Go Deep or Go Home?

Remco den Heijer[1], Tom Viering[2], Yuko Kato[3], Taylan Turan[4], Ziqi Wang[5], Marco Loog[6], and David Tax[7]

[1, 2, 3, 4, 5, 6, 7]Electrical Engineering, Mathematics and Computer Sciences, TU Delft

Abstract

Does a convolutional neural network (CNN) always have to be deep to learn a task? This is an important question as deeper networks are generally harder to train. We trained shallow and deep CNNs and evaluated their performance on simple regression tasks, such as computing the mean pixel value of an image. For these simple tasks we show that going deeper does not guarantee an improvement in performance.

1 Introduction

A convolutional neural network (CNN) is a category of neural networks in deep learning. CNNs are different from standard artificial neural networks because they use additional convolutional layers. They are employed for all kinds of computer vision tasks: image classification [1], object detection [2], object tracking [3], age estimation [4, 5] and human pose estimation [6].

The architecture of CNNs has been studied extensively [7]. Deep CNNs usually show a significant improvement in performance [7,8] as opposed to shallow networks. The trade-off is that deeper networks are harder to train and require more training data [9]. Thus, whether CNNs *always* have to be deep is an important problem.

Some papers in the literature study this problem directly, but only for classification tasks. Urban et al. [10] concludes that deeper networks are better for image classification tasks. Le et al. [11] states that "deep models have not yet proven to be more effective than shallow models for text classification tasks". A recent survey paper on CNNs [7] states that "the depth is an essential dimension in regulating learning capacity of the networks". We want to know if this is true for simple regression tasks.

In this paper, we empirically evaluated CNNs with an increasing number of convolutional layers on three different regression tasks: mapping an input image to the mean, median and standard deviation of its pixel values. We choose these tasks because we can compute the actual mean, median and standard deviation for each image easily. The central question we tried to answer is: do CNNs have to be deep to learn regression tasks?

For these specific task, the answer is no. Going deeper did not improve the performance on these specific tasks. In fact, for the standard deviation task deeper networks performed worse.

2 Related work

In this section we discuss the relevant literature. First, we will discuss two papers that compared deep and shallow models for classification tasks. Then, we will briefly discuss a paper that evaluated a deep CNN on a task similar to our tasks. Finally, we will reinforce why studying deep CNNs on regression problems is important.

Gregor et al. [10] trained networks on an image classification task. They formed an ensemble of state-of-the-art deep teacher networks and used those to label images. These labeled images were used to train shallow student networks. The shallow student networks were not as accurate as their teacher networks. Le et al. [11] compared deep and shallow-and-wide networks on five standard text classification and sentiment analysis tasks. On two out of five tasks, shallow-and-wide networks had a higher accuracy, although the difference was small (0.2%). The performance was equal for one tasks. For the other two tasks, the deeper network performed better, but again slightly (0.3%). These observations led the authors to conclude that deeper CNNs do not necessarily perform better than shallow ones for text classifications and sentiment analysis tasks. A limitation of this paper is that they derived their deep networks from networks created for image classification tasks. Deep CNNs created specifically for text classification might be better than shallow ones.

Liu et al. [12] used a state-of-the-art network (VGG [13]) in the image classification domain, to train on a regression task using MNIST dataset. The authors converted class labels into floats by sampling from normal distributions. Predicting those numbers was the regression task. Although they report that their method was successful, they only used one specific dataset and architecture and did not compare their results with a shallow network.

A recent analysis of regression problems using CNNs [14] states that there is a lack of "systematic evaluation of deep learning advances in regression" and "an over abundance of papers based on deep learning". The authors say that this "highlights again the importance of serious comparative empirical studies to discern which are the key blocks in deep

regression." We hope our research contributes to the understanding what these key blocks are.

3 Methodology

In this section we describe the experimental setup during the research. We first show the architecture of the networks and explain the tasks in more detail. Thereafter, we describe the dataset and hyperparameters used during training. Finally, we describe the loss function we tried to minimize.

