Repetition counting in videos using deep learning

Master's Thesis

Dhruv Batheja

D.Batheja@student.tudelft.nl

TU Delft

AIIR

Repetition counting in videos using deep learning

Master's Thesis

MASTER OF SCIENCE
in
COMPUTER SCIENCE

Dhruv Batheja
d.batheja@student.tudelft.nl
4716523

Thesis committee:
Prof. dr. M.J.T. Reinders, TU Delft
Dr. Jan van Gemert, TU Delft
Dr. Cynthia Liem, TU Delft

TU Delft

Preface

This report contains the background work for my thesis which is repetition estimation in videos using modern deep learning techniques. The goal of this project is to formalize and implement an end-to-end trainable model that can estimate repetition in a variety of videos. The educational goal of this project is to research and implement an architecture from scratch that can tackle the task at hand. This report is aimed at anyone wanting to research on the subject of repetition estimation using visual time-series data.

The last two years at TU Delft have been quite an experience. It has been a long two years of uphill learning curves which helped me grow personally and professionally. I learnt a lot about time management and juggling several things at a time and prioritizing what's important. I think I'm a better programmer, a better footballer and a more reliable person than I was before coming to TU Delft. I met a lot of exciting people who gave me a ton of beautiful memories. It is not possible to list everyone I want to thank for making my life easier during this arduous journey but, I will try to list some. First and foremost I would like to thank Dr. Jan van Gemert and Miriam Huijser for constantly supporting and mentoring me throughout the execution of this project. I would also like to thank Cynthia and Marcel for being part of my thesis committee. I could not have finished this report without the improvements suggested by Tavishi, Sanjeev, Miriam and Yeshwanth. Thanks to everyone in the "Delft Summer Football" facebook group and everyone in Ariston teams 13 and 15 for plenty of football memories.

There are friends, there is family and then there are friends that become family. Thank you Kanav, Sanjeev, Tavishi, Arka, Sasha, Nirmal, Gouri, Summi, Bhati and Rahul for being my homies away from home. I hope I make a lot of more memories with all of you. No matter where we are, the moments we shared are the moments we keep forever.

Abstract

This work tackles the problem of repetition counting in videos using modern deep learning techniques. For this task, the intention is to build an end-to-end trainable model that could estimate the number of repetitions without having to manually intervene with the feature selection process. The models that exist currently perform well on videos which are stationary but, realistic videos are rarely perfectly static. A series of intermediate experiments are performed to eventually come up with an end-to-end trainable pipeline. Techniques like the University of California Riverside's Matrix profile, bi-directional recurrent neural networks and convolutional neural network architectures that employ dilation are experimented with for the task at hand. For the experiments, a variety of videos from the Qualcomm and University of Amsterdam (QUVA) repetition dataset and the YouTube Segments (YTSegments) dataset are used which both exhibit a good number of non-static videos of real life scenarios like people exercising, chopping vegetables, etc. A proprietary Aircraft inspection dataset which contains repetition of spinning engine blades is also experimented with. The proposed model obtains a lower Mean Absolute Error than the existing deep learning architectures. Finally, the model proposed in this work is able to estimate repetitions on a variety of videos successfully in real time without manual intervention. On the task of repetition estimation, an accuracy of about 60% of correctly labelled frames (with repetitions so far) on unseen test videos is obtained.

Contents

List of Figures

1 Introduction

1.1. The Problem and Applications

This work tackles the problem of estimating repetition in videos. Humans are virtually surrounded by visual repetition. It is ubiquitous in a diverse set of domains like sports, music or cooking. Repetition occurs in several forms because of its variety in motion continuity and motion pattern. The perception of 3D motion in 2D depends on the viewpoint which is of course crucial for the cognizance of recurrence. Periodicity can denote a variety of movements including animal/human movements like a beating heart, breathing lungs, flapping of wings, running, etc. Movement of the blades of a windmill, spikes of a wheel, a swinging pendulum are also periodic. Periodicity or repetition is also ever-present in natural, industrial and urban environments like blinking lights or LEDs or traffic patterns or leaves in the wind. Repetition and rhythm are already used to approximate velocity, estimate progress and to trigger attention (Johansson, 1973). This has many applications in computer vision as well, like: it can be used for human motion analysis, action localization (Laptev et al., 2005), action classification (Goldenberg et al., 2005), (Lu and Ferrier, 2004), 3D reconstruction (Belongie and Wills, 2004), camera calibration (Huang et al., 2016), activity recognition (Brand and Kettnaker, 2000), object tracking (Briassouli and Ahuja, 2004) etc. Some other domains that can potentially benefit from repetition estimation include and not limited to: applications in high throughput biological-experiments, gaming and surveillance. Due to its applicability in a wide range of domains, this problem has enjoyed an increasing amount of attention.

At the semantic level, this task is actually well-defined. In fact, humans are able to perform this task without a lot of difficulty. The existing work shows acceptable results under the unacceptable assumptions of stationary and static periodicity. In real life, the video is rather complex and rarely stationary or perfectly periodic. The popular fourier-based measurement is more than often, not upto the mark in real-life applications. In practice, the movement of the camera makes this task inescapably hard. Other works try to manually perform feature engineering like foreground-segmentation, object localization, flow field calculation etc. to perform this task. This work tries to solve the problem under more natural and realistic conditions as nonstationary is actually the norm. This work experiments with three different video datasets as discussed in Section 3. A modern deep learning alternative to tackle the problem is proposed. Instead of manually engineering the features, the

proposed architecture lets the model learn from the labelled dataset that is provided to it. This end-to-end trainable architecture will only keep getting better as more labelled videos are provided to it with little manual intervention.

1.2. Research Scope

The aim of this research is to answers the following questions:

1. **How can a Recurrent Neural Net be used to estimate repetitions in a video?**
   Hypothesis: Recurrent Neural networks (More specifically its variants like Long Short Term Memory Networks (LSTMs) (Hochreiter and Schmidhuber, 1997)) are known to perform very well on tasks that involve temporal sequences because of their architecture which allows them to maintain a memory as a sequence is fed through them. Videos are temporal sequences of image frames, so theoretically, a big enough recurrent network should be able to tackle the task of repetition estimation by feeding it frame-image-pixels directly or frame-wise Convolutional Neural Network(CNN) features.

2. **How to use the University of California Riverside's Matrix Profile in conjunction with a Recurrent Neural Network to estimate repetitions in a video?**
   Hypothesis: UCR's Matrix Profile introduced in (Yeh et al., 2016) claims to make any time-series data mining task "trivial". It helps in efficiently finding motifs and anomalies in a multidimensional time-series signal. Theoretically, a matrix profile should be able to detect motifs (repeating patterns) in a multidimensional time-series signal (which videos are). If the matrix profile emits a reasonably reliable signal, a Recurrent Neural Network architecture like an LSTM should be able to estimate repetitions from it.

3. **How does the number of parameters required for the Recurrent Neural Network architecture change based on the Matrix Profile usage?**
   Hypothesis: The Matrix profile is a 1 Dimensional signal (as shown in (Yeh et al., 2016)), compared to the multidimensional frame-image-pixels and CNN-features. The number of trainable parameters required in the recurrent neural network architecture should be dramatically fewer for a variant which employs the Matrix Profile as compared to a variant where frame-image-pixels or CNN-features are directly fed to the Recurrent Neural Network architecture.

4. **Can a backpropagatable feature extraction step help improve the accuracy of our model?**
   Hypothesis: Since the computation of the matrix profile is just a series of differentiable mathematical operations (as shown in (Yeh et al., 2016)) on the raw multidimensional image-frames, it should be possible to backpropagate through the matrix profile. This could potentially result in a trained Convolutional Network that helps in computation of the Matrix Profile signal such that the computed Matrix Profile signal (using this resulting CNN's features) in turn improves the estimation of repetitions in videos.

2

Related work

Because of the numerous applications, there has been quite some research already done in this domain. There have been a variety of attempts in the past to tackle the problem of repetition estimation. These attempts can be broadly classified into a few categories as discussed below:

2.1. Frequency Domain Analysis

Most of the existing literature is dominated by methods that use the fourier transform or methods that incorporate wavelet analysis. The fourier transform is discussed in detail in the Appendix A1. These techniques employ fourier transform, and are very susceptible to background noise. The background noise from each frame in the video massively deteriorates the quality of these models, so they usually end up employing techniques of removing the background (Piccardi, 2004). The most popular way of removing the background is modelling the background as a *Gaussian mixture*. This gaussian mixture model technique leads to very precise results, but, this usually needs to employ a training stage and has a much higher computation cost than other techniques. One such cheaper technique is the *Median Filtering*. It provides a reasonable accuracy with limited memory and computation requirements. (Briassouli and Ahuja, 2007) uses median filtering for background subtraction. This paper presents an approach which performs the time-frequency analyses for the entire video at once. This allows it to extract several periodic trajectories from the video. This also allows one to estimate the separate independent periods simultaneously over a static background. The spatial domain information is used to extract the periodically moving objects. The main idea behind this paper is that the repeating patterns have unique signatures in the frequency space.

To analyze the periodic variations, a **Frequency Modulated** signal is first constructed where the frequency is tuned by the object displacements with time over successive frames in the video frame sequence (Djurovic and Stankovic, 2003), (Stankovi and Djurovi, 2001). These variations are however periodic in our case and hence the respective frequencies will also be periodic functions with respect to time. By incorporating the Time Frequency Distributions, the varying frequencies of the Frequency Modulated Signal can be obtained and thereby, the spatial trajectories can be computed. To build this Frequency Modulated signal, a technique called μ-propagation (from (Djurovic and Stankovic, 2003)) is used which

is a Fourier Transform at frequency. These Frequency-domain techniques involve an analysis which is spatially global and not local, which enables them to be effective against local illumination-variances (like in (Hoge et al., 2003)) and occlusions. These techniques don't make any kind of assumptions about the trajectory smoothness or the shape of the objects, so the results hold for a wide variety of videos and is much more reliable than intensity-based techniques like the ones proposed in (Barron et al., 1994), near the boundaries of the moving objects. The estimation of the periodic cycles using these techniques can also be employed in the spatial domain for tasks like motion segmentation. An unavoidable difficulty that arises with this technique is called the *localization problem* as discussed in (Young and Kingsbury, 1993). Even though the motion estimation by frequency-domain is global in nature and avoids local errors (like occlusion, illumination changes, object boundaries etc.), it doesn't contain insights about the actual spatial address of the objects in motion. This results in the need for further processing of the video using help from the spatial-domain to accurately allocate motion-estimates to the frame-pixels. Another difficulty is that these techniques are very susceptible to background noise and therefore can't be used without an expensive background subtraction step.

2.2. Spatial Correlation Methods

Trying to understand videos is one of the core problems of Computer Vision and has been researched upon for decades. Some of the oldest contributions in video understanding focussed on the development of spatio-temporal features from the video sequence for analysis. A majority of the past approaches for periodicity in motion analysis (like (Lu and Ferrier, 2004), (Cutler and Davis, 2000)) incorporate first detection of the object in successive image-frames and then computing their trajectories and finally analysis of the computed trajectories. More recent spatio-temporal approaches for generic motion analysis incorporate statistical shape priors and levelsets (like in (Cremers et al., 2004), (Chin et al., 1994)) or under smoothness-constraints, involve computation of the optical flow (like in (Brox et al., 2004)).

The technique proposed in (Polana and Nelson, 1997) identifies periodic repetitions in a sequence of images by initially aligning the image-frames with respect to the centroid of the major-moving-object in the video so that the object in consideration remains immobile in time. Then, a few lines which are parallel to the motion-flow of the chosen centroid are extracted. These are referred to as the *reference curves*. The spectral power is then computed for the frames along these *reference curves*. The measure of periodicity for each of the reference curves is calculated as the difference(normalized) between the sum of (highest amplitude) spectral energy frequency and the sum of the spectral energy frequencies half way through.

The method discussed in (Liu and Picard, 1998) makes an assumption of a static camera and employs background subtraction to extract motion information. The objects in the foreground are tracked and their trajectory path is approximated to a line using the Hough Transform (as shown in (Duda and Hart, 1971)). The hough transform is discussed in detail in the Appendix A1. The information from the temporal histories of each pixel is then further dissected using the Fourier analysis and the harmonic motion energy accumulated because of the periodic motion is computed. The given assumption in this case is of-course

that the background is static. This is because a non-homogeneous background will undermine the harmonic energy that needs to be estimated. This work differs from the work of (Cutler and Davis, 2000) as in that instead of individual pixels, areas of the image are considered(collection of pixels) which compose an object. Another change introduced in the latter is the use of a smooth image resemblance metric. Because of these changes in approach, the fourier analysis of (Cutler and Davis, 2000) is much simpler as the analyzed signals don't have a significant number of harmonics of the rudimentary frequency. Techniques like (Cutler and Davis, 2000) and (Wang et al., 2003) require an initial pre-processing step of detecting the foreground objects (like a person) in the video frames. *(Note: This step can be comparable to the background subtraction step in some approaches)*. Then the foreground objects are aligned to ensure spatial correlation with all frames in the sequence. (In the Fourier Domain Analysis techniques, this step was only required during the segmentation stage of the algorithm (lower complexity than the spatial-counterpart)). These frame-correlations are required for computing the similarity-plot per video. If we were to incorporate a spatial-only approach, the similarity plot is compared with a dense enough lattice (to make sure that the repetitions are not missed). In such cases, prior knowledge or heuristics are required to get reliable estimates of the periods.

The spatial-correlation methods can be further classified into two kinds of methods:

2.2.1. Point correspondence

Point correspondence methods have been employed in several applications like, for computing arm and leg trajectories((Seitz and Dyer, 1997)), for gait analysis ((Tsai et al., 1994)) etc. In a controlled environment setting, the point correspondences can be detected by laying out reflective markers on the objects in motion and then following their positions in the respective frames in the video sequence (as shown in (Polana and Nelson, 1997)). This results in a very precise trajectory information for the moving object or moving parts in the object. The period can be effectively estimated from this quality-data. But, real-life is not a controlled environment setting. Most of the times, it is not possible to have reflective markers placed on the moving object (e.g. in medical imagery). In such cases, the point correspondences need to be extracted from the available images as features. The methods of computing feature correspondences are highly susceptible to illumination changes, local occlusions, reflectance etc. The computation is also very expensive. Because of these problems, this method is deemed unreliable and not generally applied in practice.

2.2.2. Region Correspondence

In the region correspondence techniques, the correlation is computed for succeeding image-frames of a video and a **similarity plot** is computed (as shown in (Cutler and Davis, 2000)). Whenever an object behaves in an oscillatory/repetitive manner, say with a period of T, its periodicity can also be observed in the similarity plot. Periodicity can be assumed to be strict oscillations around a fixed state. Because of this assumption, these methods are ineffective whenever the periodicities are coupled with other motions like translation. These actually cover several useful scenarios like walking/running etc. Because of this assumption, an initial pre-processing step is mandatory which involves frame alignment based on the moving-object such that the object is stationary in the video(the frames are aligned in

such a way that the moving object is brought to the same position in each video frame), before we move on to finding correlation to compute the respective periodicities. But, this localization of objects moving periodically also needs prior knowledge and heuristics. These area-based techniques have several advantages when compared to the pixel-based techniques like enabling analysis of the robustness of an entire object which is simply not possible in pixel-based techniques. Another advantage is that these allow analysis of motion(periodic) which is not parallel to the image-plane. Whenever there is no periodic pixel variation, the pixel-based techniques won't work at all. An example of this scenario is a person walking directly towards the camera.

All of the spatial-correlation methods work in steps. The initial steps being extracting the motion trajectories using the spatial-statistics and then determining if they are repeating. These approaches are usually based on feature-matching which is unfortunately often not feasible, computationally expensive or unreliable. Placement of markers, or using feature correspondences also needs pre-processing and/or prior knowledge. Finally, these point and region correlation methods are also sensitive to noise and may lead to results which are unreliable if the quality of the videos is poor.

2.3. Approaches that define Crystallographic groups

This is a more recent approach of tackling the problem of repetition estimation which involves defining fixed cases for perception of 3D recurrence in a video comprised of 2D images (not to be confused with 3 color channels in the 2D image frames). (Liu et al., 2004) was the first work that came up with the idea of defining crystallographic groups. The main contribution of this work was that in an N-dimensional euclidean space, a finite number of crystallographic-groups(symmetry-groups) can represent an infinite number of periodic repeating patterns. The authors of this work argue that in the 2D space, there are a total of seven groups which describe the patterns(monochrome) that show signs of repeating occurrence along a single direction and seventeen groups for patterns that can show signs of recurrence along two linearly independent directions. They came up with a set of approaches to classify a periodic pattern to one of the underlying symmetry-group, it's underlying lattice and also computing the motifs.

The work published by (Runia et al., 2018) then came up with an approach that defined 18 fundamental cases of perception of 3D recurrence in 2D videos. These cases can be observed in Figure 2.1. This work builds upon the wavelet transform to handle the scenarios of non-static and non-homogeneous video environment settings. They use the flow fields and their partial differentials to extract three kinds of fundamental types of motions and three kinds of motion-continuities of inherent periodicity in 3 dimensions. This leads to a total of 9 cases. Finally, based on the 2 viewpoints (front/side) there can be a total of $9 \times 2 = 18$ fundamental cases. This work tries to effectively tackle the issues caused by camera motion and a variety of repetitions in videos by constructing a rich set of flow-based signals that vary over time. These are computed over the object segmented out from the background. Specifically, the authors measured the flow field (average-pooled) $\mathbf{F} = (F_x, F_y)$ and the respective differentials. Then, the ΔF can be estimated by calculating $\Delta_x F_x$ and $\Delta_y F_y$. The differentials are calculated by using large Gaussian derivative filters(filter size of 13×13) to get a measurement that is global in nature over the segmented foreground.

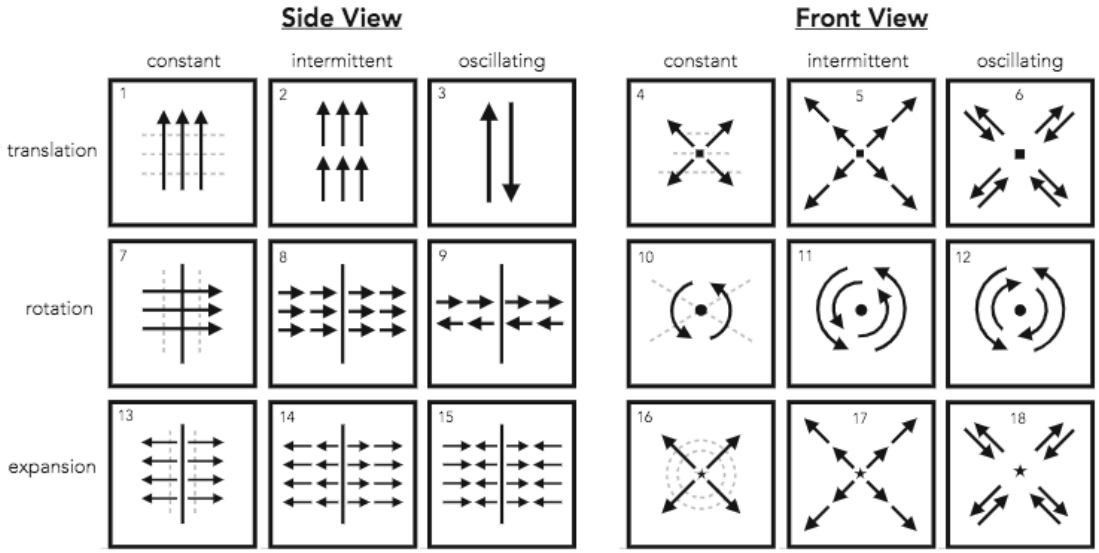

Figure 2.1: 18 Fundamental cases of recurrence-perception from (Runia et al., 2018)

Hence, the eventual measurement is a region with a small radius around the center of the object of interest. This paper relies on the motion segmentation by (Papazoglou and Ferrari, 2013) with minor modifications. They compute the dense flow field by incorporating EpicFlow (as proposed in (Revaud et al., 2015)). This work handles complex camera motion scenes and complex appearances quite well but it requires several expensive feature engineering steps like background segmentation and computation of flow fields.

2.4. Approaches that use Deep Learning

Deep learning is the use of Neural Networks as proposed in (Hinton et al., 2006). Modern deep learning techniques have added a big boost to the already rapidly evolving field of Computer Vision. Because of Deep Learning, a variety of new applications of the existing vision techniques has emerged and is rapidly becoming a part of our daily life. The introduction of AlexNet in (Krizhevsky et al., 2012) marked a huge breakthrough in the field of still-image recognition. There has been a huge interest and steady series of brilliant advances driven by innovations like multi-scale convolutions (introduced in (Szegedy et al., 2015)), use of smaller spatial filters (introduced in (Simonyan and Zisserman, 2014)), residual learning (introduced in (He et al., 2016a)), dense network connections (introduced in (Huang et al., 2017)) and many more. On the darker side, it can be argued that the domain of video processing has not yet observed its "AlexNet moment" yet. The most popular video processing task is undoubtedly the task of action recognition. Although a deep network ((I3D (Carreira and Zisserman, 2017))) currently boasts of the best results in the task of action recognition, the improvement over the best hand-crafted feature engineered approach(iDT (Wang and Schmid, 2013)) is only marginal and not as impressive.

The state of the art method was proposed by (Levy and Wolf, 2015). This work processes the video online as the frames arrive. They came up with the idea of estimating cycle length and thereby looking the counting problem from the other end. They process and analyze 20 non-consecutive frames from the video at a time and estimate the cycle length over each

such block using Convolutional Neural Networks. This information is then integrated over time by a separate flow. They break down the method into two major components:

- *The core system*: This system performs the actual counting. It processes blocks of 20 frames at a time. A region of interest(independent) is then computed for each block. This region of interest is resized and fed to a CNN which estimates the cycle length in the block which is pretrained on a synthetic dataset prepared by the authors(not actual videos).

- *The outer system*: This controls when to start and stop the counting process. This also integrates the output from the core system over time. The video is passed through the core system at several time scales to cover several lengths of repetitions. The entropy of the CNN predictions is used to determine when to start and stop counting the repetitions.

This approach relies on a synthetic pretraining step and an external system to manage the predictions. The results are impressive on the *YTSegments dataset* released with the paper but the dataset shows little variability in cycle length, camera motion, appearance of motion and clutter in the background. Also, this approach is clearly not end to end trainable.

(Karpathy et al., 2014) demonstrated a *slow fusion* model that increases the connectivity of the convolution layers in time and calculates the activations using temporal convolutions and not just spatial convolutions. They however couldn't find great success with this new approach and found that the networks operating on individual frames performed at-par with their approach. (Tran et al., 2018) discusses several kinds of spatio-temporal convolutions and experiments them for video analysis and specifically on the task of action recognition. They argue that 3D Convolutions over time have accuracy advantages over 2D CNNs over individual frames which have been the standard. They also show that using 3D convolutional filters into separate temporal and spatial components does yield some gains in the performance(accuracy) of the model.

Most of the methods proposed before have a handicap of relying on human-crafted rules to compute visual onsets and can only perform nicely on a small subset of the videos. This work by (Xie et al., 2019) aims to extract a variety of features including original-frame-pixels, frame residuals, body-pose, scene-change and optical flow and feed them to an end to end trainable network as input features. This network then re-aligns the misalignment between all the computed features using a proposed feature alignment layer. These aligned features are then further fed to a Bidirectional LSTM (introduced in (Graves and Schmidhuber, 2005)) and Conditional Random Field layers which label the sequence (as shown in (Ma and Hovy, 2016)) with their respective onsets. This approach is definitely a step in the right direction of an end to end trainable model. But the computation of several features in the first step relies on manual feature engineering and is not feasible to implement this for a real-time setting because of the high computation cost associated with each of the feature extraction steps.

3

Datasets and Preparation

This research only focuses on the visual information present in videos. The visual information in the videos boils down to a time-series of image frames. For instance, A 60-fps video has 60 image frames in each second of video.

3.1. Datasets

The following datasets were used for the experiments conducted in this research:

3.1.1. YTSegments dataset

The YouTube Segments dataset collected by (Levy and Wolf, 2015) is a collection of 100 videos mostly gathered from YouTube. This dataset contains a good mix of domains from which the videos are obtained like cooking, exercising, living animals etc. The dataset is prepared such that it is pre-segmented to only contain the repeated action. This dataset boasts of good variability in number of repetitions per video. The videos in this dataset don't display a lot of camera motion in most of the videos. The annotations for this dataset is just a total count per video. The temporal bounds per repetition in each video is not annotated. Preprocessing and manually annotating the temporal bounds of repetitions for these videos was required. Frames from a ball hopping video from this dataset can be found in Figure 3.1.

3.1.2. QUVA Repetition dataset

The Qualcomm and the University of Amsterdam Repetition dataset published with (Runia et al., 2018) is a collection of a wide variety of videos which exhibit periodicity like rope-skipping, cutting, swimming, stirring, music-making, combing etc. They collected the original untrimmed videos from YouTube. They assigned the task of labelling the intervals per-video to human annotators. After obtaining the annotations, good agreement was observed between the annotators. The videos for which there was a high overlap among the annotators were published for improving the quality of the dataset. They finally came up with the intervals by taking the intersection of temporal annotations. The repetition-counts per video and temporal bounds of each repetition in that video are finally obtained. The

Figure 3.1: Random frames of a ball-hopping video from the YouTube Segments dataset

authors argue that this is a higher quality dataset as compared to the *YTSegments* dataset proposed by (Levy and Wolf, 2015). This boasts of more variability in cycle length, camera motion, appearance of motion and clutter in the background. This dramatically increases the difficulty in both temporal dynamics and the complexity of the scene. This is a much more realistic and difficult benchmark for repetition estimation in videos. This dataset has a collection of 100 videos and it as video wise annotations that contain the frame numbers which indicate the end of a repetition cycle from the video frames. Frames from a fitness-video from this dataset can be found in Figure 4.12.

3.1.3. Aircraft inspection dataset

Another dataset that is used for the experiments is **Aircraft inspection dataset from Aiir Innovations (aiir.nl)**. During large overhauls or whenever an irregularity has been spotted by sensors during a flight, each stage of the engine is manually inspected by the mechanics to look for any anomalies. While the mechanic inspects each stage of the engine, they have to inspect each turbine blade and depending on the engine type and the engine stage currently being inspected, the total number of blades differ. Currently, the mechanic has to manually keep track of the number of blades that have been inspected to ensure all blades have been inspected. This dataset contains videos from such inspections. The dataset available has inspection videos from various angles and for different stages in the engine. This contains a collection of over 100 videos. As existing annotations, this dataset just contains the total number of repetitions per video. The annotations for temporal bounds of each repetition per video doesn't exist for this dataset yet. For video datasets like actions datasets, the repetitions are defined very well. For this dataset, it is non-trivial because one repetition depends on when you define to be the start of that repetition. Some utility scripts were built to make manually annotating the temporal bounds of the cycles in these videos much easier. Frames from a sample blade-inspection video from this dataset can be found in Figure 4.5.

3.2. Video preprocessing and Labelling

A video is a time series of image frames with an audio signal embedded into it. For the task of repetition estimation, this work will not analyse the audio signal and only consider

the visual aspects of it. So a video can be assumed to be a sequence of images as shown in Figure 3.2. Each image in this sequence is a grid of pixels. These pixels have 3 channels (values) which correspond to the intensity of the primary colors *Red, Green and Blue* as shown in Figure 3.3. These values can be in the range $[0, 255]$. Black has 0 in intensity for all the colors, so a black pixel would have R/G/B as 0/0/0. Similarly, a purely red pixel would have the intensity values 255/0/0.

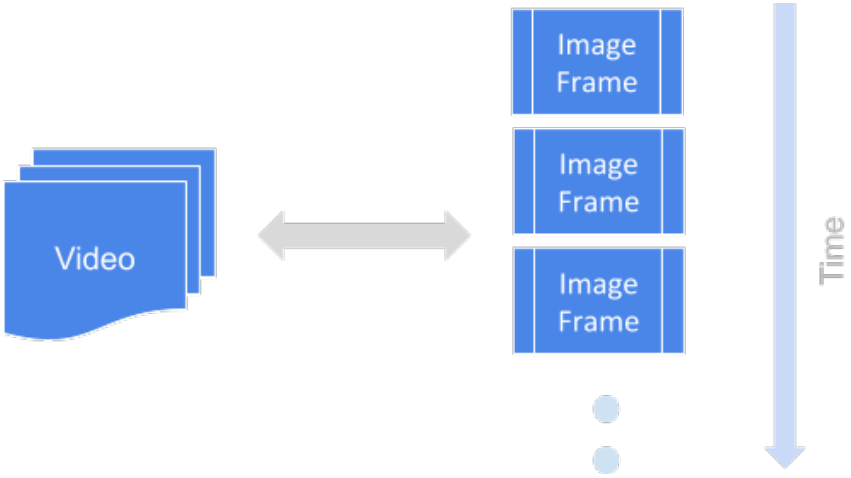

Figure 3.2: A video represented as a sequence of images

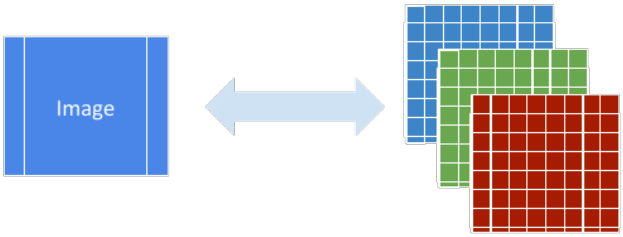

Figure 3.3: Image represented as a 3-channel grid of pixels

FPS or frames per second determines how many image-frames are present in a second of video. A high fps indicates many frames per second and hence a smoother video. As the fps increases, the size of our dataset also increases. This would mean that more information would need to be processed by the model which would require longer training times and larger model size. To avoid the aforementioned problems, for the task at hand, **all the videos are resampled to 10fps** as it is sufficient to not miss any repetition.

All the pretrained CNN models expect the images to be at least $224 \times 224 \times 3$ (Height x

Width x color channels), so all the image frames in each video were also resampled to $224 \times 224 \times 3$. This height and width is also more than enough to observe any kind of repetition in the available dataset of videos. This means a video in the prepared dataset would have $(224 \times 224 \times 3 \times frames)$ dimensions.

Now, for annotating these videos, multiple approaches were experimented with.

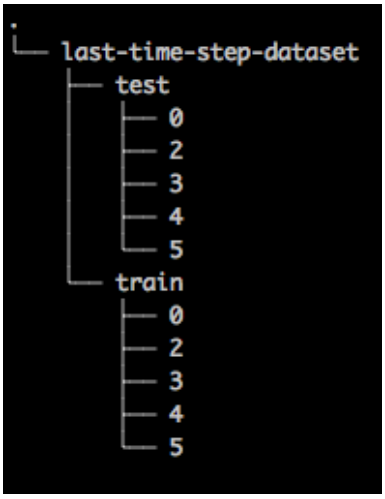

Figure 3.4: Folder structure of Last-time-step labelled dataset

3.2.1. Last time step labelling

This approach demands that the entire video is only annotated with a class label (number of repetitions in that video). The videos were labelled like this to try and backpropagate the gradient back from the last time step as discussed before. For this approach, the train and test videos are separated first to ensure that there is no overlap in the frames that were being fed to the model. So, different folders were created for train and test sets. Next, different folders were created for each of the categories (class label). For example, a different folder for (0, 2, 3, 4 ..) repetitions. Notice that there is no folder for 1 as there is no such thing as 1 repetition. (If a video has just one repetition of an event, then that event doesn't appear in the video again which implies that the video contains no repeating occurrences.) In each of these folders, trimmed videos were placed to ensure that the video had only x number of repetitions where x is the integer value of the name of the folder. It is possible to automate this as the QUVA repetition dataset has annotations with frame numbers where roughly a repetition begins/ends (onsets). For the blade-inspection dataset, a script is built which helped annotate the videos just like the QUVA dataset is annotated. A tree view of the labelled dataset folder would look like in Figure 3.4.

3.2.2. Each time step labelling

This approach demands that each frame in the video is labelled. This work came up with a novel approach of labelling the videos. The idea is that *each frame in the video is labelled with repetitions that have occurred so far.* This would mean that the initial frames in the video would be all labelled as 0. There is no such thing as 1 repetition. And after there were

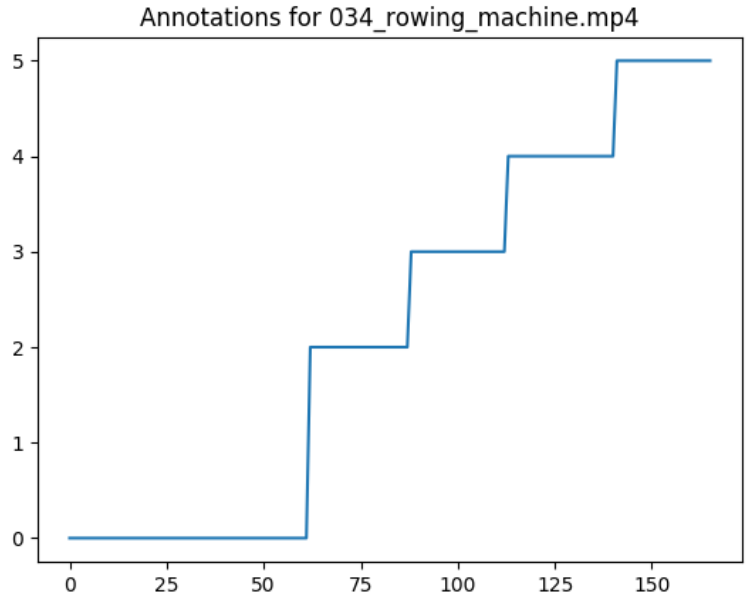

Figure 3.5: Each-time-step annotations of a video with 5 repetitions

two repetitions, the frames would be labelled as 2 and so on. The frame-wise labels for a video would look something like this [0,0,0,0,0,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5]. These annotations if visualized would look like steps. Figure 3.5 shows annotations of a video with 5 repetitions. This type of labelling would also help visualize the model prediction results in a beautiful way. The videos were labelled like this to try and backpropagate the loss back from each time step as discussed before. For both of the kinds of labelling, three separate datasets were created:

1. **Blade-inspection videos**. This dataset is a smaller subset of the entire dataset. For this dataset, the script-annotated blade inspection videos were used. This subset usually has crisp matrix profiles, so theoretically it should be easier to train a model to at least overfit on this subset. For each time-step labelling, this dataset is manually frame annotated with rough repetition occurrences just like the annotations already present for the QUVA dataset. This helps use the same automation which is used on the QUVA dataset to create each-time-step labels. This dataset has a total of 22 (manually annotated) videos.

2. **YTSegments videos**. This is a more generic dataset with a variety of videos like vegetable cutting, exercising etc. This dataset contains 100 unique videos and contained labels just like the blade inspection videos i.e. total count per video. The same annotation script is used from before(blade-counting annotation) for annotating this dataset just like the QUVA dataset is pre-annotated to have consistent annotations across all datasets.

3. **QUVA videos**. This is a more generic dataset with all kinds of videos like a guy playing tennis, a person skipping ropes, cutting cucumber etc. This dataset has more camera movement and background variation as compared to the YTSegments dataset. This

dataset contains a total of 100 unique videos. Hypothetically, it should be harder for a model to learn using this dataset because of the variability in the video-frames of this dataset. This dataset has a total of 100 (pre-annotated) videos.

3.3. Training and Testing splits

All of these datasets have a train, test and validation split of **70%, 20% and 10%** respectively. These splits are created at the beginning and ensure that the videos in each of these splits have completely different frames to avoid unreliable results. The videos from the training set will be used for actual gradient computation and weight updates in the model parameters. The videos from the validation split will be used for inspecting the loss curves and watching out for signs of over-fitting or bias. The videos from the validation split will never be used to update the weights of the model. The test split is used to compute the projected accuracy of the model. Since the model has never seen the test-set videos while training, model's accuracy on this set is a reliable estimate of the model's performance on a real world example.

4

Approach

4.1. The Matrix Profile signal

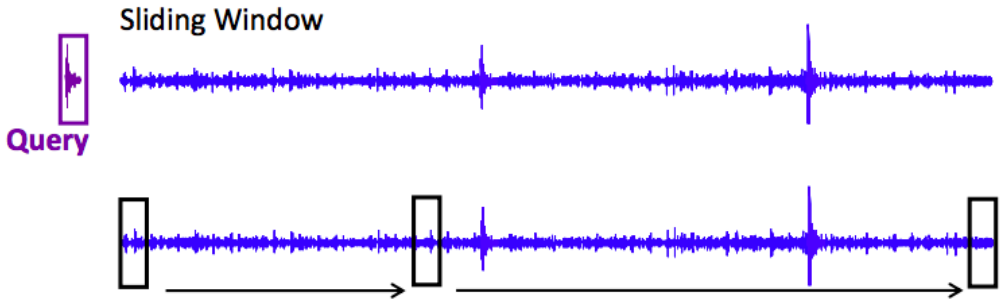

Figure 4.1: Sliding window for Matrix Profile calculation from (Yeh et al., 2016)

The matrix profile proposed by (Yeh et al., 2016) is discussed in detail in the Appendix A1. The matrix profile computation has one hyperparameter which is the window size. A window size determines how many data points(from the time-series sequence) are compared at a time. For each window in the time series, it is compared to all the other windows. Like a sliding window shown in Figure 4.1, each window gets its turn and on its turn, its distance is computed with every window in the time series. All these distance values are fill up the reference window's *column* in the distance matrix (which is shown in Figure 4.2). The type of distance used is Z-Normalized Euclidean distance.
Given two windows (time-series-sequences) x and y, where: $x = x_1..x_n$ and $y = y_1..y_n$
Z-normalized sequence $\hat{x}_i$:

$$\hat{x}_i = \frac{x_i - \mu_x}{\sigma_x} \tag{4.1}$$

Z-normalized Euclidean distance $d(x, y)$:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (\hat{x}_i - \hat{y}_i)^2} \tag{4.2}$$

.

