Increasing robustness and generalisation of CNNs

Harshitha Koppisetty
hksetty@yorku.ca
218798017

Abstract

Several Deep Learning techniques and Convolutional Neural Networks have been rapidly developing and have re However, there exist several malicious techniques that can cause a malfunction in these systems. Further, out of distribution data and noisy data are also not handled very well. In the recent years, a dedicated field of study has emerged where the objective is to increase the robustness of these techniques. In this paper, we will discuss some of the existing techniques and a new approach based on them.
Keywords- Deep Learning, CNN, robustness, generalization

1 Introduction

Convolutional Neural Networks (CNNs) were modeled after the human visual system with the objective of performing tasks like image classification and object detection. Since the emergence of AlexNet in 2012, a lot of rapid developments have been seen in this field. CNNs are achieving very high levels of accuracy (new human levels) for these tasks. But, CNNs have also proved to be overly brittle when faced with adversarial attacks or with noisy images. Several malicious attacks and techniques to lead the CNN to mis-classify images have been developed. In addition to this, CNNs also perform poorly when they encounter images of their target object in different orientations or in a different context (like the image of a cow from the top as opposed to the usual side view shown in most cases, or the image of a cow on a beach instead of in a grassy field).

This is particularly harmful because with the increase in performance of CNNs, they have been steadily incorporated into many facets of life as part of decision making processes that directly affect the larger population. It is important to develop ways to strengthen the CNNs and make them robust to noisy images, adversarial attacks and out of distribution (o.o.d.) images.

2 Increasing Robustness and Generalization

In order to develop a model that is robust and generalises well, the model needs to be developed in a manner that mimics human processing since, humans have been proved to be remarkably robust to noise, o.o.d. data and adversarial attacks. This can be done by encoding an appropriate inductive bias in four components - the training (datasets), the objective function (loss function or the function that needs to be maximized or minimized), the learning rule (mathematical logic) and the architecture (the layers and their organization) (Evans, B. D. et. al. 2022).
Some ways of incorporating the bias in the training dataset includes:

- Incorporating human uncertainty (Peterson, J. C. et. al. 2019)
- Increasing shape bias (Geirhos, R. et. al. 2018)

A method of incorporating the bias in the network (by incorporating it in the architecture) is by replacing the first convolution layer in a CNN with a fixed biologically inspired filter that restricts the features of the inputs that are being selected for processing at the early levels of the CNN (Evans, B. D. et. al. 2022).

2.1 Increasing Shape bias

In a study published by Geirhos, R. et. al. (2018), the authors provided evidence that the models that were trained on Imagenet dataset were biased towards texture rather than shape. This is different from how humans classify objects. Human object recognition and classification is based on shape perception. It was theorized that networks were learning based on texture due to it being easier than learning based on shape thus, displaying the phenomena of shortcut learning.

The authors devised experiments to see if this shortcut learning could be avoided and CNNs could be trained to be more biased towards shape. They created a stylized dataset where the Imagenet dataset was used as a base. The textures of the Imagenet images were swapped with the textures of random painting or with another texture (like the texture of an elephant on an image of a cat). They also prepared several testing datasets that were progressively more out of distribution.

It was found that networks trained on the stylized dataset were more biased towards shape and had better generalisation to o.o.d. data.

2.2 Incorporating bias by changing the architecture

Evans, B. D. et. al. (2022) published their research they modified standard architectures of CNNs to be more robust to noise and o.o.d. data. They did so by incorporating a bank of biologically inspired filters.

The first convolution layer was replaced with a bank of fixed filters. This way the filter filters out non-biologically relevant data. They experimented with several types of filters:

- low pass filter (non-biological filter to compare results to)
- Gabor filters
- Difference of Gaussian filters
- Difference of Gaussian filters followed by Gaussian filters ('Combined' front end)

It was found that the performance of the network doesn't suffer when the filter is added at the front. The main difference in performance was noted for noisy datasets and o.o.d data. The performance of the CNNs trained with the Combined front end did the best, followed by the CNNs with Gabor filters.

2.3 New Approach

Performance of the CNN was increased in the studies discussed in the previous subsections by either changing the training method or the architecture. In this project, the objective is to combine the discussed methods and observe if the performance of the models can be improved further.

To do so, a Stylized dataset is taken and passed through a Gabor filter. This filtered data is then given as input to a standard CNN (VGG16 in this case). To see how the performance of the models change, the VGG16 model is trained on original and stylized datasets (both filtered and un-filtered) - i.e 4 different training variations are compared and these models are then tested on several different datasets.