3.1 Networks

We created several networks with different number of layers. We first created a base network [15] of which we derived all other networks. We will refer to this network as 'Base'. In Figure 1 a diagram of 'Base' is shown.

We used 28x28 grayscale images as the input (see Section 3.2 for details about the dataset). The first layer was a convolutional layer with a kernel size of 5x5 and 6 channels. We did not use padding, hence the slight decrease in width and height of the intermediate feature maps. This convolutional layer was followed by a max pooling layer that reduced the dimensions by half. These two layers are repeated once more. The only difference is that the second convolutional layer uses 12 channels instead of 6. We have now come the layer that was different between different networks. This layer was stacked upon itself multiple times. In total we created 7 different networks where this layer was stacked 0, 1, 2, 3, 4, 5, and 10 times, respectively. We will refer to these networks with their number of extra layers. We used padding to prevent reduction in dimensions. After all these layers we used three fully connected layers to get to a single output neuron. We choose the Rectified Linear Unit (ReLU) as activation function for the hidden layers.

In addition to the CNNs we created a network without convolutional layers. This network consisted of an input layer of 784 neurons followed by the last three linear layers shown in 1. We refer to this network as None.

3.2 Tasks

In order to evaluate the networks, we created three simple tasks. Simple, meaning it was easy to compute the true output value of an input. Each task was a mapping $f : \mathbb{R}^2 \mapsto \mathbb{R}$. We define the mean (1), median (2) and standard deviation (3) tasks as follows:

$$\bar{x} : x \mapsto \frac{1}{N} \sum_{i=0}^{k-1} \sum_{j=0}^{l-1} x_{ij} \tag{1}$$

$$\text{med} : x \mapsto \text{median}(x) \tag{2}$$

$$s_x : x \mapsto \sqrt{\frac{\sum_{i=0}^{k-1} \sum_{j=0}^{l-1} (x_{ij} - \bar{x})^2}{N-1}} \tag{3}$$

where N is the number of pixels, k the width and l the height of a 2D array x.

3.3 Dataset

We used Zalando's FashionMNIST [16] dataset for the three tasks. This dataset consists of 60,000 training and 10,000 test samples. Let x denote a sample from dataset X. Of the training samples, we used 55,000 for training and 5,000 for validation. Each sample is a 28x28 grayscale image in ten different clothing categories. Since we focused on regression tasks, we ignored the class labels. Instead, we computed $\bar{x}$, med and s_x for each sample. We call these values targets and we denote a target with y. Let Y be the set of all targets. We will refer to the datasets used for tasks $\bar{x}$, med and s_x as MeanFMNIST, MedianFMNIST and StdFMNIST respectively.

Distribution of targets

It is important to know the distribution of the dataset targets because an unequal distribution can affect the performance of a network. This might give an impression of a very high performance, while instead there is a bias in the dataset. Figure 2 shows histogram plots of the distribution of the targets.

From the histograms it is clear that the MedianMNIST was not equally distributed. Almost half of the samples have a median of 0. The MeanFMNIST skews to the left, while the StdFMNIST skews to the right.

3.4 Hyperparameters

Most of the hyperparameters were the same between different networks. We used a batch size of 64 and a random seed of 42, unless otherwise stated. For optimization we used the Adam algorithm [17]. We fine-tuned the learning rate and number of epochs.

The learning rate is a crucial hyperparameter, since if it is too large the average loss will increase [18]. If it is too low, it might take a long time before the network converges. We fine-tuned the learning rate for each network on each dataset, using Pytorch Lightning built-in Learning Rate Finder. This algorithm trains the network for a small run, increasing the learning rate after each processed batch and notes the loss. The result is a learning rate vs. loss plot. An example is shown in Figure 3. From this plot a point with the sharpest downward slope is chosen as the learning rate. See [19, 20] for full details. We used the default parameters: start at 10^{-8}, stop at 1, try 100 learning rates and increment the learning rate exponentially.