Once the z-normalized euclidean distances of all the windows with all the other windows is computed, a completely filled in distance matrix is obtained.

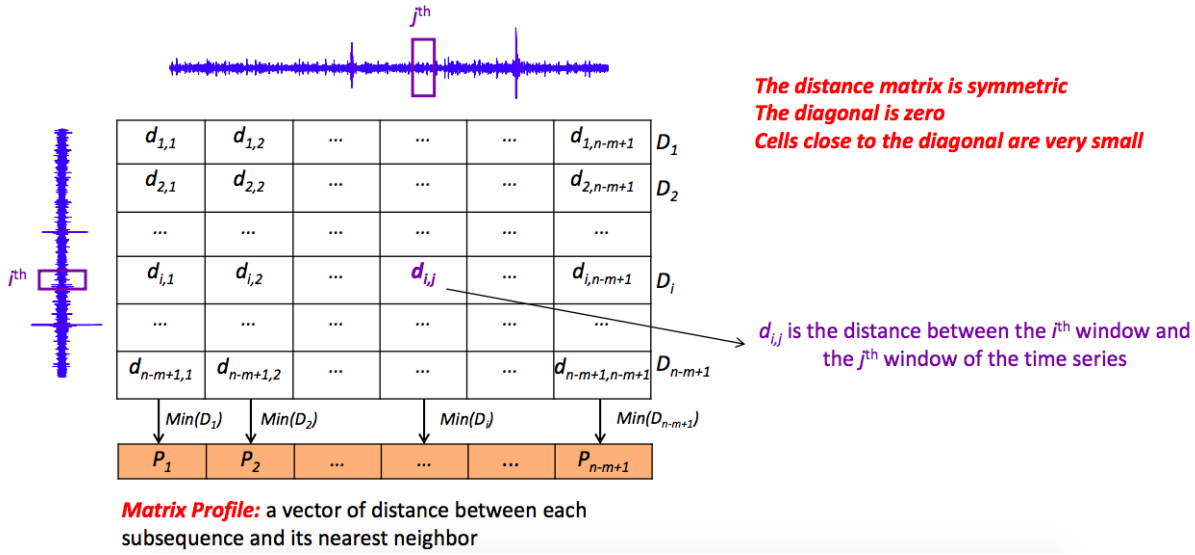

Figure 4.2: Each window is compared to every other window in the input to obtain the Distance Matrix and finally the Matrix Profile from (Yeh et al., 2016)

1. Each column in this matrix is the distance profile for that window.
2. Computing the matrix profile from this is very straightforward: Just save the minimum value from each column and there you have a 1-dimensional matrix profile.

Each item in the computed matrix profile represents the minimum distance of the corresponding time-step item in the original signal when measured against all the other items in the signal. This 1-dimensional matrix-profile is quite intuitive to look at and the peaks in the signal depict anomalies and the valleys depict motifs. There has been a lot of research done in efficient computation of the Matrix profile ((Zhu et al., 2019)) and maintaining it incrementally ((Zhu et al., 2018)).

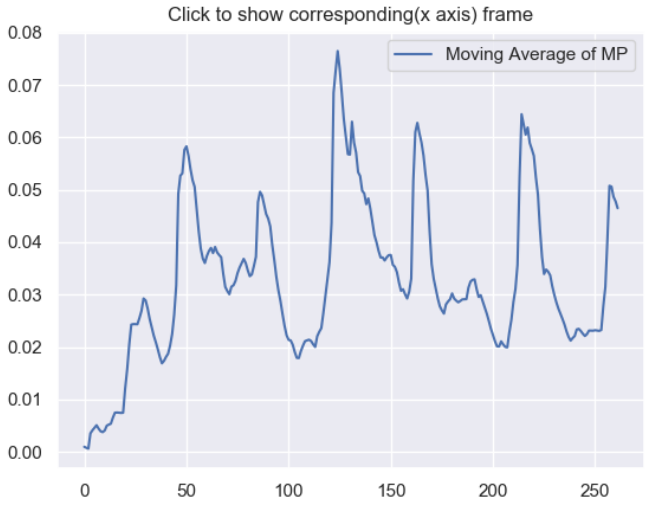

Figure 4.3: Matrix Profile for a blade inspection video (5 repetitions)

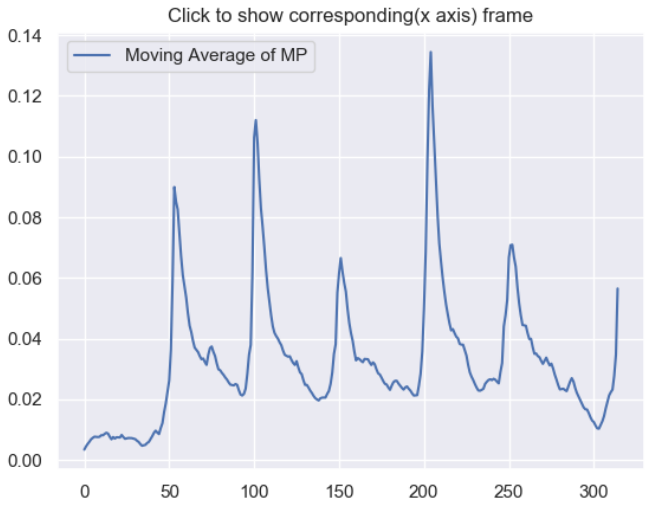

Figure 4.4: Matrix Profile for a blade inspection video (5 repetitions)

In the following experiment, some matrix profile signals are manually inspected to see if it is even possible to use it for the task at hand. The results are quite impressive for the blade counting dataset. The matrix profile looks sharp with clear valleys which indicates the presence of repeating patterns. Some matrix profiles with their respective frames can be found in Figure 4.3. The blade inspection dataset has static camera and the background usually doesn't change much. This is one of the reasons that the matrix profile output displays clear peaks and valleys. Some sample frames from the blade inspection dataset can be seen in Figure 4.5. Another crisp matrix profile for a blade inspection video can be seen in Figure 4.4.

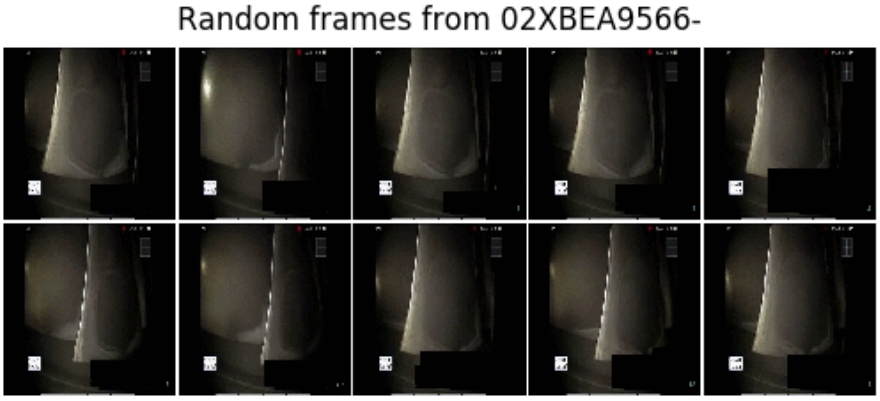

Figure 4.5: Random frames from a blade inspection video

The Matrix profile is also computed for the more generic QUVA dataset. Mixed results are obtained for this harder dataset. For some of the videos, the matrix profile output seems reliable like for chopping cucumbers video (Matrix profile - Figure 4.8, Frames - Figure 4.7)

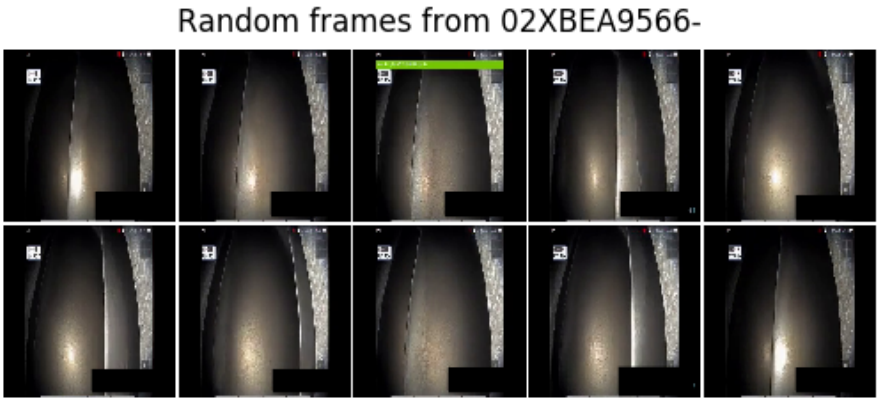

Figure 4.6: Random frames from a blade inspection video

Figure 4.7: Random frames from cucumber chopping video

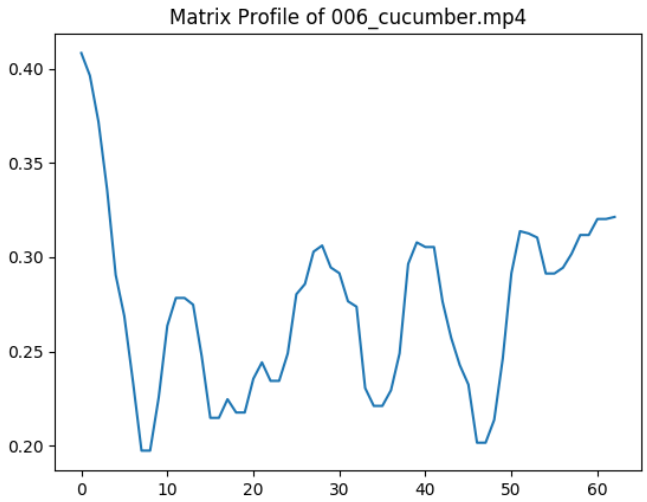

Figure 4.8: Matrix Profile for cucumber chopping video (5 repetitions)

Figure 4.9: Random frames from rowing video

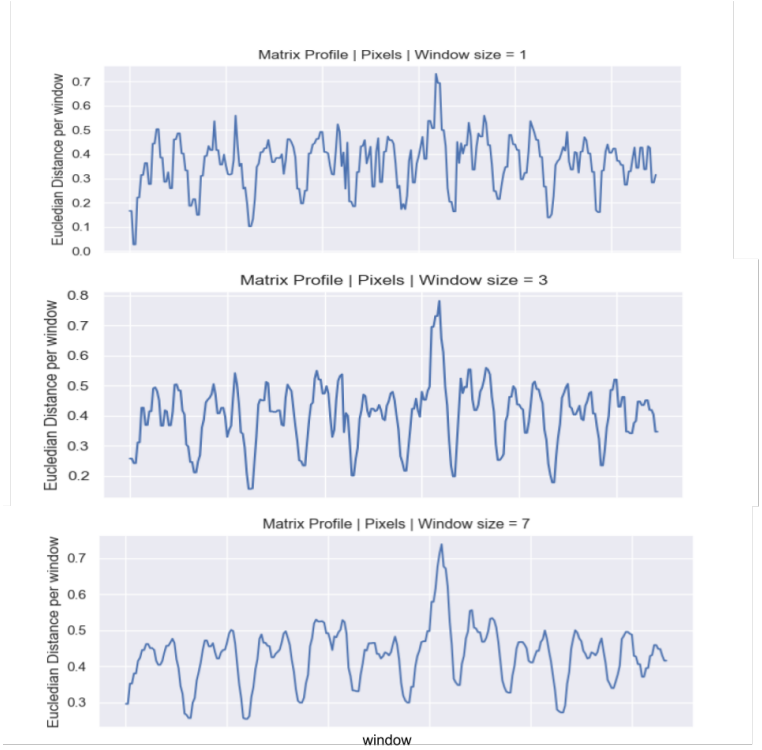

Figure 4.10: Matrix Profile for rowing video (5 repetitions)

and rowing video (Matrix profile - Figure 4.10, Frames - Figure 4.9). For both the videos dicsussed above, the matrix profile outputs clearly show 5 valleys. However, it is not at all intuitive for some videos such as for the fitness video (Matrix profile - Figure 4.11, Frames - Figure 4.12). The matrix profile output in this case shows no clear peaks or valleys while the video actually has 5 repetitions. Another interesting example is the bmx-stunt video shown in Figure 4.13. This video has an abnormal spike in it's matrix profile which is shown in Figure 4.14. The cause of this spike becomes clearer when the window-wise distance profiles are visualized. The distance profiles for the same can be seen in Figure 4.15. The spike is because the video has a few frames where there is a bright camera flash and all those frames are over exposed. One such frame is also shown in the frames of the video in 4.13. The distance profile for that window (in yellow) is much higher than all the other

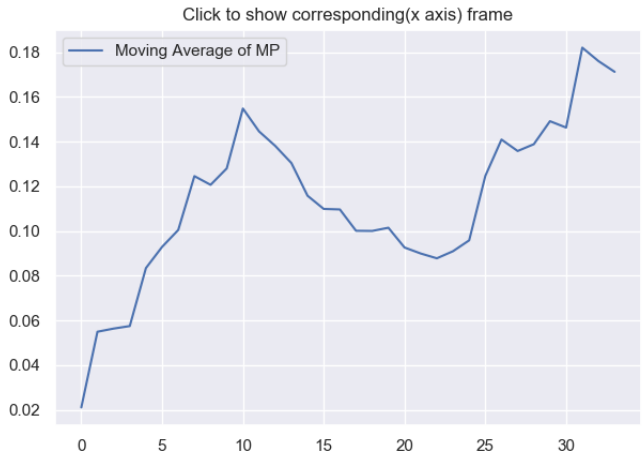

Figure 4.11: Matrix Profile for fitness video (5 repetitions)

Figure 4.12: Random frames from fitness video

profiles. Therefore, all the windows have a high distance with that window (sharp peak in the center for all distance profiles in Figure 4.15).

Figure 4.13: Frames from bmx video

The matrix profile is indeed a great feature for the task at hand but it is not always enough for all kind of videos as observed above. The reason for this is that the entire im-

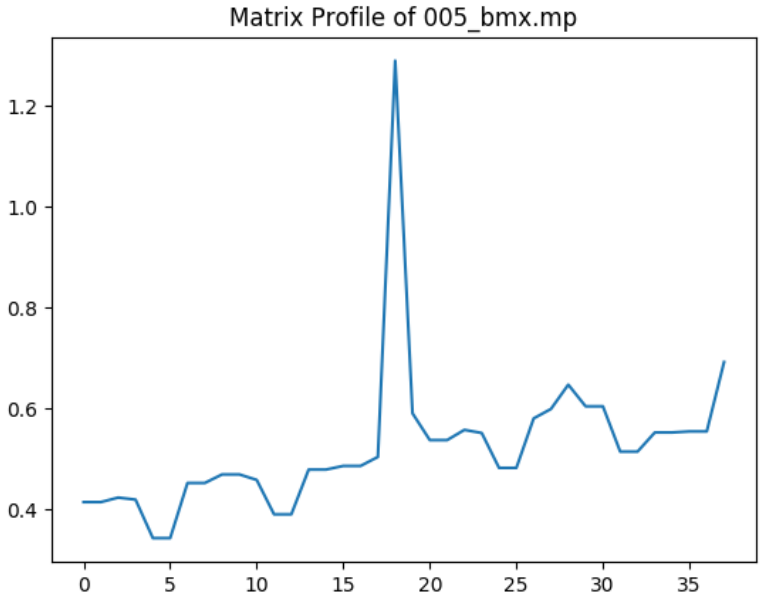

Figure 4.14: Matrix profile calculated from the bmx video

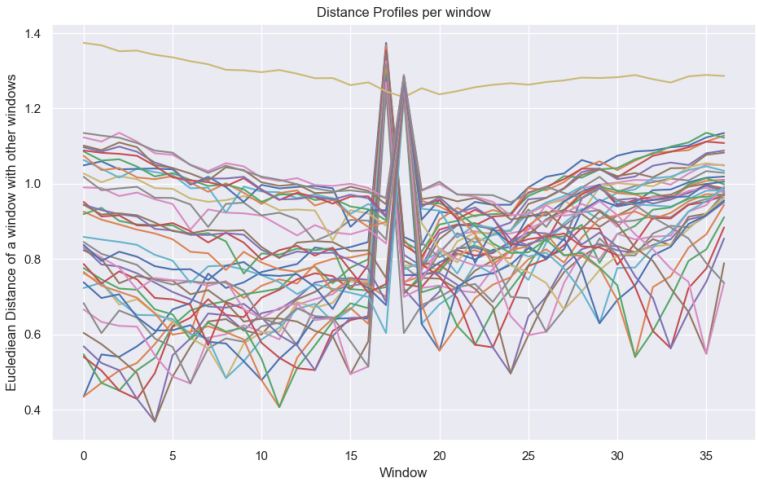

Figure 4.15: Distance profiles from the bmx video

age frame is sent for comparison in the matrix profile, and this is bound to have a lot of background information which is extra noise so, in videos where the background changes, the output of the matrix profile becomes unreliable. A possible solution to this could be to employ some background subtraction techniques to ensure that only the interest region is passed to the matrix profile from each frame (or window of frames). Since the matrix profile looks promising for most of the videos, it was decided to experiment with the matrix profile and use it for the task of repetition estimation. As discussed before, a quick intuition about the matrix profile is that if we are just able to estimate the number of valleys in the matrix profile, we would be able to estimate the number of repeating patterns in the video.

4.2. Using a Recurrent Neural Network to find peaks and valleys in the matrix profile

The matrix profile is a noisy 1D signal. Theoretically, an LSTM should be able to do well at the task of estimating peaks and valleys in a 1 dimensional signal. As a toy task for the same, this work works with a self generated synthetic 1D signal. Since a sine wave is also a 1D signal with peaks and valleys, this work experiments with sine waves as a toy task for the LSTM.

4.2.1. ToyTask: Mimicking the Matrix Profile

For creating our synthetic dataset, we first began with a sinusoidal cycle. We will refer to it as a wave-cycle in the remainder of this report.

Wave-cycles

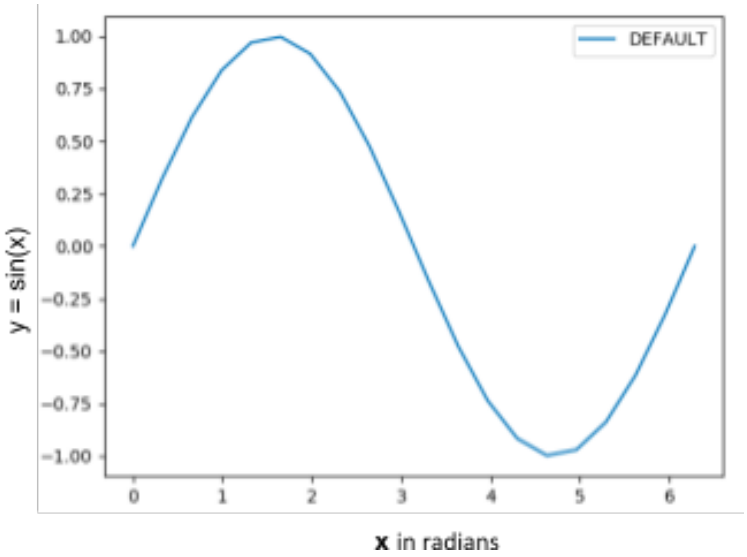

Figure 4.16: Default wave-cycle for a sine wave

A default sine wave-cycle is shown in Figure 4.16
A wave-cycle can have the following parameters:

1. G is the **Ground value**. This is simply where on the vertical axis, the wave-cycle should start. By default this would have a value of 0 as a sine wave exists at ground 0. This directly added to the value of $f(x)$

2. S is the **Scaling factor**. Scaling factor decides if the wave is squeezed or expanded in the x axis. If the stretch factor has a value greater than 1, then the wave should have larger cycles and similarly if the stretch factor has a value less than 1, then the wave will have smaller cycles.

3. A is the **Amplitude factor**. This is the height of the wave-cycle. The maximum value of a sine wave is 1, hence the default amplitude factor is set to 1. Having an amplitude factor larger than 1 will result in taller waves and an amplitude factor less than 1 will result in shorter wave-cycles.

4. P is the **Phase value**. This is the x value where the wave-cycle begins. A sine wave can begin from $-\infty$ to ∞. By default it is set to 0, so the wave-cycle begins at x=0 and goes up. If the wave-cycle begins anywhere between $\pi/2$ and $3\pi/2$, it will go down initially.

5. C is the **Crop factor**. This determines what proportion of the wave-cycle to include. It can have values in the range $[0,1]$. By default, the entire cycle should be included in the wave-cycle, so the default value for this is 1.

This wave cyle is created by using the following equations:

$$
\begin{aligned}
phase &= 0, end = 2\pi \\
phase &= phase * S \\
end &= phase + 2\pi C * S \\
f(x) &= G + (\sin(x/S) * A) \qquad \forall x \in [phase, end]
\end{aligned} \tag{4.3}
$$

.

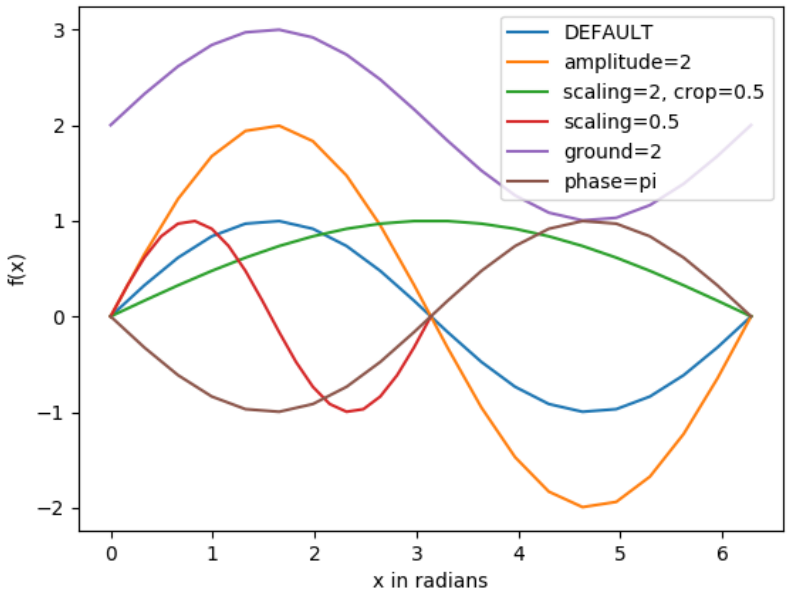

Figure 4.17: Wave cycle parameters

The default wave-cycle shown in Figure 4.16 has the default values for all the parameters: *[Ground=0, Scaling=1, Amplitude=1, Begin=0, Crop=1]*.
Please look at Figure 4.17 to have a better intuition about the wave-cycle parameters. The blue wave-cycle is the default wave-cycle. The purple cycle is identical to the default cycle, but it has a ground value of 2, so it is all shifted up in the y-axis. The orange cycle is taller than the default cycle because it has an amplitude value of 2. The brown cycle looks like the inverse of the default cycle because it begins at π instead of the usual 0. So the sine wave is shifted in the x axis by half a cycle. The red and green cycles demonstrate the effect of the scaling factor. The red wave-cycle is squeezed and the green cycle is expanded compared

to the default cycle because the scaling factors are less than one and greater than one respectively. Note that the green cycle also has a crop factor of 0.5 which is the reason only half the wave-cycle is actually created.

Stitching the wave-cycles

Not to be confused with how Fourier Transform (See Appendix A1) works, these wave-cycles are placed one after the other and are not added together for the same timesteps. The timesteps for each of the wave-cycles are non-overlapping.

"Wave-cycles in a wave are like words in a sentence" - Jan Van Gemert.

The number of data points in a wave-cycle determines how smooth the wave-cycle is. More data points in a cycle would ensure that the computed f(x) is very smooth. If we have very few data points in a given range for x, the computed f(x) would still be a sinusoidal cycle but it would be coarse and edgy. The number of data points in a cycle is kept configurable as a configurable parameter.

Now, to create a long wave from these wave-cycles, we can simply align a bunch of wave-cycles. If we ensure that the new wave-cycle begins where the last wave-cycle ended, we can end up with a smooth-looking wave. A wave-cycle can end anywhere between $G+A$ to $G-A$ (where G is ground value and A is the amplitude value for that wave-cycle). To ensure that the next wave seems continuous with the previous wave-cycle, just adjust the ground value of the new wave-cycle such that it aligns perfectly with the previous wave.

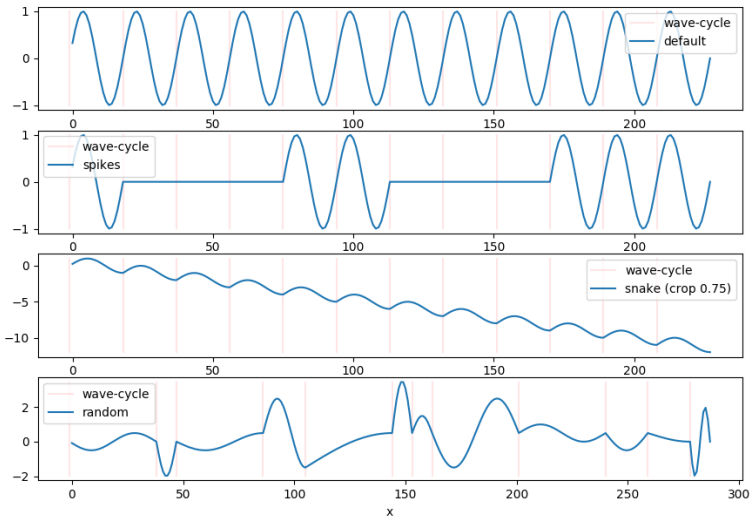

Figure 4.18: Stitched wave-cycles

Figure 4.18 shows a few stitched wave-cycles. There are 4 waves displayed in the image. The first wave is composed of all the default wave-cycles. Notice where each wave-cycle ends in the picture (marked with a transparent red vertical line). The second wave shows a

spiked wave which is created by sampling two kinds of wave-cycles:

1. Default wave-cycles.
2. Flat wave-cycles. $f(x) = G$ (G = Ground value).

The third wave is stitched from wave-cycles which have the same parameters as the default wave-cycle, apart from the *crop factor*. The crop factor in each of the wave-cycles used for this wave is 0.75. This is the reason that the wave-cycle ends at the value $G - A$ instead of ending at G. Hence the overall wave shows a downwards trend. The fourth wave shown in Figure is composed of random wave-cycles. The random wave-cycles are generated by sampling values of its parameters from the following choices:

1. **Crop factor:** can have a value from *[0.25, 0.5, 0.75, 1]*
2. **Scaling factor:** can have a value from *[0.5, 1, 2]*
3. **Amplitude:** can have a value from *[0.5, 1, 2]*
4. **Phase values:** can have a value from *[0 , π]*
5. **Ground values:** This value depends on where the last wave-cycle ended.

The waves generated can be easily subsampled to get a wave of a given *sequence length*. The sequence length in real-world would be determined by how many frames the video has. For the toy tasks, we ensured that the sequence length was never more than a few hundreds. For longer sequences, it gets harder for the gradients to flow backwards because of the problem of vanishing gradients. Vanishing gradients is a very common problem in neural networks which usually occurs when the computation graph becomes too big to effectively backpropagate useful information. Computation Graphs are discussed in detail in the Appendix A.

Labelling the stitched wave-cycles

The waves generated in the previous section can be easily labelled for peaks and valleys (local maximas and minimas). The key point in labelling such a wave is: each wave-cycle can have at most one peak and one valley. Other peaks and valleys can exist at the points where the wave-cycles are stitched. A few rules can easily determine whether the stitching point will be a peak or a valley. Each point in the wave can be assigned one of the following classes:

1. *Category-1*: No peak, no valley
2. *Category-2*: Peak
3. *Category-3*: Valley

Most of the points in the waves will be assigned to the *Category-1*. Figure 4.19 shows some labelled waves. The green vertical lines indicate presence of a peak (*Category-2* point) and the red vertical lines indicate presence of a valley (*Category-2* point) at that point.

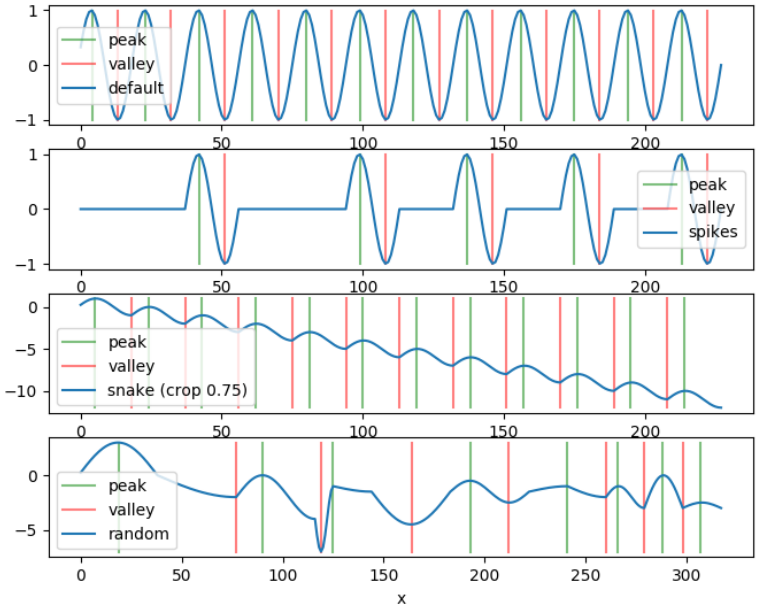

Figure 4.19: Labelled waves

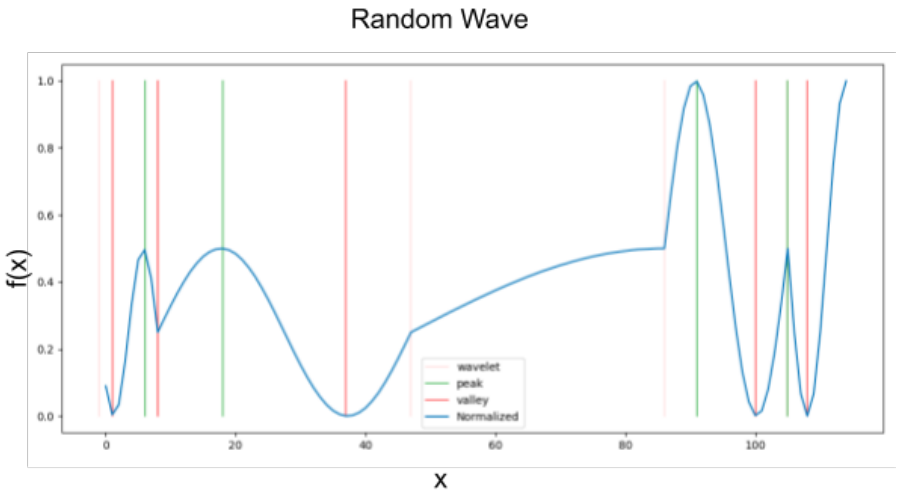

Figure 4.20: Random wave

Adding noise to the wave

The actual matrix profile is not nice and smooth like the waves generated so far. It is actually noisy and coarse (as shown in Figure 4.3). To mimic the actual matrix profile, some noise was added to the generated waves. As most of the models expect a normalized input (as suggested by (Sola and Sevilla, 1997)), the datapoints in the waves were first normalized between 0 and 1. On this normalized wave, some Gaussian noise with μ = **0,** σ = **0.05** was added. As the Gaussian distribution has 68% of its datapoints within the first σ (either side of the μ), this would ensure a maximum of 5% distortion in 68% of the datapoints in the wave. The choice of μ and σ was made after inspecting the waves to ensure the final result

looked like a typical matrix profile. Figure 4.21 shows a default wave with some Gaussian noise added to it.

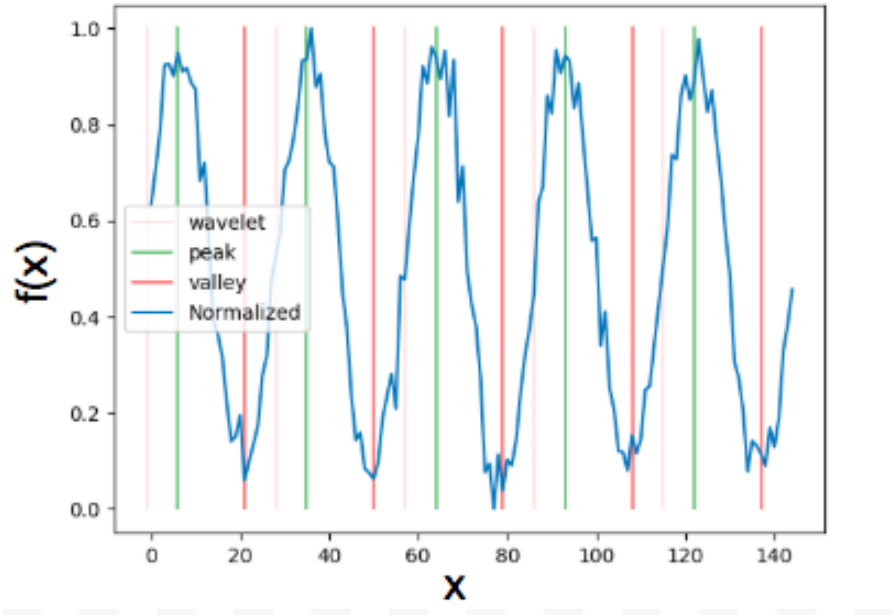

Figure 4.21: Noisy wave

Each datapoint in the dataset looks as:

1. **X** will contain the $f(x)$ values as shown in the sections above.
2. **Y** will contain the class (category) for each of the datapoints in X.

For preparing the training and validation data set, there must not be any overlap between the datasets. To ensure that, the only option was to create synthetic **random waves**. A sample random wave can also be found in Figure 4.20. The dataset has a split ratio of **70% Training set and 10% Validation set and 20% Test set**. A dataset of a total 200 waves with 160 training samples and 40 validation samples is generated. Each wave has a *sequence length* of 100 points. The sequence length of 100 points was decided as this would mimic the video dataset which after preprocessing, would have similar number of frames (at least in the same order of magnitude). This is therefore a classification task, as the model needs to predict the class for each of the points in the signal. The class can be one of the three categories (discussed above). Since there are 3 classes, it was decided to use a negative log-likelihood loss also known as the **cross entropy loss** (See Appendix A for more details). For a single training example, the loss is computed as:

$$L_i = -log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \tag{4.4}$$

The main motivation for using this type of loss is that it emits a probability score for each of the classes as the output if we employed. This makes the model training results quite interpretable and helps make informed decisions while trying to tune the parameters. For

instance, it can be observed if the model is actually learning by inspecting the class scores at the interest points (peaks/valleys).

4.2.2. Training models on this synthetic dataset

Recurrent Neural Networks

Theoretically, a recurrent neural network should be able to tackle the task of labelling the time series signal at each time step with a category (no-peak-no-valley / peak / valley). Recurrent neural networks tend to feed into themselves at each time step and maintain a small memory which helps learn patterns in the time axis. A recurrent neural network block unrolled in the time axis is shown in Figure 4.22. A simple recurrent neural network is rarely used in practice because of issues over long sequences like vanishing gradients. The most popular variant that is widely used in practice is an LSTM. It has 4 times the number of parameters in each layer compared to a simple Recurrent Neural Network. An unrolled LSTM block is shown in Figure 4.23. This architecture effectively tackles the shortcomings of a simple recurrent net. More details about recurrent neural network architectures can be found in the Appendix A.

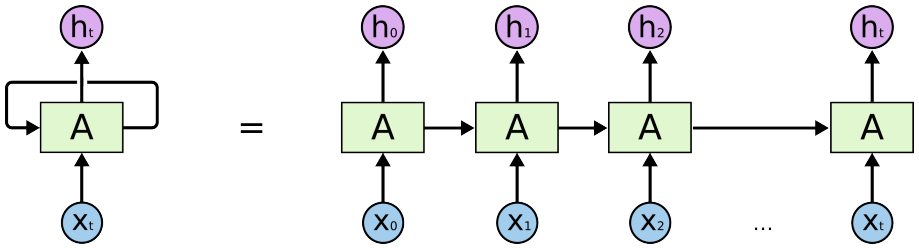

Figure 4.22: Unrolled single-layered Recurrent Neural Network from [L1]

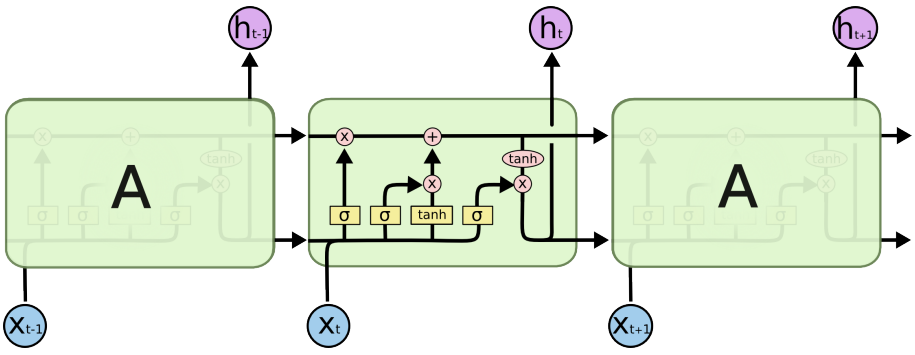

Figure 4.23: Unrolled single-layered LSTM network from [L1]

Before actually training the network, our hypothesis were:

1. Since it is a simple 1-dimensional signal, we should not require a huge model for the task of labelling each timestep with a category.