3 Existing code base

The code used by Evans, B. D. et. al. in their study was available on GitHub. The initial idea was to utilise this code and build on it. To achieve the objective of this project, the exiting code would have to be run on the required test and training datasets to compare the results.

Unfortunately, the code could not be executed. After resolving some of the syntactical errors in their code, it was run on a local machine. The system used has an Intel Graphics card which does not support some of the dependencies and NVIDIA libraries that the code relies on. Executing the code on Google Colab also had some issue, primarily related to lack of available computing resources in the free tier. Utilising Colab Pro did not help as the notebook kept crashing. The error could not be resolved and so alternatives were looked at.

A custom small scale implementation was attempted by developing the individual components and combining them. This included the following steps:

- Preparing the datasets
- Implementing a bank of Gabor filters
- Implementing a VGG16 model
- Running the code to obtain the results

4 Dataset

The original study by Evans, B. D. et. al. utilised CIFAR-10 dataset for training and testing purposes. CIFAR-10 dataset is easily available and relatively small making it good for rapid testing and development. It has 10 classes, each with 6000 images per class - i.e. 50k training images and 10k testing images.

Geirhos, R. et. al. utilised Imagenet Dataset which has 1k categories with over 1.2 million images for training and 50k images for validation, organised in 1k categories. The Imagenet dataset is roughly 150GB, so for the purpose of this project, CIFAR-10 dataset was selected.

In Geirhos, R. et. al.'s experiment, they utilised a method called Style transfer to swap the texture of the images with random paintings. The code to do so is available on GitHub. But since the this project utilises CIFAR-10 dataset, a different code repository was used to perform the style transfer (Mitzkus, B. et. al 2021). The script allows the user to stylize any dataset by following these steps:

- Download the models decoder.pth and vgg_normalized.pth from pytorch-AdaIN and move both files to the models/ directory
- Download the Style images (train.zip) from Kaggle's painter-by-numbers dataset.
- Run the stylize.py script

Since the entire Painter-by-Numbers dataset is very large (38.7GB), only the first sub-part of the dataset (train_1.zip - 5.11 GB - 11,025 images) was utilised. CIFAR-10 dataset consists of images of size 32x32. Utilising Style Transfer on such small images does not give good results as seen in figure 1.

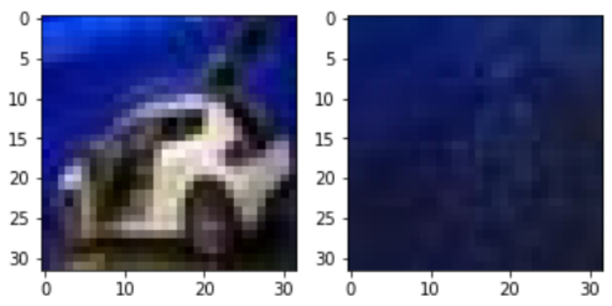


Figure 1: Style Transfer on a CIFAR-10 image, the original is on the left and the result is on the right

It is uncertain if a CNN would be able to learn any discernible features from such an image.

As an alternative, the Imagenette dataset was used. Imagenette dataset is a subset of the Imagenet dataset and has only 10 categories (tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute) - of these, in order to reduce some training time, 6 classes were taken (chain saw, church, french horn, fish, dog, golf ball) with a total of 5624 training images across these classes and 600 testing images. Style transfer was then applied to this dataset (figure 2).

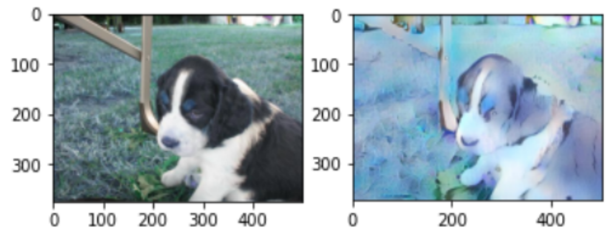


Figure 2: Style Transfer on an Imagenette sample, the original is on the left and the result is on the right

In addition to the Stylized Imagenette, some other testing datasets would be needed to compare the performance of the models. For this purpose, the following datasets were created:

- Noisy dataset 1 - A dataset generated by inducing Gaussian Noise in the Imagenette testing dataset with a standard deviation 0.001
- Noisy dataset 2 - A dataset generated by inducing Gaussian Noise in the Imagenette testing dataset with a standard deviation 0.01
- Noisy dataset 3 - A dataset generated by inducing Gaussian Noise in the Imagenette testing dataset with a standard deviation 0.1
- Edges Dataset - A dataset generated by extracting the edges of the Imagenette testing dataset (example in figure 4).
- Out of Distribution (o.o.d.) Dataset - A dataset created by randomly searching Google for 10 images of each category (example in figure 5).

