DEEP LEARNING FOR VISION & NLP

IMAGE DENOISING & CLASSIFICATION

ASSIGNMENT REPORT

Student Number: 2938740

ITNPAI1 Deep Learning for

Computer Vision & NLP

Spring 2021

PART A – IMAGE DENOISING

1. INTRODUCTION

In the field of computer vision, image processing and in particular image denoising is a popular and important research topic [1]. Good data is key to training a good network and thus it follows that excellent data stems from excellent image processing techniques and benefits immensely from continued advanced research in this field. One of the key challenges when denoising images is to remove as much of the noise as possible without losing too many of the underlying details. This is made difficult by the fact that noise, edge and texture tend to be high frequency and removing these components can easily lead to a loss of detail [2].

The task will use two traditional methods of denoising (mean and median filtering), a wavelet method and a Deep Learning method. The CNN has the advantage of not requiring user determined parameters meaning it is free to learn its own parameters for processing images. This gives the CNN enormous potential that extends beyond image denoising and into image reconstruction. A CNN trained correctly is capable of not only learning quite accurately how the image 'should' look but it is possible to enhance the image to 'super-resolution'.

2. DENOISING ALGORITHM

To complete this task in an automated process it required the development simple process:

1. Scan image folders to identify the original images (for comparison) and the target images containing various levels of noise. The original images will be the driver for which target files will be iterated through.
2. Call 4 functions in sequence to apply denoising techniques over the noisy images (mean filter, median filter, wavelet filter and Deep Learning).
3. Once denoised, two sets of summary statistics are calculated. The Mean Square Error (MSE) and the Structural Similarity Index (SSIM) are the chosen metrics for this task. A brief explanation will follow.
4. The denoised images will be written to file for review and used as examples of good and bad outcomes.
5. The MSE and SSIM statistics will be stored in a dictionary of results comparing the noisy and various denoised images to the original image.
6. A series of graphs will be produced to highlight the results in a visual manner.

It will be beneficial to define 'noise' in terms of an image. Noise can be represented as an additive function that transforms a clean image into something less desirable (for the most part). The following equation [3] is often used to describe the problem:

$$\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y}) + \boldsymbol{\eta}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{g}(\boldsymbol{x},\boldsymbol{y})$$

The term $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$ represent the image. The degradation caused by the noise additive $\boldsymbol{\eta}(\boldsymbol{x},\boldsymbol{y})$ is added to the image to produce the noisy image $\boldsymbol{g}(\boldsymbol{x},\boldsymbol{y})$. Image restoration techniques attempt to estimate and remove the noise leaving the best possible estimate of the original image $\hat{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{y})$. Note that the f-hat symbol does not show clearly here.

3. LIBRARIES / FUNCTIONS

In addition to the traditional library imports such as time, os and numpy the algorithm did call upon some more specialist functions for this task. These are described briefly below:

- OpenCV (Open Source Computer Vision Library) is one of the core libraries for image processing and computer vision tasks. This is a cross-platform library that is widely used and highly optimised for almost all CV tasks.
- Skimage (Scikit-Image) is commonly referred to as image processing for Python and contains a collection of image processing algorithms. This is also an open source library.
- medianBlur filter function is used for performing the denoising for median filter. This works by replacing the pixel at the centre of the kernel (user defined size) with the median value of all neighbouring pixels within the convolution kernel. This is classed as a low-pass filter.
- Blur filter function is used for performing the denoising for mean filter. Similar way to the medianBlur filter apart from replacing the centre pixel with a simple mean of all pixels within the convolution kernel.
- denoise_wavelet function* is used for performing wavelet denoising and is found in the Skimage library. There are two methods (VisuShrink and BayesShrink). The user can also select to use the filter in soft or hard mode (soft is preferred for providing the best estimate of the original image). The user can also select the number of wavelet decomposition levels to use.

* "To match the implementation of Wang et. al. [4] - set gaussian_weights to True, sigma to 1.5. Use_sample_covariance to False."

4. RESULTS

4.1.MSE PERFORMANCE

In terms of this absolute error metric the mean and median filters performed well in reducing the MSE compared to the levels seen in the noisy images at all levels with the mean being slightly better than the median. The gaps between the images decreasing as the noise increased highlights that they both increasingly struggling to differentiate themselves. Blurring was observed and is a known side effect of this type of denoising. In respect to the wavelet filter, it initially outperformed both the mean and median filters however as the noise increases the MSE scores become increasingly erratic. This could be due to the fact the wavelet is working harder to produce a higher level of denoising and thus appearing to increase the MSE more than the other filter.

The MSE for all filters spiked around images 0007 and 0008 as per the graphs. A quick analysis of the pictures shows that 0007 is a very flat image, almost monochrome with distinct features in the horizontal and vertical plane in the foreground of the image. Similarly, 0008 contains some very strong edges in the 3 totemic figures which would lead to larger differences in the kernel pixel values being observed as the filters work around these strong edges and features.