2. A bidirectional network should be able to do a better job than a unidirectional LSTM model. The reason for this is that a bidirectional model has information about the wave even from the opposite direction and hence gives it more confidence about it being in a peak or a valley at any point in time.

In this report, while talking about training of neural network architectures, **epochs** are mentioned as a metric for training durations. *One Epoch means the network is trained with all the training-samples once.* Several model sizes for the LSTM are experimented with. To ensure the experiments yield some tangible results without a lot of variance, the number of *hidden layers in the network are fixed to 2*. For the experiments, only the number of hidden units in each layer are varied. Since there exists a label (class) for each point in the time series datapoint, it is possible to backpropagate the *loss back from each timestep* prediction. This is like spoon-feeding the LSTM to learn better and faster. The backpropagation in this approach can be observed in the figure: 4.24. Notice how the loss is backpropagated at each time step when unrolled in time. The minimum sized model that does at least as

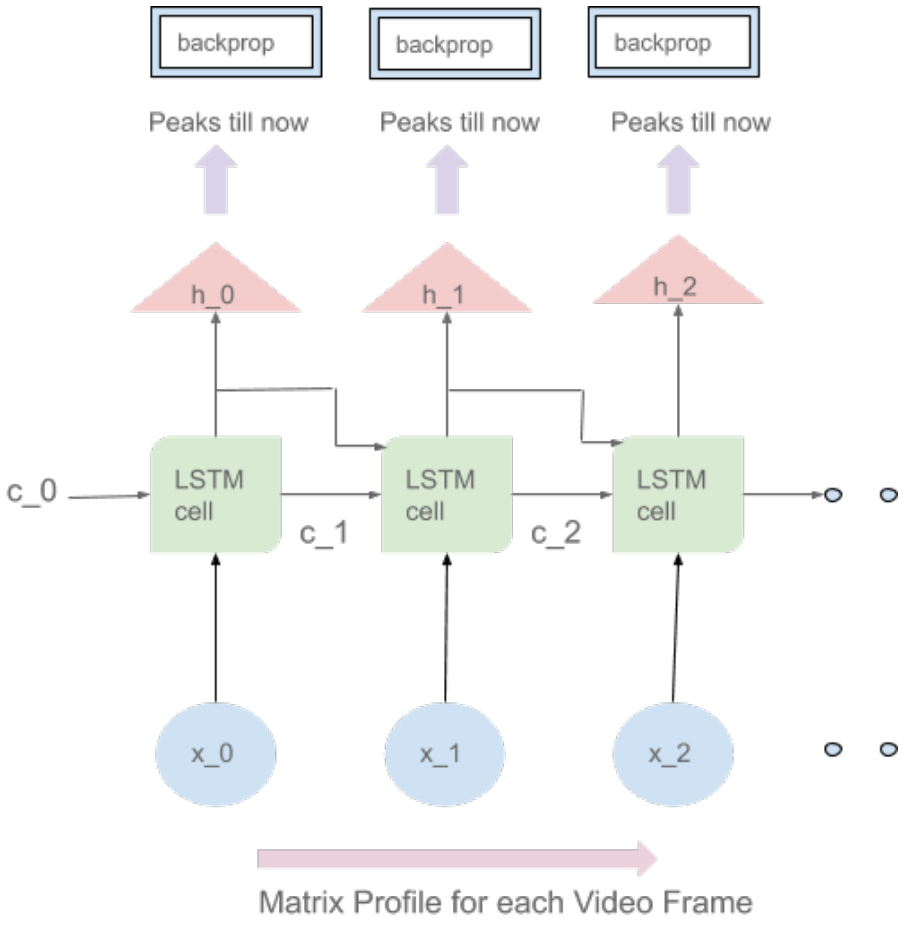

Figure 4.24: LSTM backpropagation architecture

good as the models with more number of parameters was with at most 20 hidden neurons in each hidden layer. As expected, the bidirectional LSTM does better even with fewer number of training parameters (using 10 hidden neurons in each layer). Figure 4.25 shows the training and testing losses of two LSTM architectures while training for 50 epochs. The bidirectional variant, even with fewer training parameters shows a lower *cross entropy loss*. Using this trained model for predicting classes on a test set signal is shown in Figure 4.26. The peak and valley probabilities in the output show that the LSTM is pretty confident of its predictions. The LSTM does well on this task and displays clear understanding of peaks and valleys in a 1-dimensional signal.

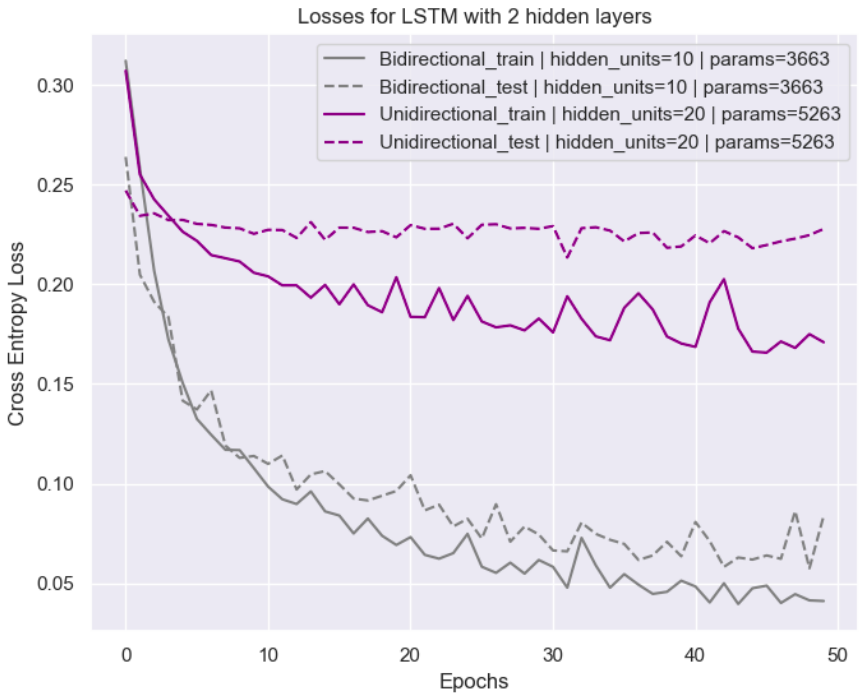

Figure 4.25: LSTM losses for 50 epochs

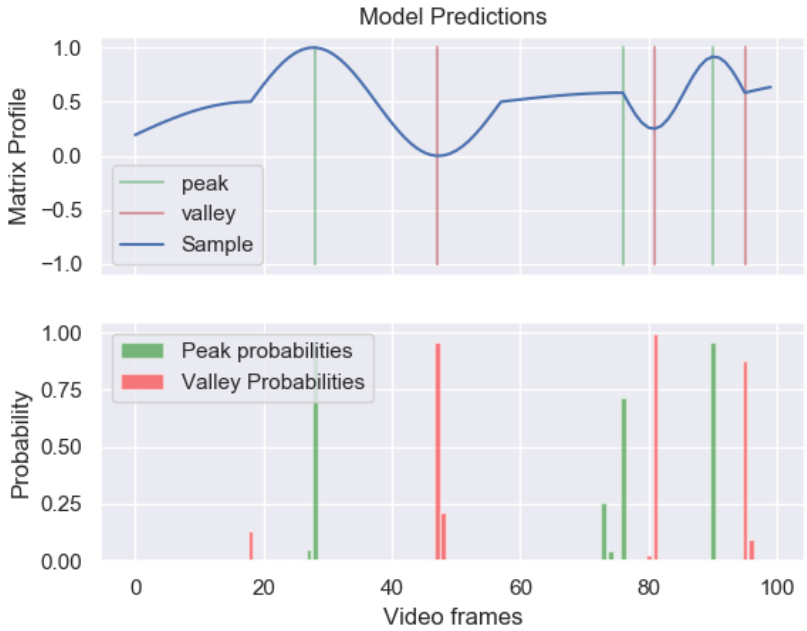

Figure 4.26: Trained LSTM prediction on a test set signal

Convolutional Architectures

Recently, there has been a lot of research done on using convolutional architectures for time series tasks as well. The main motivation for this is that the recurrent neural networks need to be fed the time series sequence sequentially which leads to a network that is

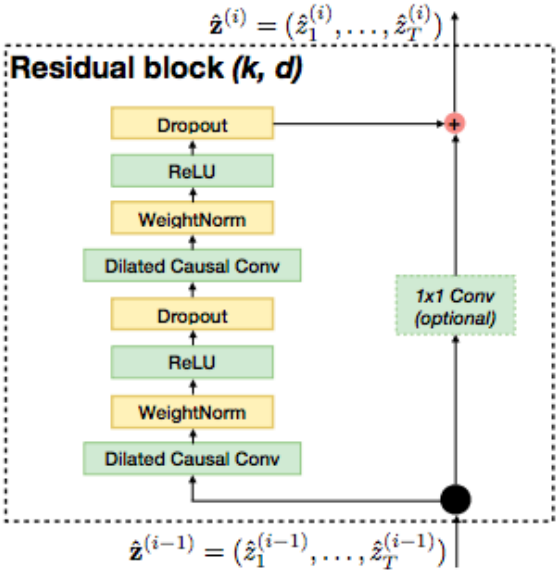

Figure 4.27: Building block of TCN from (Bai et al., 2018)

much slower to train. Convolutional architectures look at the entire sequence at once and hence the computations can be parallelized and are much faster to train. These architecture usually employ convolutions with dilations as inspired by WaveNet (Oord et al., 2016). One such architecture is the Temporal Convolutional Network (TCN) (Bai et al., 2018). The Temporal Convolutional network is composed of several residual blocks (with skip connections). One such block is shown in Figure 4.27. As depicted in Figure 4.27, the activation function used in a TCN block is RELU. It was observed that for the task at hand, a variant of RELU, specifically ELU did much better than RELU. More details on RELU and its variants can be found in the Appendix A. A bidirectional version of a TCN is also implemented for the experiments as the bidirectional variant does much better than a unidirectional variant in case of LSTMs. The training and testing loss values of the TCN are shown in Figure 4.28. This network is also trained for 50 epochs. The cross-entropy losses for the regular TCN and the bidirectional variants do not show a lot of difference. The losses however, are still much more than the LSTM counterparts. This effect is emphasised when the predictions on test signals is observed. A couple of pretrained TCN predictions on test-signals can be observed in figures 4.29 and 4.30. The probability curves show beautiful representation of learning process, but the probability values are not nearly as confident as the LSTM counterparts which are crisp and very confident in their predictions.

The difference in loss values (between Figure 4.25 and Figure 4.28) and the difference in prediction confidence values (between Figure 4.26 and Figure 4.29) makes it clear that an LSTM is better suited to the task. These experiments showed that an LSTM can be trained to estimate peaks and valleys in the matrix profile. These also provide an intuition about what size of the model should be sufficient for the task at hand. This model, just pretrained on the synthetic sinusoidal dataset is tried on an actual matrix profile of a blade inspection video. One such result can be seen in Figure 4.31. This video has 7 repetitions and the model predicts it correctly out of the box. This model already does well for videos which

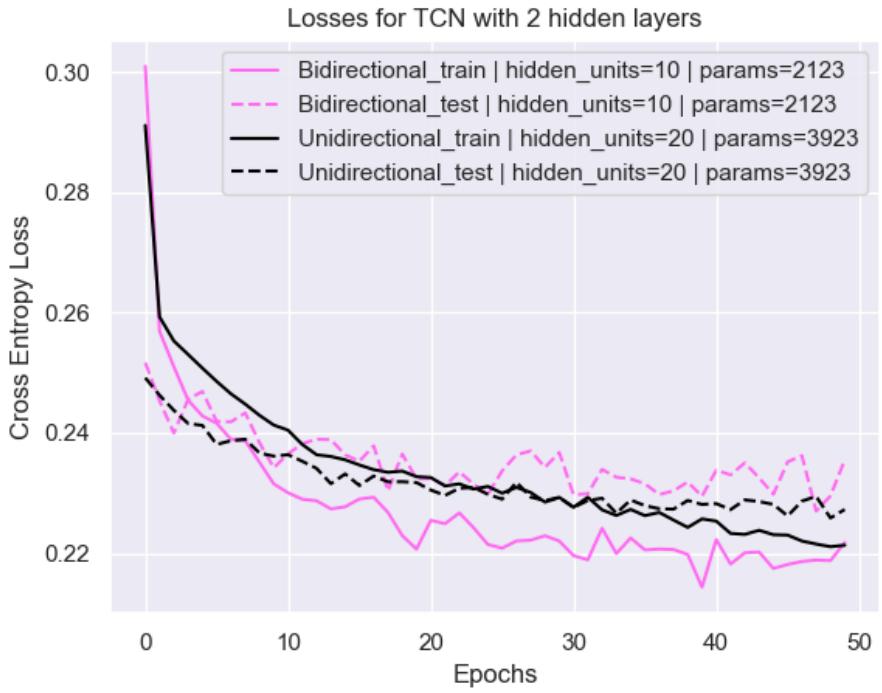

Figure 4.28: TCN losses for 50 epochs

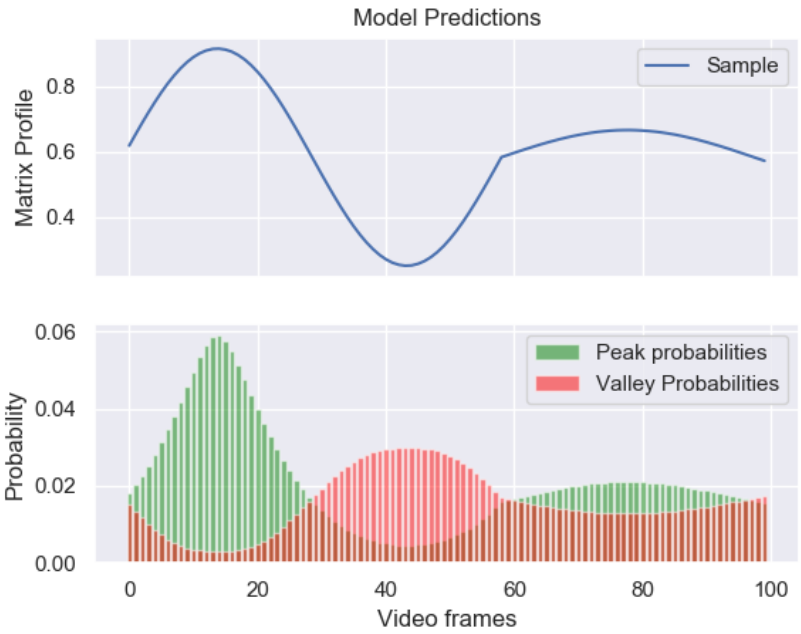

Figure 4.29: Trained TCN prediction on a test set signal

have a crisp matrix profile. These experiments show that a relatively small LSTM architecture (with only 3000 trainable parameters) is sufficient for estimating peaks and valleys in a 1-dimensional signal. The bidirectional variant performs better than a unidirectional variant even with fewer learnable parameters.

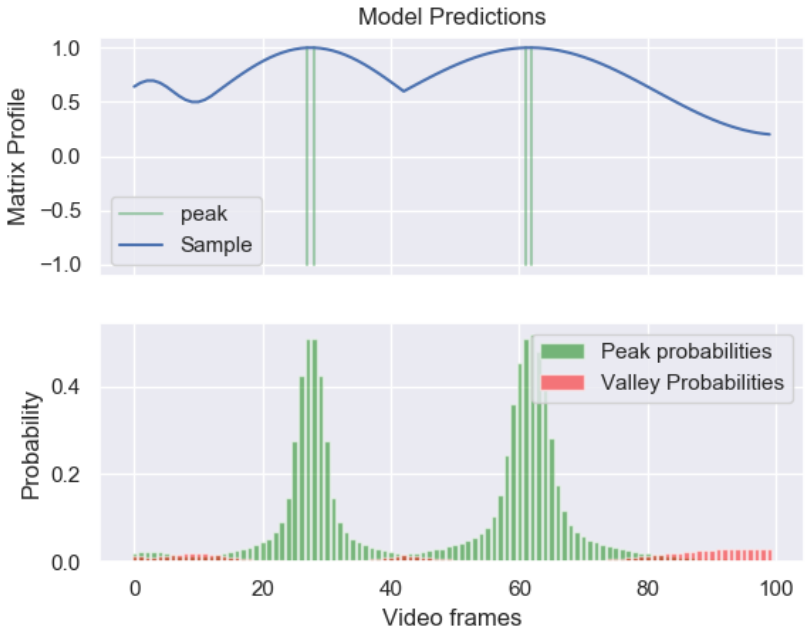

Figure 4.30: Trained TCN prediction on a test set signal

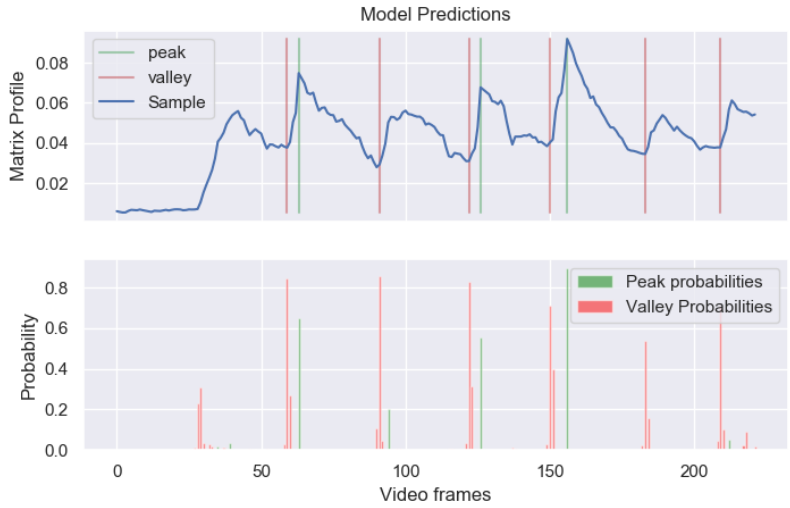

Figure 4.31: Trained LSTM prediction on a Matrix Profile (7 repetitions)

4.2.3. Backpropagating loss from the last time step

The number of peaks and valleys is a good estimate of the number of repetitions but, where exactly does the valley lie is a problem. It is not feasible to manually label the matrix profiles with peaks and valleys for all the videos. For the datasets available, there is either a count per video or rough frame numbers where the next repetition occurs. Some videos do not even have a good usable matrix profile as shown in Figure 4.12. To tackle this problem, it is clear that relying on labelling the matrix profile with peaks and valleys is not feasible. One way of labelling the videos is to have a count (class) per video. So each frame in time-series is not labelled with a class, but the entire sequence is labelled with a single class.

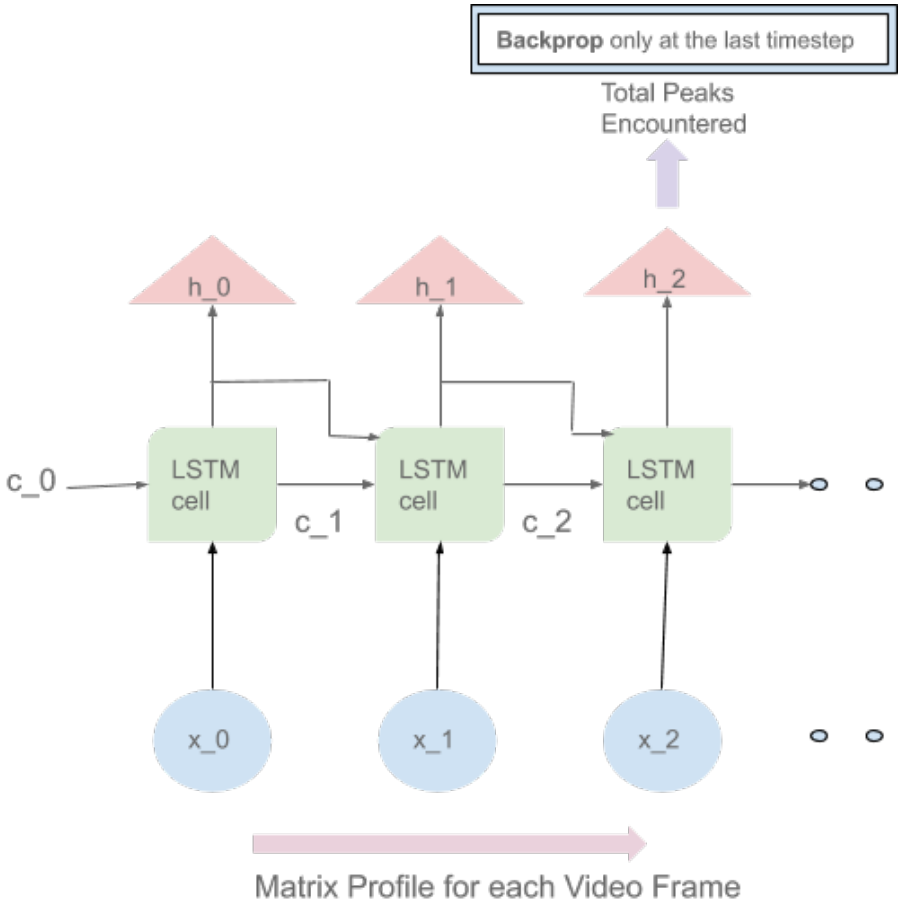

Figure 4.32: New LSTM backpropagation architecture

This means that the loss would have to be backpropagated from the last time step in the sequence. The new architecture would look something like in Figure 4.32. Note how it is different from the previous architecture shown in Figure 4.24. A network of the same size as our previous experiments with 20 hidden neurons in each layer and 2 hidden layers was used. A synthetic dataset was prepared for training this new architecture, which has sequences of fixed length (50 for our experiments). A quick estimate is that we resampled all the videos in our video dataset to 10fps, so, 50 timesteps means 5 seconds of video. Each sequence in this set would have from 0 - 5 peaks/valleys at varying positions in the signal. This was prepared using *default* and *flat-waves* with varying points-per-wave-cycle as discussed before in the wave-cycle preparation section. A sample signal is shown in Figure 4.33. The respective train and test sets were created to ensure there was no overlap. This would be the class label of the entire sequence which would be backpropagated from the last time step.
The hypotheses for this setup were:

1. It should be much harder for the network to learn any patterns, since the loss is only propagated backwards from the last time step. It should also get increasingly difficult for the model if the sequence length increases.

2. This setup should take more time and iterations to train.

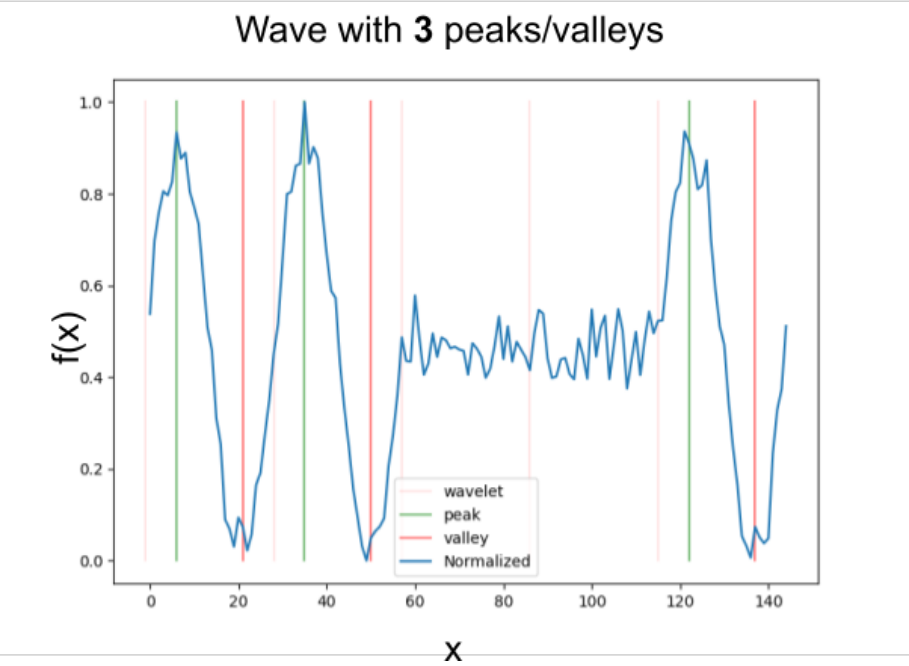

Figure 4.33: New dataset sample for Backpropagation from last step

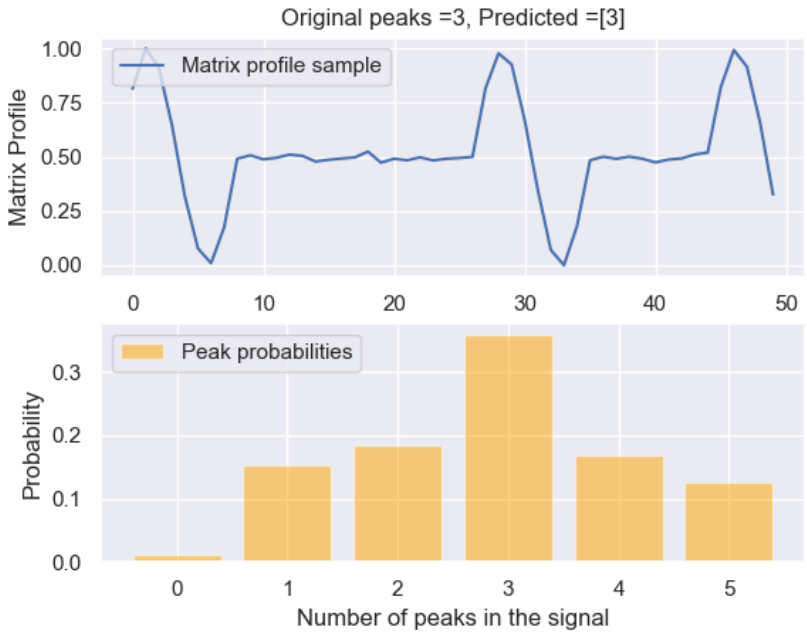

Figure 4.34: Good prediction on a last-time-step sample

As expected, the model's learning was much slower than the previous approach of LSTM spoon-feeding. The prediction results are mixed, and the confidence in the true class is not very high whenever the model makes correct predictions. A few good and bad predictions can be observed in the figures 4.34 and 4.35 respectively. It shows that it is harder to make a network understand the problem if we only backpropagate the loss from the last time step. The results were not completely unsuccessful. So, this architecture is also experimented

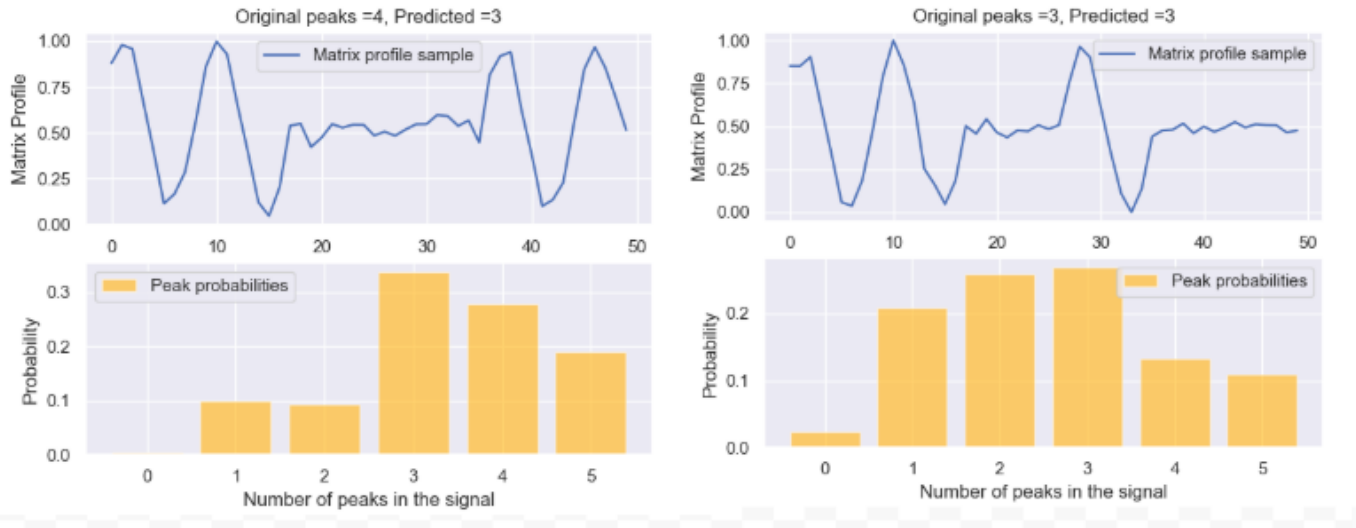

Figure 4.35: Average predictions on a last-time-step sample

with in the end-to-end pipeline.

4.3. Backpropagating through the Matrix Profile

Figure 4.36: Frames from rowing-machine video

Based on the synthetic wave toy tasks discussed in this chapter before, it is clear that a specialized recurrent neural network architecture, like an LSTM, can tackle this problem of estimating repetitions in videos. Before building an end-to-end trainable pipeline, it needs to be ensured that the computation of the matrix profile in itself is backpropagatable (the gradients should be able to flow through it). As a part of this project, a PyTorch implementation of the matrix profile computation was written which is backpropagatable.

4.3.1. Varying window size

The new backpropagatable implementation is also flexible to have varying window sizes. Our hypothesis regarding the window-size was that as we increase the window size, a range of frames would be compared at once, and that would result in a smoother matrix profile. Visualizing the results proved this hypothesis. A sample rowing-machine video was

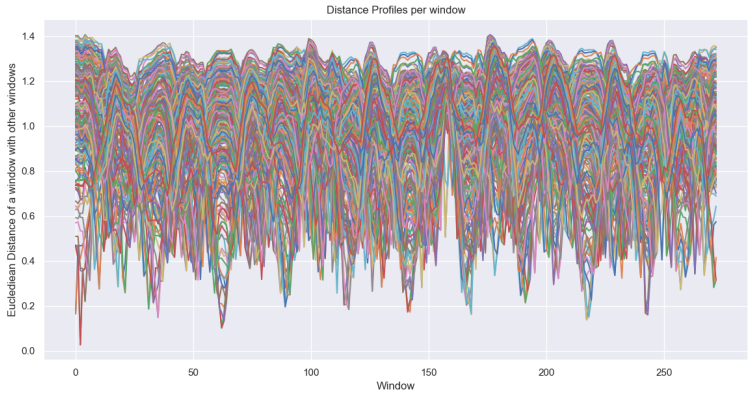

Figure 4.37: Distance profiles from rowing-machine video

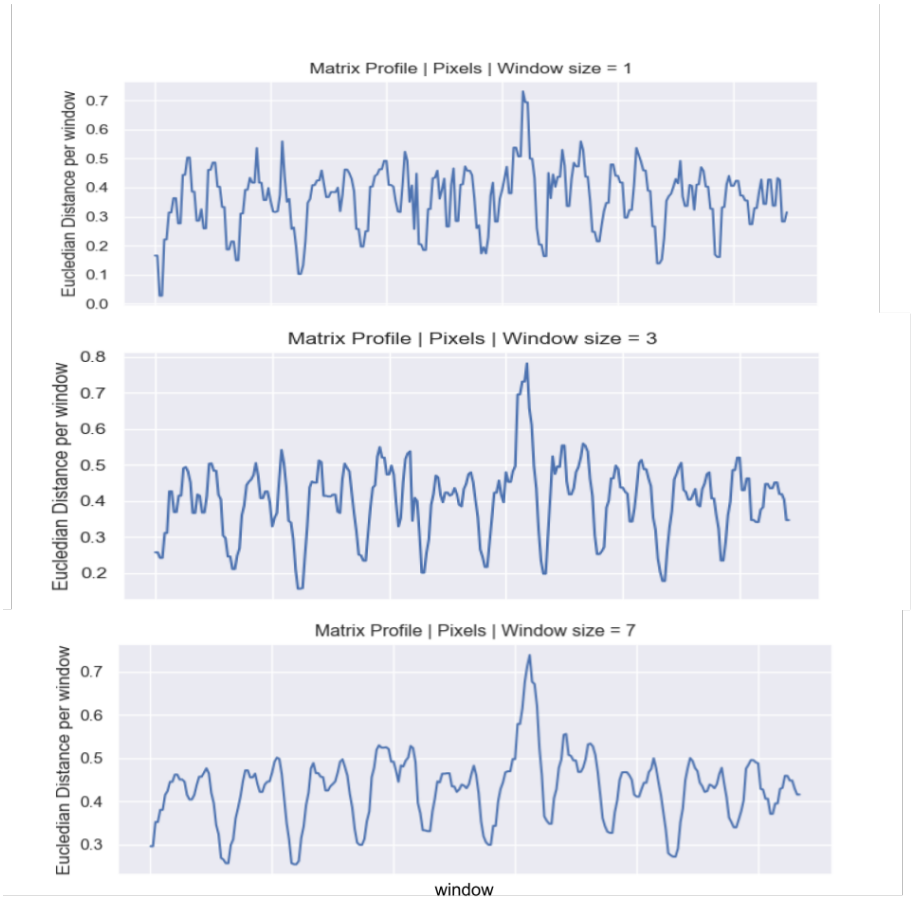

Figure 4.38: Change in Matrix profile with change in Window size

observed (frames shown in Figure 4.36). The distance profiles of this video are shown in Figure 4.37. The minimum values (for each time-step) from this distance profile is the matrix profile. The comparisons of using different window sizes(1, 3 and 7) are shown in Figure 4.38. **As expected, the matrix profile gets smoother as the window size is increased.**

4.3.2. Matrix Profile on pretrained CNNs

The matrix profile is reliable when computed using image pixels. We hypothesized that the matrix profile should be reliable when computed using CNN features as well. This could increase the speed of the computation if features from each window were compared instead of the huge number of pixels. A few pretrained convolutional neural network architectures (pre-trained on the ImageNet dataset (Deng et al., 2009)) are used to have a compressed representation of the frames. This also aids in having an end-to-end trainable model if the weights of this pretrained CNN are further tuned. Using the CNN dramatically reduces the dimensions per frame hence speeding up the calculation of the matrix profile. A few CNN architectures are compared in detail in Appendix A.14.

1. Dimensions per frame using pixels = $224 \times 224 \times 3 = 150528$
2. Dimensions per frame using CNN (ResNet-18) features = 512
3. Dimensions per frame using CNN (DenseNet-121) features = 1024
4. Dimensions per frame using CNN (SqueezeNet-1.1) features = 86528

Figure 4.39: Matrix profile using pixels vs CNN(DenseNet-121) features (window size = 1)

On comparing these CNN architectures, there is no clear winner in terms of feature map size, trainable parameters or computation graph size. In the experiments however, it was observed that ResNet, even though having the largest number of trainable parameters, trains the fastest for our task. We believe that it is because of the skip connections in its

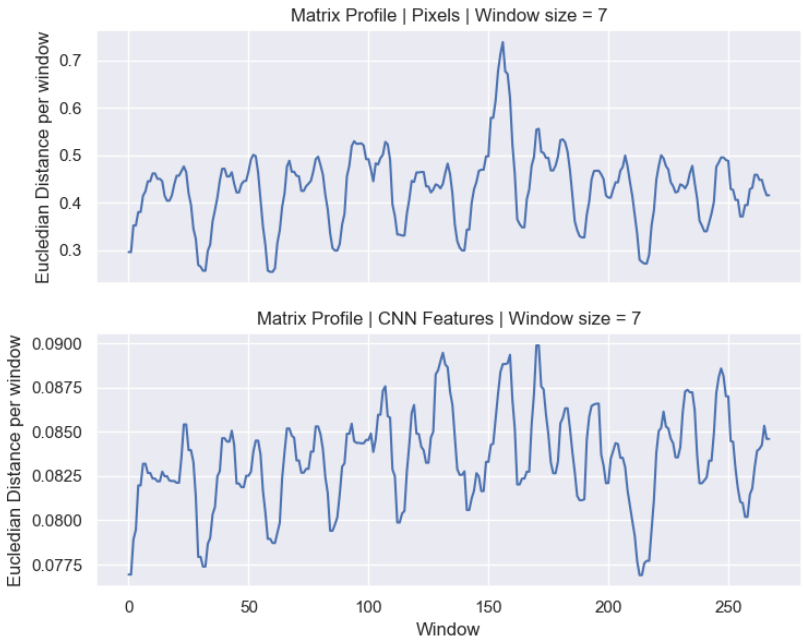

Figure 4.40: Matrix profile using pixels vs CNN(DenseNet-121) features (window size = 7)

architecture which let the gradient comfortably flow backwards. This is the reason that ResNet-18 is used for the end-to-end pipeline. To ensure that the calculated matrix profile using CNN features is reliable, the matrix profiles using pixels and using CNN features are plotted for varying window sizes in Figures 4.39 and 4.40. It can be seen in the figures that with increase in window size, the matrix profile gets smoother (like it does for matrix profile calculated using pixels). As expected, **the matrix profile using the CNN features is also decent** with an added advantage of being able to further train the CNN weights to possibly improve the calculated matrix profile signal.

