A LOOK INTO ROBUSTNESS

Mikkel Odgaard - s174371

ABSTRACT

The accuracy of neural networks keeps rising and they accomplish more and more impressive tasks. Purely looking at accuracy might be fine for lab purposes, but in the real world there are more demands. Modern neural networks are also required to be fair and unbiased. Fairness and bias are hard to quantify, so this paper shifts the focus from fairness to robustness under the assumption that a robust network is more likely to be fair. This paper investigates how various components of neural networks, such as dropout or batch normalization, affects a network's robustness and the paper proposes a simple metric called $\Delta accuracy$, that serves as a measure of robustness. This metric is made by comparing the accuracy of two MNIST datasets, the original MNIST and AlignMNIST [1]. With the proposed metric it is seen that component, such as batch normalization, does not only increase a network's accuracy, but also make it more robust to corrupted or modified data.

1. INTRODUCTION

As neural networks accomplishes more and more impressive tasks, researchers are starting to set new demands. A new field of machine learning has emerged working with fairness and bias. High accuracy is not the only sought after attribute. Fair and unbiased decisions are an important feature of modern neural networks. Fairness and bias is hard to quantify and is subject to interpretation. Companies such as IBM, Facebook and Google have made efforts in trying to reduce algorithmic bias - but there has not been set an international standard. This paper proposes a different approach to fairness with the proposition that a robust network is more likely to be fair and unbiased. A robust network is less susceptible small changes, so that changing the color, gender or beliefs of a person, should not significantly change the prediction of the network. While this proposition seemingly solves the interpretation issue of fairness, it poses another question. How is robustness measured?

Robustness is also an emerging field of machine learning and is easier to quantify. Robustness is used to describe the overall state of the network - not on an individual case by case basic. To have a fair network, it must not discriminate, even on a single prediction. One cannot say that a network is fair, if it is unfair in 1% of its predictions. However, one can say that a network is robust, if it is robust in 99% of the cases. Measuring robustness therefore shift the focus on a single prediction to all predictions. This allows us to easier investigate the problem.

So how is robustness measured? Many robustness metrics look at adversarial robustness [2] [3], a robustness that is relevant when someone is deliberately trying to fool the network. This is not the case with fairness and bias, where a general robustness is sought after. Instead, this paper will focus on the differences in accuracy between two different datasets and use this as a measure of robustness.

2. METHODOLOGY

In order to measure robustness, the network must be exposed to two different datasets. One, which is the expected input that the network is trained on, and a second one with new unseen, slightly transformed, data. This paper uses two different datasets, the original MNIST dataset and AlignMNIST [1], a dataset containing random transformations (fig. 1) from the original dataset. The transformations will be refereed to as 'corrupted', to keep in line with current literature. The network is evaluated on both the MNIST dataset and the AlignMNIST dataset. The difference in accuracy, called $\Delta accuracy$ is used as a simple robustness metric.

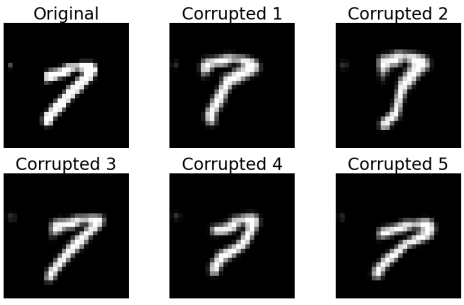

Fig. 1. An example from the original dataset and 5 AlignMNIST transformations of it

The accuracy from the AlignMNIST dataset, denoted $\hat{x}$, is calculated as the mean of the accuracies, y_n, across the first

5 transformations of the AlignMNIST. AlignMNIST consists of 75 transformation, but the accuracy of the network was observed to be stable across all transformation, so the average of 5 of them is taken to decrease the run time.

$$\hat{x} = \frac{1}{5}\sum_{n=1}^{5} y_n \tag{1}$$

$\Delta accuracy$ is then obtained by taking the original MNIST accuracy, x, and subtracting it from the transformed AlignMNIST accuracy, $\hat{x}$. This metric has a few downfalls, which will be discussed in section 4.

