Exploring normalizing flows for anomaly detection

Chinmay Pathak

Exploring normalizing flow for anomaly detection

by

C. A. Pathak

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday August 29 2019 at 12:00 PM.

Student number:	4740327	
Project duration:	November 18, 2018 – August 29, 2019	
Thesis committee:	Prof. dr. M. J. T. Reinders,	TU Delft, chair
	Dr. J. C. van Gemert,	TU Delft
	Dr. ir. A. J. van Genderen,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Acknowledgements

First of all, I would like to thank Dr. Jan van Gemert for providing me the opportunity to work on this thesis. I am grateful for his continuous support and guidance throughout this thesis. His inputs were invaluable for my work. I truly enjoyed working with him.

I would like to thank Miriam Huijser from Aiir innovations for providing all the necessary data and guidance. Your inputs through out the work and feedback for the report helped in a big way.

I would like to thank my parents and my sister for always believing in me and supporting me. I would like to also thank my friends Maneesh, Palash, Yashavi, Rajesh, Devendra and Sneha for making my stay off the work fun and relaxing which helped me to focus on my work better. Special thanks to Shreyas Nikte for providing a place to work from for the whole duration of this work.

Last but not the least, I would like to thank all my friends at crows nest for providing me company during lunch and coffee breaks. Time flew by much faster because of you guys.

C. A. Pathak
Delft, August 2019

Contents

1

Scientific Paper

Exploring normalizing flow for anomaly detection

Abstract

Anomaly detection is a task of interest in many domains. Typical way of tackling this problem is using an unsupervised way. Recently, deep neural network based density estimators such as Normalizing flows have seen a huge interest. The ability of these models to do the exact latent-variable inference and exact log-likelihood calculation with invertible architecture makes them interesting for the task of anomaly detection. In this work we explore the such normalizing flow-based model approach for anomaly detection in the novel BoroscopeV1 dataset which contains videos of the actual industry boroscope video material and has large noise. We verify the correctness of the models on a toy dataset. We found that the black pixels and high frequency in the image affect the model likelihood adversely. The experimental evidence shows that the normalizing flow-based approach can be used for the task of anomaly detection.

1. Introduction

Anomaly detection, also known as outlier detection or novelty detection is a problem of detecting data significantly different from the normal data. Given a collection of data it is always desirable to automatically determine which instances of it are unusual. This is a fundamental machine learning task that has applications in fields ranging from medicine, astronomy, fault detection, intrusion detection [10, 16, 17, 45] Traditional algorithms used for anomaly detection face difficulties when applied on high-dimensional data such as images. Along with that these algorithms require manual feature engineering [12].

Deep neural networks have shown a great ability to solve the complex visual tasks. It automates the feature learning process and thus has become de facto approach for many computer vision task. Supervised models have proven themselves for object recognition models, but they require labels of the data samples. In anomaly detection this is the biggest challenge since most of the times the anomalies are unknown and uncommon [11]. Thus, making the construction of sizable dataset is a difficult task. In this work, we make use of novel boroscope dataset. It shares little similarities with other publicly available dataset. The dataset specifics are explained later in the article in section 5.

Anomaly detection is typically solved by the unsupervised methods by first training the model with the normal data and at test time estimating the deviance of each test sample from the trained model. Reconstruction based approach using the autoencoder is the most common way tackling this problem in the literature. [8] This works explores the anomaly detection problem using normalizing flow-based model. With the ability to do exact latent-variable inference and the exact log-likelihood calculation the flow-based models have seen some interest from the research community in recent past [14, 15, 24, 25, 44, 49].

The advantage of flow-based model for anomaly detection over autoencoder based approaches is that, minimizing the reconstruction error does not guarantee that the anomaly score of normal data to be less than that of anomaly data samples, as the anomaly score of anomalous data is not considered while calculating the reconstruction error [52]. However, minimizing the negative log-likelihood for the normalizing-flows based model for the normal data samples implies the maximizing the negative log-likelihood of the other data samples, which also includes the anomalous samples. Therefore, this work demonstrate the use of flow-based model for the task of anomaly detection using the simple objective function of negative log-likelihood.

The major contributions of this thesis work are: 1)

We show that the black pixels adversely affect the likelihood values of the flow-based model. 2) We show that the convolutional autoencoder is unable to detect small anomalies in the images and thus establish the need for a approach different that reconstruction based approach for anomaly detection. 3) We show that frequency of the image does play a role in the case of the normalizing flow-based model and convolutional autoencoder

2. Related Work

Anomaly detection (AD) has long been a topic of interest in various fields [9]. The importance of understanding the normal behaviour of a system has always been a point of interest. Traditional machine learning techniques such as Support Vector Machines (SVM) [50], k-nearest neighbors (KNN) have shown great success at outlier detection tasks [7, 40]. However, as the size of the dataset started getting larger and more complex, these traditional methods are no longer the state of the art when it comes to the task of anomaly detection as they do not scale well [8]. Especially when it comes to high dimensional complex data such as images. With the rise of deep learning in the last decade, many computer vision problems have seen a huge boost in performance and accuracy [5]. Tasks such as object detection, recognition, and segmentation are solved with the help of convolutional neural networks (CNNs) with state of the art results compared to traditional machine learning because of the automatic feature extraction capacity of the neural networks [8, 38].

Traditional methods for anomaly detection

Traditional machine learning approaches for anomaly detection include methods such as one-class classification using SVM [41] where radial basis function (RBF) kernel is used to learn a region that contains the data instances. At test time, anomalous samples are identified only if they fall outside the learned region. A variation of this method, Support Vector Data Description (SVDD) [48] defines the smallest hypersphere in the latent space describing the training samples and the anomalies are identified if they fall outside the defined hypersphere.

KNN is an unsupervised approach which is traditionally used for anomaly detection [4, 6, 18, 54]. In this approach, an anomaly score is determined using the sum of the distances from the k nearest neighbors from the test instance. In other work, V.Skvara, et al. [47] shows the applicability of the KNN for anomaly detection tasks in comparison to deep generative models such as Generative Adversarial Networks (GANs) [21], Variational Autoencoders (VAEs) [26] and Autoencoders (AEs) [23]. However, this comparison was not done on image datasets, while in our problem setting we use image data only, here deep learning methods have a clear upper hand [8]. Along with that, these traditional methods also require explicit feature engineering [39], which is not required when working with deep neural networks and for an approach involving SVM, the support vectors need to be stored for the class prediction in the classification tasks which introduces memory constraints [43]. All this makes the deep neural nets a better choice with larger and more complex data set.

Deep methods for anomaly detection

There are various approaches based on deep learning for anomaly detection. Based on the nature of the input data and the availability of labels these are classified as supervised, semi-supervised, or unsupervised.

In supervised deep anomaly detection, the anomaly detection problem is predominantly posed as a binary classification problem where all the anomaly samples are combined into one class and normal samples as another [11]. In a different setting, "none of the above category" can also be appended to the classification model. Methods such as defectnet [3] anomaly detection via image resynthesis [29] and, detection of manufacturing defect using CNN and Transfer Learning [19] are some of the examples which use supervised approach to detect anomalies. In [19] authors make use of transfer learning with ImageNet [13] and COCO [28] data set, and 4 CNN modules to overcome the problem of fewer samples to get the mean average precision of 0.957 on gdxray [33] dataset of only 2800 samples at a cost of high memory and longer time to train. The biggest drawback of this approach is that the model requires the distribution of anomalies to be known before training. This poses a big problem as one of the main challenges while working on anomaly detection is that the anomalies are rare and not known prior to the dataset [8].

The semi-supervised AD methods assume all that the training samples have only normal class labels and work on the assumption that points which are close to each other both in input and latent space share the same label. This method can be implemented using any of the models such as autoencoders, generative adversarial networks, or CNNs [8]. GAN-based methods such as ganomaly [1] and skip-ganomaly [2] produce good results with a semi-supervised approach. However Lu et.al [30], puts forward a fundamental limitation that, unless the said assumption of the relation between labelled and unlabeled data distribution holds, semi-supervised methods cannot provide any significant benefits over supervised learning. This applies to the deep neural networks as well [8].

The main challenge while working with anomaly detection is the lack of samples from the anomalous class, because of which supervised and semi-supervised methods struggle. Unsupervised methods thus are the most widely applicable approach for AD as they detect anomalies based solely on the intrinsic properties of the data [8]. There are different unsupervised methods which are used in AD such as Deep autoencoder, GANs, VAEs, auto-regressive models [36, 37, 44, 49] and normalizing flow-based models [14, 15, 24, 25].

One use case of unsupervised methods is density estimation. The generative model $p_\theta(x)$ trained on some data distribution $p(x)$ ideally should assign a high likelihood to the samples from same the distribution as the model learns the joint probability distribution of the given data. Thus, when any out-of-distribution (OOD) sample Y from $q(y)$ is fed to a density estimation model it assigns a low likelihood to it. This property of generative models is very useful for a problem setting such as anomaly detection.

Deep Autoencoders [23] are the most common and fundamental unsupervised deep models that are used for AD consisting of an encoder-decoder architecture [56, 55, 32, 22]. This model works as a dimensionality reduction method which learns the common variations from the training data. VAEs on the other hand, optimize the variational lower bound on the likelihood of the data. The samples are encoded in such a way that the data may be generated from a simple gaussian prior using the similar encoder-decoder architecture. The difference between the input and the output generated by the decoder is called the reconstruction error. For normal samples, this error ideally is zero as the model has learned the representation of normal class and can thus reconstruct it, while for anomalous samples it should be a higher value as the model does not know the distribution of anomaly samples and as a result has difficulties in reconstructing it. Based on this, when a model is not able to reconstruct the input properly for a test sample, then that sample is identified as an anomaly.

The same reconstruction-based approach is also used with GANs in methods such as AnoGAN [46] and efficient anomaly detection with GANs [53]. GANs have shown greater ability to generate images with better fidelity to the train distribution hence there is a lot of interest in developing reconstruction based approach using GANs. In AnoGAN [46] first, a GAN is trained to create samples similar to training (normal) instances. During test time, best possible generated image matching the test sample is found out iteratively. For a normal test instance, this should result in lower anomaly score which is a simple L2 distance between the test input sample and reconstructed sample, while for an anomalous sample anomaly score will be high. This has the disadvantage that, for every test sample, the best matching image needs to be generated which makes it slow.

The main difficulty regarding the reconstruction-based encoder-decoder models is the degree of compression which works as another hyperparameter that needs to be manually tuned to get the best results because of the unsupervised nature of the problem [43]. Along with that, the approximation in the inference in VAEs limits its ability to learn high dimensional deep representations. In GANs the sidestepping from the likelihood objective in the training altogether, makes it difficult to train and also there is no encoder in GANs to that maps input to the latent variables directly. None of these models provide a way to tractably calculate exact log-likelihood of a data point.

There is a recent development on the invertible generative models [15, 24, 25, 44, 49] which enables the calculation of the exact log-likelihood of the input distribution using the change of variables formula. These models can be classified into autoregressive flows and normalizing flows. Autoregressive flows such as PixelCNN, PixelCNN++ [44, 49] have the advantage of

simplicity but synthesis with these models has limited ability to parallelize and thus they are not efficient for the high dimensional data such as images. On the other hand, Normalizing flow-based models such as RealNVP [15], GLOW [25] provide the best of both worlds: it is possible to perform exact latent-variable inference and log-likelihood evaluation of the given data distribution and it is efficient to parallelize for both inference and synthesis.