Knowing for how many epochs a model should be trained is important to prevent over- and underfitting. So, we trained networks Base and 10 on the MeanFMNIST for 130 epochs. We only choose two networks because training for a large number of epochs is time consuming.

3.5 Loss function

For the loss function we used the mean squared error (MSE):

$$\text{MSE}(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where $\hat{Y}$ is the set of predicted targets and n the number of samples. This is a standard loss function for regression problems and we find it easy to interpret.

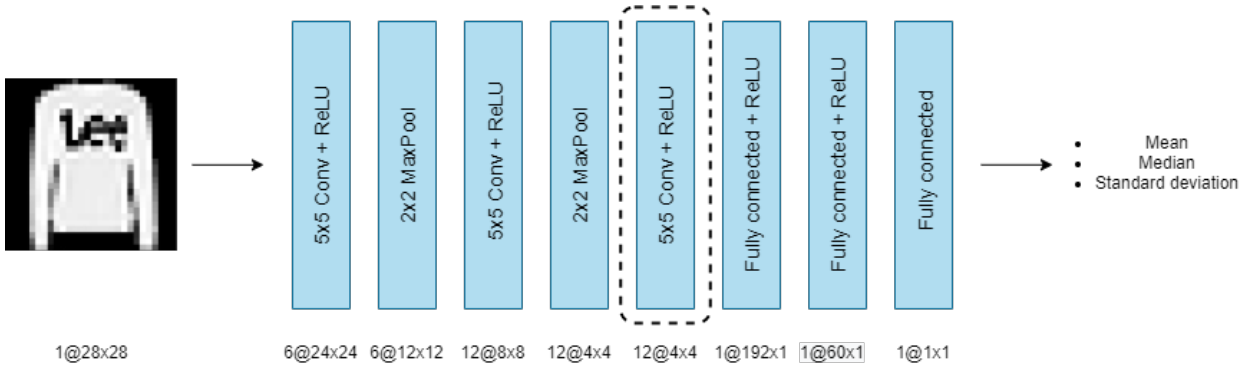

Figure 1: Diagram of base network. The dimensions are written at the bottom; for example, 1@28x28 means 1 channel and a width and height of 28. The layer enclosed with the dotted line, was repeated 0, 1, 2, 3, 4, 5 and 10 times to create different networks. The dimensions written in the rectangles are the dimensions of the kernel. We used Rectified Linear Unit (ReLU) as the activation function for the hidden layers. All networks outputted a single number.

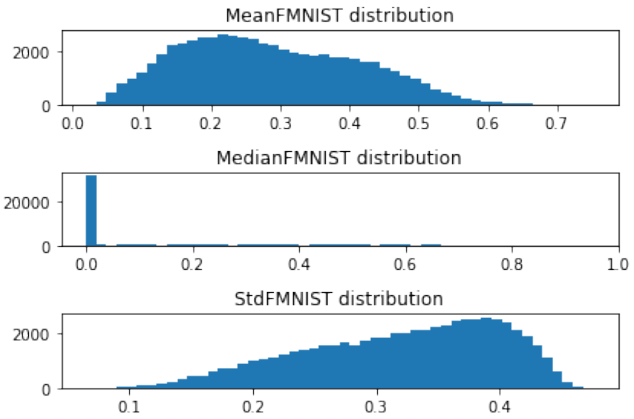

Figure 2: Distribution of targets of MeanFMNIST, MedianFMNIST and StdFMNIST datasets. Width of each bin is 0.02.

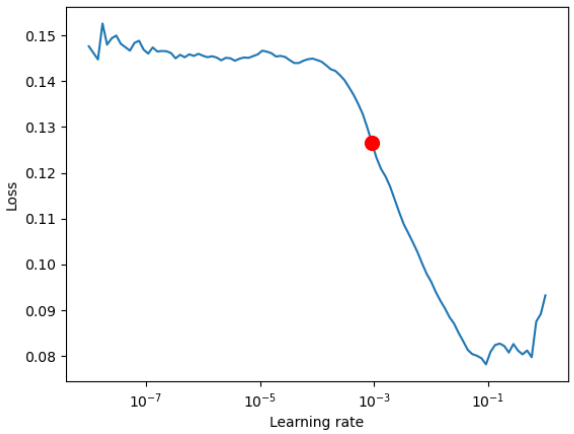

Figure 3: Example of loss vs. learning rate plot. The red dot indicates the chosen learning rate. Using network 2 with a MeanFMNIST dataset.