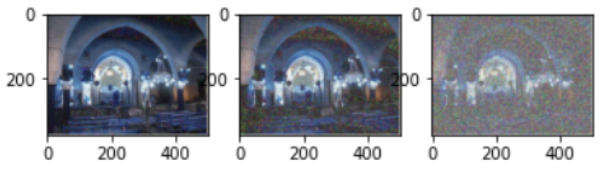


Figure 3: Left most image is the image of a church with a small amount of Gaussian Noise (Standard Deviation 0.001), the middle has Gaussian Noise with Standard Deviation 0.01 and the last has noise with Standard Deviation 0.1

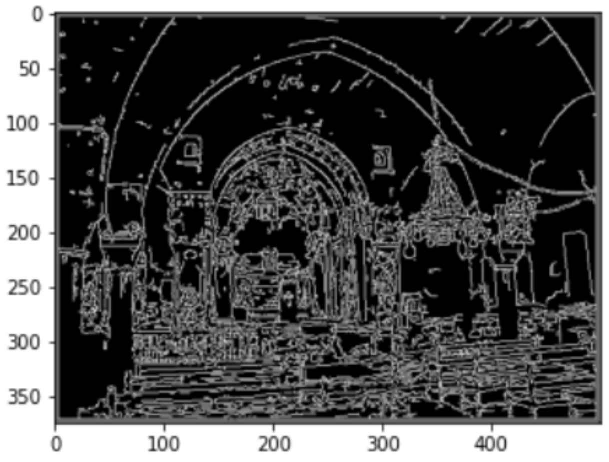


Figure 4: Image of a church with the edges extracted

5 Gabor filter bank

For the Gabor filter bank, the initial set of parameters utilised by Evans, B. D. et. al. can be seen in figure 6. Two values of 'psi' variable gives 2 different filter

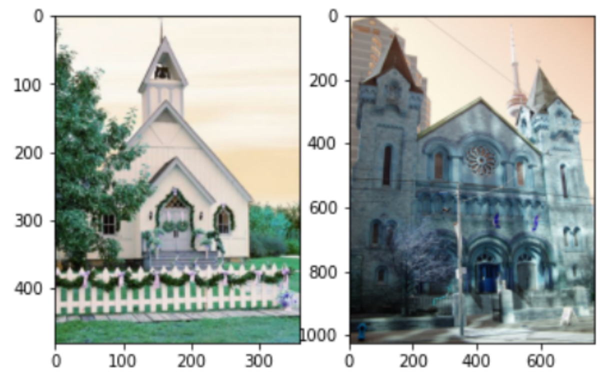
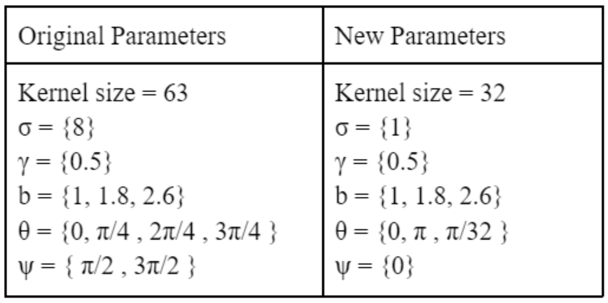


Figure 5: o.o.d sample for the class 'church'

banks, the results of which can be seen in Figure 7. It is uncertain ff enough features remain in these images for the CNN to learn from, so an alternative set of parameters were utilised (as indicated in figure 6). The results of these new parameters can be seen in the figure 7.



Original Parameters	New Parameters

Kernel size = 63
σ = {8}
γ = {0.5}
b = {1, 1.8, 2.6}
θ = {0, π/4 , 2π/4 , 3π/4 }
ψ = { π/2 , 3π/2 }
	Kernel size = 32
	σ = {1}
	γ = {0.5}
	b = {1, 1.8, 2.6}
	θ = {0, π , π/32 }
	ψ = {0}

Figure 6: Parameters of the Gabor Filter Banks

6 Architecture

For the CNN itself, a VGG16 architecture was used. The layers can be seen in Figure 19 in the Appendix. VGG16 is a relatively old model. Since then several new methods to improve accuracy and learning have been developed. To improve performance, Batch Normalization and Drop Out layers were added to the architecture. Batch Normalization provides a way to standardize the inputs to layers in a neural network by scaling the outputs of the previous layers such that they have a mean of 0 and variance of 1. This speeds up the training. Drop out layers prevent over-fitting by randomly selecting some outputs from a layer and ignoring them ('dropping' them) in each cycle of training.