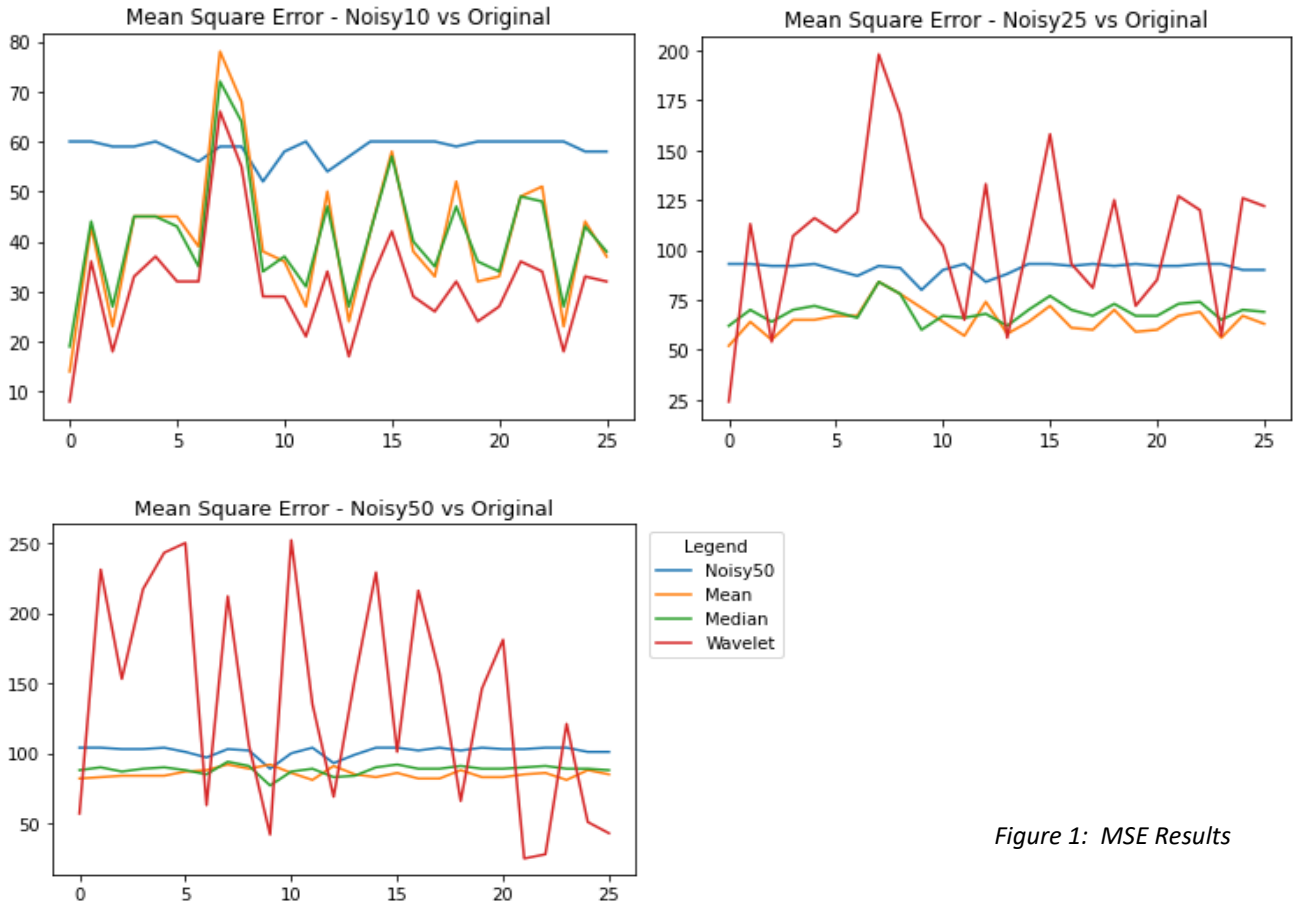
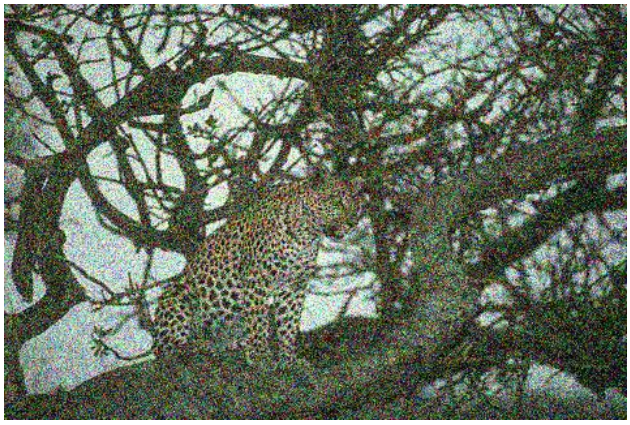


Figure 1: MSE Results

Original 0008 with strongly defined edges

Denoised noisy50 with mean filter and soft edges

Noisy50 0022 extremely noisy and distracting



Denoised noisy50 with wavelet filter and good clarity

Figure 2: Good (low MSE) and bad (high MSE) examples of MSE statistics

4.2.SSIM PERFORMANCE

The results by and large follow a similar pattern to the MSE in terms of performance, but the interpretation is very different. As noise increases the filters are better able to differentiate themselves. The wavelet filter is supreme in every image denoised with the mean filter performing increasingly better than the median filter as noise increases. Again images 0007-0009 seem to throw the mean and mean filters compared to the noisy image in a comparison against the original with the wavelet filter largely unaffected in preserving similarities with the original image (particularly image 0000 where it performs astounding well on the index). The wavelet filter averages a solid 0.9 across the noisy10 dataset making this a good choice of filter.