4 Results

The results are shown in this section. For each experiment we first describe what we have done, then show the results and finally discuss important observations.

4.1 Experiment 1. Tuning learning rates

Each time we trained and tested a network we first tuned the learning rate. We used the Pytorch Lightning Learning Rate Finder (see Section 3.3) to find the learning rate. In Experiment 3 we trained and tested each network on each dataset 10 times. In Table 1 we show the average of the learning rates per network and dataset.

	Learning rates		
Model	MeanFMNIST	MedianFMNIST	StdFMNIST
None	$2.0 \cdot 10^{-4}$	$2.5 \cdot 10^{-4}$	$2.2 \cdot 10^{-4}$
Base	$4.1 \cdot 10^{-2}$	$1.7 \cdot 10^{-1}$	$9.2 \cdot 10^{-4}$
1	$7.9 \cdot 10^{-2}$	$5.8 \cdot 10^{-4}$	$1.2 \cdot 10^{-3}$
2	$5.8 \cdot 10^{-2}$	$1.5 \cdot 10^{-3}$	$1.1 \cdot 10^{-3}$
3	$1.1 \cdot 10^{-2}$	$1.6 \cdot 10^{-2}$	$2.0 \cdot 10^{-3}$
4	$6.0 \cdot 10^{-3}$	$5.2 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$
5	$2.2 \cdot 10^{-3}$	$2.3 \cdot 10^{-4}$	$2.2 \cdot 10^{-3}$
10	$2.1 \cdot 10^{-3}$	$8.5 \cdot 10^{-4}$	$8.6 \cdot 10^{-2}$

Table 1: Fine-tuned learning rates for models on different datasets.

4.2 Experiment 2. Baseline performance

We tested three different baselines: zero, mean and median. We expect the mean baseline to have the lowest MSE, as Tom Viering stated during one of our meetings. Looking at the distribution of MedianFMNIST we also expect that the zero baseline works well for this dataset. In this experiment we only used test samples of the datasets. We first computed the mean and median of test targets for each dataset. For the zero baseline we just took 0. Then, for each baseline and dataset, we computed the MSE of the targets and the baseline. The

results are shown in Table 2, including the standard deviation (σ) of the targets. The best performing baseline (lowest loss) is shown as bold.

	MSE			
Dataset	Zero	Mean	Median	σ
MeanFMNIST	0.0981	**0.0158**	0.0160	0.1257
MedianFMNIST	0.1119	**0.0711**	0.1088	0.2667
StdFMNIST	0.1085	**0.0061**	0.0062	0.0781

Table 2: Test loss of test samples for different baselines.

From the results it is clear that the mean of the targets of the training samples is the best performing baseline. This verifies the claim by Tom Viering. We note that the median is only slightly worse than the mean for MeanFMNIST and StdFMNIST datasets. Surprisingly, the zero baseline for MedianFMNIST is not quite as good as the mean baseline.

4.3 Experiment 3. How many epochs?

We trained networks Base and 10 for 130 epochs on the MeanFMNIST dataset. The batch size was 64 and the learning rate according to Table 1. We logged the training and validation loss on every epoch. So, the training loss was the MSE over 55,000 samples and the validation loss the MSE over 5,000 samples. Figure 4 shows the results. The blue lines indicate the training loss, and the orange line shows the validation loss.

