Deep Learning

Assignment 2 – Convolutional Networks

Michael McAleer

R00143621

Part A - Part 1

Four variants of Convolutional Neural Network (CNN) were investigated, the initial baseline CNN consisting off a single convolutional layer and a single layer, and three additional CNNs with varying amounts of convolutional layers and fully connected (FC) dense layers (table 1).

CNN	Conv. Layers	Filter Count			Fully Connected Layers	Neuron Count		
Baseline	1	128			-	-		
Variant 1	1	128			1	256		
Variant 2	3	32	64	128	3	256	128	64
Variant 3	3	32	64	128	3	256	128	64

Table 1: CNN variations explored

Each of the CNNs were run for fifty epochs, with the training and validation loss and accuracy results recorded after each epoch. From the first two CNNs it is evident that the model is overfitting very quickly, with both CNNs validation loss rising after only ten epochs and continuing to rise throughout the remaining epochs. The validation accuracy of both CNNs does not increase any further after a similar number of epochs. This behaviour can be attributed also to overfitting on the training data as evident by the accuracy values getting close to 1.0 with almost no loss value recorded.

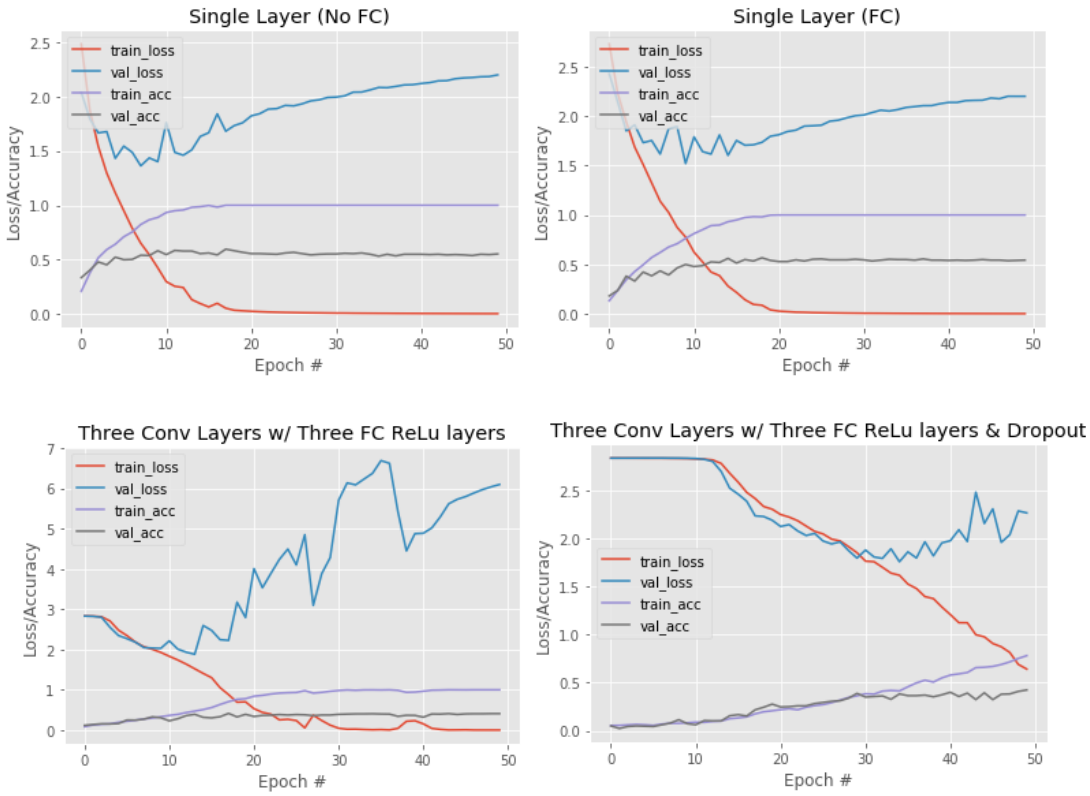


Figure 1: Training and validation results from all CNN variants investigated in part one

The next two CNNs, variants two and three, with three convolutional layers respectively present interesting results. Variant 2 with no dropout implemented shows even more severe overfitting on the

training data with the validation loss climbing to almost 7.0. The loss values for the CNN with dropout implemented demonstrate the impact of regularisation in a CNN which exhibits overfitting. The effect of removing neurons from the FC layers both reduces the number of epochs completed before the loss value starts to rise and is significantly lower throughout all epochs than the CNN without dropout. Instead of the validation accuracy improving with more convolutional and FC layers, the opposite is seen with lower validation scores across both three-layer CNNs

	Training Loss			Training Acc.			Validation Loss.			Validation Acc.		
Epoch #	**10**	**25**	**50**	**10**	**25**	**50**	**10**	**25**	**50**	**10**	**25**	**50**
Baseline	.42	.01	.004	.88	1.0	1.0	1.4	1.91	2.19	.58	.56	.55
Variant 1	.77	.01	.008	.76	1.0	1.0	1.5	1.9	2.2	.5	.55	0.54
Variant 2	1.9	.26	.3-04	.33	.92	1.0	2.02	4.49	6.09	0.30	0.37	.4
Variant 3	2.82	2.07	.64	.08	.26	.78	2.83	2.05	2.26	.07	.28	.42

Table 2: The training and validation loss and accuracy results at three checkpoints throughout training process

When looking at the performance of all four CNNs, it was not expected that the worst performing would be a CNN with multiple convolutional layers. A combination of small training data set and overfitting on that data set over a number of epochs are the two most contributory factors to the poor performance. It can be surmised that the single layer CNN performed better as there was not as much of an opportunity for the data to overfit.

Part 2 – Data Augmentation

An option when faced with a small training dataset is to augment the training data, meaning perform various manipulations on the data to create similar copies with varying degrees of height, width, zoom etc. This makes sense to address the issue in this fashion, with CNNs determining how to classify data based on patterns, the more examples of the patterns in the data, even if just very slightly altered, provide a way for the CNN to train itself on all possible minute variations of a given class.