There are a couple of papers [11, 34] which shows that normalizing flow-based methods failed to detect OOD samples when the anomalous samples had the distribution of the data with similar mean and less variance than the training data distribution. The training and test samples in that work have distinctly different images for e.g. training set images from CIFAR-10 [27] had images of dogs, trucks and horse compared to the house numbers from SVHN [35] in the test set. In our case of BoroscopeV1 the train and test classes contains the images of the similar blades. In case of blades with the defects there is a small part of the image which is different from the normal blade image. Thus, we can assume that, in this work the distribution of the normal and anomaly samples is very similar and it is unlikely that this problem will be faced in this work. Therefore, this work explores the problem of anomaly detection using simple objective function of negative log-likelihood using a normalizing flow-based model called GLOW.

3. Normalizing-flow based model for anomaly detection

In this section we explain the working of a normalizing flow-based model and how it is used for anomaly detection in this work.

Normalizing flow is composed by a series of invertible transformation functions $\boldsymbol{f}$ which transforms the target $\boldsymbol{y}$ to a latent code $\boldsymbol{z}$ sampled from a simple distribution such as Gaussian or logistic distribution. The Figure 1 shows a general block diagram of a normalizing flow-based model. Input $\boldsymbol{X}$ is passed through a series of invertible transformation to get to a latent space $\boldsymbol{z}$. While sampling $\boldsymbol{z}$ is passed through the inverse transformation that produces output $\boldsymbol{X'}$.

Let $\boldsymbol{x}$ be a high-dimensional random vector with some distribution $\boldsymbol{p(x)}$. The log-likelihood objective

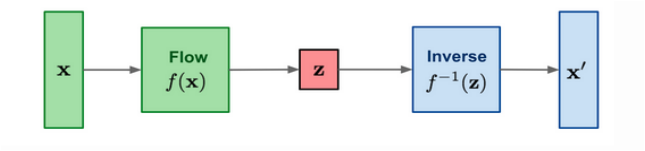


Figure 1: Flow based generative model [51].

to minimize for a flow-based model is given as:

$$\boldsymbol{L} = \frac{\mathbf{1}}{\boldsymbol{N}} \sum_{i=1}^{n} - \log p_{\theta}(x^{(i)}) \tag{1}$$

$\boldsymbol{Z}$ is a latent variable from a simple distribution, commonly a multivariate Gaussian given as $\boldsymbol{p(z)} = \boldsymbol{\mathcal{N}(z; 0, I)}$. The $\boldsymbol{x}$ is given as $\boldsymbol{x} = \boldsymbol{g(z)}$. The function $\boldsymbol{g(z)}$ is an bijective. Thus, these transformations from input space to latent space and back can be given as:

$$\boldsymbol{z} \sim \boldsymbol{p(z)} \tag{2}$$

$$\boldsymbol{x} = \boldsymbol{g(z)} \tag{3}$$

$$\boldsymbol{z} = \boldsymbol{f(x)} = \boldsymbol{g^{-1}(x)} \tag{4}$$

where the $\boldsymbol{f(x)}$ and $\boldsymbol{g^{-1}(x)}$ are the bijective functions. These invertible functions are chained together to obtain a transformation of input data sample $\boldsymbol{x}$ to latent space $\boldsymbol{z}$ as:

$$\boldsymbol{x} \iff \boldsymbol{h_1} \iff \boldsymbol{h_2} \iff \boldsymbol{h_3 ...} \iff \boldsymbol{z} \tag{5}$$

with the help of the change of variables theorem , we can define this distribution $\boldsymbol{p(x)}$ entirely in terms of an auxiliary, simpler distribution like $\boldsymbol{p(z)}$ which is a multivariate Gaussian. Thus the probability density of the model given a data-point is given as:

$$\log p(x) = \log p(z) + \log |det(\frac{dz}{dx})| \tag{6}$$

$$\log p(x) = \log p(z) + \sum_{i=1}^{K} \log |det(\frac{dh_i}{dh_{i-1}})| \tag{7}$$

The normalizing-flows based model used in this work is GLOW [25]. It is an extension of the previous two flow-based models NICE [14] and RealNVP [15] respectively. With GLOW authors have reported a significant improvement in likelihood and sample quality of natural images. The Figure 2 shows the architecture

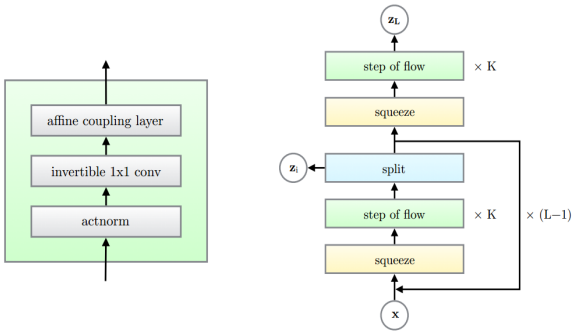


Figure 2: left one step of flow, right the multi-scale architecture of the GLOW [25]

of the GLOW model. Model consists of three components. Let us assume $\boldsymbol{p}$ and $\boldsymbol{q}$ be as the input and output of a layer with shape $\boldsymbol{h}$ x $\boldsymbol{w}$ x $\boldsymbol{c}$, where $\boldsymbol{c}$ is the number of channels and $(\boldsymbol{h}, \boldsymbol{w})$ represent the spatial dimensions.

First component as shown in left part of the figure 2 is the activation normalization layer called as actnorm. Each actnorm layer performs an affine transformation of activations using two 1 x $\boldsymbol{c}$ parameters given as: a scalar $\boldsymbol{s}$, and a bias $\boldsymbol{b}$. This transformations is written as

$$\boldsymbol{p_{i,j}} = \boldsymbol{s} \odot \boldsymbol{q_{i,j}} + \boldsymbol{b},$$

where $\odot$ is the element-wise product.

Second component is the invertible 1x1 convolutional layer. This is a generalization of a permutation operation given as,

$$\boldsymbol{p_{i,j}} = \boldsymbol{W}\boldsymbol{q_{i,j}},$$

where $\boldsymbol{W}$ is a $\boldsymbol{c}$ x $\boldsymbol{c}$ weight matrix.

Third component is the Affine coupling layer implemented similar to the one used in NICE and RealNVP. This layer captures the correlations among the spatial dimensions. The transformation is given as:

$$\boldsymbol{p_a}, \boldsymbol{p_b} = \boldsymbol{split}(\boldsymbol{p})$$

$$\boldsymbol{s}, \boldsymbol{b} = \boldsymbol{NN}(\boldsymbol{p_b})$$

$$\boldsymbol{q_a} = \boldsymbol{s} \odot \boldsymbol{p_a} + \boldsymbol{b}$$

$$\boldsymbol{q} = \boldsymbol{concat}(\boldsymbol{q_a}, \boldsymbol{qb}),$$

where NN is a neural network. Function split() and concat() perform operations along the channel dimension. GLOW uses multi-scale architecture given by the authors of RealNVP [15]. Squeeze layer is used for shuffling the variables and split layers help to reduce the computational cost.

For anomaly detection the model is trained with the normal samples only, with the negative log-likelihood as an objective function. We expect that the model learns the underlying structure of the training data samples. Thus, while testing normal samples will be more likely to be from the same distribution than the anomalous samples and thus we will get a higher negative log-likelihood value for anomalous samples. Therefore, a hypothesis can be put forward that, if the model is trained using the defect-free (normal) samples only then, during the test time, defect-free samples will produce lower negative log-likelihood than the samples containing defects. With this we can detect anomaly samples using negative log-likelihood only.

4. Experimentation

In this section we explain the different experiments performed in this work.

4.1. Experiment 1: Effect of anomaly samples in the training data

In this experiment we aim to find the effect of anomalous samples present in the dataset while training. We make a hypothesis that, if the fraction of anomalies present in the training data is considerably lesser than the normal samples, then at test time negative log-likelihood for normal samples will be lesser than the anomaly samples. It is important to know this as cleaning a dataset to have only good samples, is a difficult and time-consuming task. Thus, it is imperative for any model to know the effect of these anomalies or noise samples present in the dataset on the intended results, where a model is supposed to be trained on normal samples only.

For this experiment we make use of the MNIST dataset. We train the model with the images of one digit only, which is assumed as the normal class (digit 1). The small varying fraction of a different digit samples which is assumed to be the anomalous class (digit 8) is added to the dataset while training as shown in Table 1.

As it can be seen from Table 1 the number of anomalies (digit 8 samples) present in the training set

Case	Normal class digit	Total samples	Anomaly class digit	Total samples	% Anomalies in the dataset
1	1	6742	8	7	0.10%
2	1	6742	8	67	1.00%
3	1	6742	8	674	9.99%
4	1	6742	8	1348	20.00%
5	1	6742	8	2696	40.00%
6	1	6742	8	5851	86.78%

Table 1: Composition of training data in every case. Total samples from the normal digit class plus the total samples from the anomaly digit class create the training samples for one case

are gradually increased from 0.1% to 87% of the normal samples. Thus, the total number of training samples in each case is equal to the total normal samples (6742) and anomaly samples combined.

The Figure 3 shows the plot of negative log-likelihoods of test samples from normal and anomalous class when trained with the training data as shown in the Table 1. Ideally, the more frequent class i.e. the normal class should result in lower negative log-likelihood as model sees those samples more frequently than others. Also, As the number of anomalous samples in the training dataset increases, the negative log-likelihood of the anomalous class samples should also get better.

In Figure 3 only three cases from table 1 are shown for the ease of interpretation. From Figure 3 it can be seen that the model behaves as expected and we see that negative log-likelihood for normal samples, i.e digit one, is consistent as the number of normal samples in the dataset is constant. On the other hand, for anomalous samples we see the negative log-likelihood does get better as the plot gets pushed down, with more number of samples in the training dataset.

This shows that the model is able to learn the distribution of normal samples in spite of the presence of anomaly samples in the training data. From this we can say that the model is able to neglect the 'noise' and focus on the actual data samples while training.

This relation should hold true when the normal and anomalous class from the previous case is reversed. However, we found that this was not the case. We noticed that if we use digit 8 samples only as training data and use digit 1 as the anomaly data set while testing the model still produced better negative log-likelihood to the digit 1 test sampled which is the anomaly class in this case. This unusual behaviour is seen with other pairs of digits also. Another thing that we noticed that this behaviour is not necessarily reversible.

The Figure 4 shows that the anomaly samples i.e samples of digit 1 produced the better likelihoods than the test samples of normal class(digit 8), even when there are zero samples of anomaly class (digit 1) in the training data. We speculate that three probable reasons that could cause the model to behave in this way:

- MNIST samples are not complex enough for the model and thus the model is overfitting and simply remembering the data.
- Number of black pixels: As it is easier for the model to learn the black pixels (zero pixels). Thus the class getting better negative log-likelihood all the time has more number of black pixels.
- The class getting better negative log-likelihood is a subset of the class which is being used for training.

4.1.1 Investigating the issue of better likelihood to anomaly class samples

To check whether it is a more generalized problem for the model and not related to the complexity of the dataset we used Fashion-MNIST, a more challenging dataset than MNIST, in a similar fashion by assuming one class as normal and using another class as the anomaly class. We chose pairs of classes as shown in the Table 2. While making pairs we made sure that the pairs has the mixture of similar and different objects for training and test e.g sneaker and ankle boots, or sneaker and trousers. This way of pairing also made sure that there are classes with similar number of average black pixels in training and test image sample and there are pairs with different number of average black pixels in training and test samples as shown in Table 2.

