MSDS 458 Fall 2021 Assignment 2
Husein Adenwala

1. Abstract

The objective of this research was to understand how the architecture of a dense neural network works, specifically Convolution Neural Network impact (CNN) performance in classifying objects in RGB images. I started the experiment by evaluating the baseline performance of Multilayer Perceptron's (MLP) and CNN. I gradually added more layers to improve performance and then applied L1, L2 regularization, dropout, batch normalization and image augmentation technique to reduce variance of the models. For this research, a set of experiments was performed on the CIFAR_10 dataset, a collection of 32x32 color images of 10 classes: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks that is widely used to benchmark and test Computer Vision classification algorithms. Results from my experiment suggest that CNN performs better than MLP for image classification and increasing convolution layers and subsequent dense layers improves performance of the model; however, the model overfits and needs to regularize to reduce variance. I found dropout, batch normalization and image augmentation to be most effective in reducing the variance and improving model performance.

2. Introduction

The current generation of smartphones have highly advanced image processing that have ensured that the image quality delivered by SPCs matches that achieved with full-frame cameras (Blahnik et al, 2021). This has enabled the development of image recognition apps with deep learning models.

Looking at the success of image recognition on smartphones a nonprofit organization has consulted me in developing an image recognition model for a free object recognition app. An object recognition app is primarily for people with low vision and total blindness to help them identify objects using their smartphone.

3. Literature review

Convolution Neural Nets (CNNs) are the go-to model for computer vision. Overfitting, exploding gradient, and class imbalance are the major challenges while training the model using CNN (Joshi et all, 2019). Batch normalization and dropout are the most common techniques to reduce overfitting, and batch normalization helps overcome the problem of exploding gradients (Ye, 2020)

One of the best performing models for image recognition is Very Deep Convolutional Networks for Large-Scale Image Recognition (VGG16) (Simonyan 2015). It is a convolution neural net (CNN) architecture which was used to win the ILSVR competition in 2014. However, over the years, the deep learning architectures of CNN have become deeper to solve more complex tasks which also helped in improving the performance of classification and recognition tasks. But the additional layer to the model makes it difficult to train and the accuracy of the model starts saturating and then degrades.

Residual Network (ResNet) is one of the famous deep learning models that was introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun in their paper "Deep Residual Learning for Image Recognition" (He et all, 2015). It won 1st place in the ILSVRC 2015 classification competition. The ResNet model alleviates the issues associated with very deep CNN by adding residual block, that adds a skip connection around several layers of convolutions.

The skip connections in ResNet solve the problem of vanishing gradient in deep neural networks by allowing this alternate shortcut path for the gradient to flow through. The other way that these

connections help is by allowing the model to learn the identity functions, which ensures that the higher layer will perform at least as well as the lower layer, and not worse.

Recently, attention-based networks, such as the Vision Transformer (ViT) have achieved remarkable results compared to CNN while requiring fewer computational resources for pre-training. However, there is recent MLP-Mixer (Tolstikhin 2021) architecture based exclusively on multi-layer perceptrons (MLPs) which has a simpler architecture. It has a higher throughput and is computationally more efficient.

Data Augmentation is an important tool for image classification CNN models. As the name suggests, data augmentation enhances the image for the model, and the goal is to enable to model to extract invariant features. This reduces translation invariance and therefore reduces overfitting (Garcia et al, 2019). Also, it improves model performance (Shorten and Khoshgoftaar, 2019).

4. Methods

Research design and modeling methods

I have decided to use multilayer perceptron (MLP) and Convolution neural net (CNN) to classify colored images.

An MLP is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable. This makes MLP a good candidate for this computer vision problem.

Convolutional neural network (CNN), a class of artificial neural networks that has become dominant in various computer vision tasks, is designed to learn spatial hierarchies of features automatically and adaptively through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers.

CNN has an advantage over MLP in image classification as it can understand spatial relation between pixels of images better than MLP. This makes CNN models spatially invariant whereas MLP are sensitive to spatial variance. This enables CNN to have lower variance than MLP.

I have experimented with several combinations of multiple dense layers, convolution layers, and stacked convolution layers in my experiments. All models use the ReLU activation function which is one of the popular activation functions in neural networks because it is computationally efficient and fixes the problem of vanishing gradient. I used validation Accuracy as the metric to evaluate the performance. Finally, I used categorical cross entropy or spatial categorical cross entropy to measure the loss between labels and predictions and SoftMax activation function for the output layer.