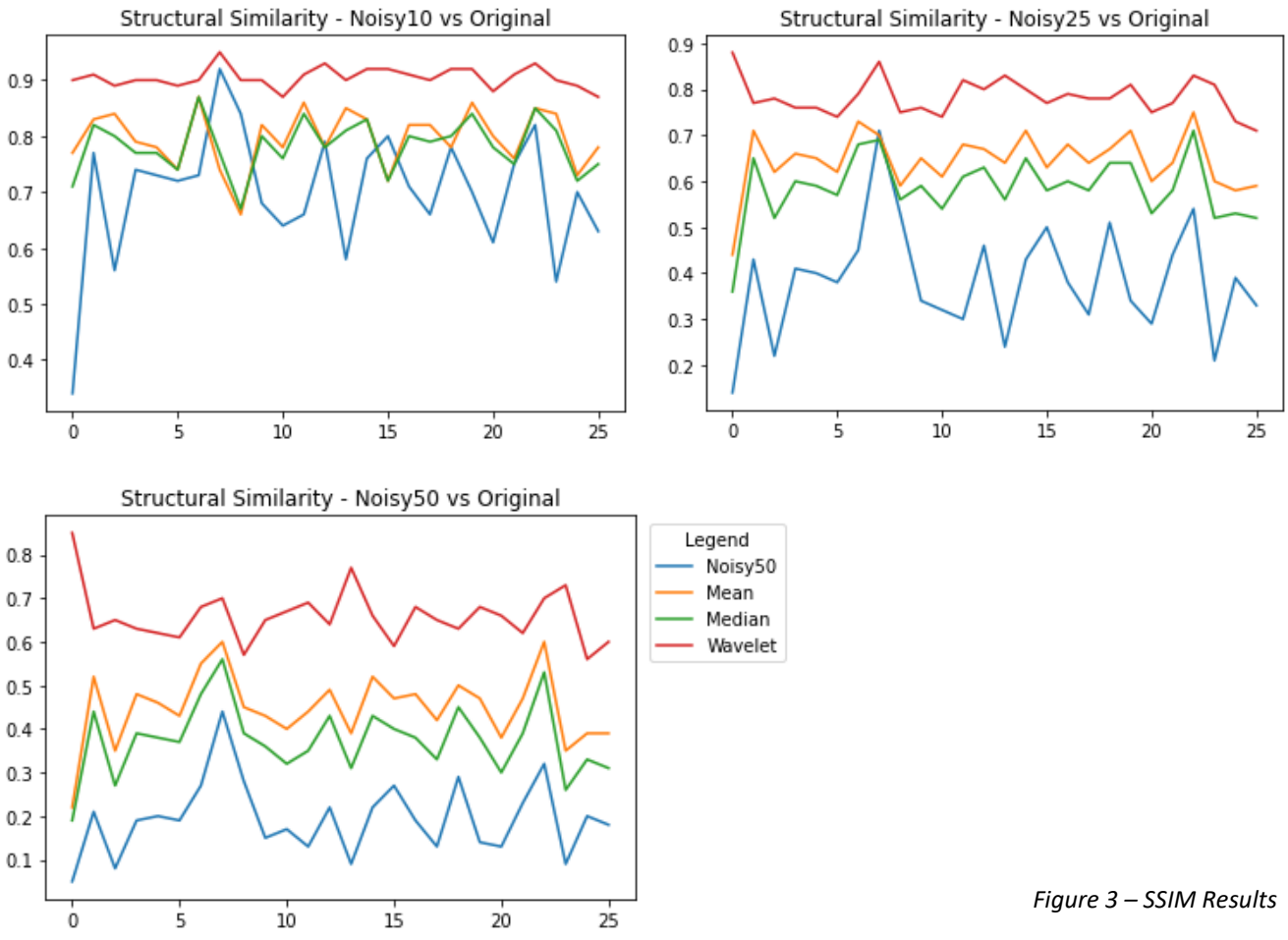


Figure 3 – SSIM Results

Noisy50 0000 with almost zero SSIM rating

Denoised noisy50 with wavelet filter and visible propellors

Original 0018 with clean and sharp lines

Denoised noisy25 with median filter and good edges

Figure 4: Good (low SSIM) and bad (high SSIM) examples of SSIM index

4.3.NON-LOCAL MEANS FILTER

A couple of other filters deserve an honourable mention, particularly the bilateral and non-local means filters. Both produced some good results during the experimentation phase with one example standing out. One of the NLM filters was tuneable to produce a stunning result on image 0000 however this tuning also led to increased numbers of artifacts on other images (cv2.fastNlMeansDenoisingColored).

Whilst the level of noise is still quite visible for these images, what stuck out is how they were able to produce a high quality representation of the aeroplane in the foreground with the star of the USAF insignia quite visible on the fuselage through the noise.

Denoised noisy50 with bilateral filter

Denoised noisy50 with fastNLMeans

Figure 5: Bilateral and fastNLMeans retaining some impressive features through heavy noise

4.4.DEEP LEARNING MODELS

Whilst not able to submit a working deep learning denoiser I would like to touch upon some theory. Convolutional autoencoders can be used to denoise images. The network model for denoising is symmetrical. The noisy images are submitted into the autoencoder as the first half of the model. In the second half of the model, the image is reconstructed using a decoder leaving the desired signal intact without any of the noise (theoretically). In the middle of the network is a central block that acts as a compressed representation of the image. The modern denoising autoencoders are an extension of the of regular autoencoders and not a specific development just for denoising images. The development of the denoising autoencounter helps the network's hidden layers to learn better and reduce the risk of overfitting the model.

5. KEY FINDINGS

Image denoising is not an easy task and requires a good knowledge of the available filters and how to use them and it comes with a steep learning curve. The MSE can be a misleading index, it was observed that as the level of noise in the images increased the wavelet filter appeared to fluctuate wildly in its results however upon closer inspection of the SSIM score and of the images themselves it proved to be a reliable filter bar one or two cases.

The sad omission of the Deep Learning results shows how difficult it is to find easy access to pre-trained models in a Python environment. Plenty of Matlab models were found and most likely owing to the availability of a dedicated de-noising library.

6. CONCLUSIONS

The conclusions from this task are that a deep level of understanding would be needed before one could call themselves an expert in image processing. In the case of filtering, we have seen that different filters react to noise and text levels in different fashions and that it is not wise to explicitly trust just one metric for the classification of how well a filter is able to denoise an image, as always, the human eye is a great tool and where a filter may possess a great structural similarity or low mean square error it may still look unacceptable for use in the real world.

Having ended up in a rabbit hole searching for denoising models, in hindsight it would have been better to attempt to build my own autoencoder to complete the task.