From Figure 5 it is clear that the Fashion-MNIST dataset also shows this problem thus, the MNIST dataset and its simplicity is not the cause of the unexpected behaviour of the model. We also see that the cases where the model failed Further, if observed closely the cases where the model is behaving unexpectedly have one thing in common. The normal class,

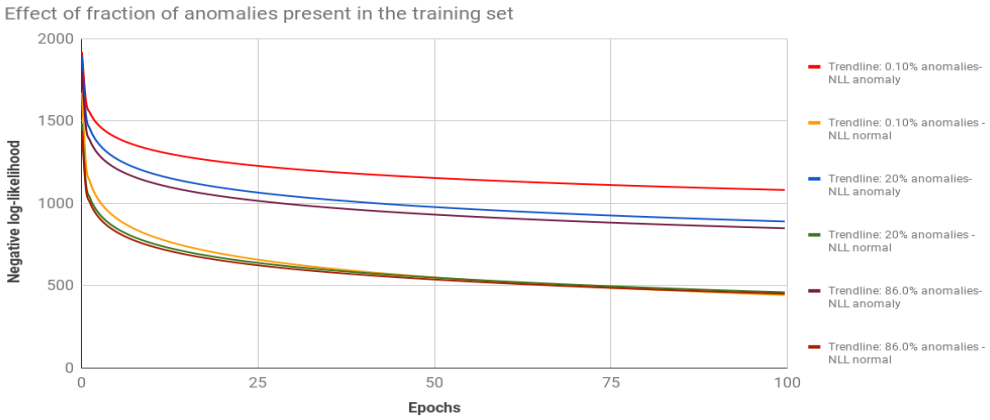


Figure 3: Negative log-likelihoods for three cases (0.10%, 20% and 86% anomalies) from Table 1. Lines represent the trend of absolute negative log-likelihood of the test samples. The Lower three plots represent the Negative log-likelihoods (NLL) of the test samples from the normal class (digit 1). Upper three plots represent the NLL of the test samples from the anomaly class (digit 8) with fraction of anomalies 0.10%, 20% and 86%. As the fraction of anomalies in the training data increases with each case, from the trend-lines we can see that the NLL of anomaly class (digit 8) test samples got better, while normal class test samples' NLL remained in the same range as the number of samples for it remained constant. We can say that the model is able to focus on the normal data and does not get affected by "noise" in the training set.

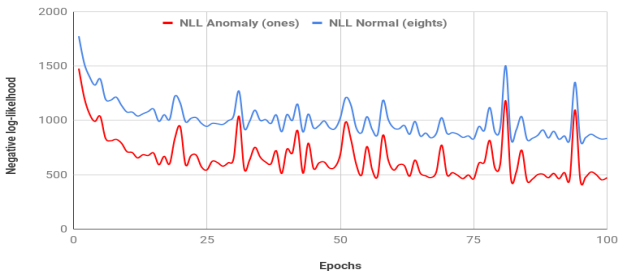


Figure 4: Negative log-likelihood plot when the normal and anomalous classes are reverse. Red represent the negative log-likelihood of anomaly class (digit 1) and blue represents the negative log-likelihood of the normal class (digit 8).

Sr No.	Normal class	Average number of black pixels	Anomaly class	Average number of black pixels
1	Sandals	532	Sneaker	519
2	Sneaker	519	Sandals	532
3	Coat	312	Shirt	290
4	Shirt	290	Coat	312
5	Sandals	532	Ankle boot	403
6	Sneaker	519	Ankle boot	403
7	Ankle boot	403	Sandals	532
8	Ankle boot	403	Sneaker	519
9	Trouser	510	Sneaker	519
10	Sneaker	619	Trouser	510
11	Sandals	532	Coat	312
12	Coat	312	Sandals	532

Table 2: Different cases used investigate the problem that causes the model to produce better log-likelihood for the anomalous samples. The table shows pairs the object class from fashion-MNIST used as normal and anomaly class. The model is trained using the samples from the normal class only. Column 3 and 5 show the average number of black pixels for the image samples from that class.

the one which is used for training has less number of black pixels than the test anomaly class. Thus, we can assume that the problem arises due to the presence of black pixels. As they are easy to learn the model will produce better likelihood if the number there are higher number of black pixels.

To validate that the problem is indeed because of the black pixels again a small is carried out. We cropped the image samples of Fashion-MNIST dataset for class shirts and class coats by keeping a tight boundary around the image sample and cropping out black pixels' area as much as possible. We resized the cropped images back to the shape of 28*28. We performed the same test as in case 4 from the Table 2 and found out

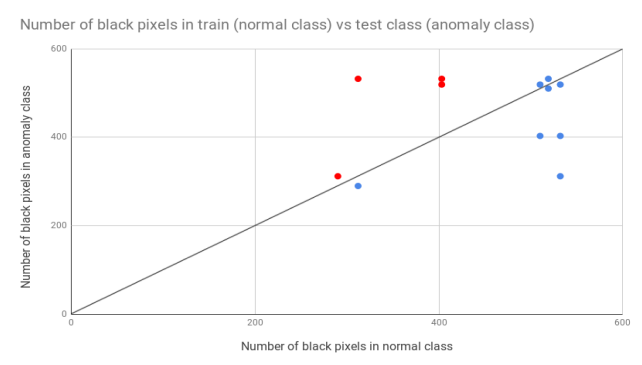
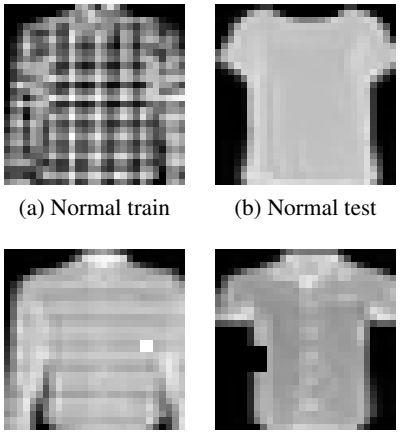


Figure 5: The plot shows the cases (in red) from 2 when the anomaly class got less negative log-likelihood than the training class. Case 4, 7, 8 and, 12 from table 2 did not behave as expected. Data-points in blue represents the cases from table 2 where model behaves as expected. All cases when model failed have have higher average number of black pixels in anomaly class than normal class.



(a) Normal train (b) Normal test

(c) Test Anomaly (d) Test Anomaly

Figure 6: The samples from the shirt class of Fashion-MNIST dataset, used in the experiment. Top row shows the normal train and test samples without the anomalies. Bottom row shows the test samples with black and white square of anomalies introduced on them

that the model behaves as expected.

There is one recent paper [42] which shows the effect of background pixels on the likelihood of the autoregressive pixelCNN model. Other than that, there is no work that shows this limitation related to the black pixels of the flow-based models.

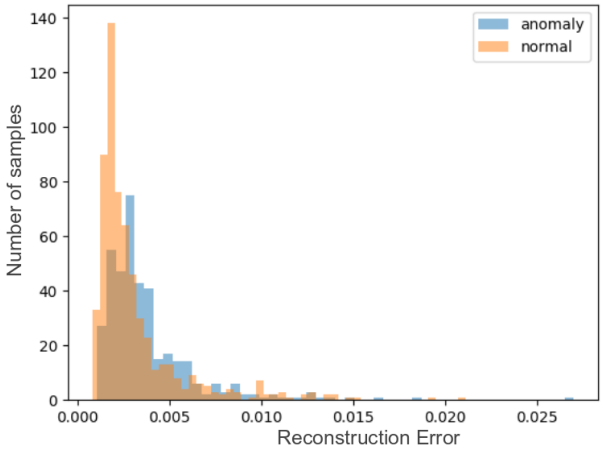
4.2. Experiment 2: Effect of the size of anomalies.

The convolution autoencoder tends to generalize while reconstructing the images as a result we may not able to identify small anomalies present on the images whereas flow-based model because of the exact log-likelihood evaluation does not suffer through this problem. Thus, we make a hypothesis that the flow-based model will be able to detect anomalies which CAE because of the generalization will not be able to detect.

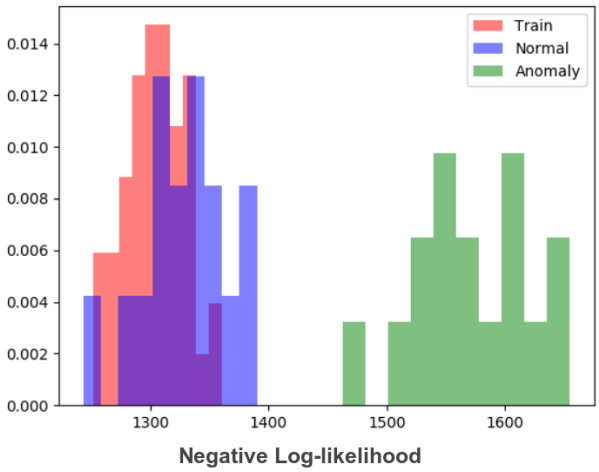
For this experiment, we manually created the anomalies in the Fashion-MNIST dataset and use that to test whether the CAE is able to detect those anomalies or not. The class 6 (shirt) from Fashion-MNIST has 6000 training and 1000 test samples. The number of black pixels present in the image is not going to affect the result in this case as the training and test samples are from the same class of shirts. Out of the 1000 test samples around 400 are marked with the anomaly. The anomalies are generated randomly over the object in the image. The anomalies varies in size of 2x2 pixels to 4x4 pixels. The Figure 6 show the image samples of training data, normal test data and anomalous test data.

This problem setting is closer to the actual problem setting of the BoroscopeV1.0. There also, the blades having anomalies are going to be exactly as similar as the normal blades, except the part where there is the addition of a scratch or dent mark somewhere randomly on the image, which is also going to vary in size from very small to large.For this experiment we trained a convolutional autoencoder with 3 convolutional layers in the encoder, max pooling with kernel size 2 and stride 2 and ReLU [20] activation along with the decoder with 3 transposed convolutional layer of upsampling, ReLU and Sigmoid activation at the last layer. Mean Squared Error (MSE) is used as the objective function. The GLOW model used for this experiment has 2 levels with 32 steps of flows in each. For coupling layers a simple convolutional network with one hidden layer is used.

The Figure 7 (a) and (b) shows the distribution of the reconstruction error and negative log-likelihood of test normal and anomaly samples for both CAE



(a) Distribution of the reconstruction error for CAE



(b) Distribution of the Negative Log-likelihood for GLOW

Figure 7: Fig (a) it can be seen that the distribution of test normal(orange) and test anomaly(blue) samples has a clear overlap because CAE tends to generalize over the anomalous part in the image and produces reconstruction error almost similar to normal samples. In Fig (b) we can see that flow-based model has a clear difference between the range of negative log-likelihood for test normal(blue) samples and test anomaly samples(green). Also test normal samples produces the negative log-likelihood values in a range similar to the training samples (red).

and GLOW, while Figure 8 shows a sample from test anomaly class with its reconstruction. From the reconstructed sample from Figure 8 we can see that the autoencoder generalized over the part of the anomaly

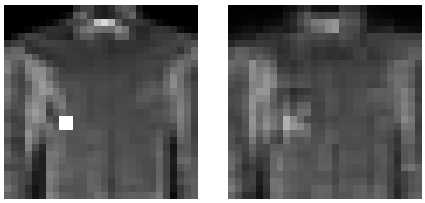


Figure 8: Left: Test sample with white anomaly square mark. Right: Reconstructed output from the CAE that generalized over the anomalous region.

and more often than not makes the pixels as similar to surrounding pixels. Even in the histogram of the reconstruction error we can see majority of the anomaly samples get the reconstruction error close to the reconstruction error of the normal test samples.

The Figure 7 (b) shows the distribution of the negative log-likelihoods for normal and anomaly test samples with the help of normalized histogram. From the Figure 7 (b) it is clear that the flow-based model is able to differentiate between normal and anomalous test samples based on the negative log-likelihood only. The model produced consistently high negative log-likelihood values for the test anomaly samples. Thus, it is possible to detect anomalies based on negative log-likelihood values only.