Data preparation, exploration, visualization

The dataset used in this research is the CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. This dataset is widely used to benchmark and test for image classification algorithms. The dataset is loaded from the Kera's framework and contains 50,000 training and 10,000 test 32x32,3 color images and the corresponding train and test image labels of the 10

categories in a NumPy array. Further, 5000 images from the training set were randomly appropriated as the validation data set.

CIFAR images have the following 10 classes

Label	Classes
0	airplane
1	automobile
2	bird
3	cat
4	deer
5	dog
6	frog
7	horse
8	ship
9	truck

Sample images

Preprocessing data:

I did min max normalization to scale the data from 0 to 1, as normalizing the data generally speeds up learning and leads to faster convergence. Also, in some of the models I used Image augmentation to reduce overfitting and improve model performance.

Augmented Image vs original image of an automobile

Implementation and programming

Model research was done using Python programming language and TensorFlow. TensorFlow is a free and open-source software library for machine learning and artificial intelligence and TensorFlow can be implemented in python by Keras. Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow which is extremely user-friendly.

I used the sci-kit learn library which has functions to create a confusion matrix and report on precision, accuracy and F1 score.

Additionally, along with the Keras image data generator library, I used NumPy and Pandas library for data scaling and augmenting the images and finally, I also used seaborn package and matplotlib library to visualize the data and results.

5. Results

I conducted 11 experiments and all the models in my experiments had Early Stopping with a patience of 10 and Relu Activation layers for dense layers and Softmax activation for output layers.

Experiment 1 & 2

Optimizer = RMS prop. No regularization, Batch size = 512.

Model in experiment 1 had a dense neural network with 3072 input nodes, two hidden layers of 512 nodes, and 10 output nodes.

Model achieved test accuracy of 49% and a train accuracy of 69.8%. It could not discriminate between different classes as you can see in the confusion matrix and box plot in Appendix A.

Model in experiment 2 had dense neural network with 3072 input nodes, three hidden layers with of 1024, 512 and 256 nodes respectively, and 10 output nodes.

Model achieved test accuracy of 51% and a train accuracy of 80.6%. This model did marginally better than model 1.

The confusion matrix highlights that the model is confusing the actual versus predicted classes and the performance is uneven between classes (see Appendix A).

The results show that increasing the width of the hidden layer by adding more nodes improves the performance of the model marginally.

Both the models have high variance, perform poorly and need regularization.

Experiment 3 & 4

Optimizer = RMS prop. No regularization, Batch size = 512

Model in experiment 3 has two convolution layers of 64, 128 kernels respectively, a dense layer of 384 nodes and 10 output nodes.

Model achieved test accuracy of 73% and a train accuracy of 100%. The model performance was uneven between classes; specifically, it performed poorly in classifying cats, dogs, birds and deer (See Appendix B).

Model in experiment 4 has three convolution layers with of 64, 128 and 256 kernels respectively, a dense layer of 384 node and 10 output nodes.

Model achieved test accuracy of 73% and a train accuracy of 99.94%. Like the model in experiment 3 the model performance was uneven between classes; similarly, it performed poorly in classifying cats, dogs, birds and deer (See Appendix B).

Key takeaway from experiment 3 and 4

From experiment 3 & 4 we can infer that CNN performs significantly better compared to MLP model, but it suffered from high variance because I did not apply regularization. Also, it is unclear if the additional layer improved performance of the model.

In the next experiments with a convolution model, I added regularization.

Experiment 5

Optimizer = 'Adam', L2 regularization and batch normalization, batch size = 512.

Model in experiment 5 had a dense neural network with 3072 input nodes, five hidden layers of 512 nodes, and 10 output nodes.

Model achieved test accuracy of 47% and a train accuracy of 63%. The additional layer made the model memorize the training data and even with regularization it had high variance (see Appendix C).

Experiment 6

Optimizer = 'Adam', L2 regularization and batch normalization and dropout (0.3), batch size = 256.

Model in experiment 6 had dense neural network with 3072 input nodes, five hidden layers of 512 nodes, and 10 output nodes.