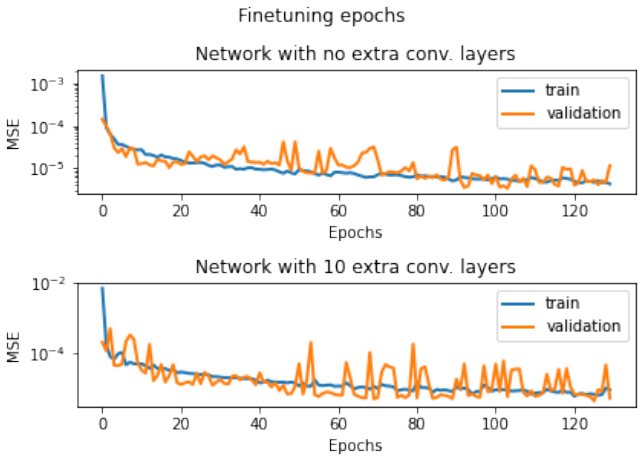

Figure 4: Training loss of two networks as function of number of epochs: zero extra layers and ten extra layers.

From Figure 4 it is clear that both the training and validation loss are decreasing. We do not observe a difference between the two networks; both end with approximately the same training and validation loss. The validation curves fluctuate but stays close to the training curves. This suggests we are not overfitting after 130 epochs, so we could still improve the performance. To keep training duration manageable, we ran all our subsequent experiments for 30 epochs. Also, after training for 30 epochs the training and validation losses are below the baseline (0.0158).

4.4 Experiment 4. Is deeper better?

In this experiment evaluated the performance of networks Base, 1, 2, 3, 4, 5 and 10 on each task. We trained and tested each network on each task ten times. In Figure 5 the average test losses are shown. The errors bars indicate the standard deviation.

When we compare the baseline losses with the test losses we see that all networks are able to learn all tasks much better than the baseline. For MeanFMNIST the best baseline has an MSE of $1.58 \cdot 10^{-2}$, while the networks all have an average test loss in the order of 10^{-5}. The average test losses of MedianFMNIST are in the range of $[8.73 \cdot 10^{-4}, 6.67 \cdot 10^{-3}]$, which is closer to the baseline, $7.11 \cdot 10^{-2}$, than the MeanFMNIST test losses. For StdFMNIST the test losses are in the order of 10^{-5}, which is significantly lower than the best baseline $6.1 \cdot 10^{-3}$. From these observations we conclude that the networks have the capacity to learn the tasks.

From the results of MeanFMNIST and MedianFMNIST we conclude that deeper does not necessarily mean better.

Looking at the results of StdFMNIST we see that networks Base, 1, 2, 3 have significantly lower test losses than network 4, 5 and 10. A possible explanation for this might be that we stopped training early. In Experiment 3 we concluded that 30 epochs was good enough, from experimenting with the MeanFMNIST. If we trained networks 4, 5 and 10 for more epochs, we might get a loss comparable to the other networks or even better. To further investigate this observation we should run Experiment 3 using the StdFMNIST dataset and train networks 4, 5 and 10 for more epochs. To get closer to the real average test loss we can also increase the number of training and test cycles.

4.5 Experiment 5. Are convolutional layers required?

We trained and tested network None (without convolutional layers) on each task. We did this ten times per task using different initial weights. The results are shown in Figure 6.

It is immediately clear that this network is able to learn the s_x task better than the other two tasks. The test losses are also listed in Table 3.

We can also compare these results with networks that have a convolutional layer. Compare the results from Figure 6 with the losses of Base network from Figure 5. We summarize the results in Table 3.

We observe that adding convolutional layers is beneficial when training on MeanFMNIST and MedianFMNIST. For StdFMNIST there is no significant improvement.

5 Discussion

The most important finding from the results is that going deeper does not necessarily improve performance for regression tasks. In the case of the s_x task, the performance decreased for deeper networks. These findings are limited by early stopping of training. We did this in order to keep training times manageable. To really know if this is true, we should fine-tune the number of epochs.