This experiment shows the limitations of the autoencoder based approach when anomalies in the images are very small. As a result, it underlines the importance of finding a better approach for the task of anomaly detection in a critical setting such as of turbine inspection where it is vital to detect smallest possible anomalies. Thus, in this work flow-based approach is used as BoroscopeV1 has contains anomalies varying in the size from very small to big deformities in the image as shown in Figure 12.

4.3. Experiment 3: Effect of frequencies

In this experiment we investigate the effect of frequencies in the image on the unsupervised deep neural nets. In experiment 1 we saw the effect of the black pixels and how does that affects the negative log-likelihood of the GLOW. The paper [42] shows that the autoregressive model pixelCNN also follow a similar behaviour. Thus, we put a hypothesis that the unsupervised deep neural nets rewards low frequency images than the images with high frequencies in it. will

be reconstructed better or will get better negative log-likelihood values compared to the high frequency images.

This means that the generative models used in this experiment i.e 1.Fully connected autoencoder, 2.Convolutional autoencoder(CAE), and 3.GLOW will be able to reconstruct the images with low frequencies with a lower reconstruction in case of AE and CAE, or will get better negative log-likelihood values in case of glow compared to the high frequency images. To test this hypothesis, we created set a small dataset explained as follows.

Toy Frequency dataset

The Figure 9 shows the samples from both classes, low and high. The low frequency class has frequencies ranging from one to five, whereas frequencies in range 11 to 15 belongs to high frequency class. To make sure that there are enough samples in the dataset, variations in the intensity are used as a form of data augmentation as well as 90 degree rotation. As a result, both low and high frequency class has 600 images each, with 120 images for each frequency.

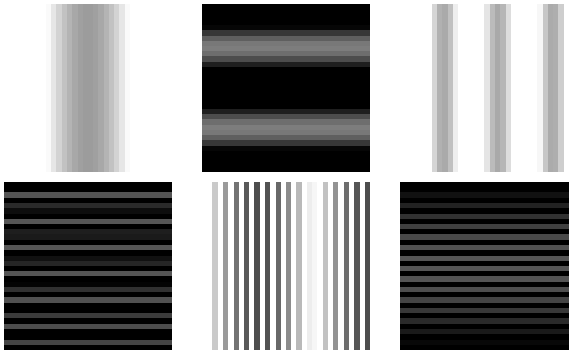


Figure 9: Samples from the toy frequency dataset. Top row shows the samples from the low frequency class with frequency one, two and three from left to right. Bottom row shows the frequency from the high frequency class with frequency eleven, twelve and thirteen from left to right

The Fully connected autoencoder has 3 fully connected layers in encoder with ReLU as an activation function. The decoder also similarly has 3 fully connected layers with the ReLU as non-linearity. Only difference is the final fully connected layer has the sigmoid activations. Convolutional autoencoder and GLOW architecture is same as from experiment 2.

The Figure 10 shows the distribution of the error values, i.e the reconstruction error for fully connected and convolutional autoencoder and negative log-likelihood for GLOW, in a scatter plot, for all the frequencies separately. From the Figure 10(a) we see that frequency has no effect on the fully connected autoencoder as all the frequencies were reconstructed in a similar range of reconstruction error values. However, from Figure 11 we can observe that the intensity of the pixels played a part in the ability of the model to reconstruct the test samples. For every frequency we feed the model in the ascending order of the intensity, we observe that with the increase in the intensity irrespective of the frequency the reconstruction error for test samples became higher.

In Figure 10(b) we see that frequency 14 and 15 affected the reconstruction ability of the CAE significantly. Similarly in case of Figure 10(c) frequency 14 and 15 test samples produced negative log-likelihood significantly higher than the rest. However, in both Figure 10(b) and (c) we can see an upward trend of the error values indicating that the frequencies does play a role in the ability in the ability of the models to either reconstruct or predict likelihood.

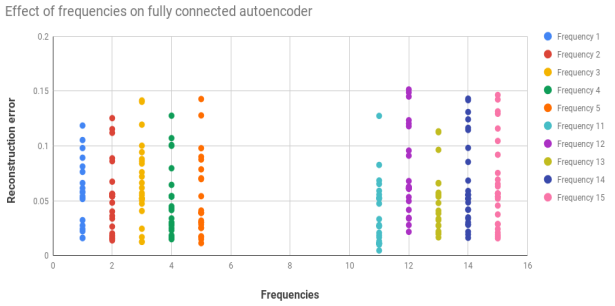
Thus, we can conclude that the fully autoencoder was not susceptible at all to the higher frequencies but is clearly gets affected by the intensities in the of the pixels in the image. Whereas CAE and GLOW did not show any such effect because of the intensities but showed a bias towards the low frequency in the images.

5. Results with BoroscopV1

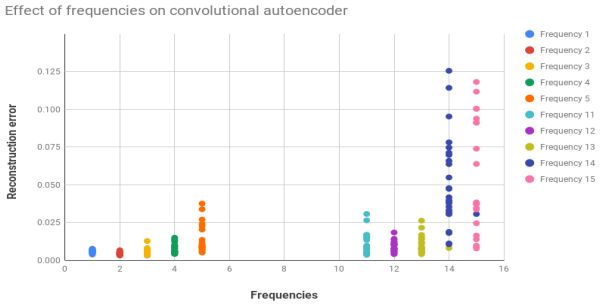
In this, we show the results obtained with the BoroscopeV1 dataset.

BoroscopeV1

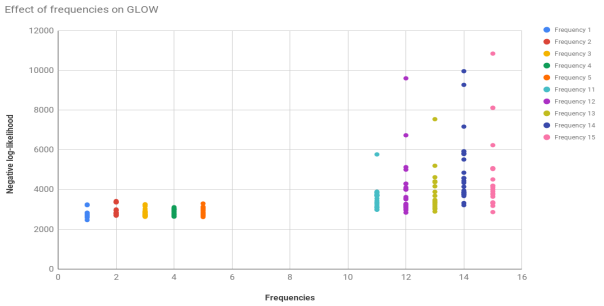
BoroscopeV1 dataset has 380 videos of blades of 2 to 10 minutes each from the various view points. Out of which, 330 videos are of clean blades while 50 contain anomalies.
The BoroscopeV1 has a test set which has a total of 800 images. Out of those 800 images, 283 have defects while the rest are validated to be free from any defect.



(a) Distribution of reconstruction error from Fully connected autoencoder. We see no effect of frequency on the reconstructions of fully connected autoencoder.



(b) Distribution of reconstruction error for Convolutional Autoencoder. We can see that the reconstructions of the CAE are affected with the frequencies



(c) Distribution of Negative log-likelihood for GLOW, Similar to CAE we see that frequency does affect the likelihood of the model

Figure 10: Figure shows the distribution of the error values for all three models for all the frequencies from low (1-5) and high (11-15) separately. Frequencies from 6 to 10 were not considered to have a clear distinction between low and high class of frequency in the images.

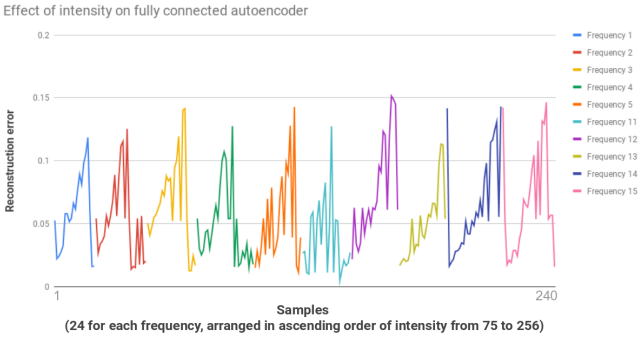


Figure 11: Reconstruction error by the fully connected autoencoder for all the frequencies separately, when each test sample for each frequency is fed in the order of increasing intensity. Y axis shows the reconstruction error. The each frequency represented by different colour on X axis has the intensity ranging from 75 to 256. We clearly see a trend that as the intensity goes up the reconstruction error goes high

This test set is used as it is while testing the model.

For training, BoroscopeV1 has 98902 images extracted at a constant rate of 300 images from each video. In this work, we use only ten percent of these frames for training purpose, meaning we effectively use only 30 images from each video. These images are first cropped with the size depending on the aspect ratio of the video and also to get rid of the unnecessary markings present on the frames in all four corners. The images in the Figure 12 shows the variety of the viewpoints and also the type of defects that are present in the dataset.

In this work we apply a patch based approach with a patch size of 32x32 rather than feeding the whole image to the model. Because, of the two reasons. First, it saves the training time. Flow-based model has the large number of transformations chained together called as “flow”, which takes a considerably more time to train even for small datasets compared to other unsupervised models. With patch size of 32x32 we hope to keep the training time reasonable. Second, the BoroscopeV1 has the images of the blades from different viewpoints. Thus, on a whole image level it is possible that two image samples look completely different. However, on a patch level, these images are almost similar. This reduces the complexity while training, as we do not need to have multiple Gaussian dis-

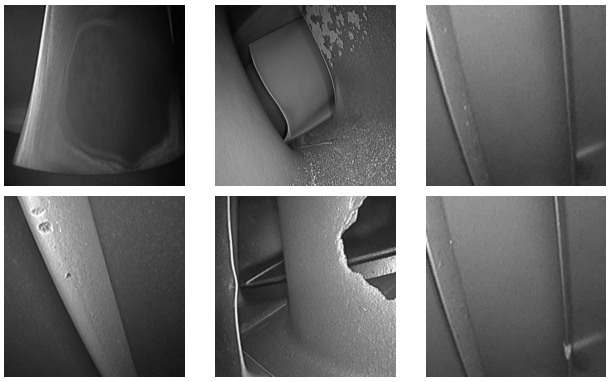


Figure 12: Samples from the BoroscopeV1 after cropping. Top rows shows the healthy samples, bottom row shows the different anomalies. Right image on top and bottom row show the difference between a normal and anomalous sample.

tributions as a prior to map these two different blades. Also with this approach we are able to locate anomalous area within the image at a patch level.

The train and test sets were composed as mentioned in the previously with around 800 samples in the test set and 9800 samples for training set. Figure 13 show the heatmap generated by the model on some samples.

In some case such as figure 13 first row, the model works extremely well. It exactly detect the patch with the anomaly. Similarly in the second and third row of the figure 13 we observe that the anomaly patches did got identified correctly. However, in second and third row the texture of the blade did produce some some false positives. Also the fact that majority of the anomalous region for all three samples from figure 13 ended up under a single patch helped in identifying those anomalies correctly.

figure 14 shows some of the failure cases. We see from the first two rows from figure 14 that the reflection from the blades was detected as an anomalous region. The reflections on these blades does resemble the scratched surface of a blade and thus model identifies the reflections as anomalies too. This can be resolved by either adding more training samples because it will help model see more samples with bright patches and thus not identify anomalies solely on the intensity values or by creating a better dataset by keeping a precautions that the reflections will not occur.