To address the issue of the small amount of training data in the flower data set augmentation techniques were applied, multiple variants of augmentation were implemented to determine the most successful combination (table 3).

	Height	Width	Shear	Zoom	Rotation	Vertical Flip	Horizontal Flip	Dropout
Baseline	0.1	0.1	0.2	0.3	20	True	False	No
Increase	0.3	0.3	0.3	0.5	45	True	True	No
Decrease	0.1	0.1	0.1	0.1	10	True	True	No
Combo	0.1	0.1	0.2	0.4	45	True	True	0.1

Table 3: Augmentation scale values for various input parameters

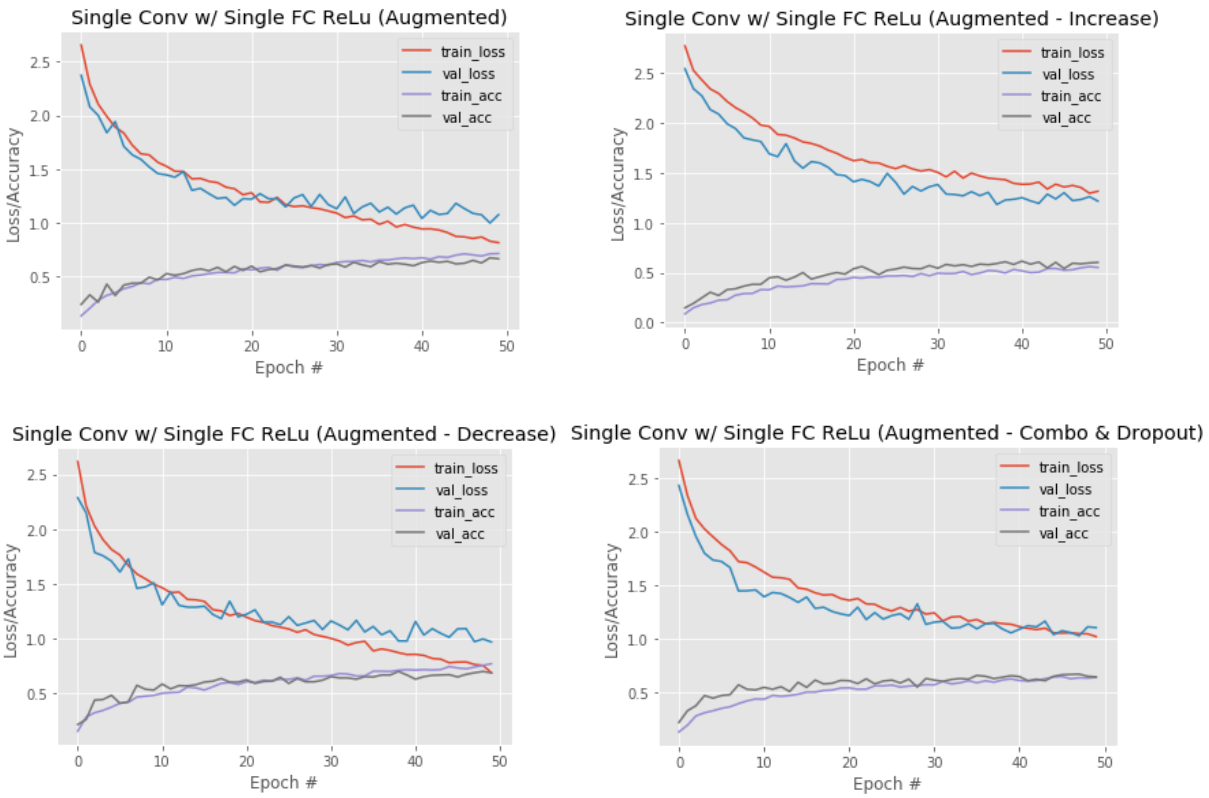


Figure 2: *Training and validation results from all data augmentation variants investigated in part two*

The baseline used in the data augmentation analysis is extracted from the example used in the lecture notes, from there the values were increased, decreased, and a combination of both used. The results from all these tests can be seen in figure 2. Not surprisingly the data augmentation has a positive effect on the validation loss and accuracy achieved by the CNN. The most successful augmentation profile, without dropout applied, is that with the lowest input values for manipulation scale. This makes sense as CNNs training on flower classification would rely heavily on small details within the flowers, manipulating or distorting these images too much is likely changing them to the point of not being useful as training data. With flower classification, parameters such as shear should be kept as low as possible as they are 'destructive' on portions of an image, however others such as height, width, zoom, and rotation may be explored further as these would not impact the boundary integrity of the flowers themselves within the images.

Applying dropout with data augmentation implemented adds additional noise to the training process, providing more desirable loss values whilst not impacting the accuracy score in any great manner. From the trendlines in the graph in figure 2 representing augmentation with dropout, it can be seen that the loss value is still continuing to slightly decrease even after fifty epochs, so if lower loss values are a key factor of model then it would make sense to increase the epoch counts until the validation loss converges at its global minimum as the training time is so short.

Part 3 – CNN Ensembles

Using the best performing CNN network so far, the CNN from part-2 with the decreased data augmentation values (dropout applied in addition to previous configuration), a CNN ensemble was constructed that would train the same fixed-structure CNN multiple times to gain multiple model predictions to improve the classification by averaging predictions and retrieving the largest predicted value for each validation image across all models. Table 4 shows the results of the CNN ensemble.

Model #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Acc	0.6794	0.6412	0.6235	0.6529	0.5588	0.6971	0.6588	0.7088	0.6529	0.6706
Final Ensemble Accuracy			**0.7029**							

Table 4: CNN Ensemble validation accuracy values after the training process has finished for each model

Whilst the CNN Ensemble does not provide a final validation accuracy higher than all the models predicted, it does provide an accuracy very close to the best accuracy achieved overall. This lack of an improvement may be attributed to the data set being still small even with augmentation and a lack of diversity as all the base-models were identical except for the initialisation of their weights when the networks are first compiled. A lack of diversity in an ensemble is an issue as all the values being averages are so similar, meaning that the process of averaging all model predictions presents the possibility of the same class being selected across all models.