There has been a lot of debate on where the Drop Out layers should be added. Some approaches include:

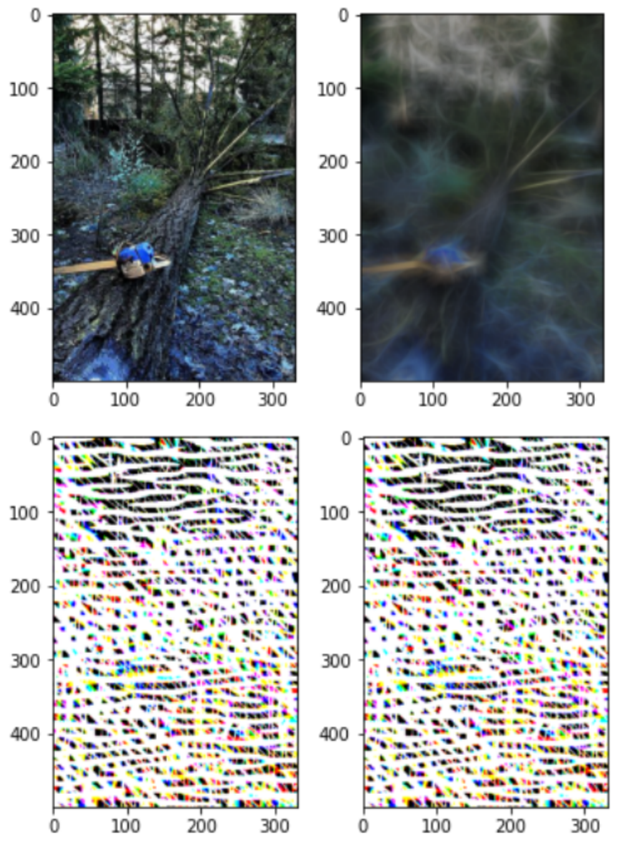


Figure 7: The top-left image is the original image. The top-right image is the filtered output using the new parameters. The bottom two images are the filtered output with the original parameters

- Adding the Drop Out layers between the Dense Layers.
- Adding the Drop Out layers between the convolution layers, before the non linear activation functions
- Adding the Drop Out layers between the convolution layers, after the non linear activation functions.

To decide on an architecture, all three variations were tried. Applying the Drop Out layers between the Dense layers gives the best results. When Drop Out layers were added between the Convolution layers (before or after them) a lot of useful information is lost. And while there is no appreciable change in Accuracy, the validation loss increases by a lot.

Finally, the updated architecture was trained. The new architecture is as shown in Figure 20 (in the appendix).

In both architectures, a Stochastic Gradient Descent (SGD) optimizer was used with learning rate = 1e-4 (0.0001) and momentum = 0.9. There are