Other than that, if the anomalous region is partially

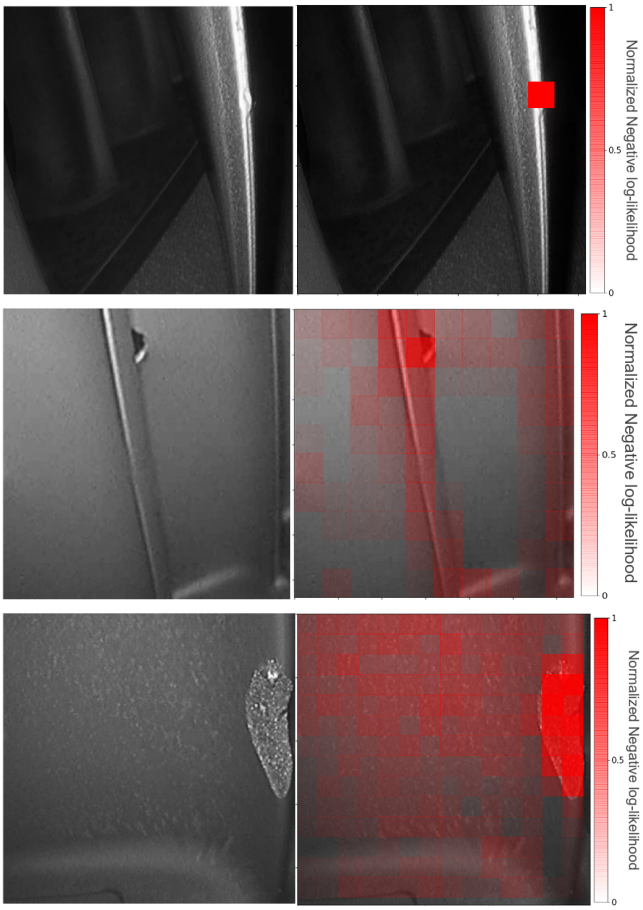


Figure 13: Left column: Test input samples containing anomalies, Right column: Corresponding generated heatmap. Anomalous part of the blade results in a darker red patch in the corresponding heatmap. Darker the red patch higher the negative log-likelihood. We see that model is able to detect the anomalies. There is clear distinction in the predicted likelihood of the normal patch and anomalous patch.

covered by the patch then that also affects the ability of the model to detect the anomalies. All three samples from figure 14 show this issue. This can be resolved by keeping the stride length as minimum as possible but, it has the trade off response time. For e.g a 320x320 image generates 100 patches if the stride is kept as 32. i.e. For every actual test sample we need to create, test and then generate heatmap for 100 patches. This trade off depends on application.

Third factor affecting the model is the black pixels. From experiment 1 we showed that the black pixels be-

ing easy to learn cause a problem when predicting the likelihood of the data sample. Sometimes the background of a blade is black shadow and that gets exposed because of the anomaly in the blade. As a result what model sees is the black shadow and not the actual blade and thus the model does not identify that as an anomaly.

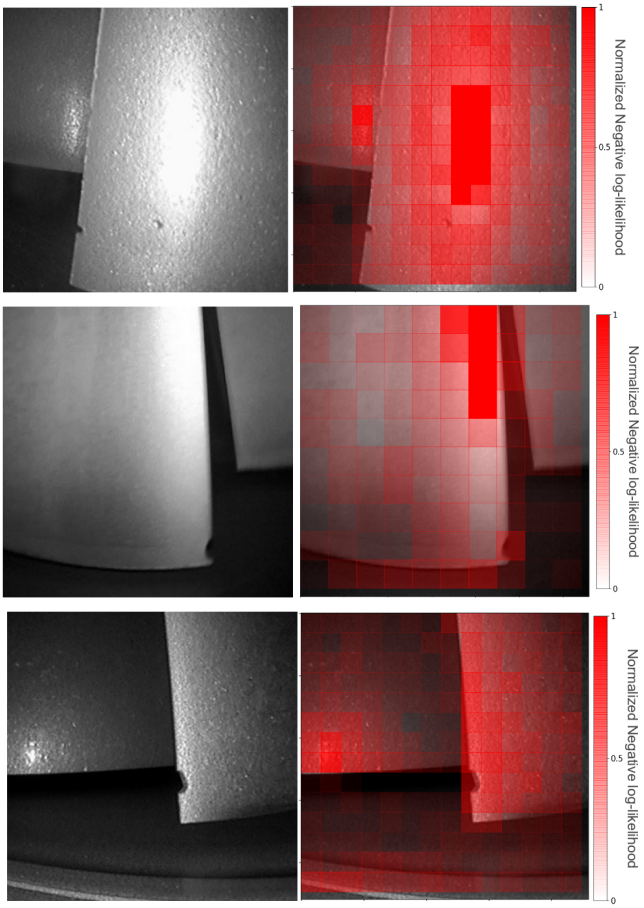


Figure 14: Left column: Test input samples containing anomalies, Right column: Corresponding generated heatmap. Part of the blades having anomalies results in a darker red patch in the corresponding heatmap. Darker the red patch higher the negative log-likelihood. This figure shows some of the failed cases. Reflections from the shiny surface of blades, the area of the anomalous region under the patch, black pixel, texture of the blade played a role in the ability of the model in identifying the correct anomalies.

One important thing to note here is that because the images are investigated in patches and there is no information sharing regarding the nearby patches while calculating the likelihood, it may happen that the bigger anomalies might go undetected or partially detected as the model does not have a global context understanding and thus it may miss some anomalies. Other than that the model does perform well in detecting small anomalies because of the patch-based approach. The middle image in 14 and the first image in 13 are the perfect examples of that.

6. Conclusion

This work explores the problem of anomaly detection with the normalizing flow-based model. In this work we first verified the approach on the toy datasets with help of small experiments. We were able to show that the flow-based model is able to detect the acute anomalies present in the images which convolutional autoencoder did not. This establishes the need of a approach other than the reconstruction based methods. We believe through this work that the ability of the normalizing flow-based model to be able to do the exact log-likelihood evaluation could provide that alternative approach. We showed that black pixels causes the flow-based models to give a better negative log-likelihood value. We also showed that convolutional autoencoder and GLOW show a bias towards the low frequencies in the image whereas fully connected autoencoder does not. However, fully connected autoencoder is affected by the intensities of the pixels in the image.

In case of a more complex BoroscopeV1 dataset, we showed promising results where normalizing flow-based model was able to detect the anomalies in the image precisely. However, in some cases it failed to differentiate between the normal and anomalous samples we speculate it could be either because of the reflections from the blades or the black pixels. Also, because of the patch based approach some anomalies which landed on the border of the patches could not be detected.

Recommendations

Anomaly detection is a not an easy problem than it seems from the first glance. The unavailability is of the anomalous data samples is the biggest challenge while working on problem like this. In this work we explored

this problem with the normalizing the flow-based models. The some of the disadvantages discussed below can be resolve with simple steps which could be the immediate direction for the future research such as the effect of reflections in detecting the anomalies in the images can be resolved with the help of more data, as we are only using 10 percent of the BoroscopeV1 it is possible to add more training samples to generate better results, or the problem of anomalous region partially covered by the patches can be resolved by changing the stride lengths to make sure there is an overlap between the patches such that, anomalous part is not missed because of the patch border or making the patch size bigger depending upon the size of anomalies in the dataset.

Keeping in mind the different viewpoints from the BoroscopeV1 we another direction for future work which could be interesting to explore is the conditional normalizing flow. This paper [31] shows the use of conditional glow for structured prediction task of inpainting. The flow-based models have shown the state of the art results in the synthesis of the new samples because of their invertability and the exact log-likelihood calculation ability. Thus it will be interesting to explore this approach for anomaly detection.

References

[1] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian Conference on Computer Vision*, pages 622–637. Springer, 2018. 3

[2] S. Akçay, A. Atapour-Abarghouei, and T. P. Breckon. Skip-ganomaly: Skip connected and adversarially trained encoder-decoder anomaly detection. *arXiv preprint arXiv:1901.08954*, 2019. 3

[3] N. Anantrasirichai and D. Bull. Defectnet: multi-class fault detection on highly-imbalanced datasets. *arXiv preprint arXiv:1904.00863*, 2019. 2

[4] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 15–27. Springer, 2002. 2

[5] A. Bahnshen. Building ai applications using deep learning, Jun 2017. 2

[6] R. J. Bolton and D. J. Hand. Unsupervised profiling methods for fraud detection. 2001. 2

[7] S. Byers and A. E. Raftery. Nearest-neighbor clutter removal for estimating features in spatial point processes. *Journal of the American Statistical Association*, 93(442):577–584, 1998. 2

[8] R. Chalapathy and S. Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019. 1, 2, 3

[9] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009. 2

[10] S.-B. Cho and H.-J. Park. Efficient anomaly detection by modeling privilege flows using hidden markov model. *computers & security*, 22(1):45–55, 2003. 1

[11] H. Choi and E. Jang. Generative ensembles for robust anomaly detection. *arXiv preprint arXiv:1810.01392*, 2018. 1, 2, 4

[12] L. Deecke, R. Vandermeulen, L. Ruff, S. Mandt, and M. Kloft. Image anomaly detection with generative adversarial networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 3–17. Springer, 2018. 1

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 2

[14] L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014. 1, 3, 4

[15] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 1, 3, 4, 5

[16] H. Dutta, C. Giannella, K. Borne, and H. Kargupta. Distributed top-k outlier detection from astronomy catalogs using the demac system. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 473–478. SIAM, 2007. 1

[17] E. Eskin. Anomaly detection over noisy data using learned probability distributions. 2000. 1

[18] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer, 2002. 2

[19] M. K. Ferguson, A. Ronay, Y.-T. T. Lee, and K. H. Law. Detection and segmentation of manufacturing defects with convolutional neural networks and transfer learning. *Smart and sustainable manufacturing systems*, 2, 2018. 2

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016. 8

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2

[22] S. Hawkins, H. He, G. Williams, and R. Baxter. Outlier detection using replicator neural networks. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 170–180. Springer, 2002. 3

[23] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006. 2, 3

[24] E. Hoogeboom, J. W. Peters, R. v. d. Berg, and M. Welling. Integer discrete flows and lossless compression. *arXiv preprint arXiv:1905.07376*, 2019. 1, 3

[25] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018. 1, 3, 4, 5

[26] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 2

[27] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014. 4

[28] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2

[29] K. Lis, K. Nakka, M. Salzmann, and P. Fua. Detecting the unexpected via image resynthesis. *arXiv preprint arXiv:1904.07595*, 2019. 2

[30] T. T. Lu. Fundamental limitations of semi-supervised learning. Master's thesis, University of Waterloo, 2009. 3

[31] Y. Lu and B. Huang. Structured output learning with conditional generative flows. *ArXiv*, abs/1905.13288, 2019. 14

[32] R. Mehrotra, A. H. Awadallah, M. Shokouhi, E. Yilmaz, I. Zitouni, A. El Kholy, and M. Khabsa. Deep sequential models for task satisfaction prediction. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 737–746. ACM, 2017. 3

[33] D. Mery, V. Riffo, U. Zscherpel, G. Mondragón, I. Lillo, I. Zuccar, H. Lobel, and M. Carrasco. Gdxray: The database of x-ray images for nondestructive testing. *Journal of Nondestructive Evaluation*, 34(4):42, 2015. 2

[34] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*, 2018. 4

[35] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011. 4

[36] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016. 3

[37] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016. 3

[38] P. Oza and V. M. Patel. One-class convolutional neural network. *IEEE Signal Processing Letters*, 26(2):277–281, 2018. 2

[39] M. Pal and G. M. Foody. Feature selection for classification of hyperspectral data by svm. *IEEE Transactions on Geoscience and Remote Sensing*, 48(5):2297–2307, 2010. 2

[40] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *ACM Sigmod Record*, volume 29, pages 427–438. ACM, 2000. 2

[41] G. Rätsch, S. Mika, B. Schölkopf, and K.-R. Müller. Constructing boosting algorithms from svms: An application to one-class classification. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (9):1184–1199, 2002. 2

[42] J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. A. DePristo, J. V. Dillon, and B. Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *arXiv preprint arXiv:1906.02845*, 2019. 8, 9

[43] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft. Deep one-class classification. In *International Conference on Machine Learning*, pages 4393–4402, 2018. 2, 3

[44] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017. 1, 3

[45] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International Conference on Information Processing in Medical Imaging*, pages 146–157. Springer, 2017. 1

[46] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International Conference on Information Processing in Medical Imaging*, pages 146–157. Springer, 2017. 3

[47] V. Škvára, T. Pevnỳ, and V. Šmídl. Are generative deep models for novelty detection truly better? *arXiv preprint arXiv:1807.05027*, 2018. 2

[48] D. M. Tax and R. P. Duin. Support vector data description. *Machine learning*, 54(1):45–66, 2004. 2

[49] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016. 1, 3

[50] V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013. 2

[51] l. weng. Use of deep learning features in log-linear models. *Log-Linear Models, Extensions, and Applications*, 2018. 4

[52] M. Yamaguchi, Y. Koizumi, and N. Harada. Adaflow: Domain-adaptive density estimator with application to anomaly detection and unpaired cross-domain translation. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3647–3651. IEEE, 2019. 1

[53] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*, 2018. 3

[54] J. Zhang and H. Wang. Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. *Knowledge and information systems*, 10(3):333–355, 2006. 2

[55] D. Zhao, B. Guo, J. Wu, W. Ning, and Y. Yan. Robust feature learning by improved auto-encoder from non-gaussian noised images. In *2015 IEEE International Conference on Imaging Systems and Techniques (IST)*, pages 1–5. IEEE, 2015. 3

[56] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. 2018. 3

2

Introduction

2.1. Introduction

Determining which instances stand out as dissimilar compared to other while when analyzing real-world data-sets has become a common need. Such insances are known as anomlaies. The domain of the project is to detect anomalies on the jet engine blade images. These anomalies occur mainly due to different types wear and tear on the blades, and in size from a small scratch or dents to a large deformation of the blade. A new dataset called BoroscopeV1 consisting the images 2.1 of jet engine blades was created for this purpose.