Although dropout was implemented in this ensemble, the rate was set at 0.1 so the regularisation impact is minimal. To further increase diversity in the ensemble, the most obvious thing to do would be to have a variation of models. With each model there could be varying layer counts, filter counts, FC neuron counts, dropout rates, regularisation types (L1, L2, or both), augmentation severities, kernel sizes, pool sizes, and much more. In essence, there is an almost infinite amount of combinations of models that could be included in an ensemble, even if the only differences between models is the hyper-parameter values. From this early stage in use of CNN ensembles, it can be assumed that a good approach would be to tinker with multiple models until a series of strong candidate models are apparent, these could then make up the ensemble and provide a good framework from which to continue.

Part B – Transfer Learning

Part 1 – Feature Extraction

When exploring feature extraction in CNNs, a number of pre-trained CNN models and machine learning (ML) classifier algorithms were tested, table 5 includes all variants. A wide range of ML classifiers were examined as they all exhibit very different behaviour when determining the output classification. Note that the table is for listing only, it does not reflect the tests in which all ML classifiers were tested with all pre-trained CNNs.

Pre-Trained CNN	ML Classifier
VGG16	AdaBoost
VGG19	Gaussian Naive Bayes
Xception	k-Nearest Neighbours (kNN)
InceptionV3	Logistic Regression
ResNet50	Multi-layer Perceptron (MLP)
	Quadratic Discriminant Analysis
	Random Forest (RFC)
	Stochastic Gradient Descent (SCG)
	C-Support Vector (SVC)

Table 5: Pre-trained CNNs and ML classifiers used in feature extraction testing

AdaBoost (Adaptive Boosting) – AdaBoost is a member of a collection of boosting algorithms which have the single aim of producing highly accurate classifiers. Although they are generally low in their individual accuracy, when combined with other low accuracy algorithms in an ensemble they have the ability of tracking failed predictions in models. They also have the added benefit of not being affected by the overfitting problem. AdaBoost combines multiple classifiers top increase the overall accuracy of the classifiers by building a strong classifier using multiple poorly performing classifiers. The basic concept of AdaBoost is to set weights of classifiers in each iteration of the training phase that it tries to accurately predict unusual observations. AdaBoost was selected to in this test to determine if this boosting technique could provide higher accuracy rates than other non-boosted classifiers. Unfortunately this did not materialise and the AdaBoost classifier resulted in very low accuracy rates of approximately 10-20% correct predictions. The poor results returned may be attributed to the incorrect usage of the classifier, that is, not within a CNN ensemble where it can get the opportunity to combine multiple poor predictions to get a stronger more accurate end-prediction.

Gaussian Naïve Bayes – Gaussian Naïve Bayes (GNB) is a member of the Naïve Bayes classifiers, based on the Bayes' theorem but focusing on gaussian distribution of data as opposed to the Multinomial or Bernoulli variants. As described eloquently on Hyper Physics [1], gaussian distribution is a continuous function which approximates the exact binomial distribution of events, often described as a "bell-shaped

curve". One drawback of GNB is that it works best when the dataset it is working on is very large, which would explain why its performance with the various pre-trained CNN models was not impressive, the highest accuracy rate achieved was 67.9% on the InceptionV3 model, the lowest being 20.2% on the ResNet50 model. It is likely that there was just not enough samples of each class in the training data to give GNB the best opportunity to perform well.

k-Nearest Neighbours – The k-Nearest Neighbours (kNN) algorithm is very simple yet very successful in its operation on determining the output class (or regression value) of a given data point based on the k-nearest neighbours in the feature space. In this test, the results were non-weighted and only the 3 nearest neighbours were taken into consideration when voting. Results from the various pre-trained CNN models with kNN classification were not as accurate as anticipated, with accuracy values in the 63-67% range across all models except ResNet50 which seen an accuracy value of 29%. These numbers could likely be improved with fine-tuning the hyper-parameters for the number of nearest neighbours taken into consideration in the vote and if those neighbour distances should be used to determine weighted voting.

Logistic Regression – Logistic Regression (LR) sees data fit into a linear regression model which is then acted upon by a logistic function to predict the target categorical dependent value [2]. In the case of the flowers data set multinomial LR is used as the dataset contains more than two target categories which are not ordered. In terms of performance, LR can be seen as the most consistent of all algorithms tested, with accuracy values in the 82-88% range (except ResNet50 again with 55.8%) and the top of all individual classifier results per pre-trained CNN. It should be noted that in the results, whilst the MLP classifier came out on top with an accuracy score of 87.6% when running the full test program, when the first test was run with only the LR classifier in-use with the VGG16 model, the accuracy then was 88.2%, the highest accuracy seen across all tests so far.

Multi-Layer Perceptron – The Multi-Layer Perceptron (MLP) classifier is the only classifier used in the tests which is designed specifically for use in deep learning models. MLP works by training on the data set and determining correlations between the inputs and outputs, adjusting the weights and bias values of the model to minimise error. MLP is a feed forward network, meaning they implement both forward and backward passes, in forward passes the decision of the output is determined, on backward passes the partial derivative of the error function with reference to the weights and biases that are updated using gradient descent with the intention of getting closer to the error minimum. The MLP classifier was one of the strongest classifiers tested, with accuracy values putting it in the top 3 of all pre-trained model results, and in the combined run of tests it achieved the highest accuracy with the VGG16 model with 87.6%. The success of the MLP algorithm is likely as a result of its feed forward behaviour, with the forward and backward passes updating the weights and biases more effectively than other classifiers.

Quadratic Discriminant Analysis - Quadratic Discriminant Analysis (QDA) is very similar in behaviour than the Gaussian Naïve Bayes classifier in that they both use Baye's rule and Gaussian density to each class, the key difference being QDA using a quadratic decision boundary assuming that the observations

from each class are drawn from Gaussian distribution. Discriminant analysis is used to determine which variables discriminate between two or more natural occurring groups [3]. QDA was by the worst performing of all the classifiers tested, consistently ranking at the bottom for every pre-trained CNN. The impact of a small data set on the GNB classifier may be compounded here with the QDA classifier, with not enough data for the classifier to make clear determination between observations of the various classes in the dataset.