4.3.3. Gradient Video

Before moving to the end-to-end experiments, it needs to made sure that the gradients flow from the model prediction vs count label back into the LSTM and through the Matrix profile computation graph back into the CNN block. To actually see if there is some useful information being passed back, an experiment was conducted to visualize the gradient flow. To see if the gradients flow back properly with meaningful information, the CNN block is removed from the pipeline as shown in Figure 4.42. And the video frames are made trainable. What this means is that the frame-image-pixels are made tunable, so the gradient information can flow into them. The last time step of the prediction is compared to the actual label of the video (as discussed in the Last time step labelling section above) and is where the gradient originates and it flows back into each frame of the video. Now, the gradient information per frame in the video is gathered and a video is created out of the gradients. A video that we used for this experiment is the bmx video (as shown in Figure 4.13 before). After making the frame pixels trainable, the computed gradient information video is shown in 4.41. The gradient video shows a nice interest region in the pixels where

Figure 4.41: Gradient video computed by making the video-frame-pixels trainable.

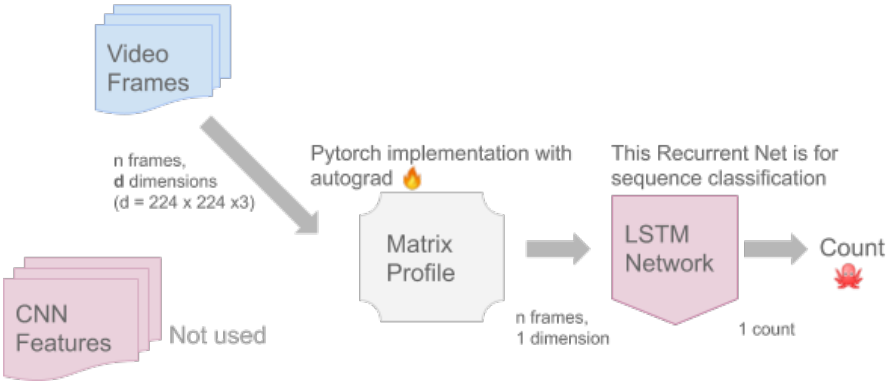

Figure 4.42: End-to-end pipeline without CNN block for Gradient video computation

the movement actually occurs in the video. It clearly highlights the moving region in the video for each frame. We believe this could be experimented with further for tasks like interest region computation or background subtraction from videos. This experiment shows that the gradients flow backwards properly even from just the last time step.

4.4. End-to-end trainable pipeline

The final end-to-end trainable pipeline is shown in Figure 4.43. A series of experiments were conducted that would change one variable at a time to be able to definitively answer the research questions. All the experiments conducted so far work with only segments of this pipeline. For example, the sinusoidal experiments at the beginning aid in finalizing the LSTM block (and its size required for a given task) that could understand a 1-dimensional signal that matrix profile generates which is the last block in the pipeline. Similarly, the CNN with matrix profile experiments were conducted to ensure that the output of a convolutional network can be used for computing the matrix profile instead of the raw pixel values. This section covers the end-to-end flow. Some challenges encountered while training this pipeline are discussed in Appendix A.15.

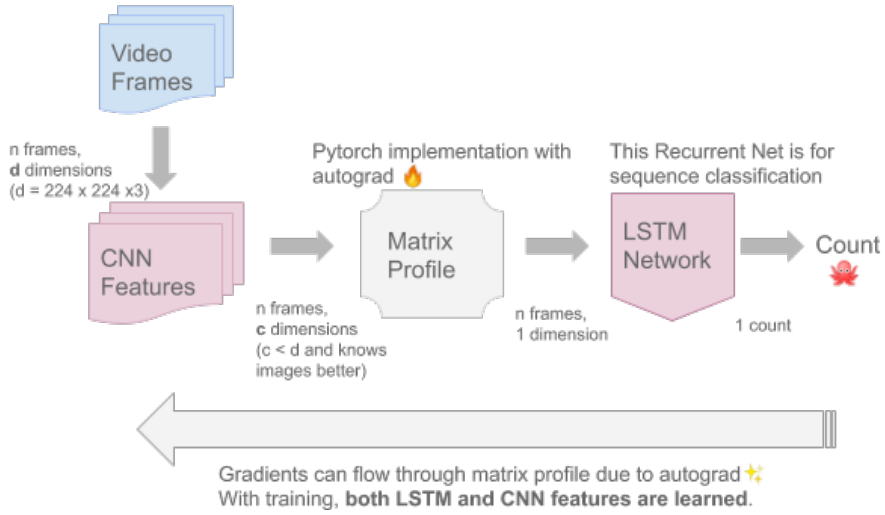

Figure 4.43: End-to-end trainable pipeline

4.4.1. Changing one variable at a time

Theoretically, with an infinite amount of labelled videos for training and an unlimited amount of memory in the available GPUs, it should be possible to estimate repetitions in all kinds of videos by just feeding the raw pixel values into a big enough LSTM network. We have neither of those, so we need to learn features from the images to reduce the dimensions of the input to the LSTM network. And we do not even have an unlimited memory, so we want our LSTM network to be as small as possible. Through our upcoming experiments, we will try to come up with light versions of the model that perform at least as good as the heavier models. The key idea is that the lighter the model is, the more feasible it is to use for a real-world problem.
Several instances of the model were deployed with slight modifications in each one. The slight changes ensure that one variable is changed at a time to understand the effect of the variations. It is a configurable plug and play pipeline and the following configurations are experimented with:

1. **CNN Type**: This can have a value between *DenseNet-121 or SqueezeNet-1.1*. Based on this configuration, the Convolutional Neural Network block is decided. For either of these networks, the input to the Convolutional Block (per time step) is an image of dimensions $(224 \times 224 \times 3)$. The output of this convolutional block is of dimensions 512 if it is a *ResNet-18*, 1024 if it is a *DenseNet-121* and 86528 if it is a *SqueezeNet-1.1*.

2. **Matrix Profile Flag**: This is a boolean flag which can have a value of True/False *(on/off)*. If this flag is turned *on*, only then the matrix profile block is used in the pipeline. For instance, if this flag is turned off, the CNN features are directly fed to the next block which is the LSTM block. In this case, the input for the LSTM block would be of dimensions 1024 (if CNN used is DenseNet-121), 86528 (if CNN used is SqueezeNet-1.1) or 512 (if CNN used is ResNet-18).

3. **Matrix Profile window-size**: This field is only used if the *Matrix Profile flag* is set to

on. This field determines the window-size which is a hyperparameter used in calculation of the matrix profile. (See the Matrix Profile section above for more details).

4. **LSTM Hidden Layers**: This field determines the number of hidden layers employed in the LSTM block in our pipeline. By default, we have set this parameter to 2 for our experiments. This configuration can help us experiment with the complexity of our model later on if needed.

5. **LSTM Hidden Neurons(per layer)**: This field determines the number of neurons in each hidden layer of the LSTM block. Because of our initial wave-cycle experiments, we could make informed guesses about this parameter. Based on the choices of the other configuration fields, this field would be modified (as shown in this section later).

6. **CNN Backpropagation**: This is a boolean flag which can have a value of True/False *(on/off)*. If this flag is turned *on*, only then will the weights of the CNN be tuned. For initialization of the weights of the CNN (either of the two variants), we use the pretrained weights (pretrained on ImageNet (Deng et al., 2009)). If this flag is not turned *on*, the image frame to CNN feature calculation step will not be a part of the computation graph (more details on the computation graph in the Appendix). In that case, the end-to-end computation graph would be much smaller than in the case of this flag being *on*. Our hypothesis regarding this parameter was that by tuning the weights of this CNN, we could possibly polish the computed matrix profile signal. (The CNN would adjust itself to make it easier for the matrix profile to detect motifs). If turned *on*, this would make the training of the pipeline drastically slower, but the possible rewards are worth the wait.

Based on the configuration options discussed above, there can be any number of combinations to converge to a final model. But, from our experience with the experiments conducted so far, we came up with the following combinations:

1. **Model: t0**

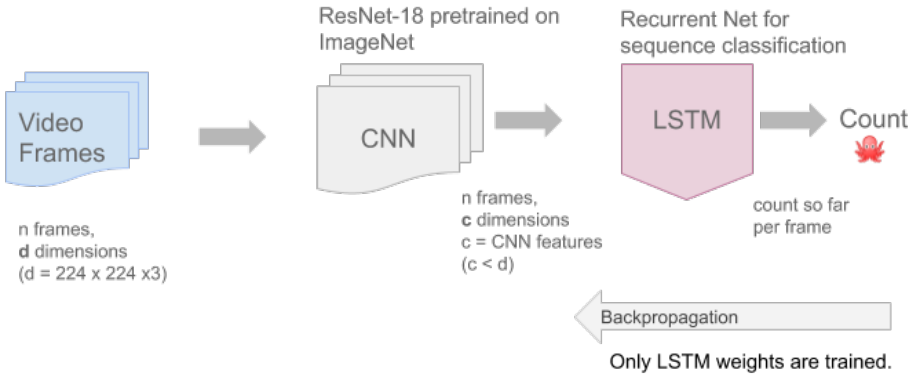

Figure 4.44: Architecture of models **t0** and **t1**. The CNN features are directly fed to the LSTM block.

- *CNN Type*: ResNet-18
- *Matrix Profile Flag*: *off*

- *Matrix Profile window size*: Not applicable (as Matrix Profile Flag is off)
- *LSTM Hidden Layers*: 2
- *LSTM Hidden Neurons (per layer)*: 100
- *CNN Backpropagation*: *off*

Figure 4.44 shows the architecture of this model. This is a relatively lightweight pipeline without the Matrix Profile. The idea of training this pipeline is that feeding the CNN frames directly to a light LSTM. Our hypothesis regarding this variant is that it would begin to learn some patterns but the performance of this pipeline would not be as good as the heavier LSTM variant.

2. **Model: t1**

- *CNN Type*: ResNet-18
- *Matrix Profile Flag*: *off*
- *Matrix Profile window size*: Not applicable (as Matrix Profile Flag is off)
- *LSTM Hidden Layers*: 2
- *LSTM Hidden Neurons (per layer)*: 2000
- *CNN Backpropagation*: *off*

Figure 4.44 shows the architecture of this model. This is a heavier Pipeline without the Matrix Profile. The idea of this pipeline is that we have enough number of hidden neurons to actually fit the task at hand. Our hypothesis is that this variant would perform better than the variant with a lighter LSTM.

3. **Model: t2**

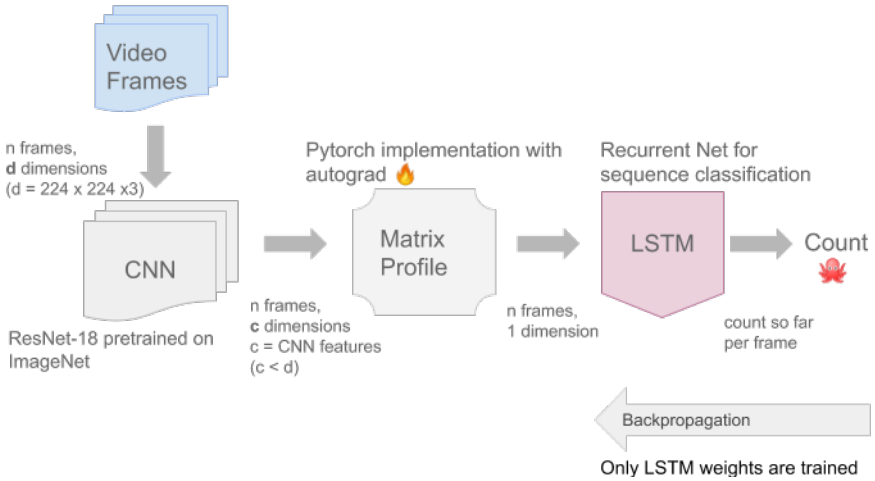

Figure 4.45: Architecture of models **t2** and **t3**. The CNN features are used to compute the Matrix Profile and that is fed to the LSTM block.

- *CNN Type*: ResNet-18
- *Matrix Profile Flag*: *on*

- *Matrix Profile window size*: 1
- *LSTM Hidden Layers*: 2
- *LSTM Hidden Neurons (per layer)*: 2000
- *CNN Backpropagation*: *off*

Figure 4.45 shows the architecture of this model. This is a heavier pipeline with the Matrix Profile.

4. **Model: t3**

- *CNN Type*: ResNet-18
- *Matrix Profile Flag*: *on*
- *Matrix Profile window size*: 1
- *LSTM Hidden Layers*: 2
- *LSTM Hidden Neurons (per layer)*: 100
- *CNN Backpropagation*: *off*

Figure 4.45 shows the architecture of this model. This is a relatively lightweight pipeline with the Matrix Profile. The idea of using matrix profile in this pipeline is that if it is possible to use the matrix profile to greatly reduce the size of the model required for the task of repetition estimation.

5. **Model: t4**

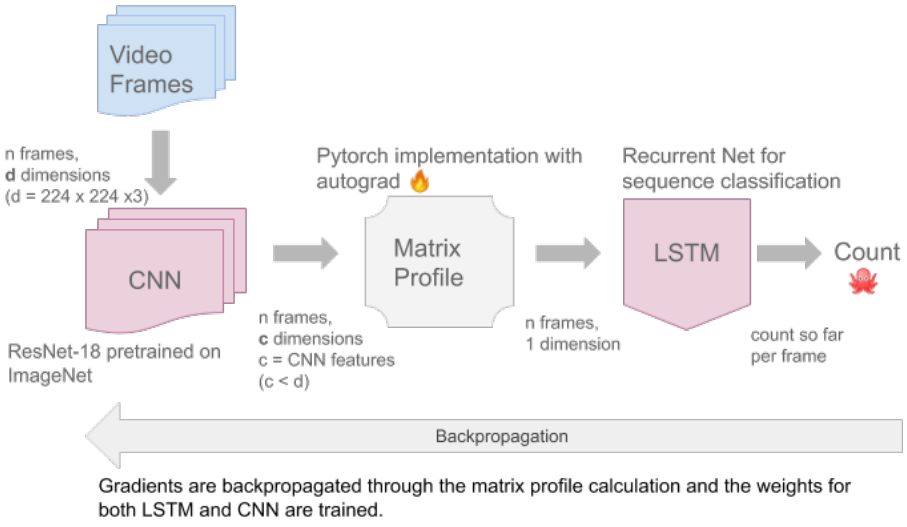

Figure 4.46: Architecture of model **t4**. The CNN features are used to compute the Matrix Profile and that is fed to the LSTM block. The CNN weights are also trained for this model.

- *CNN Type*: ResNet-18
- *Matrix Profile Flag*: *on*
- *Matrix Profile window size*: 1

- *LSTM Hidden Layers*: 2
- *LSTM Hidden Neurons (per layer)*: 100
- *CNN Backpropagation*: *on*

Figure 4.46 shows the architecture of this model. This model is a relatively lightweight pipeline with the Matrix Profile but its CNN backpropagation is turned *on* leading to a huge computation graph. The goal of this pipeline is to train the CNN in such a way that the computed matrix profile using this gets better (it becomes easier to detect repetitions from the computed matrix profile signal). This model would answer the question if backpropagating through the matrix profile actually helps.

These models are trained on all the three datasets for 500 epochs or a max-run-time of 3 days (whichever happens before) on a pascal GPU machine. The training-losses, validation-losses and epoch-accuracies can be found in the Appendix B.

5

Results and Discussions

The model-configurations discussed at the end of the last section are all trained separately on all the three datasets. (Note: These results are obtained using *each time-step labelling* as discussed before). Each frame denotes a class to which it belongs (repetitions so far). Now, to make sure that there is no bias, it is ensured that each video in the dataset has exactly 5 repetitions. The videos which have more than 5 repetitions are trimmed so that they meet this constraint.

Measuring accuracy: For each time-step labelled data, a label for each frame in the video is present which denotes the number of repetitions so far. There are several ways to measure accuracy. Existing literature (like (Runia et al., 2018) and (Levy and Wolf, 2015)) uses *Mean Absolute Error* to compute errors in their final estimations. Mean Absolute Error is the difference in actual and predicted number of repetitions. These works also do not use a separate test set for evaluation. This would mean only considering the output of the last frame for our architecture since an output is produced for each frame by this work's proposed model. This is not a good indicator of the accuracy of the proposed model. A technique to evaluate such an online approach is to look at all the predicted labels of all the frames (from the videos in the test set) and compare them to the actual labels to compute an overall accuracy. This approach however, has an ingrained flaw. If a video has a lot of frames (as compared to other videos in the test set), the accuracy of that video would skew the overall accuracy of the model which should not be the case. Since all the videos have exactly 5 repetitions, a video could have more frames if the action is repeated slowly in the video (temporal cycle length is large). To overcome this flaw, this work proposes the approach of computing accuracy per video (from the test set) by comparing individual frame predictions with the labels and then calculating an overall accuracy of the test set by taking an average of all the accuracies. This approach gives an equal weight to each video in the test set.

5.1. Blade Inspection dataset

This is the smallest dataset (with each-time-step labelling) with 20 videos. The repetitions in this dataset are slower than the other two datasets. Since each video in all subsets contains exactly 5 repetitions, this dataset has the maximum number of frames per video. Since

each of these videos feature long repetition cycles, the videos in this dataset have a lot of frames. Backpropagating gradients through the matrix profile computation step leads to a very big computation graph which doesn't fit in the GPU memory. This is the reason this dataset was resampled to 5 frames per second. All the training and validation loss plots can be found in the Appendix B.

- Figure B.1 shows training and validation losses using *t0 architecture*. It also shows the test-accuracy per epoch while training. Figure B.2 shows accuracies per video distribution. The training and validation loss curves show a steady decrease in loss and the best accuracy(number of correctly labelled frames) obtained was about 56%.

- Figure B.3 shows training and validation losses using *t1 architecture*. It also shows the test-accuracy per epoch while training. Figure B.4 shows accuracies per video distribution. This architecture uses a heavier LSTM variant and shows signs of overfitting on the training set. The loss in train set keeps decreasing while the loss in the validation set starts to rise which indicates that the model is overfitting on the training set. Since this dataset had very few videos and lots of trainable parameters, this was expected. This could be rectified by employing some regularization techniques like L1/L2 regularization, dropout etc. (A few regularization techniques are discussed in detail in the Appendix A). As expected, the best accuracy obtained using this architecture was about 61% and better than the lighter LSTM architecture (t0 architecture). The accuracy difference was not significant however.

- Figure B.5 shows training and validation losses using *t2 architecture*. It also shows the test-accuracy per epoch while training. Figure B.6 shows accuracies per video distribution. This architecture uses the Matrix Profile and uses a heavy LSTM. The accuracies obtained by this architecture were at-least as good as the t1-architecture (around 60%) which indicate that the Matrix Profile is useful for the task at hand instead of using the CNN features directly.

- Figure B.7 shows training and validation losses using *t3 architecture*. It also shows the test-accuracy per epoch while training. Figure B.8 shows accuracies per video distribution. This architecture uses a light LSTM with the Matrix profile. The accuracies obtained with this architecture were again at-least as good as the heavier *t1 architecture* and *t2 architecture* variants. The accuracy obtained using this architecure was 66% which is the best accuracy obtained so far compared to the other architectures. This helps prove our hypothesis that employing the matrix profile can help compress the model size required for the task of repetition estimation.

- Figure B.9 shows training and validation losses using *t4 architecture*. It also shows the test-accuracy per epoch while training. Figure B.10 shows accuracies per video distribution. The training of this model for the blade dataset shows a lot of movement in loss curves and obtains a best accuracy of 66%. This accuracy is not better than the *t3 architecture*, but, is still better than any other architecture and doesn't disprove thy hypothesis that backpropagating through the matrix profile can be beneficial for repetition estimation.

5.2. YTSegments dataset

This dataset contains about 100 videos (with each time-step labelling). This dataset contains a variety of videos like cooking, exercise etc. The repetitions in this dataset are fairly quick compared to the blade dataset, so the number of frames per video in this dataset is lower (as all the videos have exactly 5 repetitions). All the training and validation loss plots can be found in the Appendix B.

- Figure B.11 shows training and validation losses using *t0 architecture*. It also shows the test-accuracy per epoch while training. Figure B.12 shows accuracies per video distribution. Because of the variety of videos in this dataset, the loss curves show more variance around the trend line. The best accuracy obtained on this dataset was around 51%. As expected this dataset has a variety of videos and has more videos which results in a slightly lower accuracy compared to the blade dataset. Toward the end the training shows some signs of overfitting as the training loss decreases while the validation losses start rising.

- Figure B.13 shows training and validation losses using *t1 architecture*. It also shows the test-accuracy per epoch while training. Figure B.14 shows accuracies per video distribution. Again the heavier LSTM. This architecture uses a heavier LSTM variant. It doesn't yet show any signs of overfitting. This could be because the model is still learning its massive amounts of parameters and the amount of training data is not too small like the blade dataset. This architecture again obtains a better accuracy of 56% compared to the lighter LSTM variant.

- Figure B.15 shows training and validation losses using *t2 architecture*. It also shows the test-accuracy per epoch while training. Figure B.16 shows accuracies per video distribution. This architecture uses the Matrix Profile and uses a heavy LSTM. The accuracies obtained by this architecture were around 57% and again at-least as good as the *t1-architecture* indicating the usefulness of the Matrix Profile.

- Figure B.17 shows training and validation losses using *t3 architecture*. It also shows the test-accuracy per epoch while training. Figure B.18 shows accuracies per video distribution. This architecture uses a light LSTM with the Matrix profile. The accuracies obtained with this architecture were around 56%. Just like the blade dataset, a lighter model with the Matrix profile gets accuracies as good as the heavier models indicating that the with the matrix profile, the required model size is smaller.

- The *t4 architecture* involves backpropagating throught the matrix profile calculation and training the pretrained CNN weights. Since the repetitions on this dataset were quick, the size of the computation graph was feasible. The validation loss seemed to hint overfitting as the training loss kept decreasing but, the accuracies obtained by this model were 63% which is the best so far. This experiment shows that **it is indeed possible to backpropagate through the matrix profile**.

5.3. QUVA dataset

This dataset also contains about 100 videos (with each time-step labelling). It contains videos with lots of camera movements and varying repetition cycle lengths. All the training

and validation loss plots can be found in the Appendix B.

- Figure B.21 shows training and validation losses using *t0 architecture*. It also shows the test-accuracy per epoch while training. Figure B.22 shows accuracies per video distribution. The training and validation curves show a steady decrease and later shows signs of overfitting just like the same architecture with the *YTSegments dataset*. The best accuracy on the test-set obtained using this architecture was a mere 40%.

- Figure B.23 shows training and validation losses using *t1 architecture*. It also shows the test-accuracy per epoch while training. Figure B.24 shows accuracies per video distribution. The best accuracy obtained was 45% using this architecture.

- Figure B.25 shows training and validation losses using *t2 architecture*. It also shows the test-accuracy per epoch while training. Figure B.26 shows accuracies per video distribution. This architecture uses the Matrix Profile in conjunction with a heavy LSTM. The best accuracy obtained by this architecture(46%) was quite similar to the *t1 architecture* variant.

- Figure B.27 shows training and validation losses using *t3 architecture*. It also shows the test-accuracy per epoch while training. Figure B.28 shows accuracies per video distribution. Even with a light LSTM this architecture shows accuracies of 46% which are similar to the heavier LSTM variants just like for the other two datasets.

- The *t4 architecture* which involves backpropagating throught the matrix profile calculation and training the pretrained CNN weights was successfuly performed on the quva dataset as well. The quva dataset contains videos of a variety of cycle lengths but the cycle lengths are never as big as the blade dataset videos. Just like for the *YTSegments dataset*, the loss curves showed hints of overfitting, but the best accuracy(50%) obtained by this backpropagation through the matrix-profile approach exceeded all the other architectures. **This again proves our hypothesis that backpropagating through the matrix profile is possible and potentially useful.**

The accuracies on the test sets of Aircraft Engine Blades, YTSegments and QUVA datasets for all of the model architectures are plotted in Figure 5.1. The trend in the plot shows that the models making use of the Matrix profile (*t3 and t4 architectures*) obtain the best results in terms of accurately labeled frames. The experiments comparing *t0 and t1 architectures* show a difference of about 5% for all datasets. This shows that estimating repetitions using a recurrent network fed with CNN features requires a heavy RNN to better learn the task at hand. The *t3 architectures* for all datasets consistently gave accuracies equal to or better than the heavier *t1 and t2 architectures*. This helps us deduce that computing the matrix profile is actually useful for estimating repetitions and helps us reduce the size of the model without compromising on accuracy. The *t4 architecture* showed good promise by obtaining the best accuracies (with a margin of about 5%) for both *quva and ytsegments datasets*. By looking at the results of the *t4 architecture* on the datasets, it can safely be concluded that it is possible to backpropagate through the step of computing the matrix profile and update the CNN weights such that it helps the upstream LSTM to more effectively use the computed Matrix profile signal. This means that backpropagating through the matrix profile could potentially be used for several other tasks where matrix profile already shines to further improve the results in that specific domain. This also means that other problem

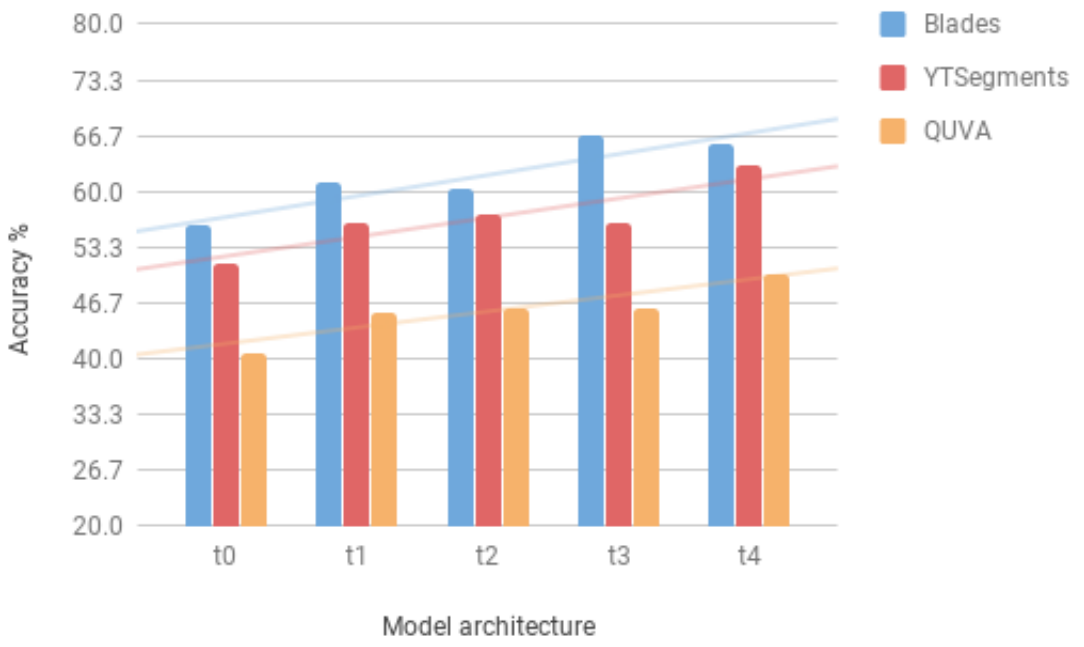

Figure 5.1: Model architecture accuracies for Aircraft Engine Blades, YTSegments and QUVA datasets.

	Blades	YTSegments	QUVA
(Levy and Wolf, 2015)	3.454	2.87	3.076
t4 architecture	**0.409**	**1.383**	**0.641**

Table 5.1: Mean Absolute Error comparison of the *t4 architecture* with (Levy and Wolf, 2015) on the prepared datasets with 10 fps and 5 repetitions per video.

specific mathematical operations could be backpropagated through to further improve results.

The best performing architecture (*t4 architecture*) is also compared with the pretrained model published by (Levy and Wolf, 2015) on the prepared dataset for **Mean Absolute Error**. The prepared datasets contain all the videos with exactly 5 repetitions. The model published by (Levy and Wolf, 2015) predicts a count per video which can be compared with 5 to get the error in prediction (if any) and the error is averaged over all the videos in the dataset. The count per video is obtained from the proposed *t4 architecture* by only considering the model's prediction on the last time step. The Mean absolute errors for both these models can be found in the Table 5.1. It clearly shows that on the prepared dataset, the proposed architecture easily outperforms the model published by (Levy and Wolf, 2015) (which possibly requires dataset specific configurations judging by higher errors on datasets with longer repetition cycle lengths).

6

Conclusions and Recommendations

6.1. Conclusions

This work suggests a new way to look at the problem of repetition estimation. It proposes a novel approach of labelling the dataset by labelling each frame of the video with a class (repetitions so far). This results in an inherently online approach to tackle the problem. It is observed that this kind of labelling aids in faster and better learning in recurrent architectures as compared to assigning a label to the entire video (or to just the last time step). After labelling the dataset like this, the Matrix Profile is experimented with and it is observed that it is useful for the task of repetition estimation. Furthermore, a recurrent neural network architecture like an LSTM can be trained using video frames or using pretrained CNN (like ResNet) features from video frames. An LSTM like this would need to be quite heavy (with a lot of trainable parameters). The size of this network can be dramatically reduced by using the Matrix Profile and feeding the Matrix profile (computed from the CNN features) to the upstream LSTM instead of feeding the CNN features to LSTM directly. The weights of this pretrained CNN (ResNet-18) are also trained by backpropagating through the Matrix Profile computation step and it is observed that the accuracy of the model increases for both all the three (Blade inspection, QUVA and YTSegments) datasets. The fact that the matrix profile computation is backpropagatable can be quite beneficial to several domains that employ the Matrix Profile or other domains which employ some problem specific mathematical operations like the Matrix Profile.

6.2. Recommendations

The problem of estimating repetitions in videos is harder than it seems at first glance. There could be several ways to improve the performance and learning of the proposed architecture. Currently, the datasets were labelled with each-time-step labelling manually by a single person. This could be rectified by having several people label the videos and aggregating their results to have consistent repetition cycle estimates. For the matrix profile calculation, a window size of 1 is used in the end-to-end pipeline experiments. Different values for this hyperparameter can be cross-validated to see what works best for the dataset at hand. Another improvement could include cross-validating with several loss-types like the Margin Hinge Loss etc. instead of the employed Negative Log-Likelihood Loss. and with different activation functions like *tanh* or *sigmoid*. Different types of loss and activation functions

can perform better for different kinds of problems. A few popular loss types and activation functions are discussed in detail in the Appendix A. More boost in performance could be obtained by increasing the size of the dataset by using video augmentation techniques like rotation, shear etc. The computation graphs of several matrix profile implementations could be inspected to chose the smallest one or the most parallelizable one to enable faster convergence. Smaller computation graphs as proposed by (Veit and Belongie, 2018) can be experimented with. The matrix profile output is not necessarily aligned perfectly with our idea of repetition-cycle end/begin. The Feature Aligning Network recently proposed by (Xie et al., 2019) also seems promising to align the matrix profile properly with the labels. A completely different way to tackle this problem could be by using the Triplet Loss proposed by (Schroff et al., 2015). To use the Triplet Loss, videos with the same number of repetitions could be grouped together as *positive* examples for the *anchor* and any other video could be *negative* examples.

Bibliography

[L1] Link-1 understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2019-08-18.

[L2] Link-2 opencv optical flow. https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html. Accessed: 2019-08-18.

[L3] Link-3 calculus on computational graphs: Backpropagation. https://colah.github.io/posts/2015-08-Backprop/. Accessed: 2019-08-18.

[L4] Link-4 activation functions in neural networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Accessed: 2019-08-18.

[5] Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

[6] Barron, J. L., Fleet, D. J., and Beauchemin, S. S. (1994). Performance of optical flow techniques. *International journal of computer vision*, 12(1):43–77.

[7] Belongie, S. and Wills, J. (2004). Structure from periodic motion. In *International Workshop on Spatial Coherence for Visual Motion Analysis*, pages 16–24. Springer.

[8] Brand, M. and Kettnaker, V. (2000). Discovery and segmentation of activities in video. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (8):844–851.

[9] Briassouli, A. and Ahuja, N. (2004). Fusion of frequency and spatial domain information for motion analysis. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 175–178. IEEE.

[10] Briassouli, A. and Ahuja, N. (2007). Extraction and analysis of multiple periodic motions in video sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1244–1261.

[11] Brox, T., Bruhn, A., Papenberg, N., and Weickert, J. (2004). High accuracy optical flow estimation based on a theory for warping. In *European conference on computer vision*, pages 25–36. Springer.

[12] Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308.

[13] Chandola, V. (2009). Detecting anomalies in a time series database varun chandola, deepthi cheboli, and vipin kumar.

[14] Chin, T. M., Karl, W. C., and Willsky, A. S. (1994). Probabilistic and sequential computation of optical flow using temporal coherence. *IEEE Transactions on Image Processing*, 3(6):773–788.

[15] Cremers, D., Osher, S. J., and Soatto, S. (2004). Kernel density estimation and intrinsic alignment for knowledge-driven segmentation: Teaching level sets to walk. In *Joint Pattern Recognition Symposium*, pages 36–44. Springer.

[16] Cutler, R. and Davis, L. S. (2000). Robust real-time periodic motion detection, analysis, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):781–796.

[17] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[18] Djurovic, I. and Stankovic, S. (2003). Estimation of time-varying velocities of moving objects by time-frequency representations. *IEEE Transactions on Image Processing*, 12(5):550–562.

[19] Duda, R. O. and Hart, P. E. (1971). Use of the hough transformation to detect lines and curves in pictures. Technical report, Sri International Menlo Park Ca Artificial Intelligence Center.

[20] Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer.

[21] Goldenberg, R., Kimmel, R., Rivlin, E., and Rudzsky, M. (2005). Behavior classification by eigendecomposition of periodic motions. *Pattern Recognition*, 38(7):1033–1043.

[22] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610.

[23] Harris, C. G., Stephens, M., et al. (1988). A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer.

[24] He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[25] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer.

[26] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

[27] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[28] Hoge, W. S., Mitsouras, D., Rybicki, F. J., Mulkern, R. V., and Westin, C.-F. (2003). Registration of multidimensional image data via subpixel resolution phase correlation. In *Proceedings 2003 International Conference on Image Processing (Cat. No. 03CH37429)*, volume 2, pages II–707. IEEE.

[29] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[30] Huang, S., Ying, X., Rong, J., Shang, Z., and Zha, H. (2016). Camera calibration from periodic motion of a pedestrian. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3025–3033.

[31] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

[32] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[33] Johansson, G. (1973). Visual perception of biological motion and a model for its analysis. *Perception & psychophysics*, 14(2):201–211.

[34] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732.

[35] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[36] Laptev, I., Belongie, S. J., Perez, P., and Wills, J. (2005). Periodic motion detection and segmentation via approximate sequence alignment. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 816–823. IEEE.

[37] Levy, O. and Wolf, L. (2015). Live repetition counting. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3020–3028.

[38] Liu, F. and Picard, R. W. (1998). Finding periodicity in space and time. In *ICCV*, pages 376–383. Citeseer.

[39] Liu, Y., Collins, R. T., and Tsin, Y. (2004). A computational model for periodic pattern perception based on frieze and wallpaper groups. *IEEE transactions on pattern analysis and machine intelligence*, 26(3):354–371.

[40] Lu, C. and Ferrier, N. J. (2004). Repetitive motion analysis: Segmentation and event classification.

[41] Lucas, B. D., Kanade, T., et al. (1981). An iterative image registration technique with an application to stereo vision.

[42] Ma, X. and Hovy, E. (2016). End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

[43] Mueen, A., Keogh, E., Zhu, Q., Cash, S., and Westover, B. (2009). Exact discovery of time series motifs. In *Proceedings of the 2009 SIAM international conference on data mining*, pages 473–484. SIAM.

[44] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

[45] Papazoglou, A. and Ferrari, V. (2013). Fast object segmentation in unconstrained video. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1777–1784.

[46] Piccardi, M. (2004). Background subtraction techniques: a review. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, volume 4, pages 3099–3104. IEEE.

[47] Polana, R. and Nelson, R. C. (1997). Detection and recognition of periodic, nonrigid motion. *International Journal of Computer Vision*, 23(3):261–282.

[48] Revaud, J., Weinzaepfel, P., Harchaoui, Z., and Schmid, C. (2015). Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1164–1172.

[49] Runia, T. F., Snoek, C. G., and Smeulders, A. W. (2018). Real-world repetition estimation by div, grad and curl. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9009–9017.

[50] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

[51] Seitz, S. M. and Dyer, C. R. (1997). View-invariant analysis of cyclic motion. *International Journal of Computer Vision*, 25(3):231–251.

[52] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[53] Sola, J. and Sevilla, J. (1997). Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on nuclear science*, 44(3):1464–1468.

[54] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[55] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385.