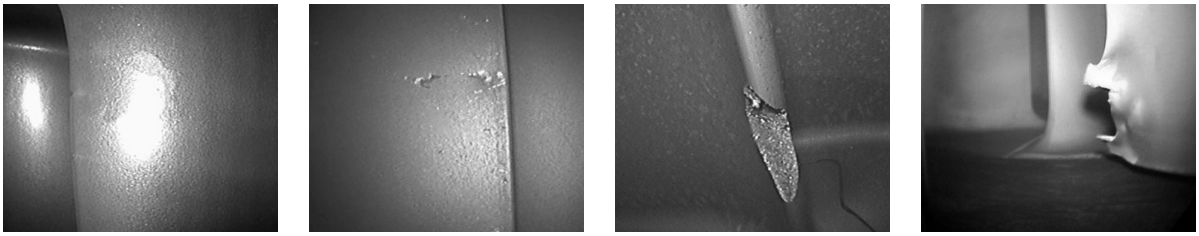


Figure 2.1: Sample images showing variations in the anomalies from the BoroscopeV1 dataset

Jet engine blades some times get scratched or deformed during their operation. As its malfunction can cause a engine failure which could prove life threatening to the passengers travelling in the aircraft. Detection of it while inspection is important. Thus accurate detection of these anomalies occurring in the form of wear and tear in the image of an jet engine blade is extremely important. As the whole operation is done on images this can be modelled as a computer vision problem.

Over the past decade, deep learning techniques have seen a significant rise in the real-world applications specifically in the field of computer vision owing to rising computational power, increased storage and extensive methods of data collection. Convolutional neural networks have shown great applicability when it comes to the tasks such image classification, object detection, segmentation and some more. Detection of features and classification of an image based on that, is a computer vision problem. Feature representation of an image can be done in many ways. One could also use human annotations for describing each image and then classifying based on that, but it is not a practical and feasible approach. Traditional machine learning approaches expects the features to be fed directly to the classifier instead of learning those based on the given data. A neural network can generate its own hidden features which can be effectively used for the detection and classification tasks.
In supervised learning all the labelled training examples are fed into the model, with this model learns the representation of all the classes of the data. This is certainly a desired way as one can optimize the patterns and structures based on the labels. The BoroscopeV1 dataset has a very uneven distribution of the data samples with high percentage of images with no defects which are referred as ‘healthy’ or ‘normal’ samples, and very small percentage of images with defects which are referred as ”unhealthy”,

or ‘anomalous’ samples. As the data distribution is so uneven, supervised model will not see enough images from unhealthy class during training to learn its representation. And consequently will not be able to detect the anomalies accurately as the discriminative model such as supervised classification, tends to bias the predictions towards the class with more number of samples. Thus instead of learning to predict an output depending on input data, an approach which will learn the inherent structure of the input data will be much more useful in this kind of problem setting. Unsupervised learning does exactly that.

Uneven distribution of the training data with high number of normal samples is a classic setting for an anomaly detection problem. Unsupervised representation learning has become very dominant in the task of anomaly detection in the recent years with rise of variational autoencoders, generative adversarial networks, autorgressive and normalizing flow-based models. Anomaly detection a well known sub-domain of the unsupervised learning is a challenging task because of the high dimensional structure of the images. This work shows the normalizing flow-based approach for the detecting the anomalies.

2.1.1. Motivation

Reconstruction using convolutional autoencoder is the most widely approach for the anomaly detection on images. However, the convolutional autoencoder have the disadvantage when detecting small defects in the image as they tend to generalize over that small region which results in lower reconstruction error values. Along with that, choosing a right degree of compression is also a cause of problem because it works as a hyperparameter that needs to be manually tuned and choosing the right value is hard due to the unsupervised nature of the problem setting. Thus, it is important to look for an approach that tries to learn the underlying structure of the data accurately such that we can predict the likelihood of the test samples. Thus, normalizing flow-based approach is used in this work.
The flow-based generative models are able to do exact latent variable inference and log likelihood evaluation. Whereas in variational autoencoders it is possible to only approximate the value of latent variables corresponding to a datapoint. GANs have no encoder at all to infer the latents.

2.1.2. Research Question

The sole objective of this research work is to be able to detect and locate the anomalies of various shapes and size on the images. The aim is to show that normalizing flow-based models can be used to detect to anomalies with a simple objective function of negative log-likelihood.

- *Can we use normalizing flow-based models with simple objective function of negative log-likelihood for anomaly detection?*

While working it was broken down in small experiments as:

- How does the model behave in the presence of anomalies or noise in the training set.
- How does the model detect the acute anomalies present on the image compared to conventional reconstruction based approach.
- What is the effect of low and high frequency images on the ability of the model to predict the likelihood?

3

General Background on Deep Learning

This chapter provides a general background theoretical information on neural networks needed for clear insight. We start with the basic overview of what neural networks are, how they work. Followed by a detailed explanation on the working of convolutional neural networks (CNNs) in classification. We also look into different types of convolution operations that are used in different unsupervised models which are discussed in the next section and also in the related work section in the 1st chapter of the report 1.

3.1. Neural Networks

Neural networks or artificial neural network is made up of a connected acyclic graph of nodes also called as neurons. The neuron is a basic unit of a neural network where computation happens[11]. Figure 3.1 shows the mathematical model of such neuron. The output of a neuron is given by the equation 3.1. This output is then passed through a nonlinear activation function f which gives the final output y as equation 3.2. The equation 3.1 can be interpreted as an affine transformation as scaling with the weights w and then the shifting of the origin with bias b.

$$u = \sum_{i=1}^{n} w_i x_i + b \tag{3.1}$$

$$y = f(u) \tag{3.2}$$

The deep learning is is a subset of neural networks where a multiple layers are stacked one to each

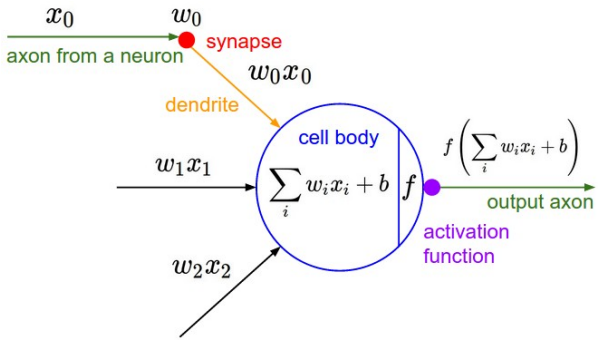


Figure 3.1: Mathematical model of a single neuron in an artificial neural network. A neuron consists of inputs $x_1, \ldots, x_n$ and weights $w = w_1, \ldots, w_n$, a bias b and an activation function f [6]

other to create a hierarchy between the input and the output layer. The Any model that uses more than two layers used in the model is referred as a deep model. Figure 3.2 illustrates a simple three-layer Neural network. Each layer consists of several neurons. Every connection between the neurons exchanges information with the help of weights and activation function. These weights are trained

with the help of backpropagation during training with different objective function depending on the task. The neural network in Figure 3.2 is a fully connected neural networks where every neuron from the previous layer is connected to the every neuron in the next layer. These neurons respond to the different combinations of the inputs from previous layers also neurons within a layer do not share any connection.

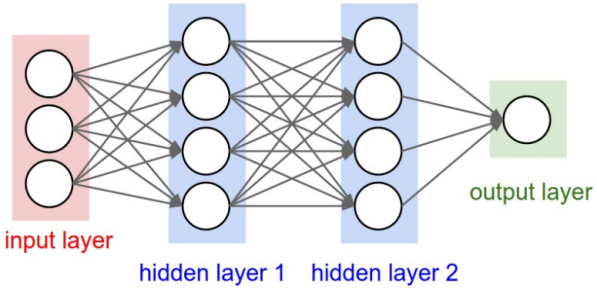


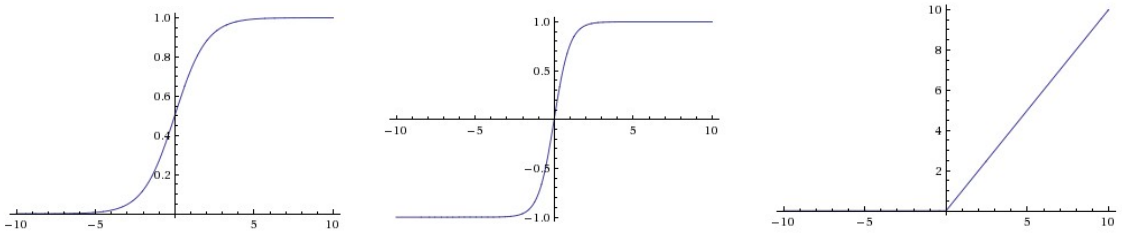
Figure 3.2: A simple three layer fully connected network with 2 hidden layers and an output layer

3.2. Activation Functions

Activation function determines the output that a node will generate, based on its input. It also introduces the non-linearity in the output which helps in capturing the complex arbitary functional mappings between the input and output. Sigmoid, Hyperbolic tangent (Tanh), Rectified Lienar Units (ReLU), Softmax are some of the most commonly used activation functions [10]. Figure 3.3 illustrates these activation functions.

The sigmoid fuction as can be seen from the figure (a) of 3.3 does not have a zero-centered input and suffere from the problem couple of drawbacks such as vanishing gradients and slow convergence [12]. Tanh ovrcomes the problem of slow convergence but still suffers from the problem of vanishing gradients. The main advantage of tanh over sigmoid is the zero centered output there by aiding the back-propagation process.