$$\Delta accuracy = \hat{x} - x \tag{2}$$

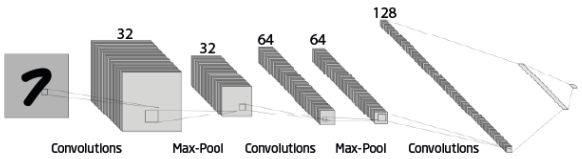

Fig. 2. Modern version of LeNet-5

The architecture can be seen on figure 2 and achieves an accuracy of 99.76%. It follows the LeNet-5 design [4] with some modern improvements:

- Two stacked 3x3 convolutional layers that replaces the 5x5 filter
- 5x5 max-pooling in the pooling layers
- ReLU activation replaces sigmoid
- Added batch normalization
- Added dropout
- Increased number of feature maps (channels)

However, this paper will not just be testing a single network. This would give a measure of robustness for this specific architecture, but it will not reveal anything meaningful. Instead, the network is gonna be stripped of key features, such as removing dropout or batch normalization, and these modified networks will then be evaluated and a $\Delta accuracy$ will be obtained. Doing this for different iterations of the network, each missing a key component, should reveal differences in network behaviour and be able to reveal if some component are vital, not only for accuracy, but also for robustness. Five different iterations is tested. Each iteration is given a name, as indicates by the parentheses.

1. Removing the decaying learning rate (StepLR)
2. Removing batch normalization (BatchNorm)
3. Removing dropout (Dropout)
4. Replacing pooling with a convolution, as a way to have learnable pooling layers (ConvPool)
5. Replacing max-pooling with average-pooling (AveragePool)

These 5 different iterations will be evaluated and their differences will be the basic of the discussion.

3. RESULTS

Figure 3 shows two things: All versions of the network are able to maintain a high accuracy on the original MNIST dataset. Secondly, there is a significant drop in accuracy between the two datasets. These two observations indicate two separate things.

Firstly, each network is still able to perform without the key component, albeit their ability is slightly hindered (mostly so without batch normalization). This is important, as if each network had been fundamentally broken after removing a key component, the impact of the results would be questioned. The robustness of a network that only gets 50% of its predictions right, undermines the fundamentals of robustness. A network that continuously makes errors cannot be tested as robust, as unexpected inputs cannot be produced, if the original inputs cannot be interpreted. Secondly, the significant drop proves that either two statements must be true. Either the AlignMNIST dataset is harder, thereby achieving a lower accuracy score, or the network fails to apply its learned knowledge on new data. The first statement cannot be answered without rigours testing of the AlignMNIST dataset, something that falls outside the scope of this paper. The second statement would indicate some sort of robustness or rather lack of it. It is not this paper's aim to determine which statement is true, therefore, for the sake of simplicity, the second is assumed to be true, as it motivates the robustness evaluation.