Random Forest – Random Forest Classifier (RFC) operate in a similar fashion to the Decision Tree classifier in that decision forests are created at training-time and outputting a class prediction, the difference is that the RFC constructs multiple random decision tree forests and outputs the mode of the class predictions (note, in regression tasks the output would be a mean prediction). The random nature of the RFC also helps to prevent overfitting on deep networks as random feature subsets are created during the decision tree creation, but this also has the negative impact of slowing down the classification process due to the extra computational power required. Within the tests run, RDF presented average results, with accuracy values in the 77%-81% range. It is difficult to determine why the RFC did not perform as well as expected as it is a very widely used algorithm in machine learning due to its simplicity, however with some fine-tuning of the classifier hyper-parameters on the best performing model it is likely that this accuracy could be further improved. The impact of the small training dataset must also be taken into consideration, it is possible the lack of data and random nature of the classifier impacted the output predictions more than anticipated.

Stochastic Gradient Descent - Stochastic Gradient Descent (SGD) classifier implements regularised linear models such as logistic regression and support vector machine (SVM) with SGD training, where the gradient of the loss is estimated for each sample and the model is updated with a decreasing learning rate. Unlike the LR classifier which SGD uses in its operation, it did not result in similar high accuracy values with ranges in the 75%-85% range (except ResNet50 with 17%). The SGD classifier is one of the most parameter heavy classifiers implemented in these tests, so it is likely that the default values used were not as successful as what may have been achieved with fine-tuning to bring it more in line with LR classifier results.

C-Support Vector - C-Support Vector (SVC) is a variant of the Support Vector Machine (SVM) classifier with the C value taken into consideration, where C is the penalty parameter of the error term. What is means is that a large value for C will results in a small margin of error, a low value will result in a large margin of error. There is no one correct value for the C value, it is largely dependent on the dataset, so assistive techniques such as gridsearchCV can be implemented to help determine the best C value for a dataset. Using a low C value of 0.025 to allow for a large margin of error and a linear kernel for class determination, the SVC classifier produced some very strong accuracy results, consistently placed in the top three performing classifiers across all models, and in the top 3 overall classifiers across all classifiers and models with an accuracy of 86.4% with the VGG16 model. The success of this classifier is likely

attributed to the larger margin of error afforded in the penalty of the error calculation but working in favour in this instance where it results in higher successful prediction.

Part B – Part 2 – Fine-Tuning

For the fine-tuning process the VGG16 pre-trained CNN model was used as it was the best-performing in previous tests with transfer learning, the top three overall accuracy results were all achieved using this model.

When fine-tuning a number of tests were run which featured alterations to model parameters such as:

- unfreezing pre-trained model layers – iteratively unfreezing layers, in cases more than one, with each phase of training on the model
- removing layers and defining new model output layers – defining new output layers yielded higher than expected accuracy results in part A so worth looking at with more robust training of weights and bias incorporated
- adding new FC layers – adding one or more new FC layers with varying amounts of neurons in each
- dropout regularisation – applying dropout regularisation of varying rates after each FC layer to mitigate the impact of overfitting and noise introduced with augmented training data
- changing the optimiser used in the model – various optimisers were tested to determine which performed best and if one optimiser could excel at optimisation over another, for example using boosting algorithm AdaDelta with adaptive learning rates
- data augmentation – augmenting the training data to provide more data for better output class determination in the validation data, the settings used were the best variant from the data augmentation tasks earlier, the 'decreased' test settings
- skipping phase 'A' and going straight to phase 'B' in the fine-tuning process
    - this test was born out of incorrect interpretation of the requirements of fine-tuning, but resulted in accuracy values considerably higher than any model implemented throughout both Parts A & B of this assignment
- ensemble the CNN models to achieve higher overall prediction accuracy

Test 1: Fine tune from 5th convolutional layer with only single new FC layer of ReLu neurons

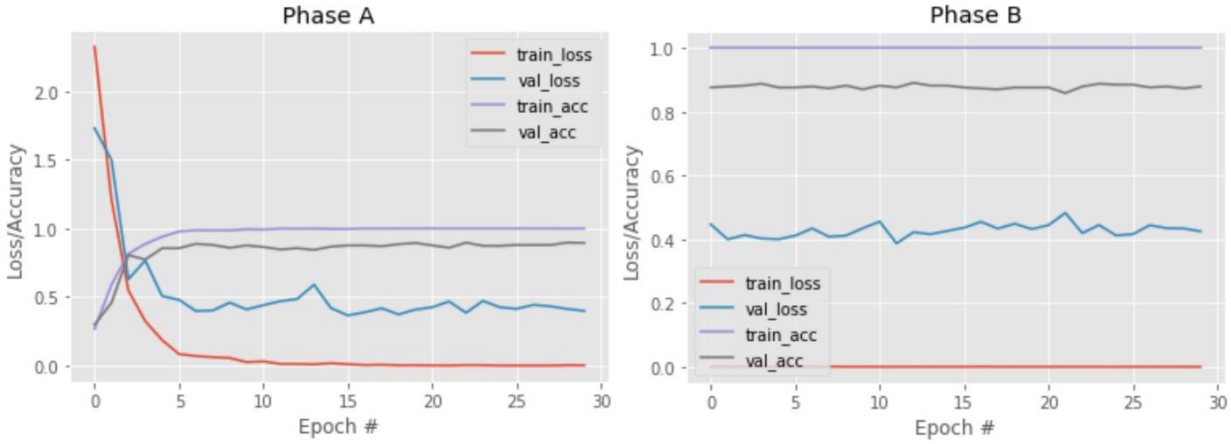
Results from test 1 show the model converging very quickly in phase 'A', with the validation accuracy and loss levelling off around the 5th epoch and values remaining consistent out with no evidence of overfitting. Unfreezing the 5th layer for phase 'B' and training those layer weights does not have any tangible impact on the model accuracy, instead of rising the both the validation loss and accuracy decrease slightly. Throughout the entirety of phase 'B' all loss and accuracy values remain consistent with no improvement

seen. In effect, all training after the 10th epoch in phase 'A' are unnecessary with no noticeable improvements seen.