[56] Stankovi, S. and Djurovi, I. (2001). Motion parameter estimation by using time-frequency representations. *Electronics Letters*, 37(24):1446–1448.

[57] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[58] Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., and Paluri, M. (2018). A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459.

[59] Tsai, P.-S., Shah, M., Keiter, K., and Kasparis, T. (1994). Cyclic motion detection for motion based recognition. *Pattern recognition*, 27(12):1591–1603.

[60] Veit, A. and Belongie, S. (2018). Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18.

[61] Wang, H. and Schmid, C. (2013). Action recognition with improved trajectories. In *Proceedings of the IEEE international conference on computer vision*, pages 3551–3558.

[62] Wang, L., Tan, T., Hu, W., and Ning, H. (2003). Automatic gait recognition based on statistical shape analysis. *IEEE transactions on image processing*, 12(9):1120–1131.

[63] Xie, Y., Wang, H., Hao, Y., and Xu, Z. (2019). Visual rhythm prediction with feature-aligning network. *arXiv preprint arXiv:1901.10163*.

[64] Yeh, C.-C. M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Silva, D. F., Mueen, A., and Keogh, E. (2016). Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1317–1322. IEEE.

[65] Young, R. W. and Kingsbury, N. G. (1993). Frequency-domain motion estimation using a complex lapped transform. *IEEE Transactions on image processing*, 2(1):2–17.

[66] Zhu, Y., Imamura, M., Nikovski, D., and Keogh, E. (2019). Introducing time series chains: a new primitive for time series data mining. *Knowledge and Information Systems*, 60(2):1135–1161.

[67] Zhu, Y., Yeh, C.-C. M., Zimmerman, Z., Kamgar, K., and Keogh, E. (2018). Matrix profile xi: Scrimp++: time series motif discovery at interactive speeds. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 837–846. IEEE.

A

Appendix I: Detailed Explanations

A.1. Matrix Profile

The problem of finding similarity joins or all-similar-pairs has been researched upon extensively since several decades. There hasn't been a great amount of progress in feasibly finding similarity joins for time series datasets. The paucity of advancement could be blamed at the intimidating nature of the problem. Even for relatively moderate sized datasets, the typical nested-loop approach can take months to converge to a solution. There have been several techniques proposed to obtain a speed-up like early-abandoning, indexing, triangular inequality, lower bounding etc. But all of these techniques only produce a small speedup of a couple orders of magnitude at best. (Yeh et al., 2016) proposed a technique which claims to be scalable even for exceptionally huge datasets. This algorithm can be used as an anytime-algorithm in cases of large datasets and emit quality approximations in an acceptable amount of time. This algorithm for similarity joins also provides the fastest known approach for finding motifs and discords in time series which are both extensively studied problems. There are several advantages of employing the Matrix Profile for data mining tasks on time series over techniques like indexing, hashing etc. like:

- It doesn't provide any false positives for tasks like discovery of motifs and discord or for joins on time series etc. **It is therefore exact.**
- Opposed to a lot of other algorithms like spatial access algorithms, the matrix profile doesn't require any tuning and is therefore simple and free of any hyperparameters.
- The matrix profile computation doesn't need a ton of space and only requires space in a linear order of magnitude of the timeseries along with a constant overhead. This allows the computation to be performed in the Random Access Memory. Hence, the space complexity is linear(efficient).
- The time complexity of the algorithm is also constant in subsequence length which is a very lucrative attribute for a motif/discord/join algorithm. The matrix profile can actually be computed in deterministic time and it is possible to precisely estimate the time that would be required to calculate it.
- It is possible to incrementally maintain the matrix profile. This makes it usable for extremely long sequences of multidimensional data. This means that the algorithms

proposed in (Mueen et al., 2009) and (Chandola, 2009) can exactly maintain time-series motifs and discords respectively using streaming data. Only the extreme(maximum and minimum) values need to be kept track of while the matrix profile grows incrementally.

- The computation of the Matrix Profile also allows for blazingly-fast approximate solutions which could be used for extremely large datasets.
- The matrix profile is not adversely affected by some missing data. The approach doesn't provide any false negatives even if some data is missing.
- The algorithm is massively parallelizable which allows it to use several cores of a CPU or even GPU to speedup the calculations.

Given all these bankable attributes, the Matrix Profile finds applicability in a wide range of data mining tasks on time series. The papers exhibit the utility of the algorithm on several time series problems like semantic segmentation, novelty discovery, motif discovery, contrast set mining etc.

A.2. Fourier Transform

For all intents and purposes everything on the planet can be portrayed by means of a waveform - a function of space, time or some other variable. For example, electromagnetic fields, sound waves, a plot of Voltage Standing Wave Ratio versus recurrence, the cost of your preferred stock versus time, and so on. The Fourier Transform gives us an interesting and incredible method for conceptualizing these waveforms. All waveforms, known to mankind, are in reality simply a combination(sum) of straightforward sinusoids of various frequencies. While this appears to be made up, it is valid for all waveforms. This goes for phone signals, TV signals the sound waves that are emitted when you talk. When all is said in done, waveforms are not comprised of a discrete amount of frequencies, but actually a continuous collection of frequencies. The Fourier Transform is the scientific device that tells us the best way to deconstruct the waveform into its sinusoidal parts. This has a huge number of applications and helps us in better understanding the universe by giving a new way of looking at the world. It is widely used in a variety of fields in engineering and science like for material rupture estimations in astrophysics, in space travel for long distance communication, in forecasts and market signals analysis etc.

A.3. Hough Transform

The hough transoform is a popular technique used in computer vision and digital image processing for feature extraction. The main goal of using this technique is to find reasonable instances of objects by using a voting methodology. This casting a ballot strategy is done in a parameter space, and the object candidates are secured from the local maxima from the *accumulator-space* that the technique builds for calculating the transfor. The earlier-version of Hough transform focussed on finding lines in a picture, yet later the Hough transform has been stretched out to recognizing places of self-assertive shapes, most generally ellipses and circles as discussed in (Duda and Hart, 1971). In computerized investigation of digital images, a problem regularly emerges of identifying straightforward shapes, for example, straight lines, ellipses or circles. In a lot of the cases an edge detection

step can be utilized as a pre-preparing stage to get image pixels or image points that are on the ideal curve in the image space. Because of blemishes in either the image information or the edge locator, be that as it may, there might miss focuses or pixels on the ideal curves just as spatial deviations between the perfect line/ellipse/circle and the noisy edges as they are acquired from the edge indicator. Thus, it is regularly significant to bunch the extricated edge highlights to a fitting arrangement of lines, circles or ellipses. The motivation behind the Hough transform is to address this issue by making it conceivable to perform groupings of edge-points into candidates(up for voting) by playing out an unequivocal voting-system over a lot of image objects that are parameterzed.

A.4. Optical Flow

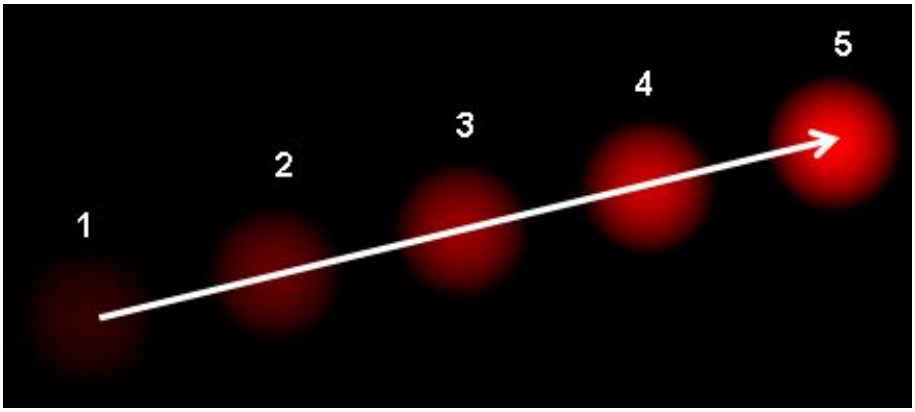

Figure A.1: Optical flow of a moving ball from [L2]

Optical flow is the pattern of evident movement of an object, edges, and surfaces in an ocular scene occuring due to the relative movement between a scene and the observer. It can also be coined as the spread of discernible velocities of the motion of patterns of brightness in an image. It is in-fact a 2 Dimensional vector field in which, each vector denotes the motion of a certain pixel or a pixel region from the previous frame to the next. Figure A.1 shows a moving ball and its position in five frames (consecutive). The big arrow denotes the direction of the displacement vector. Optical flow finds applications in several fields like video stabilization, structure from motion ((Belongie and Wills, 2004)), video stabilization etc. This algorithm works on a few big assumptions that for an object, the pixel intensities don't change with successive frames and that the pixel-neighbours have similar kind of motion trajectory. A popular open source method to compute optical flow is the Lucas Kanade method proposed by (Lucas et al., 1981) whuch takes a 3×3 patch around a point of interest and then computes the trajectory of these 9 points as a whole. Usually in practice, a corner detector like the Harris corner detector ((Harris et al., 1988)) is first used to compute corners which are interest points and then the optical flow for those points is computed. This optical flow calculation usually fails whenever there is a big movement since it only relies on pixel's local surroundings in successive frames. To overcome this, *pyramids* are used. As you move up in the pyramid from the base, the small motions disappear and the large motions become smaller. The optical flow calculation can then be performed along with the changed scale. The *Lucas Kanade method* is readily available as a function to use in the OpenCV library.

A.4.1. Dense Flow Field

The Lucas Kanade approach computes the optical flow for a feature set which is sparse in nature. For example, we typically compute the optical flow of interest points like corners as discussed above. (Farnebäck, 2003) proposed an algorithm to compute the dense flow field which is the flow(motion trajectory) for all of the pixels in the image frame. This work proposes a two-frame algorithm for motion estimation. It employs quadratic polynomials by making use of the *polynomial expansion transform* to approximate the neighborhood of the consecutive frames.

A.5. Mixture of Gaussians

A Gaussian mixture model is a probabilistic model that makes an assumption that all data points are produced from a blend of a limited number of Gaussian distributions with parameters that are unknown. One can consider mixture models as extrapolating k-Means clustering to fuse knowledge about the covariance structure of the data just as the center points of the inactive Gaussians. It is also a kind of a clustering technique. As the name suggests, each of the clusters is a part of a different gaussian distribution. This results in a probabilistic and flexible approach to model the data leads to soft-assignments unlike the k-means clustering algorithm which employs hard assignments. So there is always a probability for a datapoint to belong in a cluster. In fact, each of the distributions(part of the mixture) will have a non zero probability over generating any datapoint. This technique enjoys application in a wide range of fields which involve modelling or clustering in unsupervised settings.

A.6. Computation Graphs

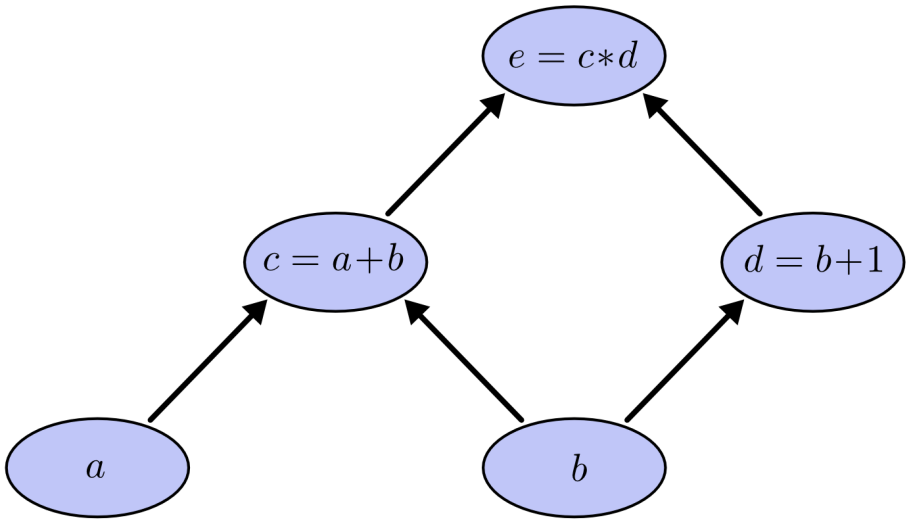

Figure A.2: Computation graph of an expression from [L3]

For calculating derivatives of an expression, we need to use the chain rule. Chain rule is just breaking down the expression to its smallest form to calculate the derivative(gradient) for each step. Computation graph is the breakdown of the expression required to calculate the derivative of the expression. There have been great advancements in frameworks

that enable building of this computation graph effectively and automatically calculate the gradients. The most popular ones being PyTorch and Tensorflow. Backpropagation is the flow of gradients back in a neural network's computation graph. It is the single most important algorithm that has enabled training of deep neural networks computationally feasible. Deep Learning is not the only field that benefits from backpropagation. It is extensively used for tasks like numerical computing, weather forecasting etc. At its core, it is a trick to compute derivatives rapidly. To get a better intuition about computation graphs, we will look at an example. Suppose we have an expression:

$$e = (a + b) \times (b + 1) \tag{A.1}$$

In this equation, the initial variables are a and b. we can rewrite this equation with a few more intermediate variables as follows:

$$c = a + b$$

$$d = b + 1$$

$$e = c * d$$

Now the computation graph for this expression would look something like the figure A.2. Whenever one node is used for calculating another node, there is an arrow from former node into the latter. Now if we calculate the derivative of e which sits at the top of the graph, we can see how the gradients would flow backwards from it into each of the nodes under it. Inspecting the computation graphs can give us tips about the learning of the network and possibly look for problems like *vanishing or exploding gradients* in neural networks.

A.7. Feedforward Neural Networks

A feedforward network is a kind of neural network that has a series of linear mathematical operations performed at the nodes of the networks and then multiplied with a non linear activation function before reaching the final output. When the information is flowing forward through such a network, the information never goes through the same node more than once. It is different from recurrent architectures as the latter involves looping the information through the same nodes several times before forwarding it across for the output layer. Multi Layer perceptrons are the simplest kind of feed-forward nets. These nets are fairly simple with the input example being fed on one end and the output is compared to the expected output on the other end and the loss is backpropagated to update the weights in the network to actually fit to the training data. It is usually trained till it minimizes the error it is making while making the predictions. A network like this can then be used on data that the network hasn't even seen before as the network learns patterns in the multidimensional data which are applicable to even the test set. This is the reason that the results get better and better as we acquire more and more data. Convolutional networks are also a special variant of the feedforward network built specially for working with 3-chanelled (Red/Green/Blue) image data. Unlike the recurrent networks, feedforward networks have no understanding of order in time and it looks at the input all together. Technically it only remembers the formative snapshots of training.

A.8. Activation Functions

Activation functions are the mathematical equations that help compute the output of a typical neural network architecture. Usually this function is applied to each neuron in the network. This function determines if the neuron is activated or not based on the input given by the user so that the model makes the correct prediction based on weather the input to that neuron is relevant for the prediction. Activation functions usually clamp the output of the input to a small range like $[-1,1]$ or $[0,1]$ which leads to a normalizing effect. Another property of the commonly used activation functions is that they are not computationally demanding and their derivatives are very easy to calculate as this function is applied to millions of neurons in a network and needs to be computed again for each training sample(or batch). Modern neural networks use backpropagation to have the gradients flow backwards in the computation graph and using simple activation function (with a simple derivative) helps speed-up the flow and thereby faster convergence.

A.8.1. Linear Activation Functions

Several linear transformations can be squished to a single linear transformation. If the activation function is a linear function, as the data is propagated forward in the network, no matter how deep the network is, the last layer will be a linear function of the entry layer. Hence, a linear activation function would render having a deep network useless as it would be equivalent to having a single linear layer which is equivalent to a simple linear regression model. Such a model has very limited flexibility and can't cope with the complexity of real world problems.

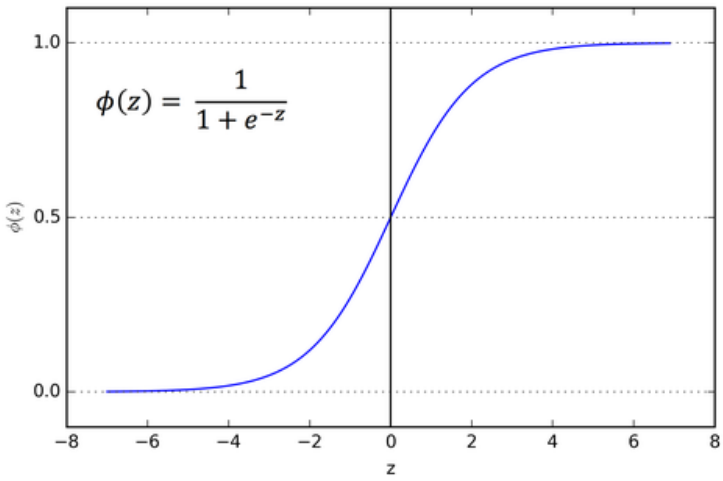

Figure A.3: Sigmoid Activation Function from [L4]

A.8.2. Non-Linear Activation Functions

Most of the modern neural network architectures use non-linear activation functions. These allow the model to have complex computation graphs between the inputs and outputs of the network which is essential for learning and fitting to complex multidimensional datasets like images, audio, video etc. Because of the non-linearity, these functions allow stacking up of several layers in the deep network which are essential for the model in

learning high level features. A few popular activation functions are discussed below:

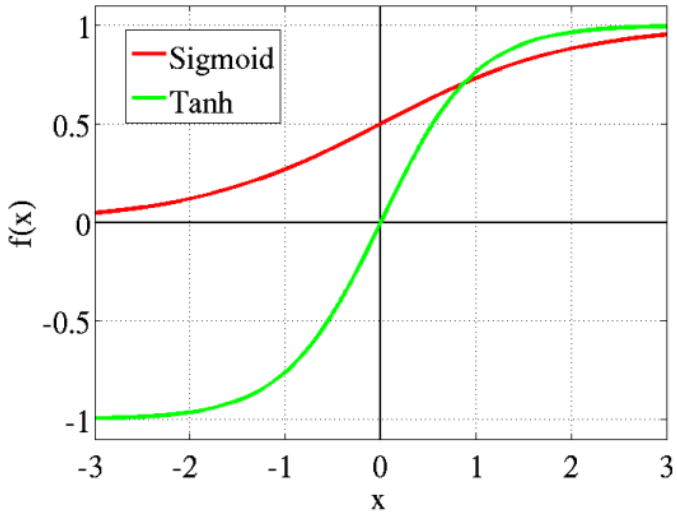

Figure A.4: Tanh with Sigmoid Activation Functions from [L4]

- **Sigmoid or Logistic**: This function ensures a smooth gradient and prevents any jumps in the output. It squeezes the output between 0 and 1. Figure A.3 shows the plotted sigmoid function.

$$f(x) = \frac{1}{1+e^{-x}}$$

$$f'(x) = \frac{f(x)}{1-f(x)}$$

  This function suffers from the problem of vanishing gradient for very high or very low values of the input. This can cause the network to learn nothing or being too slow to converge to an accurate prediction. The outputs of this model are not zero centered and this is also computationally demanding.

- **Tanh or hyperbolic tangent**: It is very similar to the logistic sigmoid function but it's range is from -1 to 1. The shape of this function compared to a sigmoid function is shown in the figure A.4.

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

$$f'(x) = 1-f(x)^2$$

  This function is zero centered and makes it easier to work with strongly positive or negative inputs. This can also suffer from the problems that the sigmoid function suffers from like vanishing gradients.

- **ReLU - Rectified Linear Unit**: It is a very simple activation function and the most popular in deep learning literature. This function clamps the output below 0 to 0 and keeps the output the same as the input otherwise. Figure A.5 shows the ReLU function compared to the sigmoid function.

$$f(x) = max(0,x)$$

$$f'(x) = 1\ (if\ x > 0)$$
$$f'(x) = 0\ (if\ x <= 0)$$

This function is computationally very efficient and usually the network converges pretty quickly using this. This suffers from a problem called the dying ReLU problem when inputs are close to zero or are negative in which case the network is unable to learn.

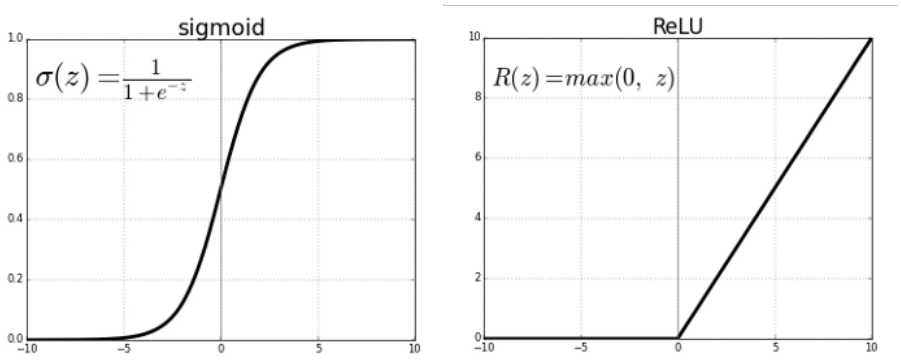

Figure A.5: Sigmoid(left) with ReLU(right) Activation Functions from [L4]

- **Leaky ReLU and variants**: Over time, several new variants of the popular ReLU function have been proposed like the Leaky ReLU. This function doesn't clamp the value below 0 to 0 but lets some fraction leak out (hence the term leaky ReLU). Because of a positive slope even in the negative region, this allows for learning(backpropagation) even for inputs that are negative. A shortcoming of this is that the results are not always consistent for the negative input values.

A.9. Residual Learning or Skip Connections

In a traditional feedforward neural network, each layer feeds into the next layer. While residual blocks enable each layer feeding into the next layer and into more layers a few hops away. This idea was quite ground breaking and vastly improved the performance of state of the art networks on a variety of tasks (ResNet introduced by (He et al., 2016a)). Theoretically, as we increase the number of layers in a network, its learning capacity should increase and should result in better accuracies since neural networks are universal function approximators. But, in practice, the deeper networks don't always perform as well as the shallower variants due to the problem of vanishing gradients. Figure A.6 shows one such example. The idea was that it was needed to skip the extra layers and atleast match the accuracy of the shallower sub-networks and hence the skip-links or residual links were designed. These are also referred to as identity shortcut connections. These act like highways for the gradient to flow backwards and not die on the way thus enabling the creation of very deep networks.

A.10. Batch Normalization

Batch normalization was first proposed by (Ioffe and Szegedy, 2015). This is typically used to improve the stability of the neural network even with eratic input data. The idea is that

this step normalizes the output of the previous activation layer by subtracting the batch mean from it and dividing the remaining value by the batch standard deviation. The activation outputs are not optimal anymore because the weights are transformed(shifted or scaled). This needs to be un-normalized and that is handled by the Gradient Descent while trying to minimize the loss function. This step of batch normalization introduces two new trainable parameters in each layer which are the mean (μ) and standard deviation (σ). Batch normalization relies on Gradient Descent to do the denormalization by changing only these two trainable weights for each activation instead of changing all the weights and losing stability of the network. This stability results in a normalizing effect and the results show signs of improved robustness.

A.11. Optimizers

While training the neural network, we need to tweak and change the trainable weights of the model to reach a configuration that minimizes the loss function to make predications as accurately as possible. To decide when,how and by how much to change the weights of the model, we rely on optimizers. In response to the output of the model with respect to the loss function, the optimizer updates the model weights and hence ties the model parameters with the loss function. Typically, the optimizer tries to go in the direction where the loss is decreasing by looking at the slope(gradient) of the loss function by taking small steps (decided by the learning rate) in the right direction. It is not possible to know what the weights need to be from the beginning, but with trial and error and inputs from the loss function, the optimizer can help the model converge to a local minima. A few popular optimizers are discussed below:

- **Stochastic Gradient Descent**: This calculates the gradient for only a subset of the training examples for every pass of gradient descent instead of calculating it for all of the training examples. It usually employs random examples or batches at a time for each pass.
- **Adagrad**: This approach changes the learning rate such that some weights in the network have different learning rates than the others. It is usually impressive whenever where are missing input examples and the dataset is sparse. One problem with this approach is that the learning rate eventually gets too small over time.
- **RMSprop**: This approach is a variant of the Adagrad discussed above. It was developed by *Professor Geoffery Hinton*. It only gathers gradients in a fixed window instead of gathering all the gradients for momentum. It tries to solve issues that the Adagrad approach left open.
- **Adam**: This is the most popular optimizer and is widely accepted in deep learning literature. Adam stands for "adaptive moment estimation". It is a way of calculating gradients by making use of the past gradients. It also benefits from the concept of momentum by adding fractions of the older gradients to the present one.

A.12. Regularization

The most common problem in machine learning is of Overfitting. Overfitting means that the model learns to do well on the training set, but, performs poorly on the data that it

hasn't seen before. This usually happens if one of the parameters is weighed too much and dominates the entire formula. This can be usually rectified by employing some kind of regularization in the optimization process. This is usually performed by adding a small piece into the loss function that discourages large weight values by penalizing them. This would mean that the it will be penalized for not only wrong predictions but also for huge weight values even if the predictions are accurate. This ensures that the weights stay sane and generalize well on unseen data. A few popular kinds of regularization techniques are discussed below:

- **L2 regularization**: This is the most common form of regularization used in practice. It can be simply implemented by punishing the squared magnitude of all parameters directly in the objective(loss) function. That means that for every weight(w) in the network, we add $\frac{1}{2}\lambda w^2$ to the loss function (where λ is the strength of regularization). Because of this, the L2 regularization discourages peaky weights and prefers diffused weight vectors. This forces the network to make use of all the features instead of using just a few features a lot.

- **L1 regularization**: This is another relatively common type of regularization. In this, for each weight w in the network, we add $\lambda|w|$ to the objective function (where again λ is the regularization strength). Using this type of regularization leads to the network using only a few of the input features and becomes sparse unlike the L2 regularization. This makes the model immune to noisy input values.

- **Dropout**: Another popular form of simple and effective regularization is the Dropout as proposed in (Srivastava et al., 2014). It complements other types of regularization discussed above and can often be used in conjunction with either of them. This is implemented by letting a neuron train with some probability p (which is a hyperparameter) on each pass and setting it to zero otherwise. This leads to a more robust model which generalises better on unseen datasets.

A.13. Loss Functions

Optimization algorithms typically use a function to evaluate a proposed candidate solution and that function is called the objective function. The solution proposed would be the set of trainable weights in our network. We usually need to minimize or maximize the objective function to converge on the best candidate solution(with the best score). Usually, in case of neural networks, the objective function is referred to as the loss function or simply *loss* which we seek to minimize. Several functions could be employed to gauge the error of a given set of weights(candidate solution). Using iterative updates to the model weights, the high dimensional terrain of the objective function can be navigated and a local minima can be reached. A popular framework for finding estimates of parameters from past training data is the **Maximum Likelihood Estimate**. It uses cross-entropy to measure the error between two probability distributions. For classification problems where each input variable needs to be mapped to a class label, the probability of the example belonging to each of the classes can be predicted. For the training examples, the probability of the actual class would be kept to 1 and the others to 0. Finally, the weights that the model-training converges on would ensure that the correct class gets the highest probability while predicting labels. Under the inference framework of Maximul Likelihood Estimate, the most pre-

ferred loss function is the **Cross Entropy Loss (also known as th Negative Log Likelihood loss)**. It computes a score that quantifies the average difference between the predicted and actual probability distributions for all labels(classes) in the problem. If the number of classes(labels) we have is j, and the number of training examples we have is N, the Cross Entropy Loss per training example is formally expressed as:

$$L_i = -log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \quad j \neq y_i$$

Another commonly employed loss is the **Multiclass Support Vector Machine(SVM) loss also known as the margin loss**. It computes a score for each class such that the correct class has the highest score. This loss doesn't emit convenient probabilities like the cross entropy loss does. The margin is denoted by Δ. The margin loss per training example is formally expressed as:

$$L_i = \sum_{j \neq y_i} max(0,\ s_j - s_{y_i} + \Delta)$$

The full multiclass Margin/Cross-Entropy loss can be expressed as:

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W)$$

As shown above, to compute the overall loss we need to average this loss(margin-loss or cross entropy loss) per training sample and add a regularization term which is denoted by $R(W)$ and λ denotes the regularization strength. In practice, both these losses usually lead to comparable results and the differences are usually very small.

A.14. Evaluating CNN architectures to use in the pipeline

Since each frame in the video has too many pixels and each pixel has 3 values in it (corresponding to the intensity of red, green and blue in the pixel), there are too many dimensions in the video. To make use of the recent work already done in the field of computer vision and more specifically, Convolutional Neural Networks (CNN), we evaluated some popular CNN architectures. We only considered the popular architectures as it would be easier to obtain standardised ImageNet pretrained weights for these models. A CNN architecture usually has a series of Convolution layers coupled with max-pooling layers and non-linear activations (discussed in detail in Appendix). These are then connected to fully connected layers which leads up to the number of classes in the dataset the same as the neurons in the last layer. For all the architectures trained on the ImageNet, the number of neurons in the last layer is 1000 as there are a thousand classes in ImageNet. Now, these models, pretrained on ImageNet can be used for a wide array of problems. The initial few layers of the network usually learn to identify simple features like edges, colors and corners etc. As the depth of the layers increase, they usually identify more and more complex features like detecting complete or partial objects in the images. The final fully-connected layers can then be disposed off when we just need the image features and a new fully connected layer can be attached to this pretrained CNN. Based, on the architecture employed, the number of features obtained from the CNN differs. A few metrics that can be compared for these architectures are:

1. Number of trainable parameters (size of the model)

2. Top-5% error (checks if the actual output class is among the top 5 predictions). This would indicate that the model is more or less accurate.

3. Top-1% error (checks if the actual output class is the same as the top prediction). This would indicate how precise the model is.

A few popular models with their top-1% errors, top-5% errors and trainable parameters are shown in the table A.1.

Model Name	Trainable Parameters	Top-5 error%	Top-1 error%
ResNet-18	11,689,512	10.92	30.24
AlexNet	61,100,8407	20.91	43.45
VGG16	138,357,544	8.5	26.63
VGG11	132,863,336	10.19	29.62
Inception-v3	27,161,264	6.44	22.55
SqueezeNet-1.1	1,235,496	19.38	41.81
DenseNet-121	7,978,856	25.35	7.83

Table A.1: Comparing Pretrained CNN architectures

Looking at the trainable parameters of a CNN is a good estimate of how big the network is and how time-consuming it will be to train, but, it is not the definitive answer. Based on the architecture incorporated, the computation graph of a bigger network (more trainable parameters) can actually have fewer number of operations performed on them, which would lead to faster training and easy flow of gradients. Cross-validation and dissecting each architecture is the only way to identify which CNN architecture would perform the best for our use case. Based on our intuitions about the architectures and by looking at the the trainable parameters, the models that we shortlisted were the popular ResNet by (He et al., 2016a), DenseNet by (Huang et al., 2017) and SqueezeNet by (Iandola et al., 2016). Out of the available pretrained ResNet models, we stuck with the smallest ResNet-18 for our experiments which has 11 Million trainable parameters (including the fully-connected layers at the end). SqueezeNet has a relatively small model size with only about 1.2 Million trainable parameters while DenseNet has about 8 Million trainable parameters. Our hypothesis was that ResNet, even though having the largest number of trainable parameters, would perform well for our task because of the skip connections in its architecture which let the gradient comfortably flow backwards.

A.14.1. ResNet-18

The **universal approximation theorem** states that a big-enough simple feedforward network with a single layer is enough to approximately represent any given function. This of course leads to problems like a massive layer and also often leads to overfitting. The research as a whole follows the trend of coming up with deep architectures to avoid the aforementioned issues. The answer is not just stacking up layers. The deeper the network

gets, the more prominent the issue of vanishing gradients becomes. As the computed gradient is propagated backwards from the last layer, a series of multiplications can actually make the gradient very small (almost zero). This results in no information reaching the initial layers while training. As a result, the performance of such an architecture often shows signs of saturation or rapid degradation. Figure A.6 is taken from (He et al., 2016a) shows train and test errors of simple CNN architectures being trained on CIFAR-10. It shows that as the depth of the network increases, there is a higher training and test loss and hence, worse performance.

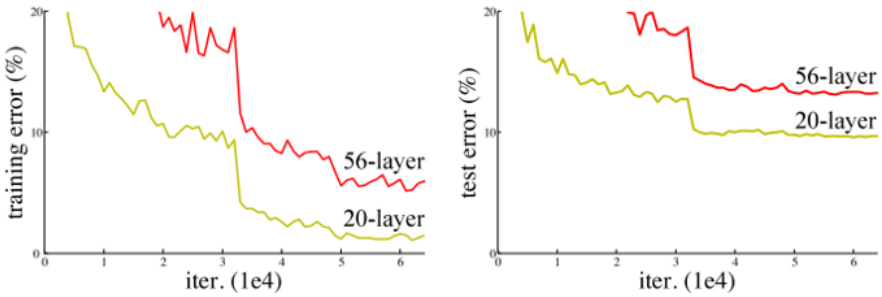

Figure A.6: Image from (He et al., 2016a) - CNN-training on CIFAR-10

Before ResNet came out, there were several ways to tackle the vanishing gradient problem but none of them performed well and could be widely used. One such technique was adding an auxiliary loss in one of the middle layers as an extra measure of supervision (like in (Szegedy et al., 2015)). There was no solution that really solved the issue properly. The main contribution of the paper was introduction of the *identity shortcut connections* which were like direct links that could skip one or more layers. One such block is shown in Figure A.7.

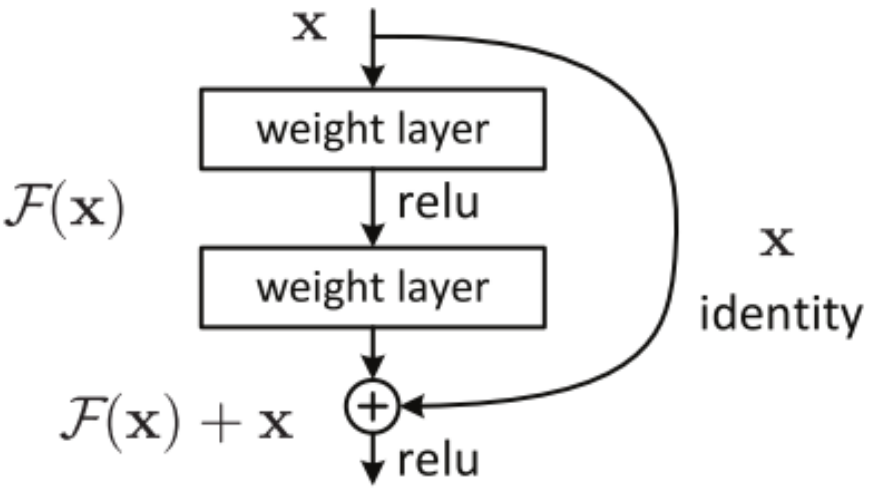

Figure A.7: ResNet Residual Block from (He et al., 2016a)

The authors hypothesized that using this architecture, if the depth of the network is increased, the performance should only get better and never degrade. This is because it

was like appending identity mappings to the existing network and the resultant network should never have a training error which was higher than its shallower variants. They came up with the idea that letting such stacked layers learn a residual mapping should be easier than having the network learn the underlying mapping directly. The residual block shown in Figure A.7 should assist the network to do this. Such skip-connections were not introduced for the first time by ResNet. (Srivastava et al., 2015) introduced one such trick in their work on Highway Networks called the gated-shortcut-connections. These connections limit and decide the amount of information that is allowed to flow through the connection. The idea is also surprisingly similar to the forget-gate of a cell in Long Short Term Memory Networks by (Hochreiter and Schmidhuber, 1997). Hence it is safe to assume that the ResNet was a variant of the Highway Networks. The hypothesis was strong but the experiments didn't reveal the desired results. The deeper variants (1202-layer ResNet) of the net were not necessarily better than the shalower counterparts (110-layer ResNet) as was hypothesized before. The same authors improved their work in (He et al., 2016b) following the intuition that the **gradient highways** need to be kept clear instead of exploring the wider solution space by using a parameterized gate. This resulted in getting results which were in line with the original hypothesis. A deeper network would always lead to at-least as good results as its shallow counterparts. ResNet gained its well deserved popularity in several computer vision tasks because of these compelling results.
For our experiments, we used the ResNet-18 which is the smallest ResNet variant trained on ImageNet. The feature map of a ResNet-18 which inputs an image of dimensions ($224 \times 224 \times 3$) is a mere 512. These 512 features effectively capture the high dimension complexities of an RGB image with much higher dimensions. This would help reduce the computation costs when computing the Matrix Profile using these features or if these features were directly fed to an LSTM. The highway connections would also lead to a simpler computation graph overall.

A.14.2. DenseNet-121

A Dense Net architecture has quite a few residual connections (skip connections) as inspired by the ResNet. The model shows very good accuracy given its relatively small size of 8 million trainable parameters. Each layer in a DenseNet is directly connected to every other layer in a feed-forward fashion (within each dense block). For each layer, the feature maps of all preceding layers are treated as separate inputs whereas its own feature maps are passed on as inputs to all subsequent layers. The rough architecture is shown in Figure A.8. The DenseNet is bigger than the SqueezeNet but it has an advantage: the number of features that are projected out from a pretrained DenseNet is a mere 1024 compared to the feature output of a SqueezeNet.