The ReLU proposed by [10] is the most widely used activation function. It overcomes the problem slow convergence of the sigmoid as well as over come the vanishing gradients problem by use of identity for positive values and thus offers better performance and generalization capacity in deep learning than Sigmoid and tanh [2, 17],



(a) Sigmoid (b) Tanh (c) ReLU

Figure 3.3: Sigmoid (a) squashes real numbers to range between [0,1] where as tanh (b) maps the real numbers to range between [-1,1]. ReLU (c) is zero when x < 0 and linear with slope 1 when x > 0. [6]

3.3. Training a Neural Network

The trainable parameters of the neural network i.e weights and biases are initialized with random values and updated during the training. The problem of training is exactly as same as minimizing loss function. The learning process of a neural network tries to minimize this loss function. The loss function is usually the difference between the ground truth and the value predicted by the model which is computed as the sum of the squared difference between the target values and network output. This can be solved

using the traditional optimization methods of gradient descent, by optimizing the parameters based on the gradient.
In deep learning due high number of parameters and complexity it is computationally inefficient to calculate all the gradients. Backpropagation [9] is used to solve this problem. There are different techniques which facilitate the training of deep complex neural networks with the help of backpropagation. Stochastic Gradient Discent (SGD) [1](cite), Adagrad [4], Adam [7] are a few frequently used examples.

3.4. Convolutional Neural Networks

The convolutional neural networks (CNNs) are the special type of neural networks which are built upon the artificial neural networks. Different hidden layers are used in CNNs than ANNs. These hidden layers are 'Convolutional Layer', 'Pooling layer', and 'Fully connected layer'. Figure 3.4 shows a general block diagram of a CNN in a classification task. Let's look at all the layers one by one.

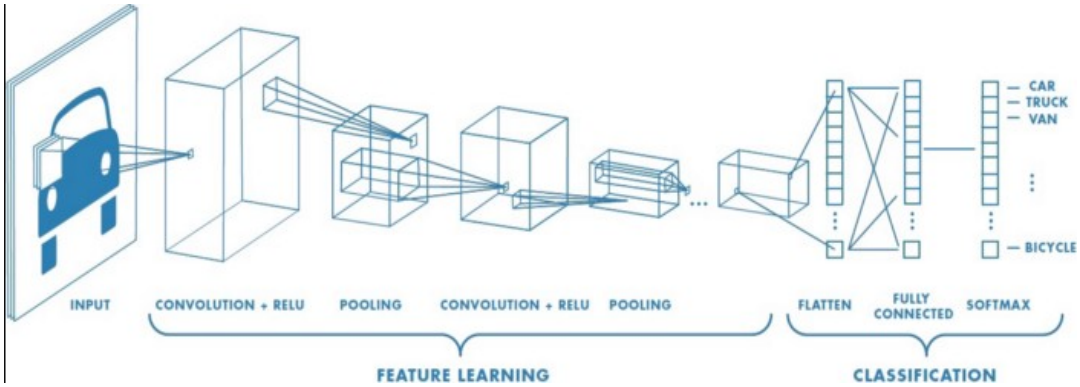


Figure 3.4: A typical block digram of CNN consisting multiple convolutional layers with ReLU activation function, pooling layer and full connected layer at the end with the softmax predictions. [13]

3.4.1. Convolutional Layer

It is impractical to connect all the neurons from the previous layer to the next as in 3.2 for high dimensional data such as images. CNNs consists of convolutional layers which provide an efficient solution for this. Figure 3.5 shows the illustration of a convolutional layer. As it can be seen from the figure, the filter of size WxHxD (here D is 1) is convolved over a region of the input volume to generate a single pixel value. Numerous convolution operations are per formed over an input where each convolution operation uses different filter. Thus, we end up different feature maps. These different feature maps are put together to generate an output of one convolutional layer. These filter are slided over the whole input volume to generate output from that local region of pixels. This is the parameter sharing scheme used in convolutional layers to reduce the number of parameters.

It is important to discuss to special types of convolutions which are used in the models discussed in the 1.

Transposed Convolution

It is also known as deconvolution. As it can be seen from the figure 3.4, convolution operation shrinks the input volume. However, in may cases such as generating high resolution images, semantic segmentation or in the decoder part of an auto-encoder we need to perform the up-sampling. Traditionally up-sampling can be done with different interpolation schemes. Modern architectures such as neural networks however, tend to learn this transformation.
A transposed convolution however does not exactly deconvolves the previous convolution operation done on the image. If an input of 5x5 is undergone an convolution operation to create a feature map of 2x2. Transposed convolution on the output feature map 2x2 carries out a regular convolution operation only but reverses its spatial transformation. The transpose convolution we make sure that the output feature map is same as what we started with (5x5 in this case). Such that, it reconstructs the

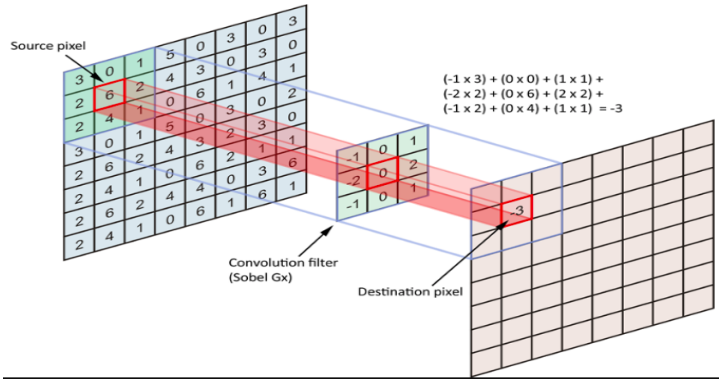
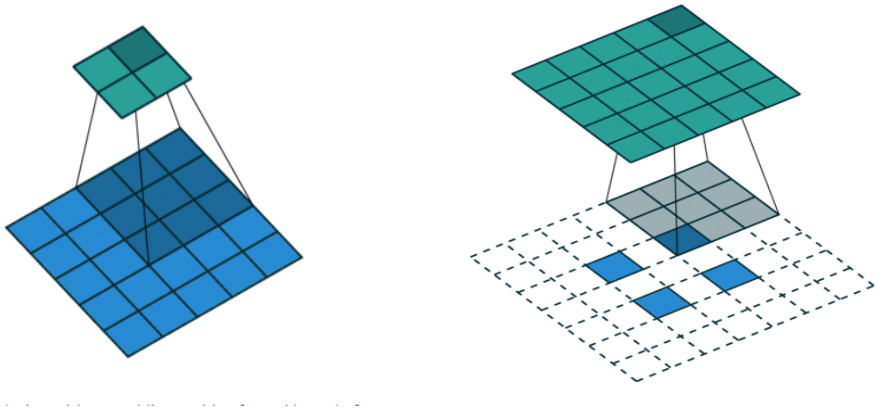


Figure 3.5: Illustration of the convolution operation in a 2D case. [3].



(a) Convolution (b) Transpose Convolution

Figure 3.6: Left shows the normal convolution operation on a 5x5 feature map resulting in 2x2 area. Right shows the transpose convolution on 2x2 feature map generating the an output of 5x5

spatial resolution from before and performs convolution with the help of some padding. This is not the mathematical inverse of the convolution process but for encoder-decoder architectures it's very useful.

1x1 Convolution

The figure 3.7 shows the illustration of 1x1 convolution where input tensor has the dimensions WxHxD and the filter size is 1x1xD. After convolution the output tensor is of size WxHx1. If N such convolutions are applied we will end up with the output layer of dimension WxHxN. 1x1 convolutions facilitates the dimensionality reduction for efficient computation and the feature pooling capacity. One more advantage of this as described in [14] is that, we can again apply the non-linearity after the convolution which helps the model to learn more complex function.

3.4.2. Pooling layer

Convolutional layers are generally followed by pooling layer. This layer helps in reducing the spatial dimension of the input volume to limit the parameter size and control overfitting. It operates independently on every depth slice of the input using MAX operation. 2x2 filter size with a stride of 2 is the most commonly used pooling operation. It reduces the spatial dimensions by 2 folds in both X and Y direction as shown in image 3.8

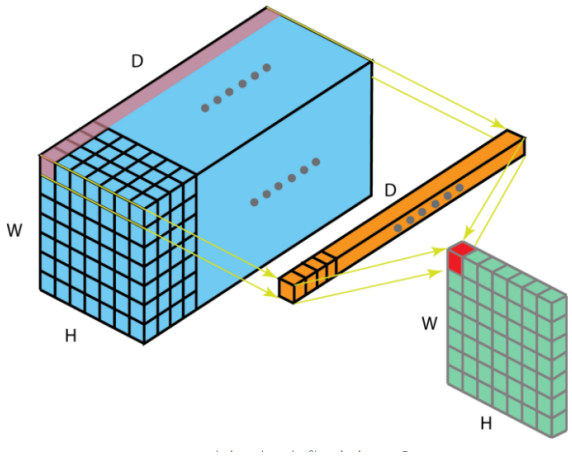


Figure 3.7: Illustration of 1x1 convolution where filter size is 1x1xD.

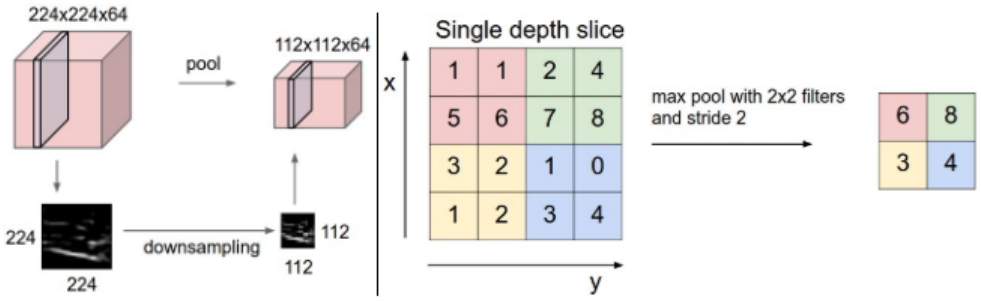


Figure 3.8: Pooling layer down-samples the volume spatially. Left image shows reduction in the spatial dimension except depth using pooling. Right image shows how MaxPooling works [6]

4

Unsupervised Deep Learning Models

In previous chapter we discussed the basic of deep learning and a working of simple convolutional neural network. In this chapter we will look at the models which work when there are no labeled training examples. These are the unsupervised models used in deep learning. This chapter will give a brief explanation of the working principle of a few such models such as, Autoencodres, Variational autoencoders (VAE) [8], Generative Adversarial Networks (GANs) [**?**] and which are referred throughout this report.

4.1. Autoencoders

Autoencoders are the most widely used unsupervised models. Autoencoders apply backpropagation, by setting the output values to be equal to the inputs. Figure 4.1 shows a simple fully connected autoencoder with one hidden layer. There are many use cases of the autoencoders such as anomaly detection, image denoising to name a few.

Autoencoder, by design, is a dimensionality reduction algorithm which learns to model the common variation in the data samples and ignore the noise. An autoencoder consists of 4 parts 'Encoder', 'Bottleneck', 'Decoder', 'Reconstruction loss'. Encoder is the part where model learns to reduce the input dimensions and map the compressed input data on the bottleneck layer.

Bottleneck layer is the layer which holds the compressed representation of the input data from an encoder. Decoder is the more often than not a mirror image of encoder. It learns to reconstruct the data from the compressed representation to be as close to the original input as possible. Reconstruction loss is the loss function which is used to measure the reconstruction quality of the decoder.