Test 1 Settings:

Optimiser	AdaDelta
Loss	Sparse Categorical Cross Entropy
FC Layers Added	1
FC Layer ReLu Neurons	256
Phase 'B' Trainable Layer	Block5_conv1
Dropout Rate	0.1
Epochs	30

Test 1 Results:



Phase	Loss	Accuracy
A	0.3989910822306924	0.89411765
B	0.42533477389746727	0.87941176

Test 2: Fine tune with new FC layer of ReLu neurons and make 5th layer trainable followed by 4th layer

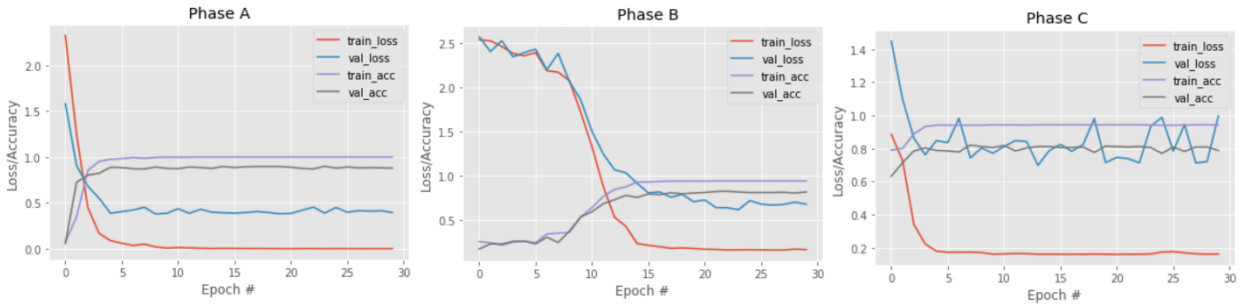
The same setting for test 1 were implemented in test 2, except for this time Adam optimiser was used with the addition of another phase and trainable layer which would be made trainable in the new phase. The drastically different results from test 1 can be attributed to the optimiser alone, with the AdaDelta optimiser performing better across phases in the earlier epochs, the Adam optimiser in all phases goes through a brief period at the start where it takes a number of epochs for the newly trainable layer to learn the features of the training data. As each phase is completed, the impact on the validation loss and

accuracy values is negative, with both the loss increasing and the accuracy decreasing. This could be explained by the VGG16 pre-trained model weights moving further and further from the ImageNet weights it began with and losing its strong accuracy prediction abilities.

Test 2 Settings:

Optimiser	Adam
Loss	Sparse Categorical Cross Entropy
FC Layers Added	1
FC Layer ReLu Neurons	256
Phase B & C Trainable Layer	Block5_conv1 (B), Block4_conv1 (C)
Dropout Rate	0.1
Epochs	30

Test 2 Results:



Phase	Loss	Accuracy
A	0.39547875462209475	0.87941176
B	0.6796931485727649	0.81764704
C	0.9960503981393927	0.7882353

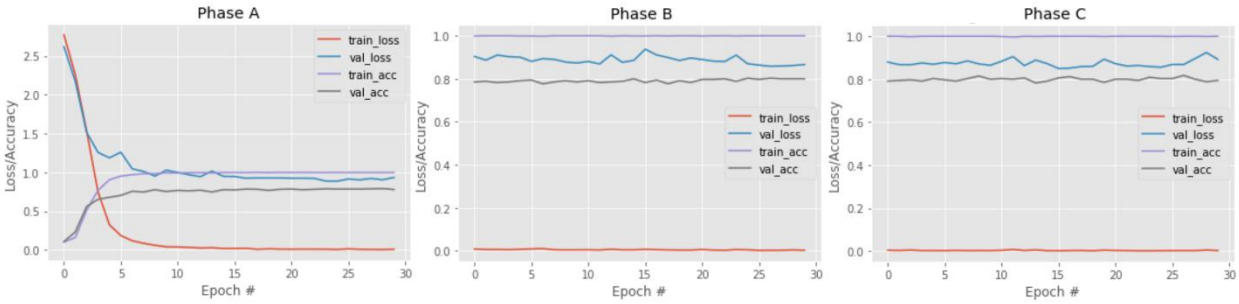
Test 3: Fine tune with new FC layer of ReLu neurons, remove 5th layer and make 4th layer trainable followed by 3rd layer

Test 3 attempted to replicate the success with removing later layers in the pre-trained CNN and outputting from an earlier layer, whilst making the two last layers trainable iteratively as the phases progress. Whilst the results from Phase A did present similar results to earlier tests with removing layers from the CNN, the validation loss and accuracy only improved slightly but the trends show no consistent improvement, with convergence being reached halfway through the first phase.

Test 3 Settings:

Optimiser	SGD
Loss	Sparse Categorical Cross Entropy
FC Layers Added	1
FC Layer ReLu Neurons	256
Phase B & C Trainable Layer	Block4_conv1 (B), Block3_conv1 (C)
Model Output Layer	Block4_pool
Dropout Rate	0.1
Epochs	30

Test 3 Results:



Phase	Loss	Accuracy
A	0.9336114673053517	0.7794118
B	0.866312014355379	0.8
C	0.8925463683464948	0.7941176

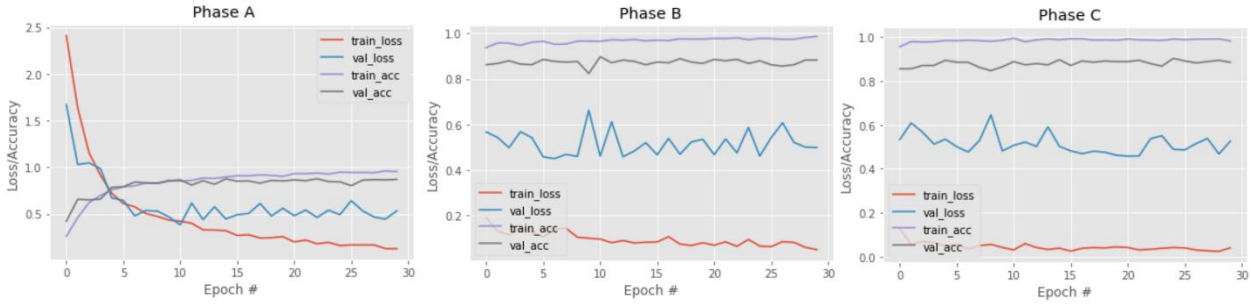
Test 4: Using augmented training data fine tune with new FC layer of ReLu neurons and make 5th layer trainable followed by 4th layer