A.14.3. SqueezeNet 1.1

SqueezeNet is one of the smallest networks that performs as well as the AlexNet with much fewer parameters. There are several advantages of smaller CNNs like: these require less communication across servers during distributed training, require less bandwidth to export a new model from the cloud to a client device and are also deployable on hardwares with limited memory (like FPGAs). The squeezenet architecture employs several clever tricks to reduce the model size like:

1. Replace the original 3x3 filters with 1x1 filters.

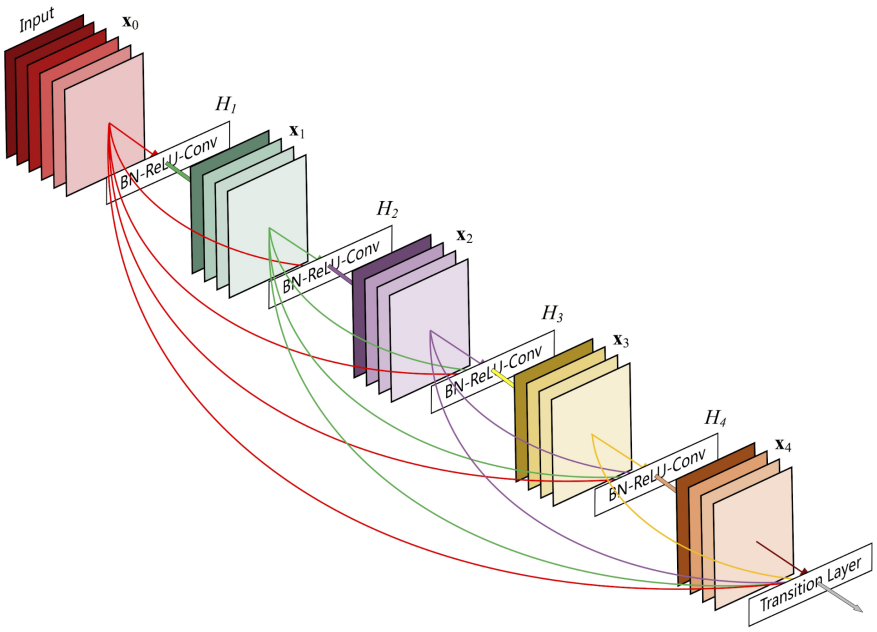

Figure A.8: DenseNet Architecture from (Huang et al., 2017)

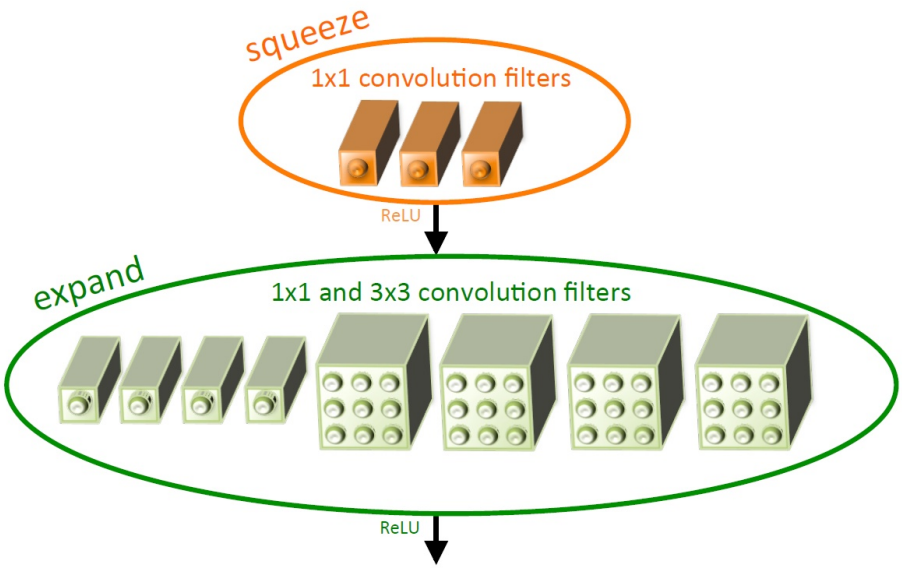

Figure A.9: Squeeze and Expand layer from a SqueezeNet from (Iandola et al., 2016)

2. Reducing the number of input channels using squeeze and expand layers (depicted in Figure A.9)

3. Perform downsampling much later in the network, so the convolution layers have relatively larger activation maps.

The third point above is the reason that the feature map of the squeezenet (size ≈ 86,000) is much larger than a DenseNet feature map (size ≈ 1,000) for an input image of same dimensions. Hence, by looking at all of these CNNs, there was no clear winner. SqueezeNet is a smaller model, but the ResNet and DenseNet boast of a better accuracy on ImageNet and a

smaller feature map. Since there is no clear winner in terms of computation graph size, we decided to experiment with all of these architectures.

A.15. Challenges

Early challenges encountered while training the pipeline:

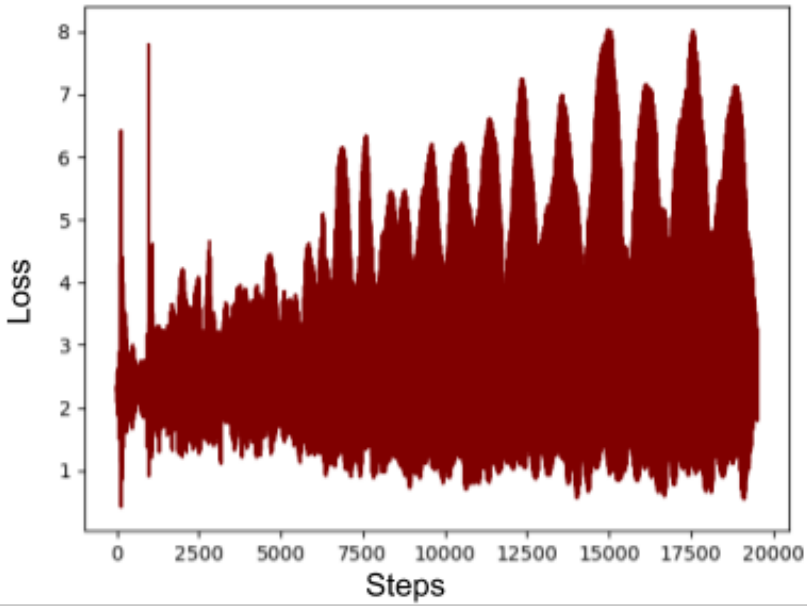

Figure A.10: Flickering loss while training on last-time-step labelled dataset

- NaNs while computing the matrix profile. NaNs are Python's (Not a Number) objects. This is a floating point value that you end up with, if you perform a calculation whose result can't be expressed as a number. Any calculations that are performed with NaN, result in NaN. These NaN values accumulate into the model weights and disrupt the entire learning process and intermediate results. Look at the following example which caused NaNs while calculating the matrix profile:
$\sqrt{(x-1)^2} = |x-1|$. Now, the derivative of this is:

$$\frac{d|x-1|}{dx} = \begin{cases} 1, & \text{if } x > 1 \\ -1, & \text{if } x < 1 \\ NaN, & \text{if } x = 1 \end{cases}$$

  A quick fix to this is to add a very small number ε to $x-1$ making it $|x-1+\varepsilon|$ to avoid a NaN gradient.

- *Memory Leaks*: A memory leak occurs when the program doesn't free up resources(RAM) that are not in use and keeps acquiring new resources. With each epoch of training, the memory usage of a typical deep learning model should be stable and not increasing. We faced this issue of leaking memory in our model and after several hours of

debugging and carefully optimizing our code, we still couldn't find any issue with our code. The problem was with *Pytorch's Dataloader* implementation. We found that it was an open Pytorch issue (More information can be found in). The issue thread suggested some workarounds which avoid the memory-leak and ensure that the model training consumes predictable amounts of memory.

- A CNN with randomly initialized weights (instead of using pretrained-on-ImageNet weights). It turns out that learning spatial image features by using just the gradient information from the LSTM is not enough. We couldn't even get the network to overfit on a small training segment. Learning from this we concluded that the repetitions-so-far gradient information from the LSTM is not enough to train the CNN block from scratch. Hence, we decided to use pretrained-on-ImageNet weights for initializing the weights of the CNN block.

- Using the Last-time-step labelling on videos. This approach meant that we would backpropagate the gradient only from the last time step back into the LSTM and then further back into the CNNs. The wave-cycle experiments showed that this approach struggled even on a 1 dimensional dataset and it was too hard for the network to learn anything using this approach. The training loss as shown in the figure A.10 was just flickering and showed no signs of a downward trend even after 100s of epochs. After these underwhelming results, we decided to help the LSTM to learn by feeding it Labels at each time step by using the *Each time step labelling* (discussed above).

B

Appendix II: Loss Curves and Accuracies

This appendix contains plots for training-losses and accuracies for all the model types discussed at the end of the Chapter 4. All of the plots with histograms indicate the number of videos in the test set that had the same (or similar) accuracies. Each of these plots is discussed in detail in the Chapter 5.

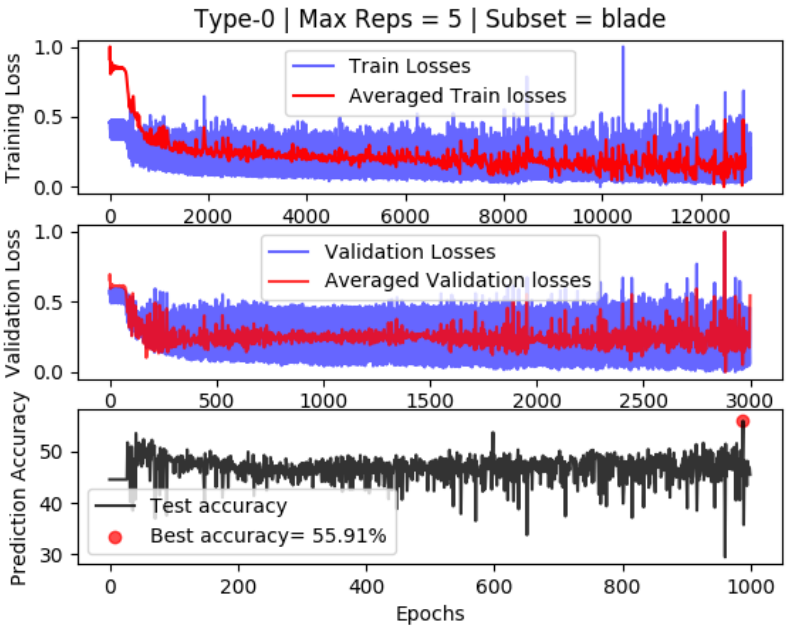

Figure B.1: Losses and accuracies of Model-Type-0 on subset=blade

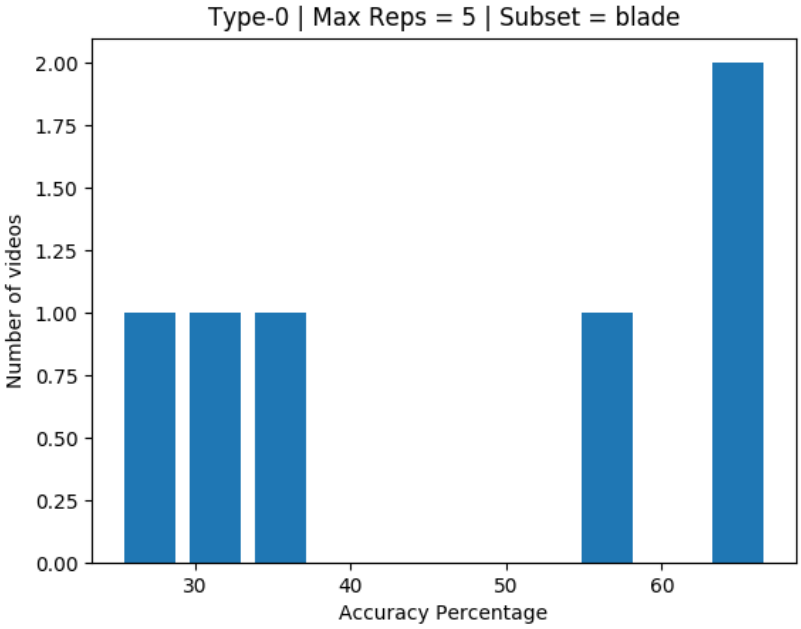

Figure B.2: Model accuracies per video(in the test-set) Model-Type-0 on subset=blade

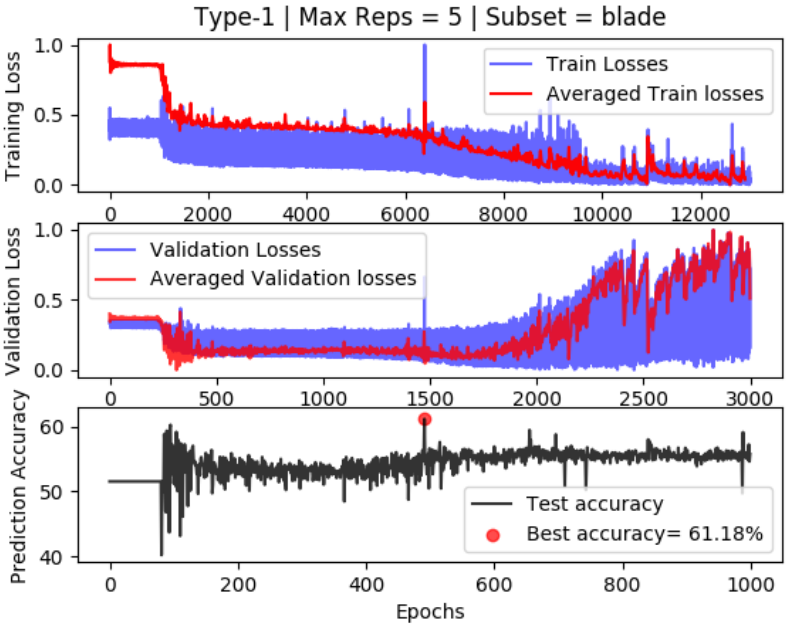

Figure B.3: Losses and accuracies of Model-Type-1 on subset=blade

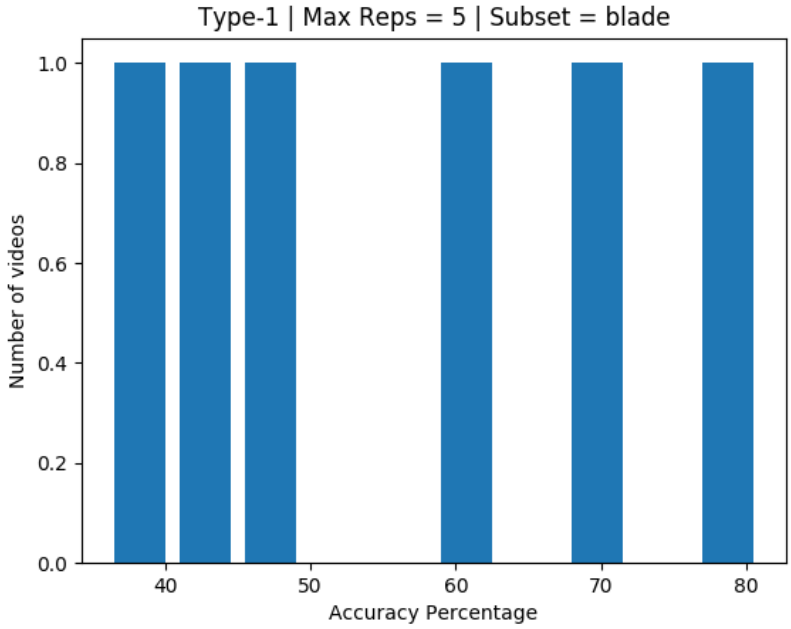

Figure B.4: Model accuracies per video(in the test-set) Model-Type-1 on subset=blade

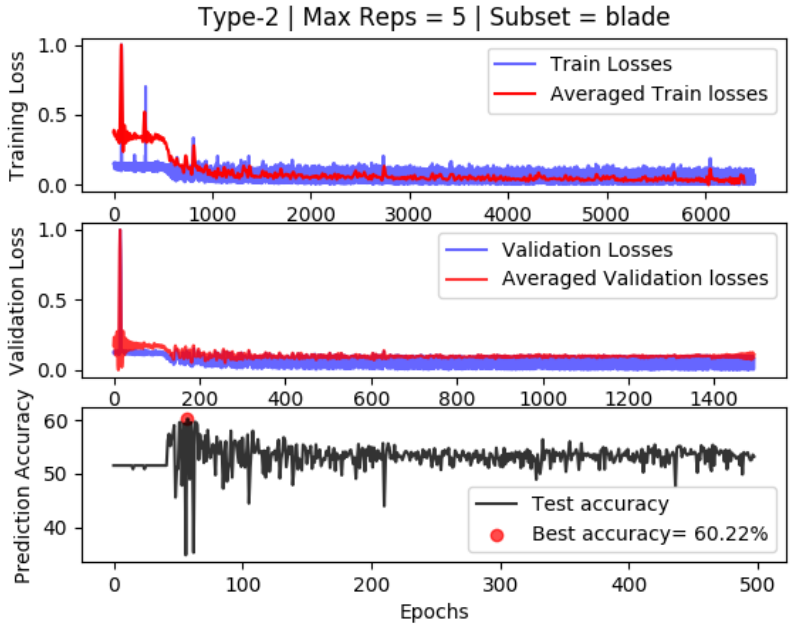

Figure B.5: Losses and accuracies of Model-Type-2 on subset=blade

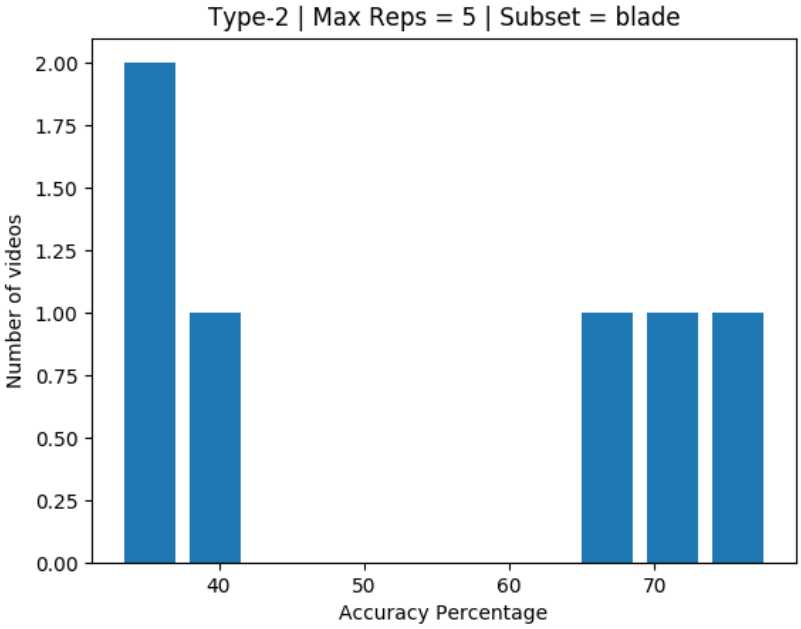

Figure B.6: Model accuracies per video(in the test-set) Model-Type-2 on subset=blade

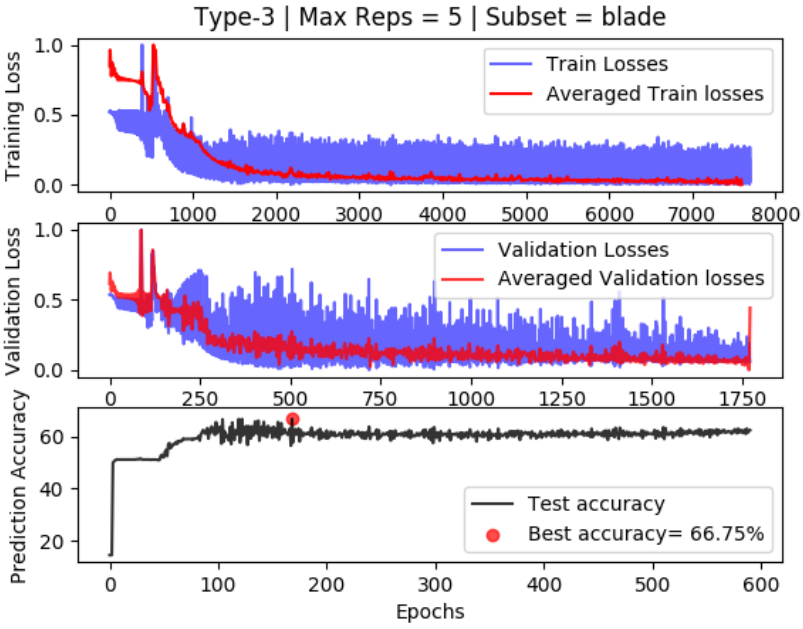

Figure B.7: Losses and accuracies of Model-Type-3 on subset=blade

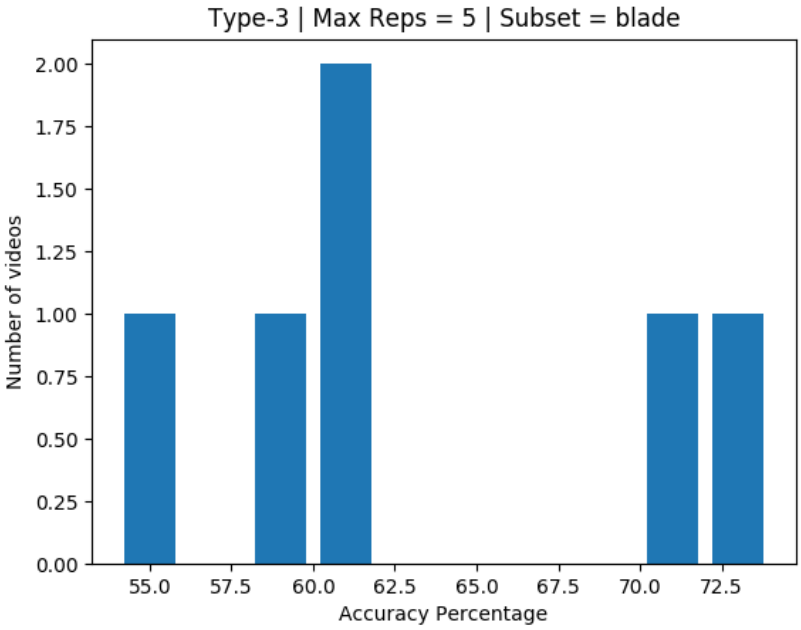

Figure B.8: Model accuracies per video(in the test-set) Model-Type-3 on subset=blade

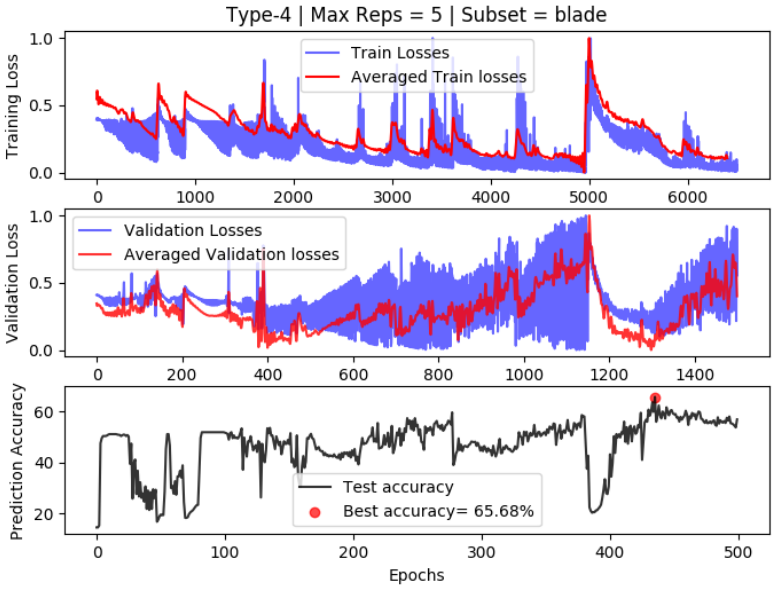

Figure B.9: Losses and accuracies of Model-Type-4 on subset=blade

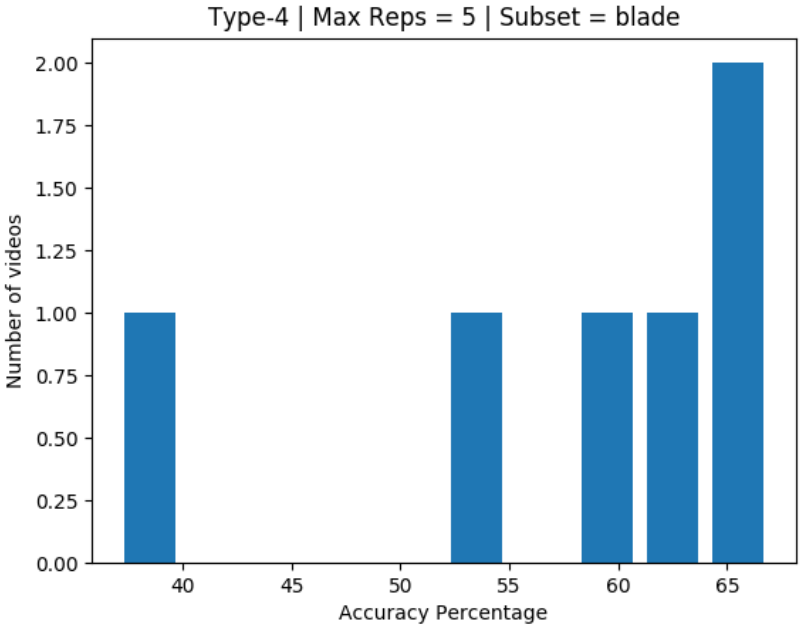

Figure B.10: Model accuracies per video(in the test-set) Model-Type-4 on subset=blade

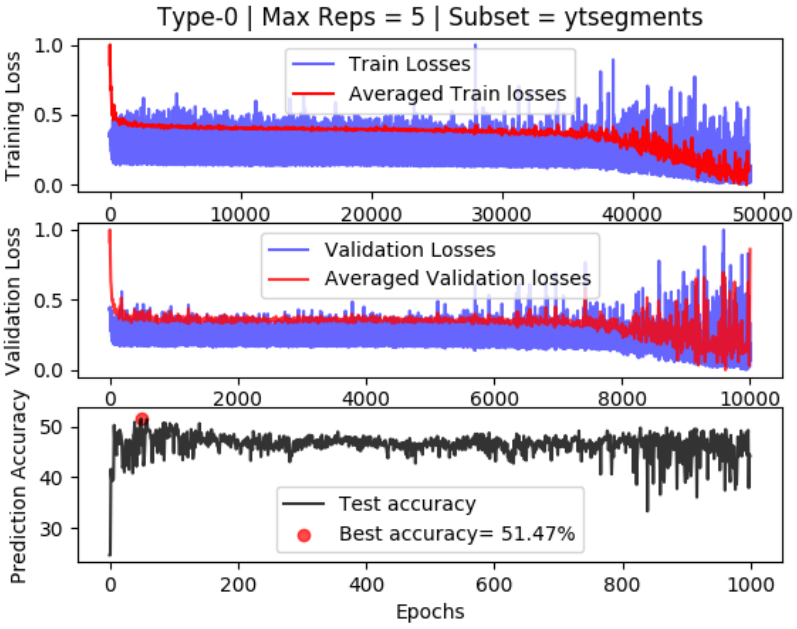

Figure B.11: Losses and accuracies of Model-Type-0 on subset=ytsegments

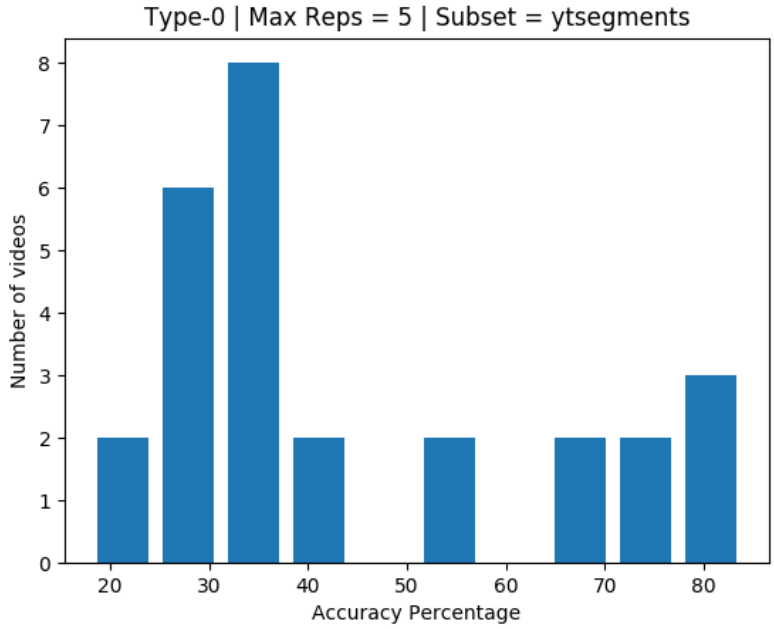

Figure B.12: Model accuracies per video(in the test-set) Model-Type-0 on subset=ytsegments

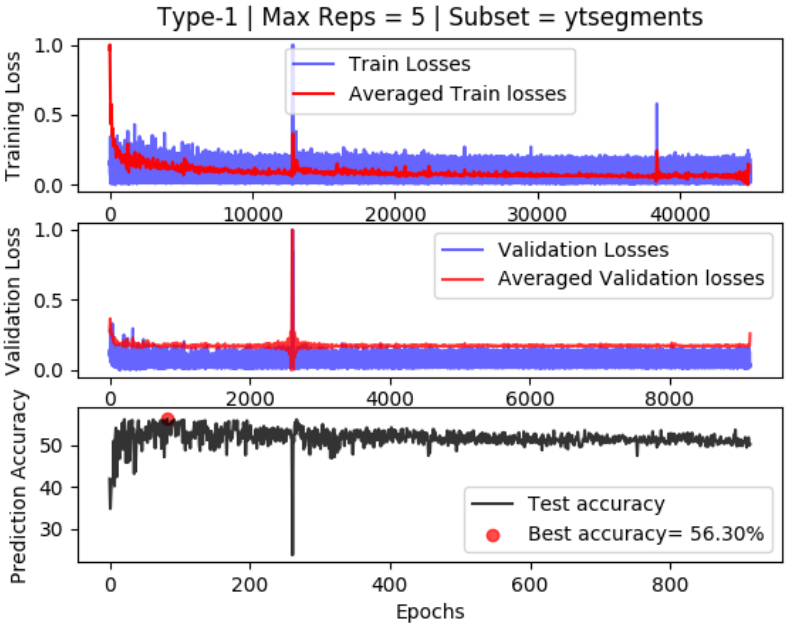

Figure B.13: Losses and accuracies of Model-Type-1 on subset=ytsegments

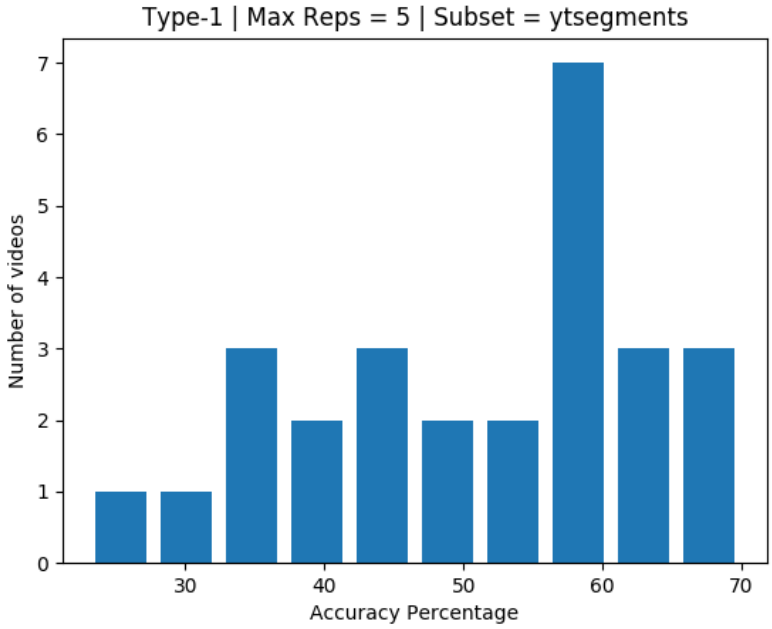

Figure B.14: Model accuracies per video(in the test-set) Model-Type-1 on subset=ytsegments

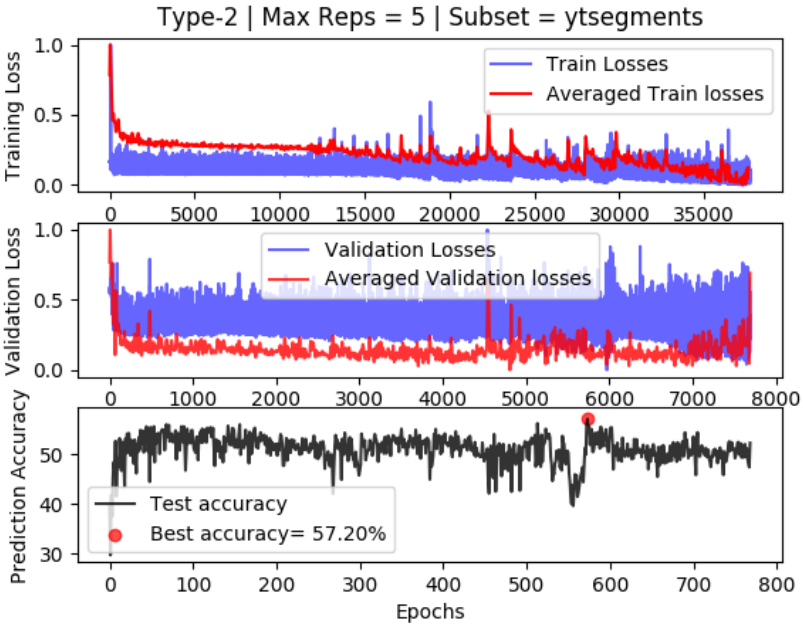

Figure B.15: Losses and accuracies of Model-Type-2 on subset=ytsegments

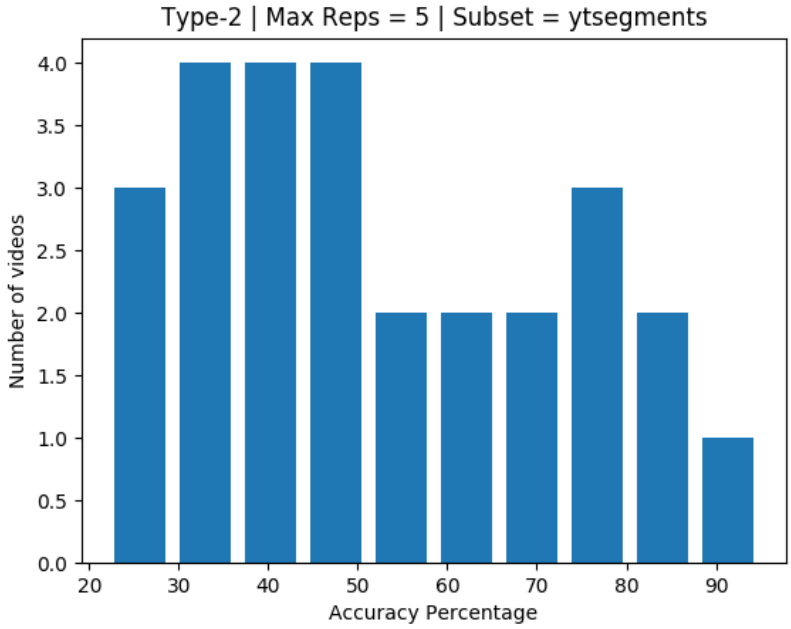

Figure B.16: Model accuracies per video(in the test-set) Model-Type-2 on subset=ytsegments