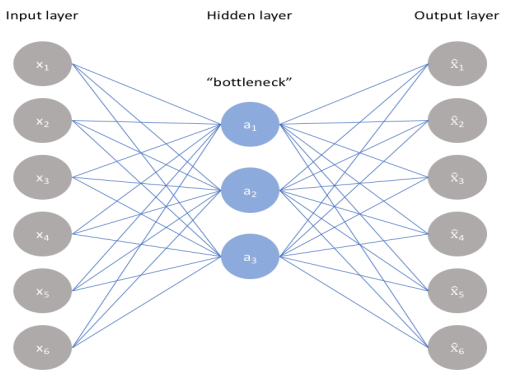


Figure 4.1: Fully connected autoencoder with one hidden layer

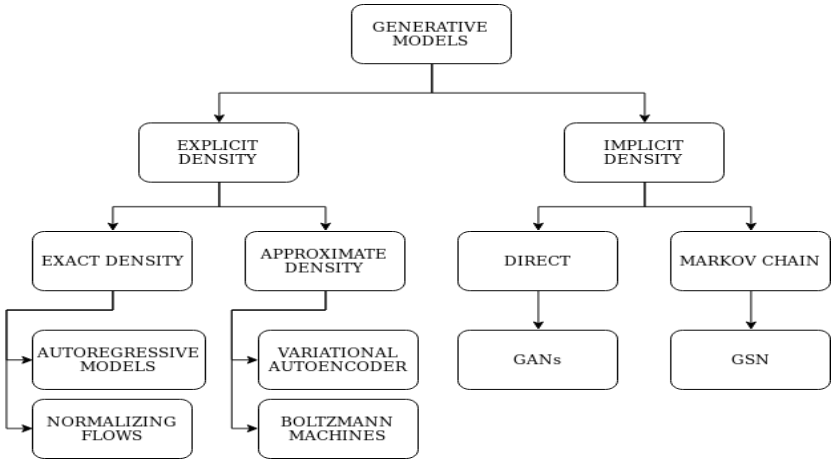


Figure 4.2: Caption

4.2. Generative models

There is a set models that learns the underlying structure of the data and are able to generate the samples from the distribution similar training samples. These are knows as generative models. The Figure 4.2 show the generative models. Generating new samples, density estimation, clustering, dimension reduction are a few typical use cases of generative models.
All generative models aim at learning the true data distribution of the training set so as to generate new data samples one way or another. However, it is not always possible to explicitly learn the probability density function of the real data. Thus different models have different ways to capture the probability density function of the data without being able to do the explicit density estimation. The figure 4.2 shows the classification of these generative models based on that.

4.2.1. Variational Autoencoders (VAE)

Vanilla autoencoder maps the corresponding input to its encoded vector only. It is not possible to generate new samples with a vanilla autoencoder. To be able to generate similar images with some variabliity we need to learn the probability distribution of the training data. VAE is a probabilistic graphical model. It learns a low-dimensional latent representation of the training data called as latent variables. These latent variables z, store the useful information about the input data x, which is related to be able to generate correct reconstructed samples.
When doing inference about the latent variables $p(z|x)$, we use the posterior $p(z|x) = p(x,z)/p(x)$. Most of the times, the denominator is intractable in a high dimensional space because of which VAE use approximate inference of the posterior. In approximate inference we start with a family of distributions over latent variable and optimize the parameters of such that we end up with a distribution as close as possible to the posterior with the help of kullback leibler (KL) divergence. However, we have no knowledge of the posterior thus we use evidence lower bound.
The figure 4.3 shows the general block diagram of a VAE. The architecture of a VAE is similar to

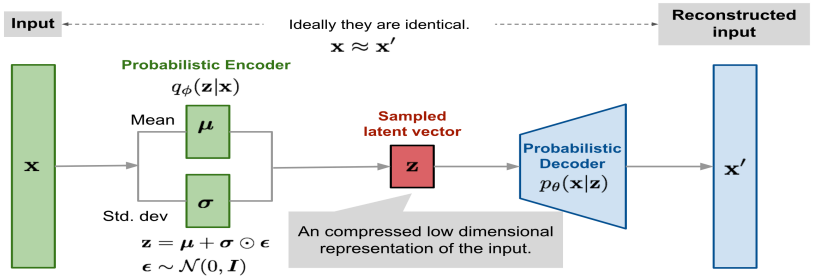


Figure 4.3: Block diagram of an variational autoencoder [16].

autoencoder except a few important differences such as, vanilla autoencoders use only pixel reconstructions without any f-divergence to measure the loss. VAE uses the reparameterization trick to allow the flow of gradients. To sample the data VAE uses decoder $z\ Q(Z|X)$ where $Z = \mu(x) + \sigma^2.e$ with $e\ N(0, I)$ compared to the autoencoder which requires encoder-decoder to generate an output.

4.2.2. Generative Adversarial Networks

Generative adversarial networks (GANs) [5] a structured probabilistic generative model consists of two parts: discriminator and generator. It is a counterfeit game between generator and discriminator where generator tries to fool the discriminator. These models dont rely on any explicit density estimation methods instead in GANs adopts the game theory approach to find the nash equilibrium between the two networks, Generator and Discriminator.
Both generator and discriminator are two neural networks given as G_ϕ and D_θ respectively. For the generator a pre-defined distribution p(z) with an input vector is defined and passed through a differentiable function $G_\theta : z- > X$. The discriminator is a normal classifier which takes input from a real data (X_{real}) and fake data (X_{fake}). The discriminator is trained to distinguish between these real and fake samples. The algorithm for GAN training is as follows: For epochs 1,...,N. First, sample real and fake data and from the discriminator D(x), where the aim of the discriminator is to learn the difference between real and fake samples. Then, sample a batch of images from generator G(z), by keeping the labels of these generated samples same as real data samples. As we want to fool the discriminator. The aim of the generator is to generate samples as similar as possible to the real data samples. Figure **??** shows a simple block diagram of a GAN.

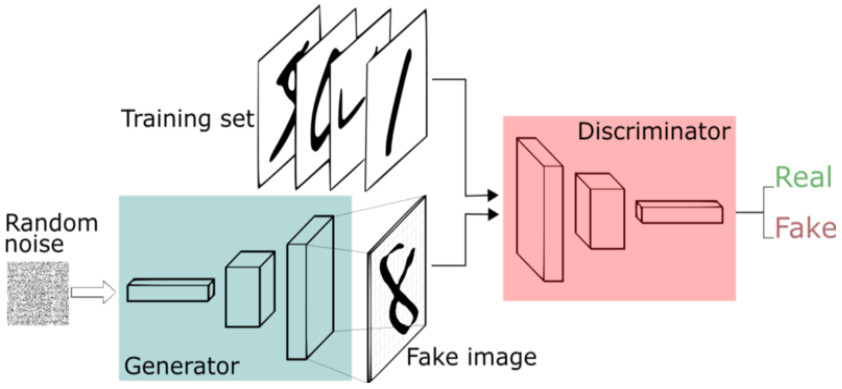


Figure 4.4: Block diagram of a vanilla Generative Adversarial Network [5]

4.2.3. Normalizing flows

Variational autoencoders can learn the feature representation however they have intractable marginal likelihoods, whereas autoregressive models provide the tractable likelihoods but no direct mechanism for learning features. Normalizing flows tries to combine the best of both worlds by providing a way for the exact likelihood estimation and feature learning with the help of change of variables formula.

Change of variables

In normalizing flow, we map the simple distribution to the complex one with the help of invertible transformations. To be able to do that we make we of change of variables theorem.Given a random variable z with known probability density function $z \sim \pi(z)$, we would transform it into a different random variable with the 1 to 1 invertible mapping function $x = f(z)$. The function f is invertible, so $z = f^{-1}(x)$. However the problems it creates is that, how to infer the unknown probability density function of the new variable, $p(x)$.

$$\int p(x)dx = \int \pi(z)dz = 1 \tag{4.1}$$

Definition of probability distribution.

$$p(x) = \pi(z) \mid dz/dx \mid \ = \pi(f^{-1}(x)) \mid df^{-1}/dx \mid \tag{4.2}$$

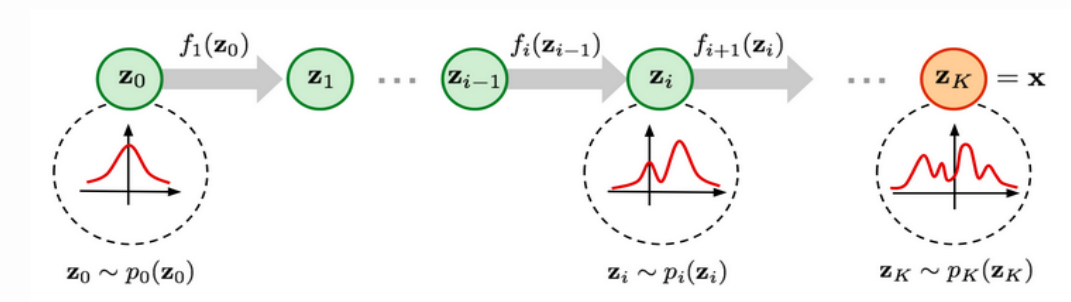


Figure 4.5: Transforming a simple distribution $p_0(z_0) to a complex one step by step using normalizing flow$ [15].

$$p(x) = \pi(f^{-1}(x)) \mid (f^{-1})'(x) \mid \tag{4.3}$$

By definition, the integral $\int \pi(z)dz$ is the sum of an infinite number of rectangles of infinitesimal width $\Delta(z)$. The height of such a rectangle at position z is the value of the density function $\pi(z)$. after substituting the variable, $z = f^{-1}(x)$ yields $\frac{\Delta z}{\Delta x} = (f^{-1}(x))'$ and $\Delta z = (f - 1(x))'\Delta x$. Here $\mid (f^{-1}(x))' \mid$ indicates the ratio between the area of rectangles defined in two different coordinate of variables z and x respectively. In a multivariate case:

$$z\ \pi(z), x = f(z), z = f^{-1}(x) \tag{4.4}$$

$$p(x) = \pi(z) \mid \frac{dz}{dx} \mid = \pi(f^{-1}(x)) \mid det \frac{df^{-1}}{dx} \mid \tag{4.5}$$

here, $det\frac{df}{dz}$ is the Jacobian of the function f, where Jacobian matrix is the matrix of the first-order partial derivatives of a function mapping f, which maps the n-dimensional input vector x to a m-dimensional output vector.
Normalizing flow transforms a simple distribution into a complex one by applying a sequence of invertible transformation functions. We repeatedly change replace the variable through this chain of transformations and eventually obtain a probability distribution of final target variable
We know the relation between each pair of consecutive variables given such a chain of probability density functions.

$$x = z_k = f_k * f_{k-1} * ... * f_1(z_0)$$

Using change of variables we can say that,

$$\log p_x = \log \pi_k(z_k) = \log \pi_{k-1} - \log \mid det \frac{df_k}{dz_{k-1}} \mid \tag{4.6}$$

Thus we can step by step back-trace the initial distribution by expanding the equation of the output x from figure 4.5 [15] and give the $\log p(x)$ as:

$$\log p(x) = \log \pi_0(z_0) - \sum_{i=1}^{K} \log \mid det \frac{df_i}{dz_{i-1}} \mid \tag{4.7}$$

The path traversed by random variables $z_i = f_i(z_{i-1})$ is the flow. Full chain formed by successive distributions π_i is called normalizing flow. The transformation function f should satisfy the follow specific structural requirements such as: The input and output dimnesions must be same, transfomation must be invertible and the computing the determinant of jacobian needs to be efficient. [15].

Bibliography

[1] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[2] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[3] A. Dertat. Applied deep learning - part 4: Convolutional neural networks., 2017. URL `https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2`.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] A. karpathy. Cs231n convolutional neural networks for visual recognition, Jun 2017. URL `http://cs231n.github.io/neural-networks-1/`.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[9] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[11] C. Nicholson. A beginner's guide to neural networks and deep learning, 2019. URL `https://skymind.com/wiki/neural-network`.

[12] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[13] Patel. S. and Pingel. J. Introduction to deep learning: What are convolutional neural networks?, 2017. URL `https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html`.

[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[15] L. Weng. Flow based deep generative models, 2018. URL `https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html`.

[16] L. Weng. From autoencoder to beta-vae, 2018. URL https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html.

[17] Matthew D Zeiler, M Ranzato, Rajat Monga, Min Mao, Kun Yang, Quoc Viet Le, Patrick Nguyen, Alan Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521. IEEE, 2013.