Augmenting the data in the fine-tuning process helped considerably with overfitting on the training data, with validation loss half of what was seen in models without data augmentation. The augmentation process did introduce noise into the model as evident by the loss spiking during the progression of epochs in each phase. Increasing the dropout rate within the model may have negated the effect of the noise in the model. Like previous models, there was no discernible improvement in validation loss or accuracy provided by including additional phases in the fine-tuning process and the model converging early on in the first phase of the training process.

Test 4 Settings:

Optimiser	AdaDelta
Loss	Sparse Categorical Cross Entropy
FC Layers Added	1
FC Layer ReLu Neurons	256
Phase B & C Trainable Layer	Block5_conv1 (B), Block4_conv1 (C)
Dropout Rate	0.1
Data Augmentation	True
Epochs	30

Test 4 Results:



Phase	Loss	Accuracy
A	0.5331715863536705	0.87058824
B	0.4977473539977588	0.88235295
C	0.5255700976016339	0.88529414

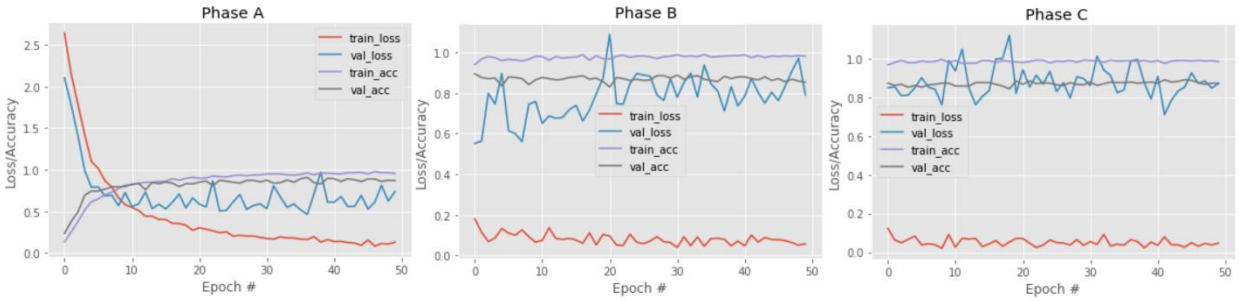
Test 5: Using augmented training data fine tune with three FC layer of ReLu neurons and make 5th layer trainable followed by 4th layer

Introducing more FC layers of ReLu neurons into the fine-tuning process did not have any impact on the validation accuracy and only served to compound the effect of overfitting on the training data. The behaviour of this model is similar to that of test 4, with convergence being reached early on in the first phase, but in all subsequent phases the model continued to overfit on the training data as the epochs progressed. Varying rates of dropout to match the size of the FC layer it is applied to would likely negate the overfitting behaviour.

Test 5 Settings:

Optimiser	AdaDelta
Loss	Sparse Categorical Cross Entropy
FC Layers Added	3
FC Layer ReLu Neurons	1024, 512, 256
Phase B & C Trainable Layer	Block5_conv1 (B), Block4_conv1 (C)
Dropout Rate	0.1
Data Augmentation	True
Epochs	50

Test 5 Results:



Phase	Loss	Accuracy
A	0.7385673554275524	0.86764705
B	0.7908998610536483	0.85294116
C	0.8723017079067755	0.87352943

Test 6: Using augmented training data fine tune with three FC layer of ReLu neurons with dropout regularisation and make 5th layer trainable followed by 4th layer

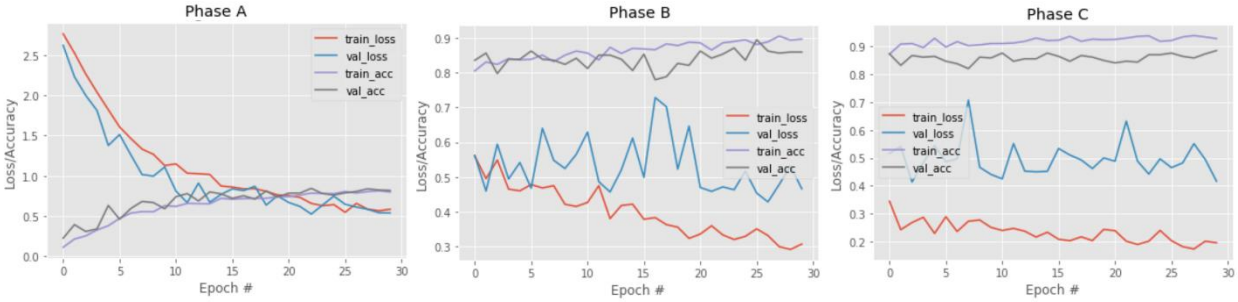
Changing the optimiser from AdaDelta in test 5 to SGD in test 6 had a very noticeable impact on the validation loss, not only was it lower after the first phase by .2, it continued to get lower after each phase. The results are a slight improvement on the best performing model so far, test 4 with only one FC layer and AdaDelta optimiser, with the loss values being lower whilst the accuracy was identical.