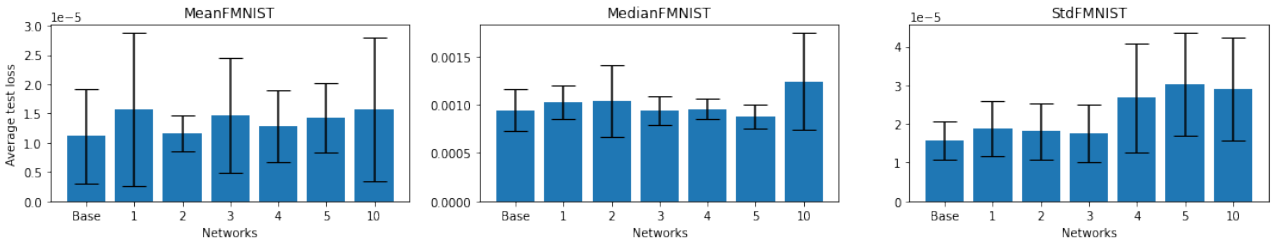

Figure 5: Average test loss for 10 training and test cycles. The errors bars show the standard deviation. From the results for MedianFMNIST network 10 we removed one outlier.

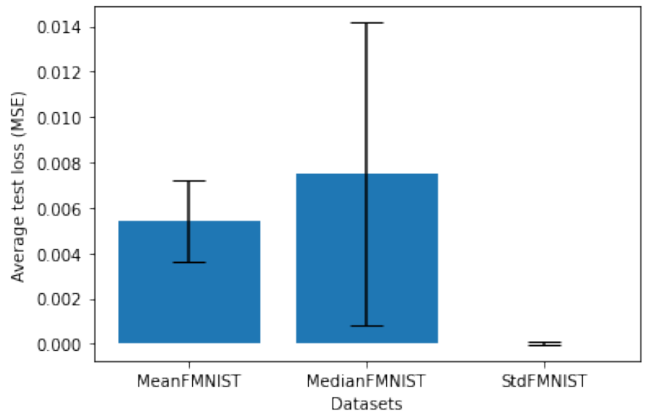

Figure 6: Average test loss over 10 training and test cycles for network None. This network does not have any convolutional layers.

	MSE	
Dataset	None	Base
MeanFMNIST	$1.21 \cdot 10^{-3}$	$1.12 \cdot 10^{-5}$
MedianFMNIST	$7.49 \cdot 10^{-3}$	$9.43 \cdot 10^{-4}$
StdFMNIST	$1.25 \cdot 10^{-5}$	$1.57 \cdot 10^{-5}$

Table 3: Comparing test losses between a network with (Base) and without (None) convolutional layers. There is no clear benefit of using convolutional layers for s_x task. For tasks $\bar{x}$ and med the test loss with convolutional layers is significantly lower.

Another important result is that for the s_x task we might not even need convolutional layers. For the other two tasks using convolutional layers showed a significant improvement in performance. We suspect this is because there is not spatial aspect in our tasks. The networks do not have to learn features from the images, which is the case in for example image classification.

During experimentation we used Adam as an optimizer. We were interested if Stochastic Gradient Descent (SGD) would be a better optimizer. We overfitted networks Base and 10 on a single batch of StdFMNIST and logged the training loss. We trained for 1,000 epochs, and did this separately for Adam and SGD. The training curves for Adam showed discontinuous jumps but was able to achieve a much lower training loss than SGD. The training curves for SGD were smooth. We are unsure what caused this behavior. This result might explain the high standard deviation in the test losses from Figure 5.

6 Conclusion

In this paper we presented the results of training shallow and deep CNNs. Our main objective was to find out whether deep networks are required for regression tasks. We evaluated the networks on three different tasks: mapping an image to the mean, median and standard deviation of pixel values. We conclude that deep CNNs are not a requirement to achieve good performance on these regression tasks. For the standard deviation task, deeper networks showed worse test results. However, this observed result could be attributed to training for a fixed number of epochs. We should further investigate if this still holds true when we train deeper networks for more epochs. Our research was also limited to only three simple regression tasks. In the future we would like to investigate whether CNNs have to be deep for more complex regression tasks, such as age or human pose estimation.