PART B – IMAGE CLASSIFICATION

7. INTRODUCTION

This task is a multiclass classification task using the CIFAR-10 dataset [5] which takes its name from the source provider (Canadian Institute for Advanced Research) and the fact that it has 10 classification outputs. It is a subset of the larger CIFAR-100 dataset of 80 million tiny images. The data was collected by some well-known names within the deep learning community – Alex Krizhevsky [6] (creator of the AlexNet network), Vinod Nair (DeepMind) and Geoffrey Hinton (no introduction required) – who all operate or used to operate out of the University of Toronto.

The collection consists of 60,000 colour images of 32x32 pixels assigned to one of 10 categories labelled from zero through nine. There are 6,000 images of each classification (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks) and is split on a 5:1 training to test ratio. The small image size is ideal for researchers to quickly iterate their algorithms and is one of the most used datasets for training machine learning and computer vision algorithms.

Alex Krizhevsky [5] quoted a baseline error rate (1 – accuracy) of 18% without data augmentation and 11% with data augmentation in his research. As this is a first attempt at a deep network during a self-learning assignment, the target will be lowered to reflect the opinion of Jason Brownlee [7] (www.machinelearningmastery.com) who cites that 80% accuracy is very achievable with excess of 90% being achieved only by high-end networks. The goal will be to end up somewhere between achievable and high-end.

8. ALGORITHMS & LIBRARIES

8.1.ALGORITHM

The networks are built in Google Colab as the network required a GPU to run in a reasonable amount of time for iterative design. The algorithm will be tested in three stages as per the brief:

1. Creation of a baseline model
2. Baseline model + parameter and hyperparameter changes
3. Improvements to the architecture (including dropout, batch normalisation)

Whilst not part of the architecture, data augmentation will be trialled to see if this helps raise accuracy. Renowned deep learning practitioner, Dr. Andrew Ng [8, 9], recently gave a live lecture around taking a model-centric versus data-centric approaches to improving data quality and be consequence, model accuracy.

8.1.1. BASELINE & IMPROVED MODEL

These two models are essentially the same only with slight tweaks to parameters as described in the results section. It is a slightly modified form of 1 block VGG network initially set to work with average pooling, 5x5 convolution and limited number of filters together with a softmax activation for the multi-classifier output layer. The baseline model was selected after trialling a small number of configurations as part of the learning process. The model was trained on basic parameters to serve as a starting point for attempting to improve the network.

Model: Baseline Model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_36 (Conv2D)           (None, 28, 28, 32)        2432
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 24, 24, 32)        25632
_________________________________________________________________
average_pooling2d_18 (Averag (None, 12, 12, 32)        0
_________________________________________________________________
flatten_18 (Flatten)         (None, 4608)              0
_________________________________________________________________
dense_36 (Dense)             (None, 100)               460900
_________________________________________________________________
dense_37 (Dense)             (None, 10)                1010
=================================================================
Total params: 489,974
Trainable params: 489,974
Non-trainable params: 0
_________________________________________________________________
```

Model: Baseline Model (Improved)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_46 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
flatten_23 (Flatten)         (None, 12544)             0
_________________________________________________________________
dense_46 (Dense)             (None, 100)               1254500
_________________________________________________________________
dense_47 (Dense)             (None, 10)                1010
=================================================================
Total params: 1,274,902
Trainable params: 1,274,902
Non-trainable params: 0
_________________________________________________________________
```

8.1.2. ADVANCED MODEL

To gain superior performance and achieve as well performing a model as possible a 3 block VGG model was developed together with batch normalisation and dropout layers to help with the massive overfitting problem that the 1 block baseline layer was suffering. Randomised data augmentation will also supplement this model for improved accuracy.

Model: Advanced Model

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 32, 32, 64)	1792
batch_normalization_18 (Batc	(None, 32, 32, 64)	256
conv2d_19 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_9 (MaxPooling2	(None, 16, 16, 64)	0
dropout_18 (Dropout)	(None, 16, 16, 64)	0
conv2d_20 (Conv2D)	(None, 16, 16, 128)	73856
batch_normalization_19 (Batc	(None, 16, 16, 128)	512
conv2d_21 (Conv2D)	(None, 14, 14, 256)	295168
max_pooling2d_10 (MaxPooling	(None, 7, 7, 256)	0
dropout_19 (Dropout)	(None, 7, 7, 256)	0
conv2d_22 (Conv2D)	(None, 7, 7, 256)	590080
batch_normalization_20 (Batc	(None, 7, 7, 256)	1024
conv2d_23 (Conv2D)	(None, 7, 7, 512)	1180160
max_pooling2d_11 (MaxPooling	(None, 3, 3, 512)	0
dropout_20 (Dropout)	(None, 3, 3, 512)	0
flatten_3 (Flatten)	(None, 4608)	0
dense_15 (Dense)	(None, 4608)	21238272
dense_16 (Dense)	(None, 1536)	7079424
batch_normalization_21 (Batc	(None, 1536)	6144
dropout_21 (Dropout)	(None, 1536)	0
dense_17 (Dense)	(None, 1000)	1537000
batch_normalization_22 (Batc	(None, 1000)	4000

```
dropout_22 (Dropout)         (None, 1000)              0
_________________________________________________________________
dense_18 (Dense)             (None, 100)               100100
_________________________________________________________________
batch_normalization_23 (Batc (None, 100)               400
_________________________________________________________________
dropout_23 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                1010
=================================================================
Total params: 32,146,126
Trainable params: 32,139,958
Non-trainable params: 6,168
_________________________________________________________________
```

8.1.3. LIBRARIES

The models were built using the Keras framework. This is a high-level API based framework that makes it relatively easy to interface with the tensorflow backend for quick prototyping and development of deep learning models.