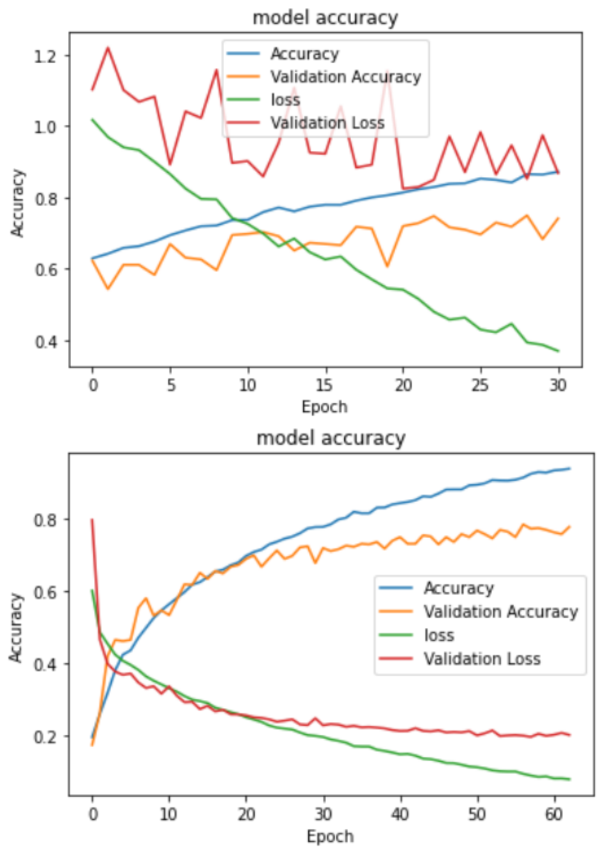


Figure 8: Outputs of two models trained on the Gabor filtered Imagenette dataset. The top image is with Drop Out layers between the convolution layers and Dense layers. The bottom image is with Drop Out layers between the Dense Layers only.

other optimizer available like the Adam optimizer. However, when an Adam optimizer was used to train the initial VGG16 architecture with the Imagenette dataset, it took approximately 5 hours 40minutes for training as opposed to the training time with SGD optimizer that took approximately 2hours 20 minutes. Thus, to speed up training the SGD optimizer was used.

The batch size of the dataset was kept at the default number of 32.

In all cases, the number of epochs was set to be 100 with an early stopping criteria to prevent overfitting. The stopping condition used monitored the 'val_loss' variable (the validation loss) with a patience of 5. Consider an epoch n, if the model's validation loss for five epochs after the epoch n was higher than it's validation loss at epoch n, then the model will stop the training process and restore the weights from epoch n.

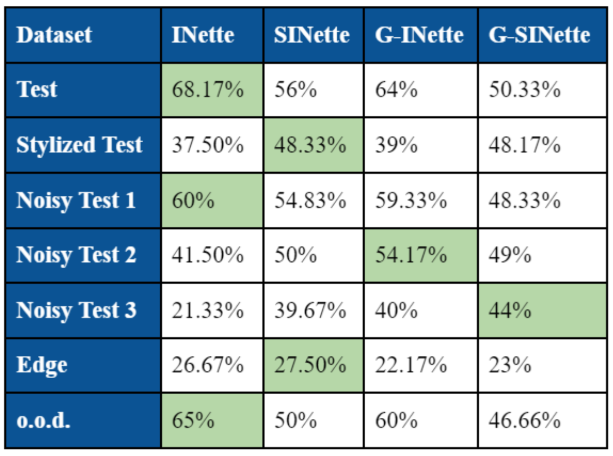
There was no particular reason why a 'ReLu' activation function was chosen aside from the fact it is a very commonly used and popular activation function for CNNs.

Next, a 'Softmax' activation function was selected for the output layer as opposed to others (like a sigmoid function) because SoftMax function's output is like a probability distribution across all labels which is preferred for a multi-class classification problem (like in this project where we have 6 classes). Sigmoid function is better for binary classification.

Finally, for the models trained on filtered data the test sets were also filtered with the same Gabor filter bank. This is because the purpose of filtering the data was to imitate the experiment by Evans, B. D. et. al. where they replaced the first convolution layer with a bank of fixed filters. The models being trained on the filtered data will thus also be tested on filtered data.

7 Results and Discussion

The results of the experiments can be seen in Figures 9 and 10.



Dataset	INette	SINette	G-INette	G-SINette
Test	68.17%	56%	64%	50.33%
Stylized Test	37.50%	48.33%	39%	48.17%
Noisy Test 1	60%	54.83%	59.33%	48.33%
Noisy Test 2	41.50%	50%	54.17%	49%
Noisy Test 3	21.33%	39.67%	40%	44%
Edge	26.67%	27.50%	22.17%	23%
o.o.d.	65%	50%	60%	46.66%

Figure 9: Results of Architecture 1 (without Batch Normalization and Drop Out)

The columns are the different training datasets -

- INette- Imagenette
- SINette - Stylized Imagenette
- G-INette- Gabor Filtered Imagenette
- G-SINette - Gabor Filtered Stylized Imagenette

The rows are the different testing datasets:

- Test- Imagenette testing dataset
- Stylized Test - Stylized Imagenette testing dataset

Dataset	INette	SINette	G-INette	G-SINette
Test	81.67%	56.83%	75.33%	52%
Stylized Test	39.17%	50.67%	48.34%	47.67%
Noisy Test 1	63.83%	38.67%	72%	49%
Noisy Test 2	49.18%	40.83%	63.83%	48%
Noisy Test 3	17.67%	21%	40.33%	37.83%
Edge	18.17%	21.67%	26.17%	18.67%
o.o.d.	66%	40%	38.33%	50%

Figure 10: Results of Architecture 2 (with Batch Normalization and Drop Out)

- Noisy Test 1 - Imagenette testing dataset with induced Gaussian noise (standard deviation 0.001)
- Noisy Test 2 - Imagenette testing dataset with induced Gaussian noise (standard deviation 0.01)
- Noisy Test 3 - Imagenette testing dataset with induced Gaussian noise (standard deviation 0.1)
- Edge - Imagenette testing dataset with the edges extracted
- o.o.d - out of distribution dataset

The detailed explanation and examples of the test sets are in Section 4.