Test 6 Settings:

Optimiser	SGD
Loss	Sparse Categorical Cross Entropy
FC Layers Added	3

FC Layer ReLu Neurons	1024, 512, 256
Phase B & C Trainable Layer	Block5_conv1 (B), Block4_conv1 (C)
Dropout Rate	0.1
Data Augmentation	True
Epochs	30

Test 6 Results:



Phase	Loss	Accuracy
A	0.5400469526648521	0.81764704
B	0.4663992456414483	0.85882354
C	0.4164209474216808	0.88529414

Test 7: Further fine-tuning experiments where 'phase A' is skipped and layers are made trainable in first pass

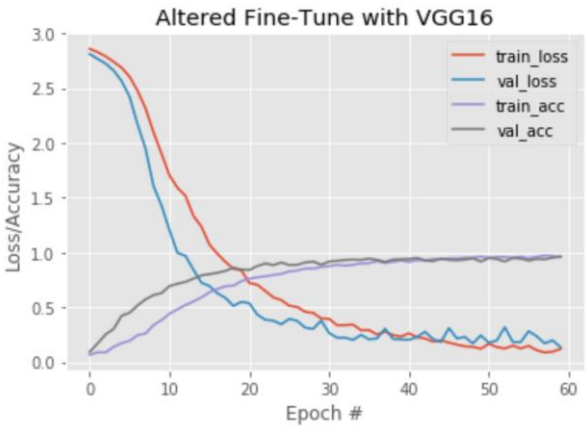
As mentioned in the section introduction, another test was born out of a misinterpretation of the fine-tuning process. The misinterpretation was phase A of the fine-tuning process, where the pre-trained model layers are left untrainable and the FC layer is the only layer with weights and bias being updated. In this test phase A was skipped and the fine-tuning process went straight to phase B and unlocked the last layer, so weights and bias could be updated. In addition, to address overfitting on the training data as epochs progress, a varying rate of dropout was applied, with the largest level of dropout applied to the FC layer with the greatest number of neurons, and the lowest dropout to the layer with the least neurons. This approach meant there was only one phase, there was no repeated passes of the model with further layers unlocked, there was only one pass with the last layer unlocked for training the weights and bias. The effect of this was astounding when compared to the performance of all tests run throughout this section. Not only did the accuracy jump from an average of 88% to 96.47%, the loss dropped further to 0.1352. the best recorded loss previous to this was 0.4164 in test 6. Although this model took longer to converge than previous tests due to the decreased learning rate, converging with these results after only forty epochs and one single pass is impressive. Overfitting has also been eliminated with the help of the

varied dropout rate as assumed in test 5 which had the same model layout but a fixed dropout rate across all FC layers.

Test 7 Settings:

Optimiser	SGD (lr=0.001)
Loss	Sparse Categorical Cross Entropy
FC Layers Added	3
FC Layer ReLu Neurons	1024, 512, 256
Trainable Layer	Block4_conv1
Dropout Rate	0.3, 0.2, 0.1
Data Augmentation	True
Epochs	60

Test 7 Results:



Epoch	Validation Loss	Validation Accuracy
1	2.8113	0.0941
10	1.4276	0.6294
20	0.5503	0.8441
30	0.3791	0.8882
40	0.2075	0.9382
50	0.2440	0.9176
60	0.1352	0.9647

Test 8: Do the same as above in test 7 with the tweaked fine-tuned network, this time run it through an ensemble network where all base-learners have same structure.

The same network from test 7 was put through a CNN ensemble, to determine if the accuracy achieved in test 7 could be further improved. Whilst it was not improved with 10 base-learners, the advantage of CNN ensembles is apparent here as the final ensemble accuracy is higher than any single accuracy achieved by any of the base-learners.

Model #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Acc	0.9265	0.9382	0.9412	0.9324	0.9500	0.9382	0.9324	0.9441	0.9441	0.9353
Final Ensemble Accuracy			**0.9559**							

Part C – Adversarial Machine Learning

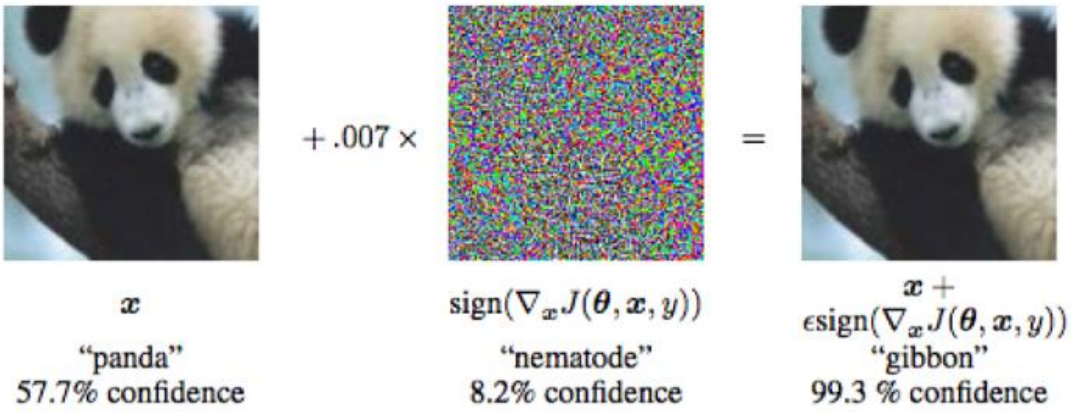
Existing Neural Networks (NN) work very well in some scenarios but not all, and they are not as robust as we can assume them to be. In classification problems using NNs, if a few pixel values are changed, a model with a very high accuracy rate may predict an entirely different class because of this change. This makes it possible to easily corrupt a model that would have been previously thought to be very accurate. This is what is known as adversarial attacks.

Adversarial attacks were identified by Ian Goodfellow in his 2014 paper 'Explaining and Harnessing Adversarial Examples' [4] whereby he argues that the primary cause of NN misclassification of adversarial examples is not as a cause of nonlinearity and overfitting as previously thought, but instead their vulnerability to adversarial perturbations is their linear nature.