Several Keras libraries were used in the model code including utils, models, layers, optimizers, wrapper and callbacks for constructing the actual model. The ImageDataGenerator library was called to provide an extra dimension to the input data by randomly manipulating the images in subtle ways to enhance the training experience (zoom, shear, pixel shifts, etc..). The dataset was also downloaded from Keras directly.

Scikit-Learn was called once to provide a StratifiedKFold function to improve the spread of the training data during cross validation

9. RESULTS

9.1. BASELINE MODEL

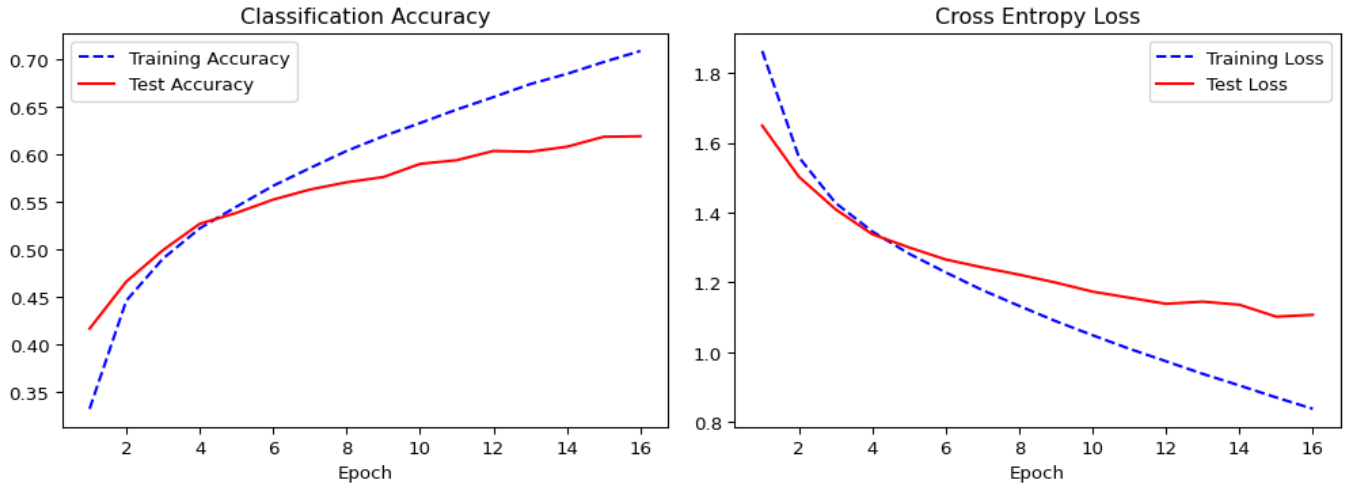
The baseline model achieved an average training accuracy of 71.3% and an average validation accuracy of 61.9% across 5 epochs in a time of 3 minutes 45 seconds on Google Colab Pro. The full baseline parameters are given below:

Parameter	Value	Comments
k-folds	5	
Epochs	16	
Batch size	64	
Optimiser	SGD	Learning rate = 0.001; momentum = 0.9

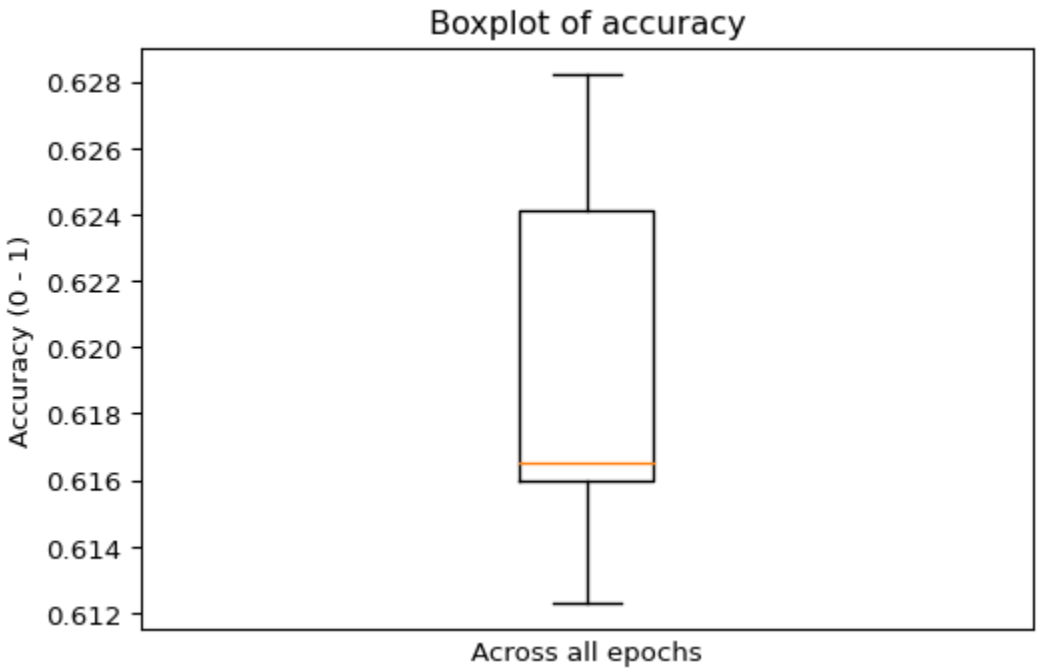
The model is clearly overfitting the training data as seen overleaf, with divergence occurring after the 4th epoch meaning that the model does not generalise well. A quick comparison of the training and validation averages against the project targets show that this is a high-bias, high-variance model:

	Error Rate	Difference	Comments
High Target	10%	-10%	
Threshold	20%	-	
Training	28.6%	+8.6%	Below minimum threshold, high bias
Validation	38.1%	+18.1%	Overfitted model, high variance, does not generalise well

A full breakdown of the data can be found in the accompanying document "Part B – Appendix – Supplementary Results".