The results of Architecture 1 are very different from Architecture 2. There is no pattern observed for Architecture 1. However, for Architecture 2, the Model trained on Gabor filtered Imagenette dataset seems to perform better when tested on the Noisy datasets and Edge dataset. The added Batch Normalization and Drop Out layers seem to provide more stable results. For performance graphs of each of the eight variation, check the Appendix.

The new approach with the Gabor Filtered Stylized dataset does not give better results for either of the two architectures (except in one case).

Further, the original training method (Imagenette dataset) performs the best on the o.o.d. dataset. This is unexpected behaviour. By comparing the class-wise accuracy of the different models on the o.o.d. dataset, it was found that the classes of 'chainsaw' and 'fish' have very poor performance. Aside from models trained on Imagenette data, the rest of the models all give below 20% accuracy for these two categories in the o.o.d set. When comparing the images in the Imagenette dataset and o.o.d. data, for both classes, in the Imagenette dataset the target object is being held by a person (person holding a fish and person holding a chainsaw), very few images exist of the target object by itself or without a human in the image. Whereas, the o.o.d data has only a few samples with a person, the rest have just the target object in various orientations. While the Imagenette trained models utilise some amount of shortcut learning to classify the images of these categories with higher accuracy, the other models can not do so. This has caused a decrease in overall accuracy of the models.

Better generalisation will most likely be seen in the reverse case where the training dataset has the target image in focus (with less clutter or distractions in the background like the Human figures for these cases) and the testing dataset has more distractions.

8 Conclusion and Future Work

Combining the training method of Geirhos, R. et. al. with the architecture of Evans, B. D. et. al. did not yield better results - for the given Hyperparameters. In general Evans, B. D. et. al.'s method gives the best results out of all variations.

Evans, B. D. et. al.'s study concluded that a combinations of Difference of Gaussian fillter bank followed by a Gabor filter bank gave better results than just a Gabor filter bank. For future work, the experiment discussed in this report can be replicated with this combined front end to see if using a Stylized dataset increases performance or now.

Additionally, some increase in performance was observed with the addition of Batch Normalization and Drop Out layers. Thus, it is possible that with fine-tuning and changing the Hyperparameters of the existing architecture, the performance of the models may change.

9 References

[1] Peterson, J. C., Battleday, R. M., Griffiths, T. L., Russakovsky, O. (2019). Human uncertainty makes classification more robust. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 9617-9626)

[2] Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., Brendel, W. (2018). ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv*

preprint arXiv:1811.12231.

[3] Evans, B. D., Malhotra, G., Bowers, J. S. (2022). Biological convolutions improve DNN robustness to noise and generalisation *Neural Networks,* 148, 96-110.

[4] Mitzkus, B., Geirhos, R., Rusak, E., Kadish, D., Mathis, M. (2021). *Stylize-Datasets* Retrieved April 17, 2022, from https://github.com/bethgelab/stylize-datasets

10 Appendix

10.1 More Results

In total, 2 architectures and 4 training methods were used in this project. The training datasets are:

- Imagenette Dataset
- Stylized Imagenette Dataset
- Imagenette Dataset filtered using a Bank of Gabor Filters
- Stylized Imagenette Dataset filtered using a Bank of Gabor Filters

The performance of the 8 variations of the models are presented in this section (figures 11-18).

10.2 Architecture details

The original VGG16 architecture is relatively old and more recent developments like Batch Normalization and Drop Out layers were not incorporated in it. In order to see if these layers could give a better results, a second architecture with these layers was also tested. The original architecture can be seen in figure 19 and the updated architecture can be seen in figure 20.