Images can be engineered to fool NNs using evolutionary techniques to create fake images that may look like an object to the human eye but changed in such a way to impact the output layers in a NN to achieve an entirely different output prediction. One of the methods used to create these adversarial examples is a type of network called a Compositional Pattern Producing Network (CPPN), a type of augmenting topology neural network [5]. CPPNs allow each node to contain a unique activation function, allowing the model to exhibit properties of all the different activation functions and create output that is patterned, symmetrical and unique. They work by implementing the following process:

1. Take a real-world image (image X for explanatory purposes), pass it through a CNN to an output prediction of 55.7% accuracy of a panda bear.
2. Using this same image, select a random neuron from the output layer that is different from the neuron that picks panda bear.
3. Apply gradient descent to the input pixels in that image in order to minimise the classification accuracy and loss with respect to the newly chose output accuracy. Instead of optimising weight in order to optimise the classifier which is the expected training behaviour in a CNN, we adjust the input pixels until they food the neural network into making a wrong output classification prediction.
4. Final step is to make sure that the artificially generated synthetic image looks as close to the original image X so it is not possible to see the difference between the two.

There are many methods implemented to make use of these CPPNs, one of the most common is Fast Gradient Sign Method (FGSM) [6]. FGSM calculates gradient descent with respect to the input image pixels and then applies a sin operation to that gradient matrix. What this matrix is calculating is for every single pixel in the input image, what would happen to the activation of my target output neuron if I increase or decrease the value of that pixel. FGSM then takes the threshold gradient matrix, multiplies it by some small number and adds it to the input image. The image [4] below gives a visual representation of this process. To keep the adversarial image as close as possible to the input image, the intention is to find the lowest possible value to use in the multiplication process that leads to a misclassification.



All deep NNs are impacted by adversarial attacks, the same as machine learning algorithms. In a lot of cases, an adversarial attack which impacts one NN is very likely to impact other NNs, this is known as cross-technique transferability [7]. Attackers are aware of this, and although they do not have access to the models to get the gradient, there are methods they can employ to determine the gradient in the model. Cloud providers such as Google, AWS, and Microsoft give users access to their own built NNs for image classification, these tools display an output prediction with probabilities included, so an attacker need only change pixels in the image and feed it through the NN again and measure the change in the output probability values to determine the gradient descent value of the NN. Unfortunately, hiding the output probability does not protect against adversarial attacks. There are broadly speaking two types of adversarial attacks:

- White box – where the gradient is known
- Black box – where the gradient is not known

The idea of black box adversarial attack is to use an unknown gradient NN to label your own training data. A collection of images, real of synthetic, and use the labels that are given back from the unknown NN as training data from which to train your own model to simulate what the unknown NN is doing internally. This is the impact of cross-technique transferability described earlier, if you can use your own local model to generate an adversarial attack, then there is a very high likelihood that the same attack will have the same impact on other unknown NNs, this is why they are so hard to defend against. Even adding noise to an image can inadvertently cause an adversarial attack by causing the NN to cross a prediction boundary thus causing the max likelihood label to change.

Lu Et Al. in their 2017 paper suggested that real-time video was not impacted by adversarial attacks sue to moving images and objects with varying angles in real-time video, this has since been proven to be an incorrect assumption. Kurakin Et Al. [8] demonstrated that generating an adversarial attack by altering an image, then printing it, then taking a picture of the printed image, did not change the impact of the image

when fed through a NN, the adversarial attack persisted throughout all copies of the original image. Other intriguing examples are the case of the 3D printed turtle that is predicted as a rifle from all angles when fed through a real-time image classifier [10] and the security researchers who were able to get the self-driving Tesla to switch lanes into incoming traffic because of a sticker they printed and put on a stop sign [11].

There are two accepted defence mechanisms against adversarial attacks:

- Reactive Strategy – next to your normal classifier you will train a second network that simply tries to detect adversarial examples, so when an adversarial attack is detected, that image is rejected. This is not an ideal solution as implementing two models and classifiers means double the computation overhead and infrastructure to run them.
- Proactive Strategy – this approach involves including adversarial examples in the training process. It has been found that this approach not only helps defend against adversarial attacks, it improves prediction accuracy rates on normal examples.

Adversarial examples provide opportunities for research to improve existing models and classifiers, new research has indicated that adversarial examples can be directly attributed to the presence of non-robust features, and after capturing these features their widespread existence in standard datasets was established [12]. Fixing issues with adversarial examples are very important for models and classifiers we have in operation today across many industries and fixing these may introduce a whole new range of machine learning applications. The research presented by Ilyas Et Al. [12] may yet go a long way to improving the robustness of modern-day NNs making them much harder to manipulate.

References

[1] Rohlf, Gaussian Distribution Function, Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/Math/gaufcn.html (Accessed: 19th May 2020)

[2] Saishruthi Swaminathan (15th March 2018) Logistic Regression—Detailed Overview, Available at: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc (Accessed: 19th May 2020)

[3] RapidMiner Studio Core, Quadratic Discriminant Analysis, Available at: https://docs.rapidminer.com/latest/studio/operators/modeling/predictive/discriminant_analysis/quadratic_discriminant_analysis.html (Accessed: 19th May 202).

[4] Goodfellow, I.J., Shlens, J. and Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

[5] Stanley, K.O., 2007. Compositional pattern producing networks: A novel abstraction of development. Genetic programming and evolvable machines, 8(2), pp.131-162.

[6] Zuo, C., 2018. Regularization Effect of Fast Gradient Sign Method and its Generalization. arXiv preprint arXiv:1810.11711

[7] Papernot, N., McDaniel, P. and Goodfellow, I., 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv preprint arXiv:1605.07277.

[8] Lu, J., Sibai, H., Fabry, E. and Forsyth, D., 2017. No need to worry about adversarial examples in object detection in autonomous vehicles. arXiv preprint arXiv:1707.03501.

[9] Kurakin, A., Goodfellow, I. and Bengio, S., 2016. Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533.

[10] Athalye, A., Engstrom, L., Ilyas, A. and Kwok, K., 2017. Synthesizing robust adversarial examples. arXiv preprint arXiv:1707.07397.

[11] Tencent Keen Security Lab (29th March 2019) Experimental Security Research of Tesla Autopilot Tencent Keen Security Lab, Available at: https://keenlab.tencent.com/en/2019/03/29/Tencent-Keen-Security-Lab-Experimental-Security-Research-of-Tesla-Autopilot/ (Accessed: 19th May 202).

[12] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B. and Madry, A., 2019. Adversarial Examples Are Not Bugs, They Are Features. arXiv preprint arXiv:1905.02175.