Figures 6: Training and Validation performance for the baseline model



Figures 7: Overall boxplot of accuracy across all epochs / folds

It can be seen from the boxplot that the better performing cross validation models far outperformed the average models.

```
Validation Accuracy: mean=61.942 std=0.583, k-folds=5
```

9.2.IMPROVED MODEL

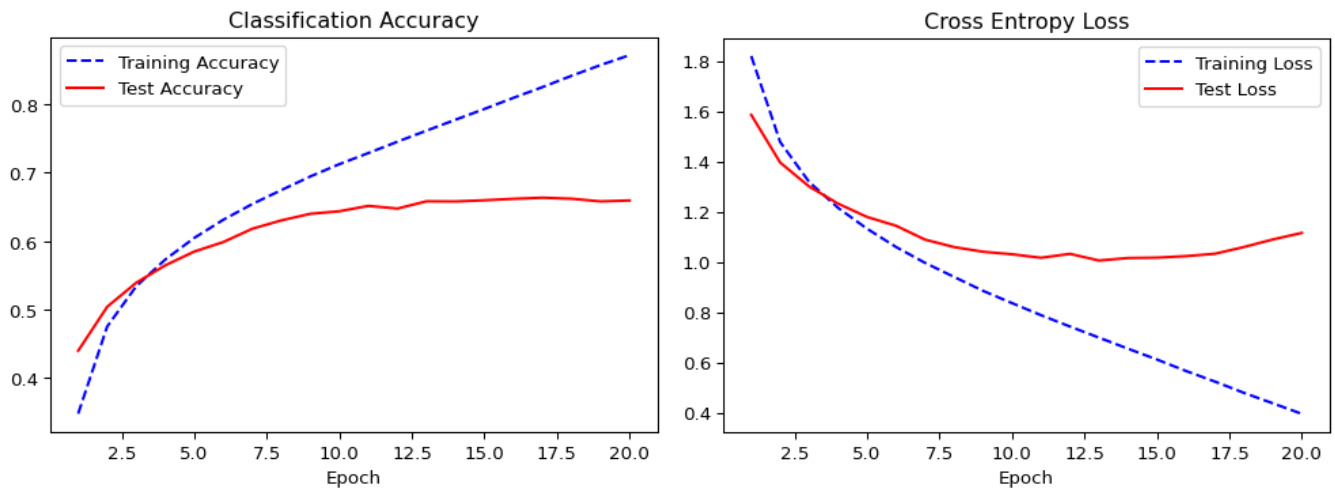
The results for the improved baseline model were marginally better with an average training accuracy of 87.9% and an average validation accuracy of 66.0% across 5 epochs in a time of 7 minutes 56 seconds on Google Colab Pro. The parameters and improvements made are given below:

Parameter	Value	Comments
k-folds	5	
Epochs	20	25% increase in training time (16 -> 20)
Batch size	32	Smaller batch
Optimiser	SGD	Learning rate halved = 0.0005; momentum = 0.9
Pooling	Max	Changed from AvgPool with no change to pool size
Conv Kernel	(3, 3)	Dropped from (5, 5)
Conv2 Layer	64 filters	Doubled number of filters from 32->64

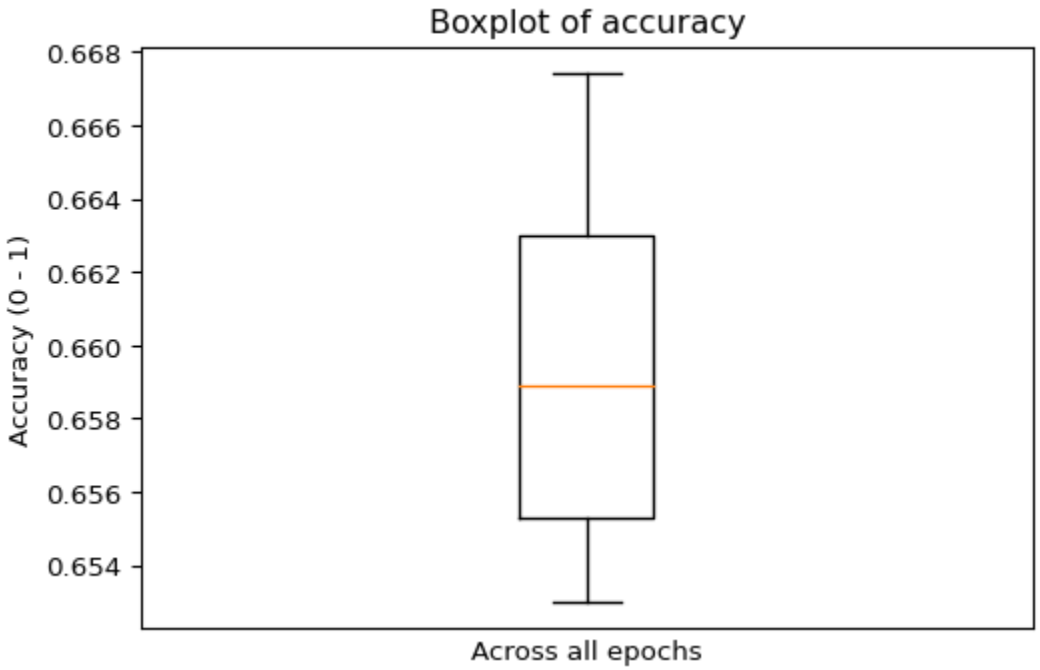
The model is overfitting even more than the baseline model however the validation set has shown a modest 4% increase. As seen overleaf, divergence is still occurring early in the training process and can be seen after the 4th epoch. After the 15th epoch the validation loss begins to increase suggesting a new approach is required for the advanced model. Comparison of the training and validation averages against the project targets show that this is still a high-bias, high-variance model:

	Error Rate	Difference	Comments
High Target	10%	-10%	
Threshold	20%	-	
Training	28.6%	-7.9%	Has surpassed the threshold and is approaching the high target figure
Validation	38.1%	+14.0%	Overfitted model, high variance, does not generalise well

A full breakdown of the data can be found in the accompanying document "Part B – Appendix – Supplementary Results".