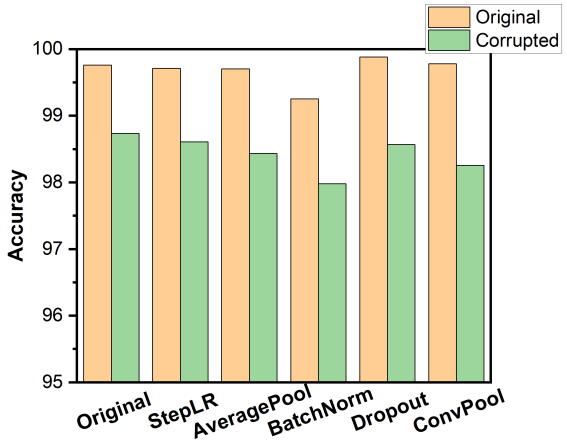

Fig. 3. Original vs. transformed (corrupted) dataset accuracy

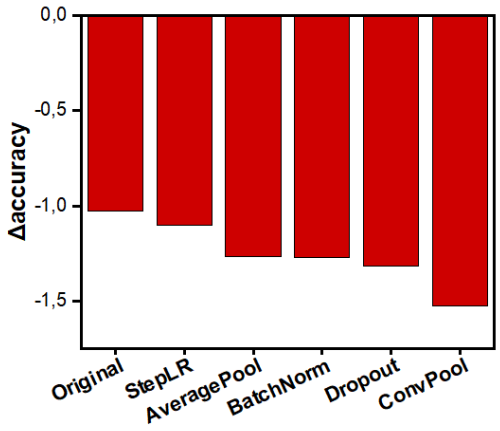

Fig. 4. The $\Delta accuracy$ of each version of the network

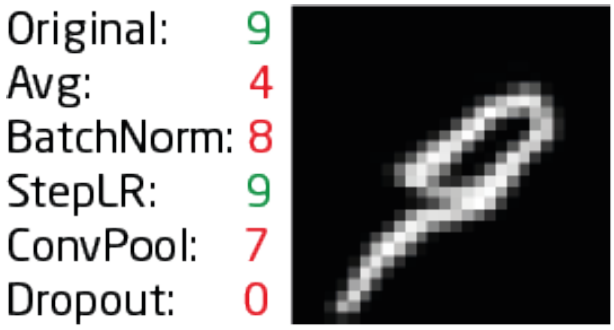

Fig. 5. Example from AlignMNIST with each network version's prediction. The correctness of the prediction is indicated by the color of the answer

One is able to dismiss two versions of the network from the robustness discussion by looking at figure 3 and 4, StepLR and ConvPool. Firstly, removing the stepped learning rate showed a very little drop in accuracy between the two datasets, which indicates that having a stepped learning rate does not increase the robustness of the network. Secondly, by using a convolutional layer instead pooling layer one neglects robustness. This is evident, as the ConvPool network has a marginally higher accuracy $(+0.02\%)$ than the original network, but suffered a large drop in accuracy when confronted with the AlignMNIST dataset. However, this does not indicate that ConvPool is a key component for robustness, as the nature of the component is having learnable parameters. These learnable parameters are able to obtain a better accuracy when presented by known data, but suffer when presented with new and transformed data. Using ConvPool instead of regular pooling strategies removes a key and unbiased calculation and introduces learnable bias into the system. In a way, it is an argument that standard pooling practices actually promotes robustness, by filtering out noise and allows the network to generalize better.

Looking further into the pooling argument, it is seen that placing max pooling with average pooling produces a higher $\Delta accuracy$, indicating that average pooling is better for robustness than max pooling. The difference between max- and average pooling for the original dataset is 0.06%, but the difference in accuracy for the AlignMNIST dataset is 0.28%. A small, but significant difference. Comparing the fundamentals of max- and average pooling, average pooling would usually come out on top when evaluating based on robustness. Fundamentally, max pooling is subject to noise, as it picks a single value out from its filter, whereas average pooling looks at the entire filter and calculates a value based on that. This very concept should strongly promote robustness, as it does not get influenced by an individual thing or pixel, but looks at the overall picture. This fits nicely into the analogy made in the introduction, where the difference between fairness and robustness was motivated.

4. DISCUSSION

A neglected thing, that was not initially brought up during section 3 is the original accuracy of the dropout version, i.e. the version of the network without dropout. It might be slightly hard to see on the figure, but the original accuracy reached was 99.88%. This is higher than the original network, that boasted an accuracy of 99.76%. This is a clear sign of overfitting and is one of the reason why the dropout version scored a high $\Delta accuracy$. It is clear why removing dropout leads to overfitting - as dropout partly tries to combat this problem. However, this is a much more serious observation than just an overfitted model. This kind of invalidity undermines the effect of the proposed metric and highlights one of its key flaws: A network can score a high $\Delta accuracy$ if its initial accuracy is high enough, as it is merely the difference between x and $\hat{x}$. In fact, the dropout network scores the 2nd highest AlignMNIST accuracy, $\hat{x}$, out of all the versions, falling just 0.04% short of the StepLR version. The StepLR version of the network, the version without a stepped learning rate, is shown, as evident on figure 4, to have the least impact on robustness, as one might also assume. Ultimately, it is hard to cement whether this particular result is fundamentally flawed, as overfitting on the original dataset might still increase the accuracy of the AlignMNIST dataset, as the transformed are derived from the original dataset. Conversely, it might be an interesting observation, as overfitting should negatively impact the performance when presented with new data - even if it resembles the original.