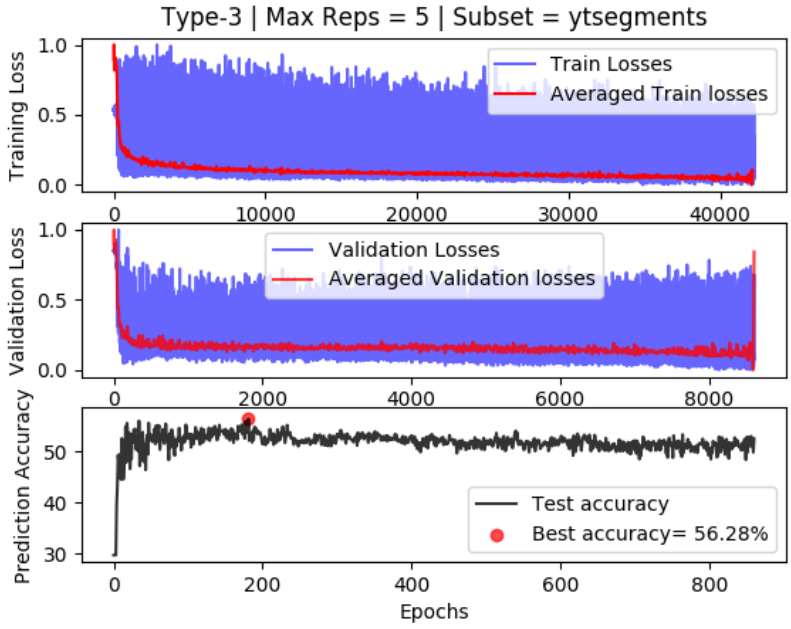

Figure B.17: Losses and accuracies of Model-Type-3 on subset=ytsegments

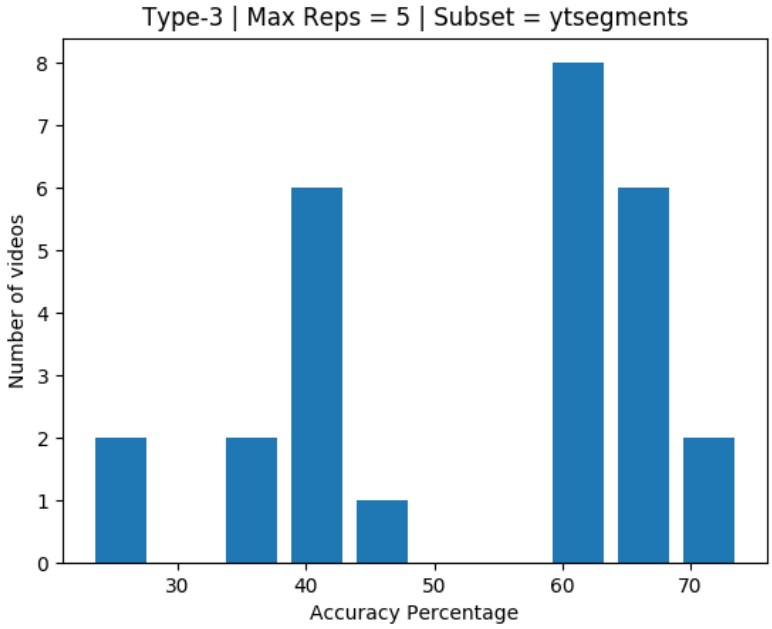

Figure B.18: Model accuracies per video(in the test-set) Model-Type-3 on subset=ytsegments

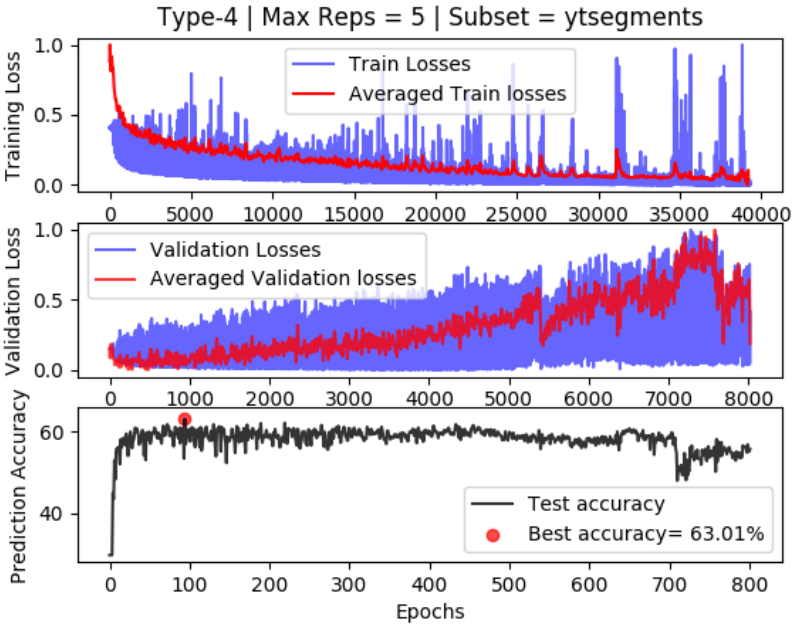

Figure B.19: Losses and accuracies of Model-Type-4 on subset=ytsegments

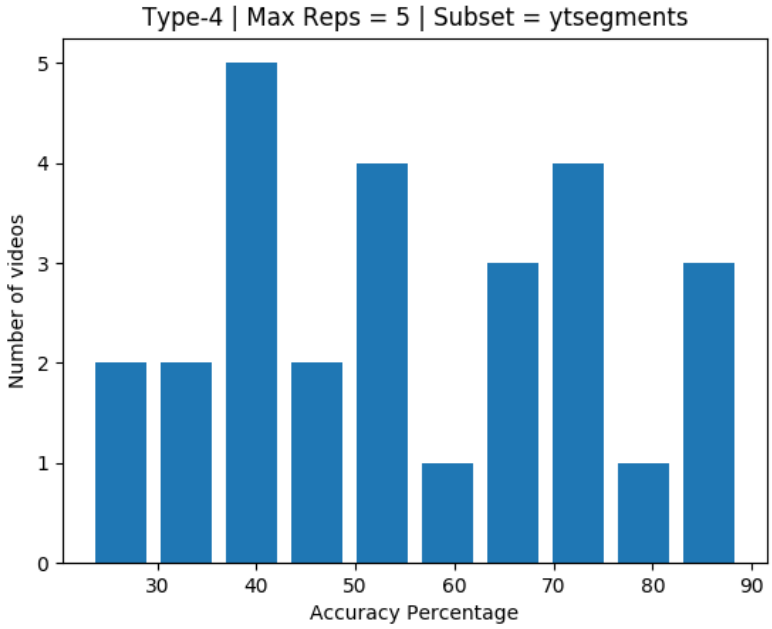

Figure B.20: Model accuracies per video(in the test-set) Model-Type-4 on subset=ytsegments

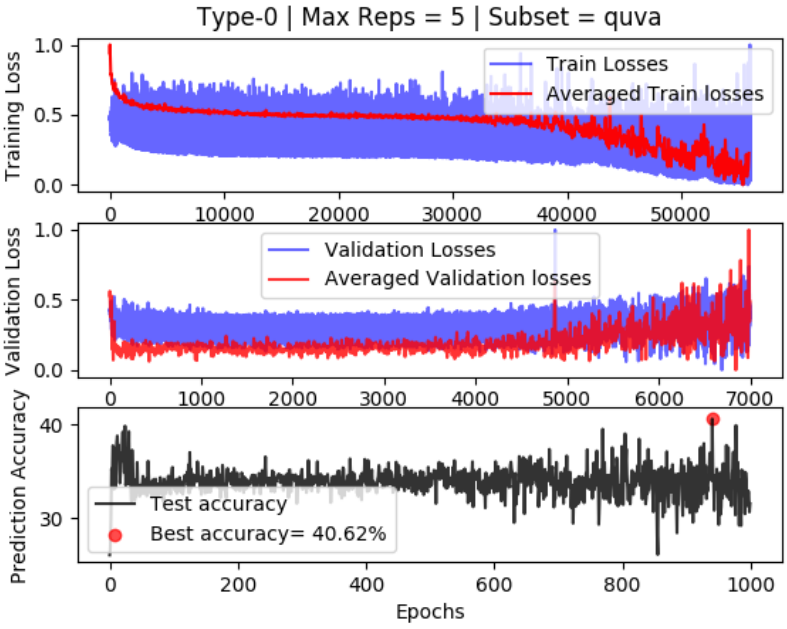

Figure B.21: Losses and accuracies of Model-Type-0 on subset=quva

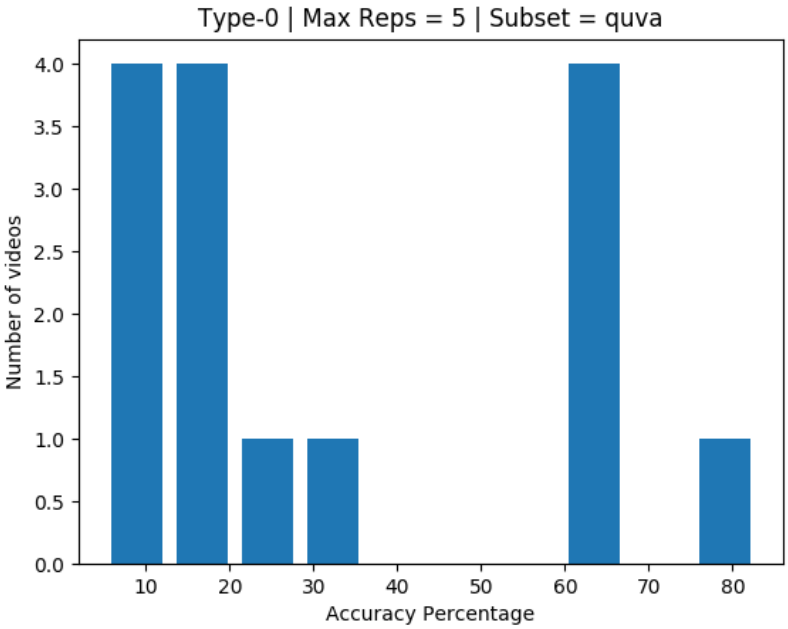

Figure B.22: Model accuracies per video(in the test-set) Model-Type-0 on subset=quva

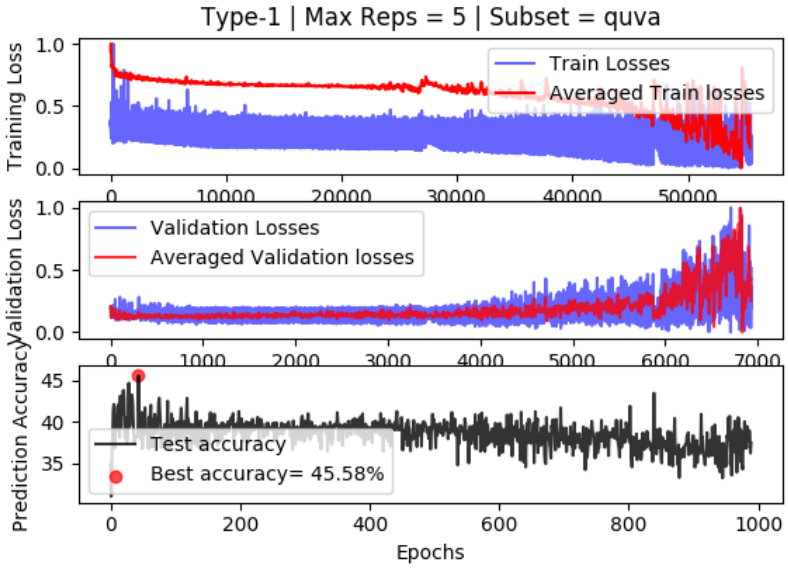

Figure B.23: Losses and accuracies of Model-Type-1 on subset=quva

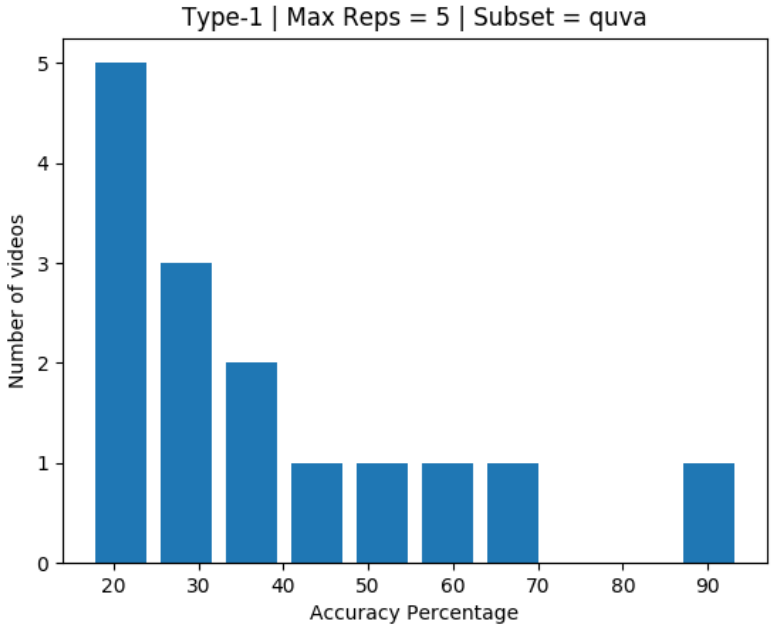

Figure B.24: Model accuracies per video(in the test-set) Model-Type-1 on subset=quva

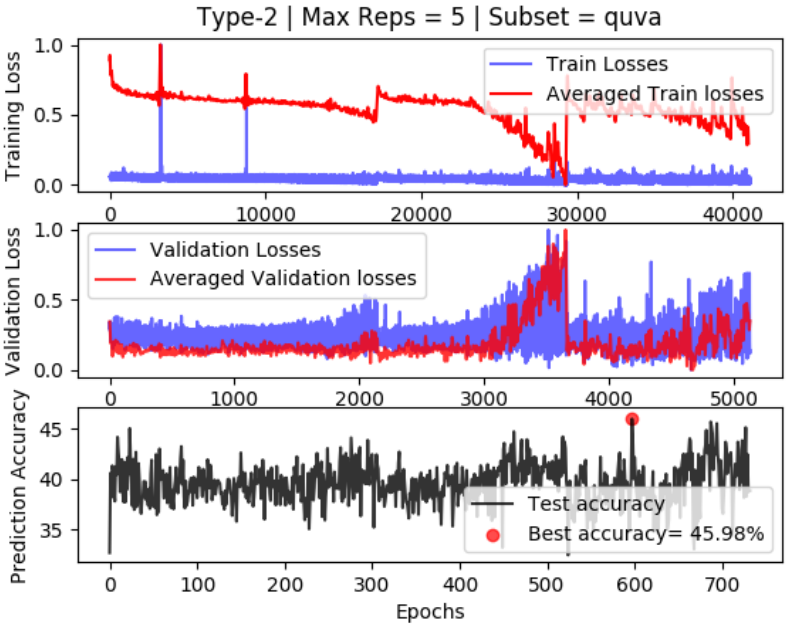

Figure B.25: Losses and accuracies of Model-Type-2 on subset=quva

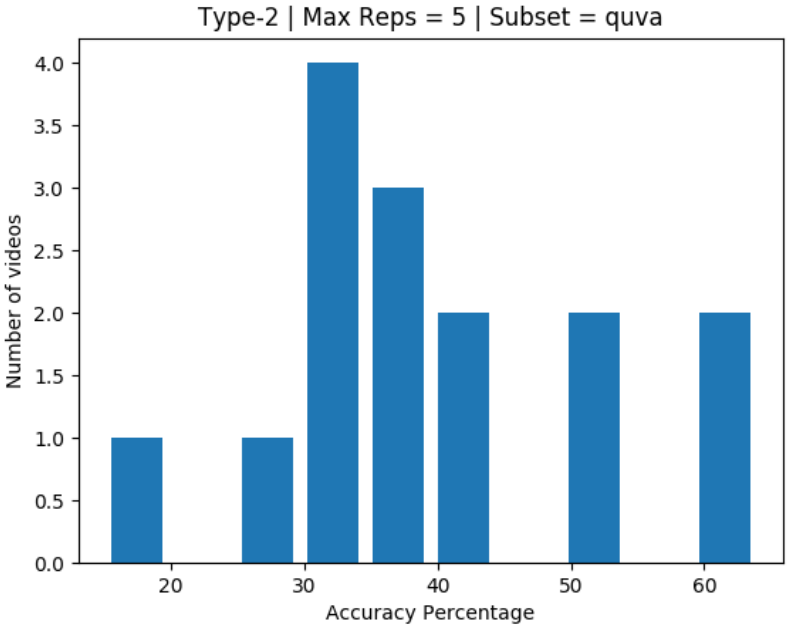

Figure B.26: Model accuracies per video(in the test-set) Model-Type-2 on subset=quva

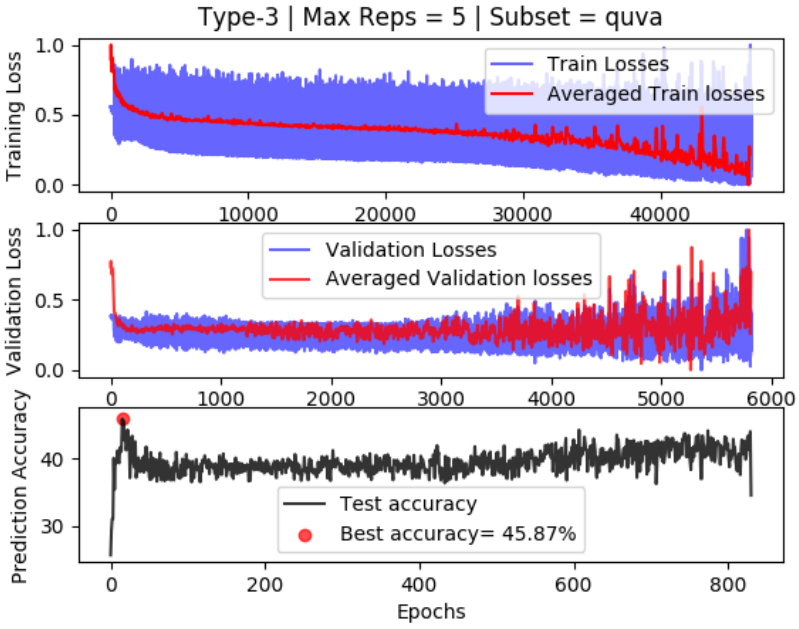

Figure B.27: Losses and accuracies of Model-Type-3 on subset=quva

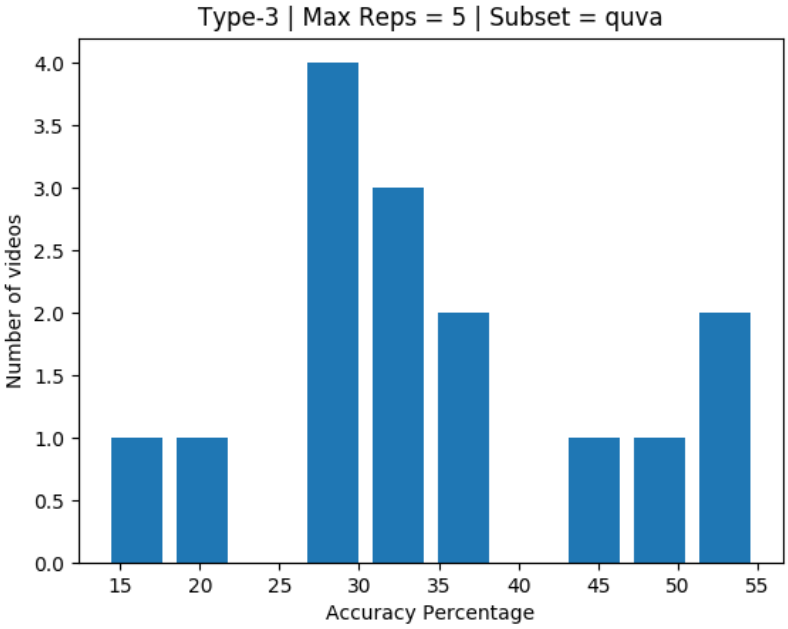

Figure B.28: Model accuracies per video(in the test-set) Model-Type-3 on subset=quva

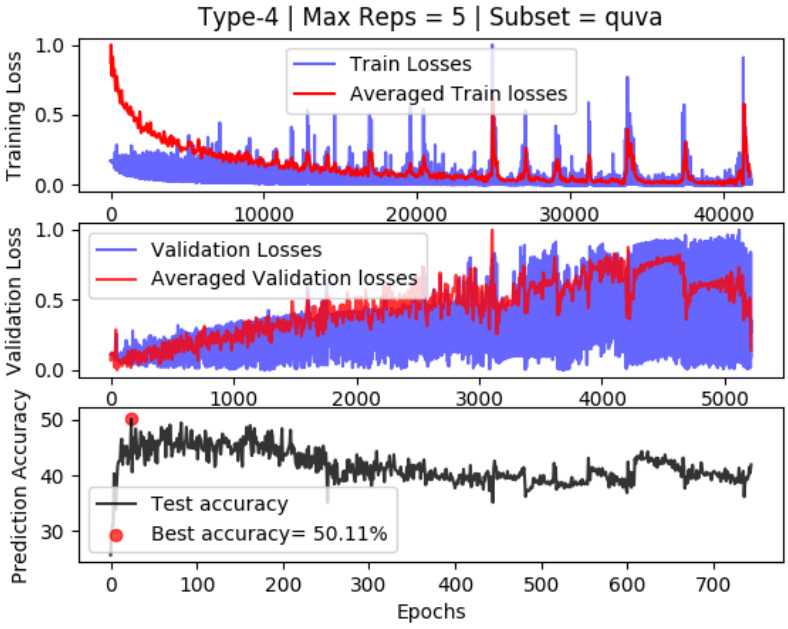

Figure B.29: Losses and accuracies of Model-Type-4 on subset=quva

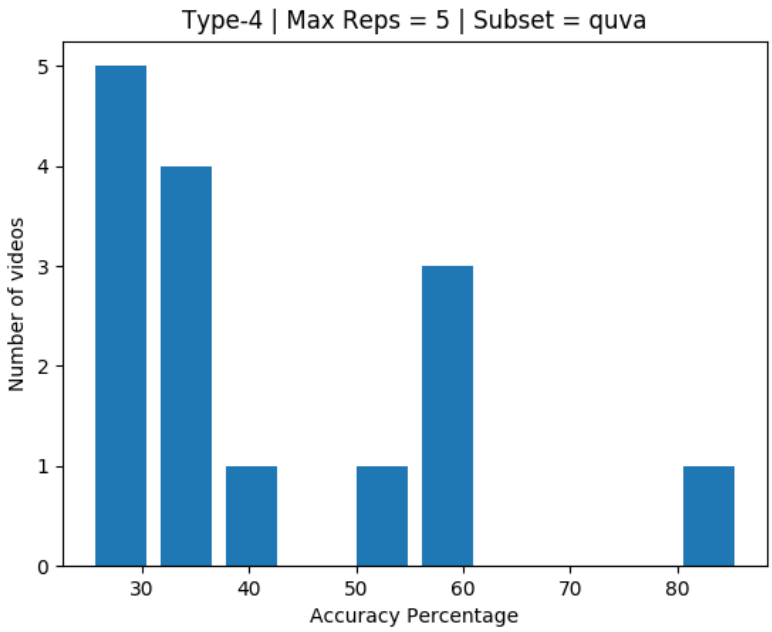

Figure B.30: Model accuracies per video(in the test-set) Model-Type-4 on subset=quva

C

Academic article

Repetition counting in videos using deep learning

Dhruv Batheja
Delft University of Technology
Master of Science in Computer Science, track Data Science
d.batheja@student.tudelft.nl

Abstract

This work tackles the problem of repetition counting in videos using modern deep learning techniques. For this task, the intention is to build an end-to-end trainable model that could estimate the number of repetitions without having to manually intervene with the feature selection process. The models that exist currently perform well on videos which are stationary but, realistic videos are rarely perfectly static. A series of intermediate experiments are performed to eventually come up with an end-to-end trainable pipeline. Techniques like the University of California Riverside's Matrix profile, bi-directional recurrent neural networks and convolutional neural network architectures that employ dilation are experimented with for the task at hand. For the experiments, a variety of videos from the Qualcomm and University of Amsterdam (QUVA) repetition dataset and the YouTube Segments (YTSegments) dataset are used which both exhibit a good number of non-static videos of real life scenarios like people exercising, chopping vegetables, etc. A proprietary Aircraft inspection dataset which contains repetition of spinning engine blades is also experimented with. The proposed model obtains a lower Mean Absolute Error than the existing deep learning architectures. Finally, the model proposed in this work is able to estimate repetitions on a variety of videos successfully in real time without manual intervention. On the task of repetition estimation, an accuracy of about 60% of correctly labelled frames (with repetitions so far) on unseen test videos is obtained.

1. Introduction

1.1. The Problem and Applications

This work tackles the problem of estimating repetition in videos. Humans are virtually surrounded by visual repetition. It is ubiquitous in a diverse set of domains like sports, music or cooking. Repetition occurs in several forms because of its variety in motion continuity and motion pattern. The perception of 3D motion in 2D depends on the viewpoint which is of course crucial for the cognizance of recurrence. Periodicity can denote a variety of movements including animal/human movements like a beating heart, breathing lungs, flapping of wings, running, etc. Movement of the blades of a windmill, spikes of a wheel, a swinging pendulum are also periodic. Periodicity or repetition is also ever-present in natural, industrial and urban environments like blinking lights or LEDs or traffic patterns or leaves in the wind. Repetition and rhythm are already used to approximate velocity, estimate progress and to trigger attention [19]. This has many applications in computer vision as well, like: it can be used for human motion analysis, action localization [20], action classification [13], [24], 3D reconstruction [3], camera calibration [18], activity recognition [4], object tracking [5] etc. Some other domains that can potentially benefit from repetition estimation include and not limited to: applications in high throughput biological-experiments, gaming and surveillance. Due to its applicability in a wide range of domains, this problem has enjoyed an increasing amount of attention.

At the semantic level, this task is actually well-defined. In fact, humans are able to perform this task without a lot of difficulty. The existing work shows acceptable results under the unacceptable assumptions of stationary and static periodicity. In real life, the video is rather complex and rarely stationary or perfectly periodic. The popular fourier-based measurement is more than often, not upto the mark in real-life applications. In practice, the movement of the camera makes this task inescapably hard. Other works try to manually perform feature engineering like foreground-segmentation, object localization, flow field calculation etc. to perform this task. This work tries to solve the problem under more natural and realistic conditions as nonstationary is actually the norm. This work experiments with three different video datasets as discussed in Section 3. A modern deep learning alternative to tackle the problem is proposed. Instead of manually engineering the features, the proposed architecture lets the model learn from the labelled dataset that is provided to it. This end-to-end trainable architec-

ture will only keep getting better as more labelled videos are provided to it with little manual intervention.

1.2. Research Scope

The aim of this research is to answers the following questions:

1. **How can a Recurrent Neural Net be used to estimate repetitions in a video?**
   Hypothesis: Recurrent Neural networks (More specifically its variants like Long Short Term Memory Networks (LSTMs) [16]) are known to perform very well on tasks that involve temporal sequences because of their architecture which allows them to maintain a memory as a sequence is fed through them. Videos are temporal sequences of image frames, so theoretically, a big enough recurrent network should be able to tackle the task of repetition estimation by feeding it frame-image-pixels directly or frame-wise Convolutional Neural Network(CNN) features.

2. **How to use the University of California Riverside's Matrix Profile in conjunction with a Recurrent Neural Network to estimate repetitions in a video?**
   Hypothesis: UCR's Matrix Profile introduced in [34] claims to make any time-series data mining task "trivial". It helps in efficiently finding motifs and anomalies in a multidimensional time-series signal. Theoretically, a matrix profile should be able to detect motifs (repeating patterns) in a multidimensional time-series signal (which videos are). If the matrix profile emits a reasonably reliable signal, a Recurrent Neural Network architecture like an LSTM should be able to estimate repetitions from it.

3. **How does the number of parameters required for the Recurrent Neural Network architecture change based on the Matrix Profile usage?**
   Hypothesis: The Matrix profile is a 1 Dimensional signal (as shown in [34]), compared to the multidimensional frame-image-pixels and CNN-features. The number of trainable parameters required in the recurrent neural network architecture should be dramatically fewer for a variant which employs the Matrix Profile as compared to a variant where frame-image-pixels or CNN-features are directly fed to the Recurrent Neural Network architecture.

4. **Can a backpropagatable feature extraction step help improve the accuracy of our model?**
   Hypothesis: Since the computation of the matrix profile is just a series of differentiable mathematical operations (as shown in [34]) on the raw multidimensional image-frames, it should be possible to backpropagate through the matrix profile. This could potentially result in a trained Convolutional Network that helps in computation of the Matrix Profile signal such that the computed Matrix Profile signal (using this resulting CNN's features) in turn improves the estimation of repetitions in videos.

2. Related work

Because of the numerous applications, there has been quite some research already done in this domain. There have been a variety of attempts in the past to tackle the problem of repetition estimation. These attempts can be broadly classified into a few categories as discussed below:

2.1. Frequency Domain Analysis

Most of the existing literature is dominated by methods that use the fourier transform or methods that incorporate wavelet analysis. These techniques employ fourier transform, and are very susceptible to background noise. The background noise from each frame in the video massively deteriorates the quality of these models, so they usually end up employing techniques of removing the background [26]. This paper presents an approach which performs the time-frequency analyses for the entire video at once. This allows it to extract several periodic trajectories from the video. This also allows one to estimate the separate independent periods simultaneously over a static background. The spatial domain information is used to extract the periodically moving objects. The main idea behind this paper is that the repeating patterns have unique signatures in the frequency space.
To analyze the periodic variations, a **Frequency Modulated** signal is first constructed where the frequency is tuned by the object displacements with time over successive frames in the video frame sequence [11], [30]. These Frequency-domain techniques involve an analysis which is spatially global and not local, which enables them to be effective against local illumination-variances (like in [17]) and occlusions. These techniques don't make any kind of assumptions about the trajectory smoothness or the shape of the objects, so the results hold for a wide variety of videos and is much more reliable than intensity-based techniques like the ones proposed in [2], near the boundaries of the moving objects. The estimation of the periodic cycles using these techniques can also be employed in the spatial domain for tasks like motion segmentation. An unavoidable difficulty that arises with this technique is called the *localization problem* as discussed in [35]. Even though the motion estimation by frequency-domain is global in nature and avoids local errors (like occlusion, illumination changes, object boundaries etc.), it doesn't contain insights about the actual spatial address of the objects in motion. This results in the need for further processing of the video using help from

the spatial-domain to accurately allocate motion-estimates to the frame-pixels. Another difficulty is that these techniques are very susceptible to background noise and therefore can't be used without an expensive background subtraction step.

2.2. Spatial Correlation Methods

Some of the oldest contributions in video understanding focussed on the development of spatio-temporal features from the video sequence for analysis. A majority of the past approaches for periodicity in motion analysis (like [24], [9]) incorporate first detection of the object in successive image-frames and then computing their trajectories and finally analysis of the computed trajectories. More recent spatio-temporal approaches for generic motion analysis incorporate statistical shape priors and levelsets (like in [8], [7]) or under smoothness-constraints, involve computation of the optical flow (like in [6]).

The technique proposed in [27] identifies periodic repetitions in a sequence of images by initially aligning the image-frames with respect to the centroid of the major-moving-object in the video so that the object in consideration remains immobile in time. Then, a few lines which are parallel to the motion-flow of the chosen centroid are extracted. These are referred to as the *reference curves*. The spectral power is then computed for the frames along these *reference curves*. The measure of periodicity for each of the reference curves is calculated as the difference(normalized) between the sum of (highest amplitude) spectral energy frequency and the sum of the spectral energy frequencies half way through.

The method discussed in [22] makes an assumption of a static camera and employs background subtraction to extract motion information. The objects in the foreground are tracked and their trajectory path is approximated to a line using the Hough Transform (as shown in [12]). The information from the temporal histories of each pixel is then further dissected using the Fourier analysis and the harmonic motion energy accumulated because of the periodic motion is computed. The given assumption in this case is ofcourse that the background is static. This is because a non-homogeneous background will undermine the harmonic energy that needs to be estimated. The work by [9] considers areas of the image (collection of pixels) instead of individual pixels as done by [22] which compose an object. It also uses a smooth image resemblance metric. Because of these changes in approach, the fourier analysis of [9] is much simpler as the analyzed signals don't have a significant number of harmonics of the rudimentary frequency. Techniques like [9] and [32] require an initial pre-processing step of detecting the foreground objects (like a person) in the video frames. *(Note: This step can be comparable to the background subtraction step in some approaches)*. Then the foreground objects are aligned to ensure spatial correlation with all frames in the sequence. (In the Fourier Domain Analysis techniques, this step was only required during the segmentation stage of the algorithm (lower complexity than the spatial-counterpart)). These frame-correlations are required for computing the similarity-plot per video. If we were to incorporate a spatial-only approach, the similarity plot is compared with a dense enough lattice (to make sure that the repetitions are not missed). In such cases, prior knowledge or heuristics are required to get reliable estimates of the periods.

2.3. Approaches that define Crystallographic groups

This is a more recent approach of tackling the problem of repetition estimation which involves defining fixed cases for perception of 3D recurrence in a video comprised of 2D images (not to be confused with 3 color channels in the 2D image frames). [23] was the first work that came up with the idea of defining crystallographic groups. The main contribution of this work was that in an N-dimensional euclidean space, a finite number of crystallographic-groups(symmetry-groups) can represent an infinite number of periodic repeating patterns. The authors of this work argue that in the 2D space, there are a total of seven groups which describe the patterns(monochrome) that show signs of repeating occurrence along a single direction and seventeen groups for patterns that can show signs of recurrence along two linearly independent directions. They came up with a set of approaches to classify a periodic pattern to one of the underlying symmetry-group, it's underlying lattice and also computing the motifs.

[28] then came up with an approach that defined 18 fundamental cases of perception of 3D recurrence in 2D videos. This work builds upon the wavelet transform to handle the scenarios of non-static and non-homogeneous video environment settings. They use the flow fields and their partial differentials to extract three kinds of fundamental types of motions and three kinds of motion-continuities of inherent periodicity in 3 dimensions. This leads to a total of 9 cases. Finally, based on the 2 viewpoints (front/side) there can be a total of $9 \times 2 = 18$ fundamental cases. This work tries to effectively tackle the issues caused by camera motion and a variety of repetitions in videos by constructing a rich set of flow-based signals that vary over time. This work handles complex camera motion scenes and complex appearances quite well but it requires several expensive feature engineering steps like background segmentation and computation of flow fields.

2.4. Approaches that use Deep Learning

Deep learning is the use of Neural Networks as proposed in [15]. Modern deep learning techniques have added a

big boost to the already rapidly evolving field of Computer Vision. Because of Deep Learning, a variety of new applications of the existing vision techniques has emerged and is rapidly becoming a part of our daily life. The state of the art method was proposed by [21]. This work processes the video online as the frames arrive. They came up with the idea of estimating cycle length and thereby looking the counting problem from the other end. They process and analyze 20 non-consecutive frames from the video at a time and estimate the cycle length over each such block using Convolutional Neural Networks. This information is then integrated over time by an *outer system* which controls when to start and stop the counting process. This approach relies on a synthetic pretraining step and an external system to manage the predictions. The results are impressive on the *YTSegments dataset* released with the paper but the dataset shows little variability in cycle length, camera motion, appearance of motion and clutter in the background. Also, this approach is clearly not end to end trainable.

Most of the methods proposed before have a handicap of relying on human-crafted rules to compute visual onsets and can only perform nicely on a small subset of the videos. This work by [33] aims to extract a variety of features including original-frame-pixels, frame residuals, body-pose, scene-change and optical flow and feed them to an end to end trainable network as input features. This network then re-aligns the misalignment between all the computed features using a proposed feature alignment layer. These aligned features are then further fed to a Bidirectional LSTM (introduced in [14]) and Conditional Random Field layers which label the sequence (as shown in [25]) with their respective onsets. There is high computation cost associated with each of the feature extraction steps.

3. Datasets and Preparation

This research focuses on the visual information present in videos. The visual information in the videos boils down to a time-series of image frames. For instance, A 60-fps video has 60 image frames in each second of video.

3.1. Datasets

The following datasets are used for the experiments conducted in this research:

3.1.1 YTSegments dataset

The YouTube Segments dataset collected by [21] is a collection of 100 videos mostly gathered from YouTube. This dataset contains a good mix of domains from which the videos are obtained like cooking, exercising, living animals etc. The dataset is prepared such that it is pre-segmented to only contain the repeated action. This dataset boasts of good variability in number of repetitions per video. The videos in this dataset don't display a lot of camera motion in most of the videos. The annotations for this dataset is just a total count per video. The temporal bounds per repetition in each video is not annotated. Preprocessing and manually annotating the temporal bounds of repetitions for these videos was required. Frames from a ball hopping video from this dataset can be found in Figure 1.

Figure 1. Random frames of a ball-hopping video from the YouTube Segments dataset

3.1.2 QUVA Repetition dataset

The Qualcomm and the University of Amsterdam Repetition dataset published with [28] is a collection of a wide variety of videos which exhibit periodicity like rope-skipping, cutting, swimming, stirring, music-making, combing etc. They collected the original untrimmed videos from YouTube. The repetition-counts per video and temporal bounds of each repetition in that video are present. The authors argue that this is a higher quality dataset as compared to the *YTSegments* dataset proposed by [21]. This boasts of more variability in cycle length, camera motion, appearance of motion and clutter in the background. This dramatically increases the difficulty in both temporal dynamics and the complexity of the scene. This is a much more realistic and difficult benchmark for repetition estimation in videos. This dataset has a collection of 100 videos and it as video wise annotations that contain the frame numbers which indicate the end of a repetition cycle from the video frames.

3.1.3 Aircraft inspection dataset

Another dataset that is used for the experiments is the **Aircraft inspection dataset from Aiir Innovations (aiir.nl)**. During large overhauls or whenever an irregularity has been spotted by sensors during a flight, each stage of the engine is manually inspected by the mechanics to look for any anomalies. While the mechanic inspects each stage of the engine, they have to inspect each turbine blade and depending on the engine type and the engine stage currently being inspected, the total number of blades differ. Currently, the mechanic has to manually keep track of the number of

blades that have been inspected to ensure all blades have been inspected. This dataset contains videos from such inspections. The dataset available has inspection videos from various angles and for different stages in the engine. This contains a collection of over 100 videos. As existing annotations, this dataset just contains the total number of repetitions per video. The annotations for temporal bounds of each repetition per video doesn't exist for this dataset yet. For video datasets like actions datasets, the repetitions are defined very well. For this dataset, it is non-trivial because one repetition depends on when you define to be the start of that repetition. Some utility scripts were built to make manually annotating the temporal bounds of the cycles in these videos much easier.