Figures 6: Training and Validation performance for the improved model



Figures 9: Overall boxplot of accuracy across all epochs / folds

It can be seen from the boxplot that the model is more consistent with the median close to the centre of the plot.

```
Validation Accuracy: mean=65.952 std=0.519, k-folds=5
```

9.3.ADVANCED MODEL

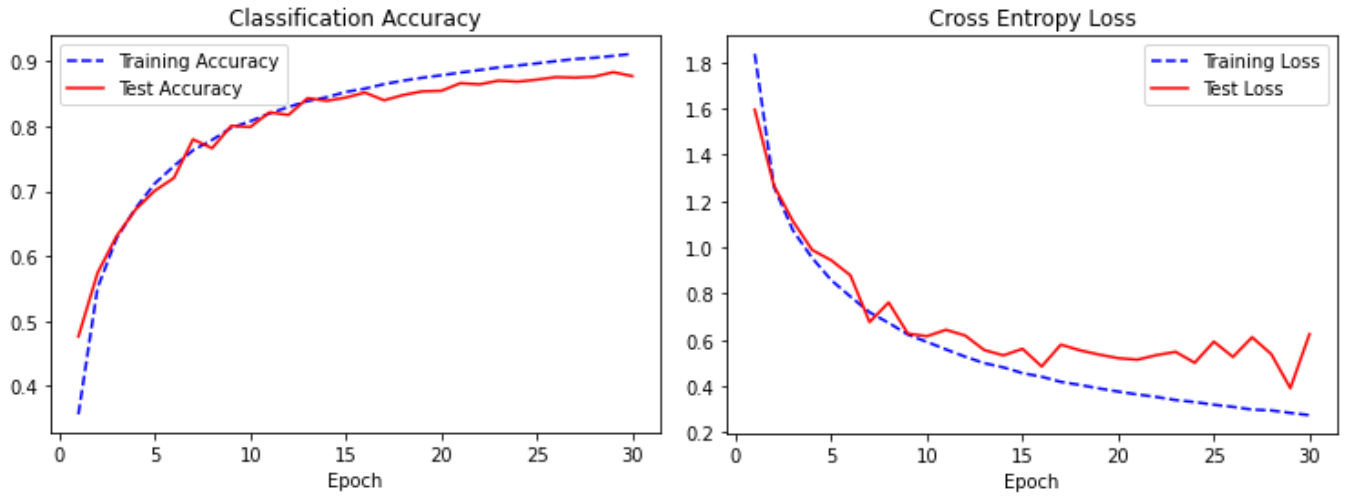
The results for the advanced model are far superior to the baseline and improved models and are approaching the higher target that was set. The model has an average training accuracy of 91.3% and an average validation accuracy of 87.1% across 30 epochs in a time of 2 hours 54 minutes on Google Colab Pro. The parameters are given below:

Parameter	Value	Comments
k-folds	10	Higher validation folds
Epochs	30	50% increase in training time (20->30)
Batch size	32	Smaller batch
Optimiser	Adam	Adam is widely regarded as a superior optimiser (default learning rate = 0.001)
Data Augmentation	Random	Height and width shift = 0.1, shear = 0.1, zoom = 0.1, horizontal flip = True
Drop-out Regularisation	Varying	Increases from 0.2 in early layers towards a maximum of 0.5 in the FC layers Helpful in decreasing the generalisation problem
Batch Regularisation	Multiple	To help stabilise training

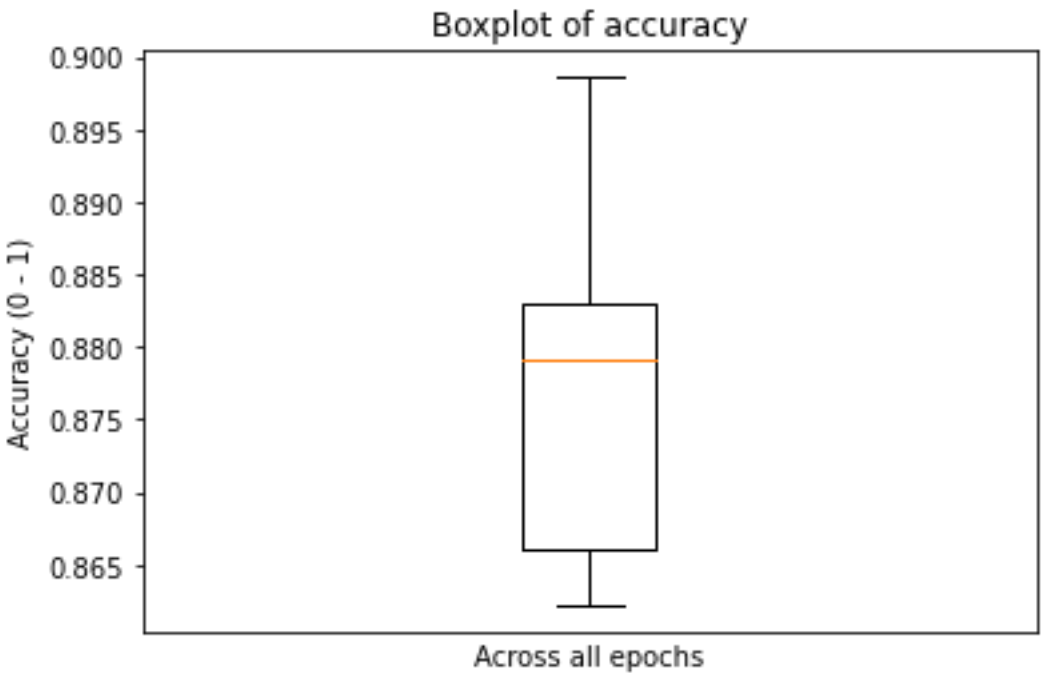
The model is no longer overfitting as before as has a variance less than 4% whilst still smashing the target set. As seen overleaf, the training and validation accuracy track each other quite closely and appear to begin a flatline at the end of training. The loss function diverges slightly as the curve flattens after 15 epochs and becomes increasingly erratic despite holding relatively flat (not increasing). Comparison of the training and validation averages against the project targets show that this is still a low bias, low variance model that exceeds the training targets:

	Error Rate	Difference	Comments
High Target	10%	-10%	
Threshold	20%	-	
Training	8.7%	-11.3%	Better than the high target
Validation	12.3%	-7.7%	Better than threshold and nearly as high as high target. Excellent results.

A full breakdown of the data can be found in the accompanying document "Part B – Appendix – Supplementary Results".



Figures 10: Training and Validation performance for the advanced model



Figures 11: Overall boxplot of accuracy across all epochs / folds

It can be seen from the boxplot that the model is relatively consistent towards the upper quartile with several low performing models dragging the lower quartile down. This could be further evidence of the erratic loss type seen after the 15th epoch.

```
Accuracy: mean=87.710 std=1.130, k-folds=10
```

10.KEY FINDINGS

There is no doubt that the additional of phased-value dropout layers together with batch normalisation cured the variance issue that the baseline model was suffering from. Without performing more experiments, it is hard to know if the model needs to be so deep or if a few layers could be removed. The gradually increasing dropout values came from a tutorial by Dr Jason Brownlee [7] after failing to make ground towards the high 80% accuracy level with various static values (ranging from 0.2 to 0.5). Changing from simple k-fold to stratified k-fold also had a positive impact on early experiments at all levels as the this ensured that the training data was evenly split across the categories.

Mixing weight decay regularisation with dropout regularisation can have disastrous consequences as one experiment took an 80% performing model down to below 20%. Overly regulating a high dropout model did not seem to be a wise option as it made it excessively difficult for the network to learn. The choice of optimiser had a bigger impact than expected. Training deeper networks and longer together with regularisation techniques can have a massive impact on the model performance however this often comes at a time cost.

10.1. TAKEAWAYS FOR FUTURE PROJECTS

During this first attempt at building an image classification model there were a few key learning points that would assist a smoother development process for the next model:

- The choice of optimiser and regularisation techniques were game changers.
- Batch normalisation and dropout layers help with taming high variance.
- Do not mix dropout and weight-decay without first testing on quickly trained models.
- The verbose setting is very helpful in gaining intuition on the training process and model performance.
- Setting up callbacks to ensure that only the best model (by validation loss) is saved during training is a great way to maximise the results of lengthy and intense training schedules and produce a high quality model.
- GPUs are a vital hardware addition on complex models.
- Whilst Jupyter Notebooks and Colab are excellent tools for fast prototyping they do hinder flexibility somewhat for more advanced coding practices.
- Keras is a great interface for prototyping but with time it would be better to become more familiar with tensorflow for the extra control and flexibility it provides.

11.CONCLUSIONS

Overall, this was an interesting and challenging project. Left to our devices to come up with something that works and to be able to understand and enact code from multiple sources without making it look like it was thrown together really does force the programmer to understand what they are doing. Whilst a great piece of code can appear to solve the trickiest of problems it can create other problems if it does not match the overall design of the system.

It is encouraging that solutions can be put together relatively quickly but thought must be given to the size and complexity of the network in relation to the available resources. Choosing a heavy hitting model may seem like a good idea at the outset but performance issues can add to the general stress if under time pressure and a free service provider such as Colab decides to limit access to resources. Understanding why the network performs the way it does is critical to making progress with training a good model and it is imperative that the model designer has an awareness of a sensible benchmark or target to aim for to compare performance and identify if the model is suffering from high bias / variance.

Having performed many experiments on many model designs and hyperparameters I feel that it is essential to keep an open mind and experiment whilst learning from the experts. As mentioned by Dr. Jason Adair in the data analytics course, you are only limited by your imagination.

12. REFERENCES

[1] M. Zhou, Comparison of CNN based and self-similarity based denoising methods. *Stanford University* http://stanford.edu/class/ee367/Winter2020/report/zhou_m_report.pdf

[2] Fan, L., Zhang, F., Fan, H. *et al.* Brief review of image denoising techniques. *Vis. Comput. Ind. Biomed. Art* **2,** 7 (2019). https://doi.org/10.1186/s42492-019-0016-7

[3] R. Gonzalez, R. Woods, Digital Image Processing. Pearson 4th Edition. Coursebook

[4] Wang, Z et al. Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing, 13, 600-612. https://ece.uwaterloo.ca/~z70wang/publications/ssim.pdf, DOI:10.1109/TIP.2003.819861

[5] A. Krizhevsky, CIFAR-10 homepage. *University of Toronto Website* https://www.cs.toronto.edu/~kriz/cifar.html

[6] A. Krizhevsky, Learning Multiple Layers of Features from Tiny Images, 2009. *Online book* https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[7] J. Brownlee, online tutorials, www.machinelearningmastery.com

[8] A. Ng, Deep Learning Specialisation course, *DeepLearning.AI* http://www.deeplearning.ai

[9] A. Ng, A Chat with Andrew on MLOps: From Model-centric to Data-centric AI. *Online lecture* https://www.youtube.com/watch?v=06-AZXmwHjo