7 Responsible Research

The result presented in this paper do not have any ethical implications because the studied tasks have no value in the real world. Instead, we will reflect on the reproducibility of our experiments. We will reflect on the six recommendations of Yale Law School Roundtable on reproducible research [21].

The first recommendation is that all source-code should be public. The code is hosted on GitHub and can be accessed through `https://github.com/Avonite/context-project`. The second recommendation is to assign a unique id to released code. We do not expect our code to change, so we did not follow this recommendation. The datasets are included in the version control, so all experiments are fully reproducible. This is not the case for the notebooks we used. If the datasets change in the future, our results will probably also change slightly. The third recommendation is to describe computing environment and software version used in the publication. All scripts ran on a HP ZBook Studio x360 G5 with a i7-8750H CPU @ 2.20GHz. The operating system was Windows 10 Education 64-bit. All software with versions can be found in requirements.txt in the repository. We added an MIT license as per the fourth recommendation. This means that anyone can freely reuse and experiment with

our code. This paper will be publicly available in the TU Delft repository, so we fulfil the fifth recommendation. All our scripts are written in Python files, i.e. plaintext, so these should still be readable in the foreseeable future. This was the sixth recommendation.

References

[1] Syed Muhammad Anwar, Muhammad Majid, Adnan Qayyum, Muhammad Awais, Majdi Alnowami, and Muhammad Khurram Khan. Medical image analysis using convolutional neural networks: a review. *Journal of medical systems*, 42(11):1–13, 2018.

[2] Kai Kang, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. Object detection from video tubelets with convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 817–825, 2016.

[3] Qiao Liu, Xiaohuan Lu, Zhenyu He, Chunkai Zhang, and Wen-Sheng Chen. Deep convolutional neural networks for thermal infrared object tracking. *Knowledge-Based Systems*, 134:189–198, 2017.

[4] Fadi Dornaika, Salah Eddine Bekhouche, and Ignacio Arganda-Carreras. Robust regression with deep cnns for facial age estimation: An empirical study. *Expert Systems with Applications*, 141:112942, 2020.

[5] Safwan S Halabi, Luciano M Prevedello, Jayashree Kalpathy-Cramer, Artem B Mamonov, Alexander Bilbily, Mark Cicero, Ian Pan, Lucas Araújo Pereira, Rafael Teixeira Sousa, Nitamar Abdala, et al. The rsna pediatric bone age machine learning challenge. *Radiology*, 290(2):498–503, 2019.

[6] Denis Tome, Chris Russell, and Lourdes Agapito. Lifting from the deep: Convolutional 3d pose estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2500–2509, 2017.

[7] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, 2020.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[9] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. *arXiv preprint arXiv:1507.06228*, 2015.

[10] Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional? *arXiv preprint arXiv:1603.05691*, 2016.

[11] Hoa T Le, Christophe Cerisara, and Alexandre Denis. Do convolutional networks need to be deep for text classification? In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] Chang Liu, Ziheng Wang, Su Wu, Shaozhi Wu, and Kai Xiao. Regression task on big data with convolutional neural network. In Aboul Ella Hassanien, Ahmad Taher Azar, Tarek Gaber, Roheet Bhatnagar, and Mohamed F. Tolba, editors, *The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2019)*, pages 52–58, Cham, 2020. Springer International Publishing.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] Stéphane Lathuilière, Pablo Mesejo, Xavier Alameda-Pineda, and Radu Horaud. A comprehensive analysis of deep regression. *IEEE transactions on pattern analysis and machine intelligence*, 42(9):2065–2081, 2019.

[15] Michael Li. Let's build a fashion-mnist cnn, pytorch style, Jan 2020.

[16] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

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

[18] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[19] Falcon W.A. Learning rate finder, 2019.

[20] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[21] Victoria C Stodden. Reproducible research: Addressing the need for data and code sharing in computational science. 2010.