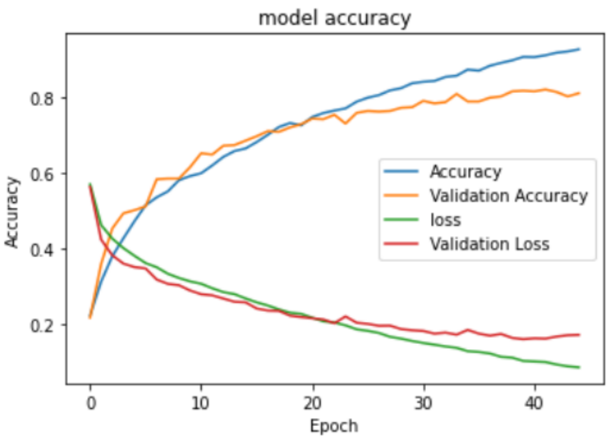


Figure 11: VGG16 trained on Imagenette dataset with Batch Normalization and Drop Out (20%)

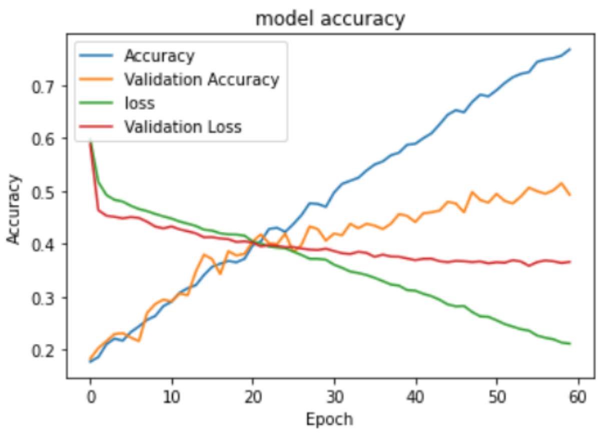


Figure 12: VGG16 trained on Stylized Imagenette dataset with Batch Normalization and Drop Out (20%)

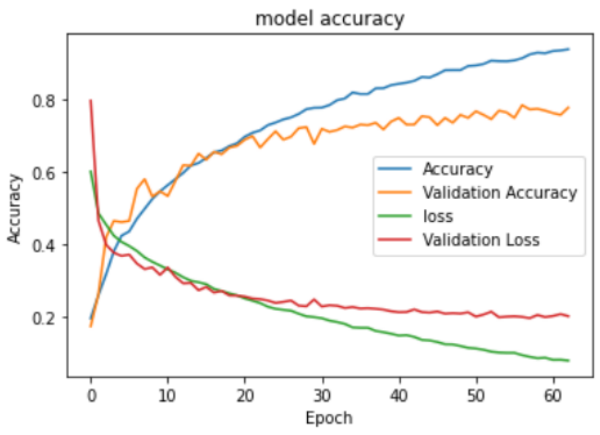


Figure 13: VGG16 trained on Gabor Filtered Imagenette dataset with Batch Normalization and Drop Out (20%)

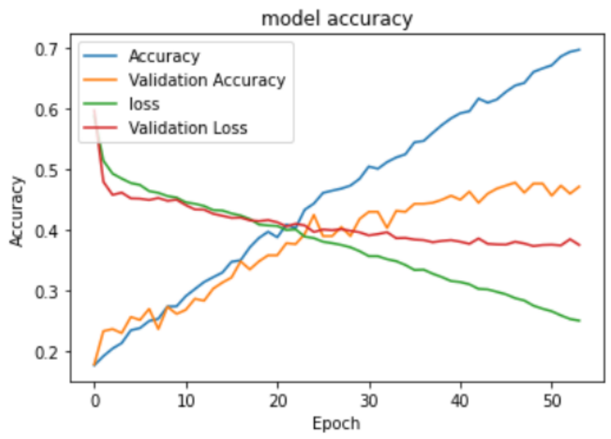


Figure 14: VGG16 trained on Gabor filtered Stylized Imagenette dataset with Batch Normalization and Drop Out (20%)