The results have been looked an quantitatively. Now it is time to dive deeper and look a single example and qualitatively try to truly understand the behaviour of the networks. A difficult example from the AlignMNIST dataset has been selected. As seen on figure 5, only two versions get this prediction right: The original network and the StepLR version. It is not par-

ticular insightful to why these two versions get it right, but it is interesting why, or rather how, they others got it wrong. What is shared by all the wrong predictions are a part truth. Average pooling predicts a 4, which closely resembles a 9, just without the top line. Batch normalization predicts an 8, closely resembling a 9, just with an extra half circle. ConvPool predicts a 7, closely resembling a 9, but without the connecting part in the middle. Dropout predicts a 0, perfectly resembling a top half of a 9. What is shared by all of this, is that the predictions are not wildly wrong. They get stuck on a part truth. The networks correctly identify most of the image, but fail draw in the last piece of information, that allows the prediction of the correct result. A robust network can not be stuck on an incorrect idea, just because it has seen something like it before. It is almost like these networks are immature and 'jump the gun' on their predictions. A robustness network can be seen as a old man, who takes his time and draws from experiences - and are thus able to generalize better. A network that is not robust, acts rapidly, guessing a 0 after only seeing the top part of the 9. Taking out key components of a network, such as batch normalization, changes the behaviour of the network. Not only by making it less accurate, but by making it less mature. These components prevents 'jump the gun'-predictions and makes the networks less susceptible for changes in the dataset.

Outside the scope of this paper lies another crucial observation. Some of these components or features, such as the decaying learning rate, also affect another important attribute of networks: The training time. Networks become more robust when exposed to more data, however more data also increases the complexity and the training time. So while some component of the neural network does not directly increase robustness, it may indirectly attribute to the cause by allowing more data to be processed faster. This means that bigger datasets can be used, more examples can be seen and the network will be able to generalize better.

5. CONCLUSION

While the differences where small, the results suggest that features such as batch normalization play a key role, not only in accuracy, but also in the robustness of the network. Although a simply experiment, and an even simpler definition of robustness, this project highlights the importance of robustness in modern networks. The two datasets are similar, in fact one is derived from the other, yet there is a significant drop in accuracy between the two. If modern neural networks are not robust and able to withstand small changes, they quickly lose value when applied to critical fields.
Neural networks are notorious for being black boxes, which is a key problem when evaluating bias, as their actions cannot be explained. But if their prediction can be trusted, by knowing it is fair and robust, neural networks can be applied in more critical fields. The hope for the future is that when evaluating a neural network, robustness is an important metric that stands side by side with accuracy, such that fairness can be achieved. Robustness does not promise fairness, but it is an important first step in the right direction.

6. GITHUB

All the code and data is made available at
github.com/mikkelfo/02456-Deep-Learning

7. REFERENCES

[1] Søren Hauberg, Oren Freifeld, Anders Boesen Lindbo Larsen, John W. Fisher III, and Lars Kai Hansen, "Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation," in *Proceedings of the 19th international Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016, vol. 51, pp. 342–350.

[2] Chirag Agarwal, Bo Dong, Dan Schonfeld, and Anthony Hoogs, "An explainable adversarial robustness metric for deep learning neural networks," 2018.

[3] Igor Buzhinsky, Arseny Nerinovsky, and Stavros Tripakis, "Metrics and methods for robustness evaluation of neural networks with generative models," 2020.

[4] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, 12 1998.