3.2. Video preprocessing and Labelling

A video is a time series of image frames with an audio signal embedded into it. For the task of repetition estimation, this work will not analyse the audio signal and only consider the visual aspects of it. So a video can be assumed to be a sequence of images. Each image in this sequence is a grid of pixels. These pixels have 3 channels (values) which correspond to the intensity of the primary colors *Red, Green and Blue*. These values can be in the range $[0, 255]$. **FPS** or frames per second determines how many image-frames are present in a second of video. A high fps indicates many frames per second and hence a smoother video. As the fps increases, the size of our dataset also increases. This would mean that more information would need to be processed by the model which would require longer training times and larger model size. To avoid the aforementioned problems, for the task at hand, **all the videos are resampled to 10fps** as it is sufficient to not miss any repetition.
All the pretrained CNN models expect the images to be at least $224 \times 224 \times 3$ (Height x Width x color channels), so all the image frames in each video are also resampled to $224 \times 224 \times 3$. This height and width is also more than enough to observe any kind of repetition in the available dataset of videos. This means a video in the prepared dataset has $(224 \times 224 \times 3 \times frames)$ dimensions.

3.3. Each time step labelling

This approach demands that each frame in the video is labelled. This work proposes a novel approach of labelling the videos. The idea is that *each frame in the video is labelled with repetitions that have occurred so far*. This would mean that the initial frames in the video would be all labelled as 0. There is no such thing as 1 repetition. And after there were two repetitions, the frames would be labelled as 2 and so on. The frame-wise labels for a video would look something like this [0,0,0,0,0,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5]. These annotations if visualized would look like steps. Figure 2 shows annotations of a video with 5 repetitions. This type of labelling would also help visualize the model prediction results in a beautiful way. The videos are labelled like this to try and backpropagate the loss back from each time step. Three separate datasets are created after labelling each with different train and test splits.

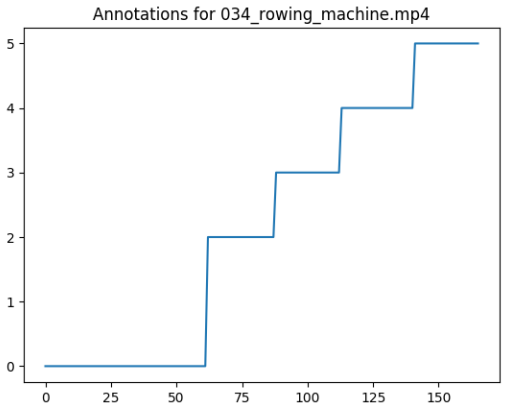

Figure 2. Each-time-step annotations of a video with 5 repetitions

3.4. Training and Testing splits

All of these datasets have a train, test and validation split of **70%, 20% and 10%** respectively. These splits are created at the beginning and ensure that the videos in each of these splits have completely different frames to avoid unreliable results. The videos from the training set will be used for actual gradient computation and weight updates in the model parameters. The videos from the validation split will be used for inspecting the loss curves and watching out for signs of over-fitting or bias. The videos from the validation split will never be used to update the weights of the model. The test split is used to compute the projected accuracy of the model. Since the model has never seen the test-set videos while training, model's accuracy on this set is a reliable estimate of the model's performance on a real world example.

4. Approach

4.1. The Matrix Profile signal

The matrix profile was proposed by [34]. Its computation relies on window size as a hyperparameter. A window size determines how many data points(from the time-series sequence) are compared at a time. For each window in the time series, it is compared to all the other windows. Like a sliding window, each window gets its turn and on its turn, its distance is computed with every window in the time series.

All these distance values are fill up the reference window's *column* in the distance matrix. The type of distance used is Z-Normalized Euclidean distance.
Given two windows (time-series-sequences) x and y, where: $x = x_1..x_n$ and $y = y_1..y_n$
Z-normalized sequence $\hat{x_i}$:

$$\hat{x_i} = \frac{x_i - \mu_x}{\sigma_x} \tag{1}$$

Z-normalized Euclidean distance $d(x, y)$:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (\hat{x_i} - \hat{y_i})^2} \tag{2}$$

.

Once the z-normalized euclidean distances of all the windows with all the other windows is computed, a completely filled in distance matrix is obtained.

1. Each column in this matrix is the distance profile for that window.

2. Computing the matrix profile from this is very straightforward: Just save the minimum value from each column and there you have a 1-dimensional matrix profile.

Each item in the computed matrix profile represents the minimum distance of the corresponding time-step item in the original signal when measured against all the other items in the signal. This 1-dimensional matrix-profile is quite intuitive to look at and the peaks in the signal depict anomalies and the valleys depict motifs. There has been a lot of research done in efficient computation of the Matrix profile ([36]) and maintaining it incrementally ([37]).

In the following experiment, some matrix profile signals are manually inspected to see if it is even possible to use it for the task at hand. The results are quite impressive for the blade counting dataset. The matrix profile looks sharp with clear valleys which indicates the presence of repeting patterns. The matrix profile for a video from the blade dataset (which has 5 repetitions) can be found in Figure 3. The blade inspection dataset has static camera and the background usually doesn't change much. This is one of the reasons that the matrix profile output displays clear peaks and valleys. The comparisons of using different window sizes(1, 3 and 7) for matrix profile calculation are shown in Figure 4. **As expected, the matrix profile gets smoother as the window size is increased.** The Matrix profile is also computed for the more generic QUVA and YTSegments datasets. Mixed results are obtained for these harder datasets and the peaks and valleys in the signal don't always show clear correlation with the number of repetitions. A possible solution to this could be to employ

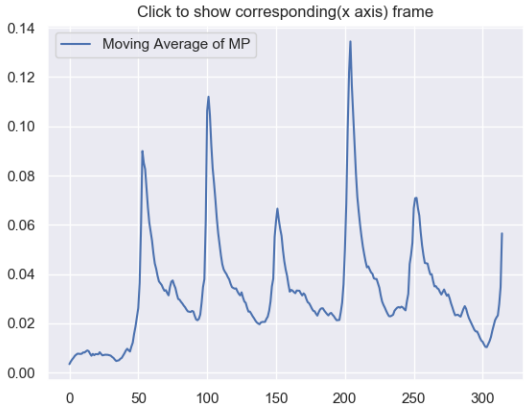

Figure 3. Matrix Profile for a blade inspection video with 5 repetitions

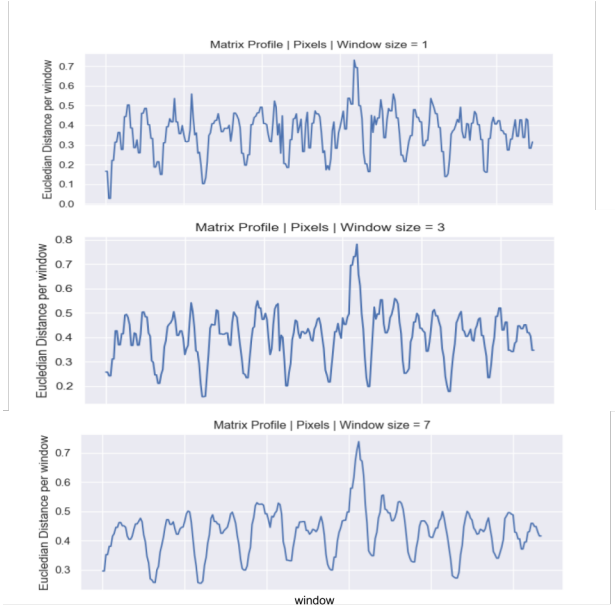

Figure 4. Change in Matrix Profile with change in window size

some background subtraction techniques to ensure that only the interest region is passed to the matrix profile from each frame (or window of frames). The matrix profile is also computed using ImageNet pretrained CNN features instead of raw pixel values and the resulting signal is decent. These results can be seen in the supplementary document.
A quick intuition about the matrix profile is that if we are just able to estimate the number of valleys in the Matrix Profile, we would be able to estimate the number of repeting patterns in the video.

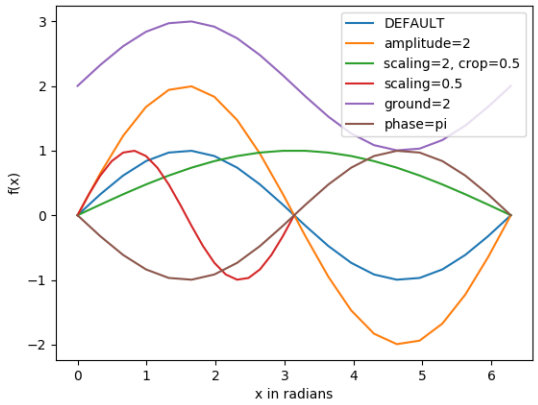

Figure 5. Wave cycle parameters

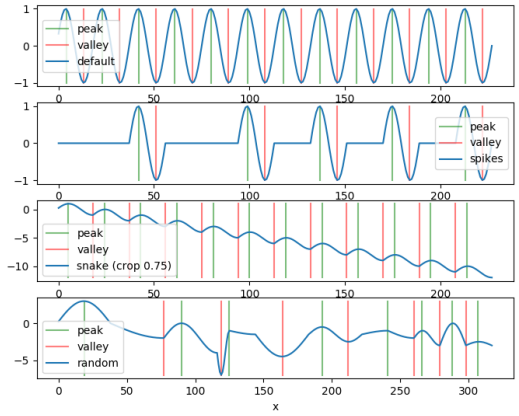

Figure 6. Labelled waves

4.2. Using a Recurrent Neural Network to find peaks and valleys in the Matrix Profile

The matrix profile is a noisy 1D signal. Theoretically, an LSTM should be able to do well at the task of estimating peaks and valleys in a 1 dimensional signal. As a toy task for the same, this work works with a self generated synthetic 1D signal. Since a sine wave is also a 1D signal with peaks and valleys, this work experiments with sine waves as a toy task for the LSTM.

4.2.1 ToyTask: Mimicking the Matrix Profile

For creating our synthetic dataset, we first began with a sinusoidal cycle. We will refer to it as a wave-cycle in the remainder of this report. To create a synthetic wave, a number of wave-cycles like shown in Figure 5 are stitched (like words in a sentence) with configurable parameters like Scaling, Amplitude, Phase, etc. A wave-cycle can have the following parameters:

1. G is the **Ground value**. This is simply where on the vertical axis, the wave-cycle should start. By default this would have a value of 0 as a sine wave exists at ground 0. This directly added to the value of $f(x)$.

2. S is the **Scaling factor**. Scaling factor decides if the wave is squeezed or expanded in the x axis. If the stretch factor has a value greater than 1, then the wave should have larger cycles and similarly if the stretch factor has a value less than 1, then the wave will have smaller cycles.

3. A is the **Amplitude factor**. This is the height of the wave-cycle. The maximum value of a sine wave is 1, hence the default amplitude factor is set to 1. Having an amplitude factor larger than 1 will result in taller waves and an amplitude factor less than 1 will result in shorter wave-cycles.

4. P is the **Phase value**. This is the x value where the wave-cycle begins. A sine wave can begin from $-\infty$ to ∞. By default it is set to 0, so the wave-cycle begins at x=0 and goes up. If the wave-cycle begins anywhere between $\pi/2$ and $3\pi/2$, it will go down initially.

5. C is the **Crop factor**. This determines what proportion of the wave-cycle to include. It can have values in the range $[0, 1]$. By default, the entire cycle should be included in the wave-cycle, so the default value for this is 1.

This wave cyle is created by using the following equations:

$$phase = 0, end = 2\pi$$
$$phase = phase * S$$
$$end = phase + 2\pi C * S$$
$$f(x) = G + (\sin(x/S) * A) \qquad \forall x \in [phase, end] \quad (3)$$

.

Figure 6 shows a few stitched waves which are labelled. The waves generated can be easily labelled for peaks and valleys (local maximas and minimas). The key point in labelling such a wave is: each wave-cycle can have at most one peak and one valley. Other peaks and valleys can exist at the points where the wave-cycles are stitched. A few rules can easily determine whether the stitching point will be a peak or a valley. Each point in the wave can be assigned one of the following classes:

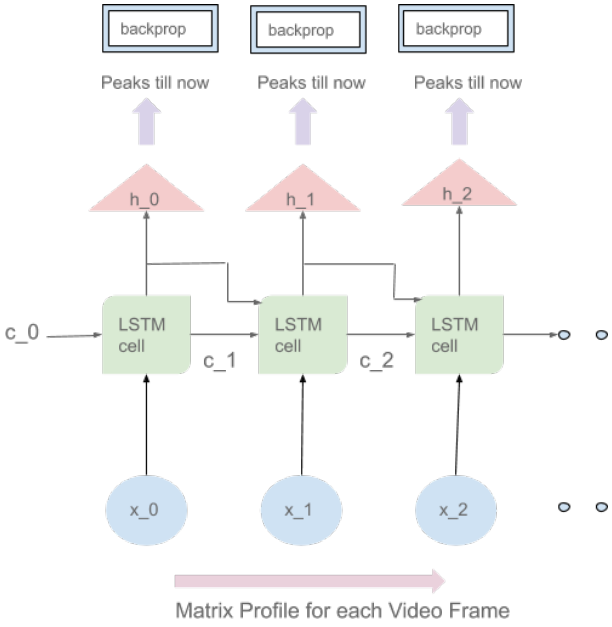

Figure 7. LSTM backpropagation architecture

1. *Category-1*: No peak, no valley
2. *Category-2*: Peak
3. *Category-3*: Valley

Most of the points in the waves are assigned to the *Category-1*. Figure 6 shows some labelled waves. The green vertical lines indicate presence of a peak (*Category-2* point) and the red vertical lines indicate presence of a valley (*Category-2* point) at that point. Some Gaussian noise with $\mu = 0$, $\sigma = 0.05$ is also added to this wave to mimic an actual Matrix Profile which is also noisy and coarse.

4.3. Training neural networks on the synthetic dataset

A few configuration of recurrent neural network architectures like the LSTM is trained on this dataset. It is observed that the minimum sized model with 2 hidden layers that performs at least as good as the models with more number of parameters is 20 hidden neurons in each hidden layer. As expected, the bidirectional LSTM does better even with fewer number of training parameters (using 10 hidden neurons in each layer). Using this trained model for predicting classes on a test set signal is shown in Figure 8. The peak and valley probabilities in the output show that the LSTM is pretty confident of its predictions. The LSTM does well on this task and displays clear understanding of peaks and valleys in a 1-dimensional signal. Some convolutional architectures like Temporal Convolutional Network (TCN) proposed by [1] are also experimented with (as shown in the supplementary document). The results for an LSTM are much better and the predictions are more confident.

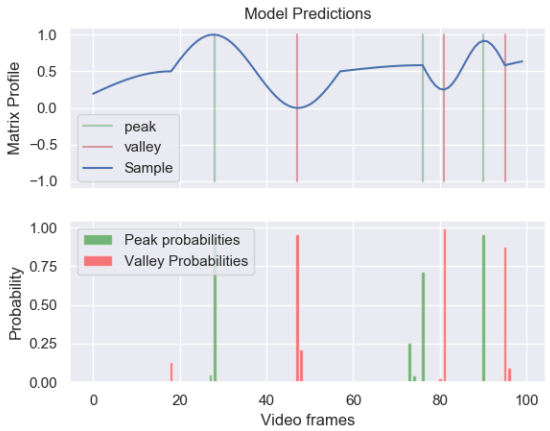

Figure 8. Trained LSTM prediction on a test set signal

These experiments show that an LSTM can be trained to estimate peaks and valleys in the Matrix Profile. These also provide an intuition about what size of the model should be sufficient for the task at hand. These experiments show that a relatively small LSTM architecture (with only 3000 trainable parameters) is sufficient for estimating peaks and valleys in a 1-dimensional signal. The bidirectional variant performs better than a unidirectional variant even with fewer learnable parameters.

Backpropagating loss from the last time step is also tried, but the results are not comparable to backpropagating from each time-step. The findings of backpropagating only from the last time step can be found in the supplementary document.

4.4. End-to-end trainable pipeline

The final end-to-end trainable pipeline is shown in Figure 9. A series of experiments are conducted that change one variable at a time to be able to definitively answer the research questions. All the experiments conducted so far work with only segments of this pipeline.

4.5. Changing one variable at a time

Theoretically, with an infinite amount of labelled videos for training and an unlimited amount of memory in the available GPUs, it should be possible to estimate repetitions in all kinds of videos by just feeding the raw pixel values into a big enough LSTM network. We have neither of those, so we need to learn features from the images to reduce the dimensions of the input to the LSTM network. And we do not even have an unlimited memory, so we want

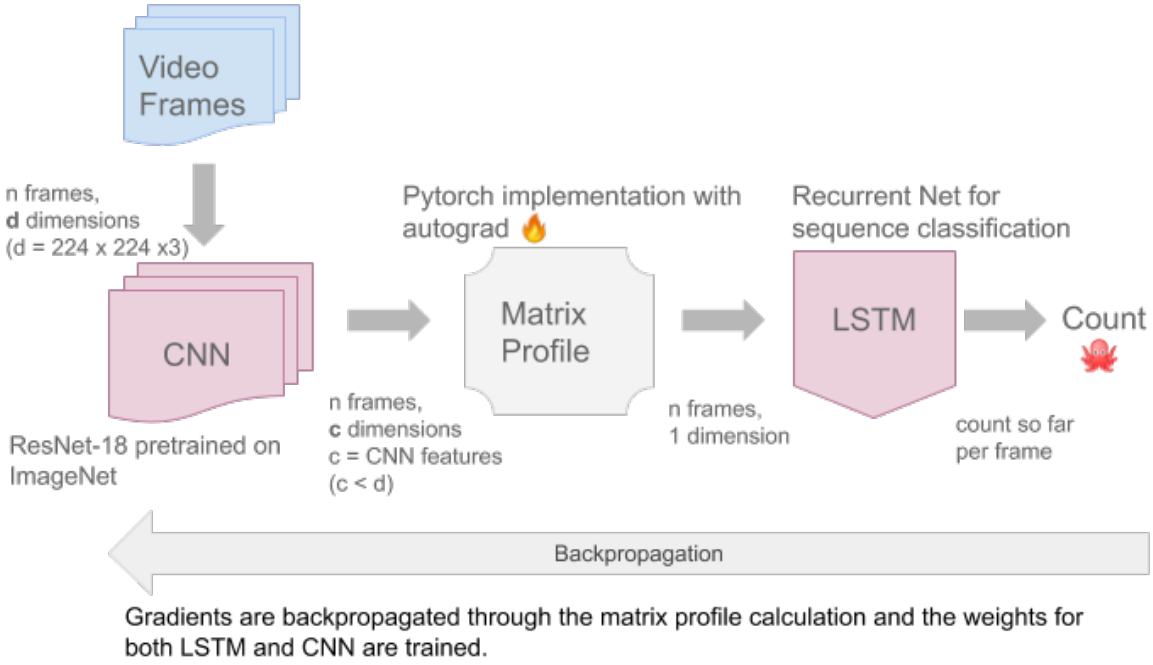

Figure 9. Architecture of model **t4**. The CNN features are used to compute the Matrix Profile and that is fed to the LSTM block. The CNN weights are also trained for this model.

our LSTM network to be as small as possible. Through our upcoming experiments, we will try to come up with light versions of the model that perform at least as good as the heavier models. The key idea is that the lighter the model is, the more feasible it is to use for a real-world problem.

Several instances of the model were deployed with slight modifications in each one. The slight changes ensure that one variable is changed at a time to understand the effect of the variations. It is a configurable plug and play pipeline and the following configurations are experimented with:

1. **CNN Type**: Based on this configuration, the Convolutional Neural Network block is decided. Several CNN architectures were evaluated like *DenseNet-121*, *SqueezeNet-1.1*. For brevity, *ResNet-18* is used in the experiments that follow. The input to the Convolutional Block (per time step) is an image of dimensions $(224 \times 224 \times 3)$. The output of this convolutional block is of dimensions 512 if it is a *ResNet-18*.

2. **Matrix Profile Flag**: This is a boolean flag which can have a value of True/False *on/off*. If this flag is turned *on*, only then the matrix profile block is used in the pipeline. For instance, if this flag is turned off, the CNN features are directly fed to the next block which is the LSTM block. In this case, the input for the LSTM block would be of dimensions 512 (if CNN used is ResNet-18).

3. **Matrix Profile window-size**: This field is only used if the *Matrix Profile flag* is set to *on*. This field determines the window-size which is a hyperparameter used in calculation of the matrix profile. (See the Matrix Profile section above for more details).

4. **LSTM Hidden Layers**: This field determines the number of hidden layers employed in the LSTM block in our pipeline. By default, we have set this parameter to 2 for our experiments. This configuration can help us experiment with the complexity of our model later on if needed.

5. **LSTM Hidden Neurons(per layer)**: This field determines the number of neurons in each hidden layer of the LSTM block. Because of our initial wave-cycle experiments, we could make informed guesses about this parameter. Based on the choices of the other configuration fields, this field would be modified (as shown in this section later).

6. **CNN Backpropagation**: This is a boolean flag which can have a value of textiton/off. If this flag is turned *on*, only then will the weights of the CNN be tuned. For initialization of the weights of the CNN (either of the three variants), the weights pretrained on ImageNet [10] are used. Our hypothesis regarding this parameter

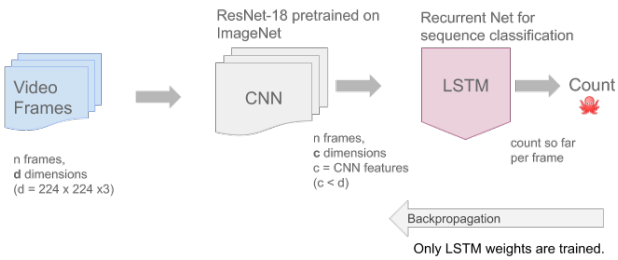

Figure 10. Architecture of models **t0** and **t1**. The CNN features are directly fed to the LSTM block.

was that if turned *on*, the CNN would adjust itself to make it easier for the matrix profile to detect motifs.

The configuration options that are kept constant for all the upcoming experiments are **CNN Type** for which a ResNet-18 is used, **LSTM Hidden Layers** for which 2 is used and **Matrix Profile window size** for which 1 is always used (if the *Matrix Profile Flag* is on). Based on the configuration options discussed above, there can be any number of combinations to converge to a final model. For brevity, the following combinations are experimented with:

Model: t0 is shown in Figure 10. This is a relatively lightweight pipeline without the Matrix Profile. The idea of training this pipeline is that feeding the CNN frames directly to a light LSTM. Our hypothesis regarding this variant is that it would begin to learn some patterns but the performance of this pipeline would not be as good as the heavier LSTM variant.

- *Matrix Profile Flag*: *off*
- *LSTM Hidden Neurons (per layer)*: 100
- *CNN Backpropagation*: *off*

Model: t1 is shown in Figure 10. This is a heavier Pipeline without the Matrix Profile. The idea of this pipeline is that we have enough number of hidden neurons to actually fit the task at hand. Our hypothesis is that this variant would perform better than the variant with a lighter LSTM.

- *Matrix Profile Flag*: *off*
- *LSTM Hidden Neurons (per layer)*: 2000
- *CNN Backpropagation*: *off*

Model: t2 is shown in Figure 11. This is a heavier pipeline with the Matrix Profile.

- *Matrix Profile Flag*: *on*
- *LSTM Hidden Neurons (per layer)*: 2000

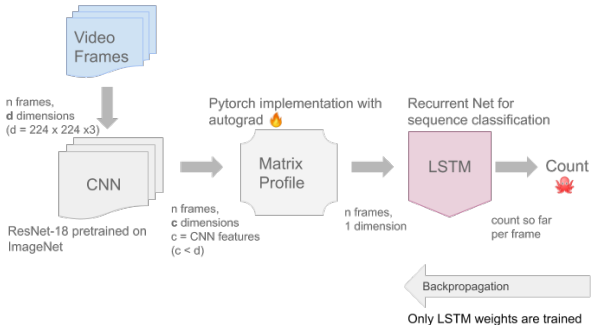

Figure 11. Architecture of models **t2** and **t3**. The CNN features are used to compute the Matrix Profile and that is fed to the LSTM block.

- *CNN Backpropagation*: *off*

Model: t3 is shown in Figure 11. This is a relatively lightweight pipeline with the Matrix Profile. The idea of using matrix profile in this pipeline is that if it is possible to use the matrix profile to greatly reduce the size of the model required for the task of repetition estimation.

- *Matrix Profile Flag*: *on*
- *LSTM Hidden Neurons (per layer)*: 100
- *CNN Backpropagation*: *off*

Model: t4 is shown in Figure 9. This model is a relatively lightweight pipeline with the Matrix Profile but its CNN backpropagation is turned *on* leading to a huge computation graph. The goal of this pipeline is to train the CNN in such a way that the computed matrix profile using this gets better (it becomes easier to detect repetitions from the computed matrix profile signal). This model would answer the question if backpropagating through the matrix profile actually helps.

- *Matrix Profile Flag*: *on*
- *LSTM Hidden Neurons (per layer)*: 100
- *CNN Backpropagation*: *on*

These models were trained on all the datasets for 500 epochs or a max-run-time of 3 days (whichever happens before) on a pascal GPU machine. The training losses, validation losses and epoch accuracies can be found in the Appendix of the supplementary document.

5. Results and discussions

The model-configurations discussed at the end of the last section are all trained separately on all the three datasets. (Note: These results are obtained using *each time-step labelling* as discussed before). Each frame denotes a class to

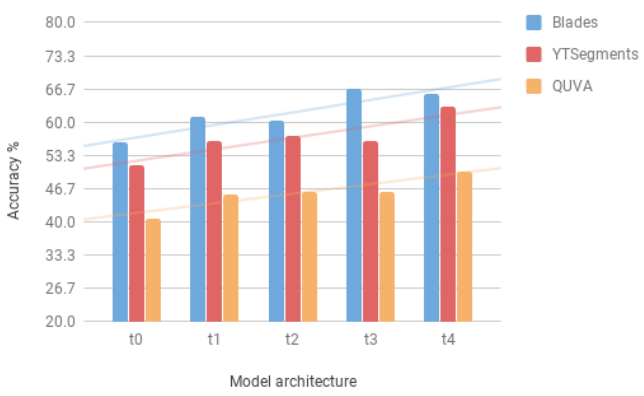

Figure 12. Model architecture accuracies for Aircraft Engine Blades, YTSegments and QUVA datasets.

which it belongs (repetitions so far). Now, to make sure that there is no bias, it is ensured that each video in the dataset has exactly 5 repetitions. The videos which have more than 5 repetitions are trimmed so that they meet this constraint.

Measuring accuracy: For each time-step labelled data, a label for each frame in the video is present which denotes the number of repetitions so far. There are several ways to measure accuracy. Existing literature (like [28] and [21]) uses *Mean Absolute Error* to compute errors in their final estimations. Mean Absolute Error is the difference in actual and predicted number of repetitions. These works also do not use a separate test set for evaluation. This would mean only considering the output of the last frame for our architecture since an output is produced for each frame by this work's proposed model. This is not a good indicator of the accuracy of the proposed model. A technique that considers the predictions per frame for all the videos for evaluating such an online approach is to look at all the predicted labels of all the frames (from the videos in the test set) and compare them to the actual labels to compute an overall accuracy. This approach however, has an ingrained flaw. If a video has a lot of frames (as compared to other videos in the test set), the accuracy of that video would skew the overall accuracy of the model which should not be the case. Since all the videos have exactly 5 repetitions, a video could have more frames if the action is repeated slowly in the video (temporal cycle length is large). To overcome this flaw, this work proposes the approach of computing accuracy per video (from the test set) by comparing individual frame predictions with the labels and then calculating an overall accuracy of the test set by taking an average of all the accuracies. This approach gives an equal weight to each video in the test set.

The accuracies on the test sets of Aircraft Engine Blades, YTSegments and QUVA datasets for all of the model architectures are plotted in Figure 12. The best accuracies obtained for each of the datasets are 66%, 63% and 50% respectively. The trend in the plot shows that the models making use of the Matrix profile (*t3 and t4 architectures*) obtain the best results in terms of accurately labelled frames. The experiments comparing *t0 and t1 architectures* show a difference of about 5% for all datasets. This shows that estimating repetitions using a recurrent network fed with CNN features requires a heavy RNN to better learn the task at hand. The *t3 architectures* for all datasets consistently gave accuracies equal to or better than the heavier *t1 and t2 architectures*. This helps us deduce that computing the matrix profile is actually useful for estimating repetitions and helps us reduce the size of the model without compromising on accuracy. The *t4 architecture* showed good promise by obtaining the best accuracies (with a margin of about 5%) for both *quva and ytsegments datasets*. By looking at the results of the *t4 architecture* on the datasets, we can safely conclude that it is possible to backpropagate through the step of computing the matrix profile and update the CNN weights such that it helps the upstream LSTM can more effectively use the computed Matrix profile signal. This could potentially mean that backpropagating through the matrix profile could be used for several other tasks where matrix profile already shines to further improve the results in that specific domain. This also means that other problem specific mathematical operations could be backpropagated through to further improve results.

The best performing architecture (*t4 architecture*) is also compared with the pretrained model published by [21] on the prepared dataset for **Mean Absolute Error**. The prepared datasets contain all the videos with exactly 5 repetitions. The model published by [21] predicts a count per video which can be compared with 5 to get the error in prediction (if any) and the error is averaged over all the videos in the dataset. The count per video is obtained from the proposed *t4 architecture* by only considering the model's prediction on the last time step. The Mean absolute errors for both these models can be found in the Table 1. It clearly shows that on the prepared dataset, the proposed architecture easily outperforms the model published by [21] (which possibly requires dataset specific configurations judging by higher errors on datasets with longer repetition cycle lengths).

6. Conclusions

This work suggests a new way to look at the problem of repetition estimation. It proposes a novel approach of labelling the dataset by labelling each frame of the video with a class (repetitions so far). This results in an inherently online approach to tackle the problem. It is

	Blades	YTSegments	QUVA
Levy & Wolf [21]	3.454	2.87	3.076
t4 architecture	**0.409**	**1.383**	**0.641**

Table 1. Mean Absolute Error comparison of the *t4 architecture* with [21] on the prepared datasets with 10 fps and 5 repetitions per video.

observed that this kind of labelling aids in faster and better learning in recurrent architectures as compared to assigning a label to the entire video (or to just the last time step). After labelling the dataset like this, the Matrix Profile is experimented with and it is observed that it is useful for the task of repetition estimation. Furthermore, a recurrent neural network architecture like an LSTM can be trained using video frames or using pretrained CNN (like ResNet) features from video frames. An LSTM like this would need to be quite heavy (with a lot of trainable parameters). The size of this network can be dramatically reduced by using the Matrix Profile and feeding the Matrix profile (computed from the CNN features) to the upstream LSTM instead of feeding the CNN features to LSTM directly. The weights of this pretrained CNN (ResNet-18) are also trained by backpropagating through the Matrix Profile computation step and it is observed that the accuracy of the model increases for both all the three (Blade inspection, QUVA and YTSegments) datasets. The fact that the matrix profile computation is backpropagatable can be quite beneficial to several domains that employ the Matrix Profile or other domains which employ some problem specific mathematical operations like the Matrix Profile.

7. Recommendations

The problem of estimating repetitions in videos is harder than it seems at first glance. There could be several ways to improve the performance and learning of the proposed architecture. Currently, the datasets were labelled with each-time-step labelling manually by a single person. This could be rectified by having several people label the videos and aggregating their results to have consistent repetition cycle estimates. For the matrix profile calculation, a window size of 1 is used in the end-to-end pipeline experiments. Different values for this hyperparameter can be cross-validated to see what works best for the dataset at hand. Another improvement could include cross-validating with several loss-types like the Margin Hinge Loss etc. instead of the employed Negative Log-Likelihood Loss. and with different activation functions like *tanh* or *sigmoid*. Different types of loss and activation functions can perform better for different kinds of problems. A few popular loss types and activation functions are discussed in detail in the Appendix of the supplementary document. More boost in performance could be obtained by increasing the size of the dataset by using video augmentation techniques like rotation, shear etc. The computation graphs of several matrix profile implementations could be inspected to chose the smallest one or the most parallelizable one to enable faster convergence. Smaller computation graphs as proposed by [31] can be experimented with. The matrix profile output is not necessarily aligned perfectly with our idea of repetition-cycle end/begin. The Feature Aligning Network recently proposed by [33] also seems promising to align the matrix profile properly with the labels. A completely different way to tackle this problem could be by using the Triplet Loss proposed by [29]. To use the Triplet Loss, videos with the same number of repetitions could be grouped together as *positive* examples for the *anchor* and any other video could be *negative* examples.

References

[1] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[2] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International journal of computer vision*, 12(1):43–77, 1994.

[3] S. Belongie and J. Wills. Structure from periodic motion. In *International Workshop on Spatial Coherence for Visual Motion Analysis*, pages 16–24. Springer, 2004.

[4] M. Brand and V. Kettnaker. Discovery and segmentation of activities in video. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (8):844–851, 2000.

[5] A. Briassouli and N. Ahuja. Fusion of frequency and spatial domain information for motion analysis. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 175–178. IEEE, 2004.

[6] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *European conference on computer vision*, pages 25–36. Springer, 2004.

[7] T. M. Chin, W. C. Karl, and A. S. Willsky. Probabilistic and sequential computation of optical flow using temporal coherence. *IEEE Transactions on Image Processing*, 3(6):773–788, 1994.

[8] D. Cremers, S. J. Osher, and S. Soatto. Kernel density estimation and intrinsic alignment for knowledge-driven segmentation: Teaching level sets to walk. In *Joint Pattern Recognition Symposium*, pages 36–44. Springer, 2004.

[9] R. Cutler and L. S. Davis. Robust real-time periodic motion detection, analysis, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):781–796, 2000.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[11] I. Djurovic and S. Stankovic. Estimation of time-varying velocities of moving objects by time-frequency representations. *IEEE Transactions on Image Processing*, 12(5):550–562, 2003.

[12] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. Technical report, Sri International Menlo Park Ca Artificial Intelligence Center, 1971.

[13] R. Goldenberg, R. Kimmel, E. Rivlin, and M. Rudzsky. Behavior classification by eigendecomposition of periodic motions. *Pattern Recognition*, 38(7):1033–1043, 2005.

[14] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

[15] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, jul 2006.

[16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] W. S. Hoge, D. Mitsouras, F. J. Rybicki, R. V. Mulkern, and C.-F. Westin. Registration of multidimensional image data via subpixel resolution phase correlation. In *Proceedings 2003 International Conference on Image Processing (Cat. No. 03CH37429)*, volume 2, pages II–707. IEEE, 2003.

[18] S. Huang, X. Ying, J. Rong, Z. Shang, and H. Zha. Camera calibration from periodic motion of a pedestrian. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3025–3033, 2016.

[19] G. Johansson. Visual perception of biological motion and a model for its analysis. *Perception & psychophysics*, 14(2):201–211, 1973.

[20] I. Laptev, S. J. Belongie, P. Perez, and J. Wills. Periodic motion detection and segmentation via approximate sequence alignment. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 816–823. IEEE, 2005.

[21] O. Levy and L. Wolf. Live repetition counting. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3020–3028, 2015.

[22] F. Liu and R. W. Picard. Finding periodicity in space and time. In *ICCV*, pages 376–383. Citeseer, 1998.

[23] Y. Liu, R. T. Collins, and Y. Tsin. A computational model for periodic pattern perception based on frieze and wallpaper groups. *IEEE transactions on pattern analysis and machine intelligence*, 26(3):354–371, 2004.

[24] C. Lu and N. J. Ferrier. Repetitive motion analysis: Segmentation and event classification. 2004.

[25] X. Ma and E. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

[26] M. Piccardi. Background subtraction techniques: a review. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, volume 4, pages 3099–3104. IEEE, 2004.

[27] R. Polana and R. C. Nelson. Detection and recognition of periodic, nonrigid motion. *International Journal of Computer Vision*, 23(3):261–282, 1997.

[28] T. F. Runia, C. G. Snoek, and A. W. Smeulders. Real-world repetition estimation by div, grad and curl. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9009–9017, 2018.

[29] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[30] S. Stankovi and I. Djurovi. Motion parameter estimation by using time-frequency representations. *Electronics Letters*, 37(24):1446–1448, 2001.

[31] A. Veit and S. Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018.

[32] L. Wang, T. Tan, W. Hu, and H. Ning. Automatic gait recognition based on statistical shape analysis. *IEEE transactions on image processing*, 12(9):1120–1131, 2003.

[33] Y. Xie, H. Wang, Y. Hao, and Z. Xu. Visual rhythm prediction with feature-aligning network. *arXiv preprint arXiv:1901.10163*, 2019.

[34] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh. Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1317–1322. IEEE, 2016.

[35] R. W. Young and N. G. Kingsbury. Frequency-domain motion estimation using a complex lapped transform. *IEEE Transactions on image processing*, 2(1):2–17, 1993.

[36] Y. Zhu, M. Imamura, D. Nikovski, and E. Keogh. Introducing time series chains: a new primitive for time series data mining. *Knowledge and Information Systems*, 60(2):1135–1161, 2019.

[37] Y. Zhu, C.-C. M. Yeh, Z. Zimmerman, K. Kamgar, and E. Keogh. Matrix profile xi: Scrimp++: time series motif discovery at interactive speeds. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 837–846. IEEE, 2018.