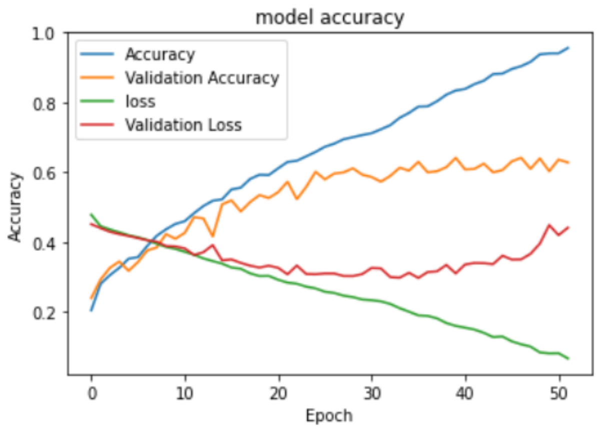


Figure 17: VGG16 trained on Gabor Filtered Imagenette dataset

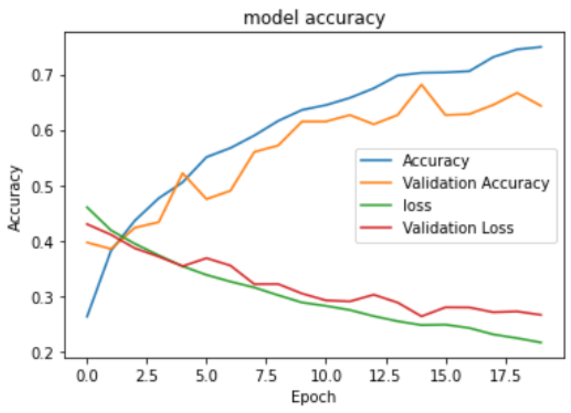


Figure 15: VGG16 trained on Imagenette dataset

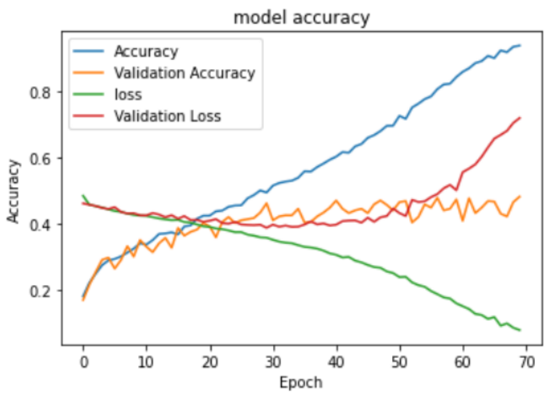


Figure 18: VGG16 trained on Gabor Filtered Stylized Imagenette dataset

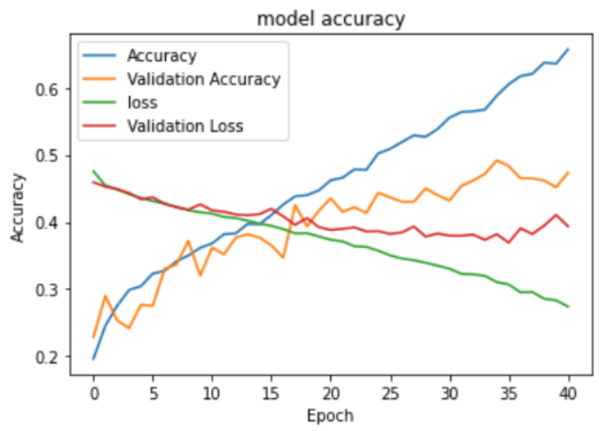


Figure 16: VGG16 trained on Stylized Imagenette dataset

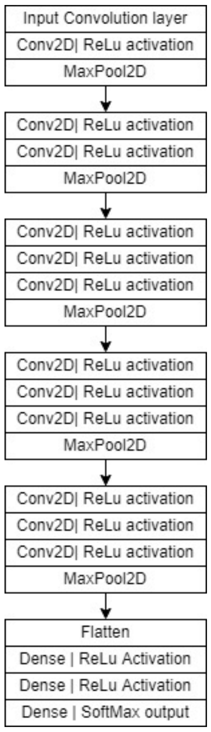


Figure 19: Initial Architecture

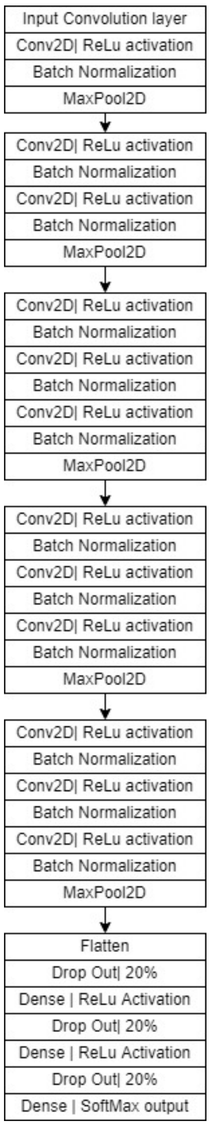


Figure 20: Updated Architecture