Model achieved test accuracy of 44% and a train accuracy of 46% (see Appendix C). The model accuracy was lower than the model in experiment 5 but it had low variance and therefore the model was not overfitting the data.

Key takeaway from experiment 1, 2, 5 and 6

From experiment 5 & 6 we can conclude that dropout was effective in reducing variance/overfitting. Also, MLP has limitations in image classification because it cannot understand or detect spatial relationships between pixels in an image. The model can perform well on training data because it memorizes the data but performs poorly on test data because its learning is not translation invariant.

In the next experiment, I only worked with CNN models.

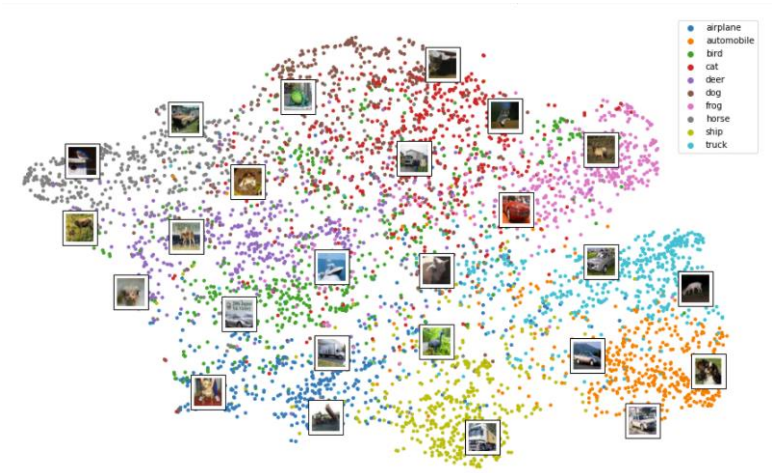
Experiment 7

Optimizer = 'RMSprop', L2 regularization and batch normalization on the dense layer and dropout (0.3) on Convolution layer, batch size = 512.

Model in experiment 7 has three convolution layers of 128, 256 and 512 kernels respectively, a dense layer of 384 nodes and 10 output nodes.

Model achieved test accuracy of 81% and a train accuracy of 99.8%. The model performance was uneven between classes; specifically, it performed poorly distinguishing between cats and dogs (See Appendix D).

I used t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce the dimension of the second to last layer of 384 dense layer nodes to 2. The T-SNE plot shows how well the 384 nodes could discriminate between each label



Key takeaway from experiment 7

Batch normalization and dropout improved model performance. In the next CNN model, I added batch normalization and dropout in every layer.

Experiment 8

Optimizer = 'RMSprop', batch normalization and dropout (0.3) on Convolution and Dense layer, padding = same on Convolution layer, batch size = 512.

Model in experiment 7 had four convolution layers of 128, 256, 512 and 1024 kernels respectively, two dense layer of 256 and 512 nodes respectively and 10 output nodes.

Model achieved test accuracy of 79% and a train accuracy of 99.71%. The model performance was uneven between classes; specifically, it performed poorly distinguishing between cats and dogs (See Appendix E).

Experiment 9

Optimizer = 'Adam', batch normalization and dropout (0.3) on Convolution and Dense layer, padding = same on Convolution layer, batch size = 512.

Model in experiment 9 had four convolution layers of 128, 256, 512 and 1024 kernels respectively, two dense layers of 256 and 512 nodes respectively and 10 output nodes.

Model achieved test accuracy of 83% and a train accuracy of 99.82%. The model performance was uneven between classes; specifically, it performed poorly distinguishing between cats and dogs (See Appendix F).

Key takeaway from experiment 8 & 9

Deep CNN model provided model performance improvements, and the Adam optimizer seemed to perform slightly better than RMSprop in deep CNN model. Padding added space around the borders and thereby improved performance by keeping information at the borders.

In the next CNN models, I increased the drop out to 0.5 to reduce variance. I made the model deeper with group or stacked convolution layers as done in VGG 16 model.

The stack layers of convolutions can learn more intricate patterns within the features mapped in the previous layer. This enabled the neural network to identify general patterns in early layers, then focus on the patterns within patterns in later layers.

Experiment 10

Optimizer = 'Adam', batch normalization and dropout (0.5) on Convolution and Dense layer, padding = same on Convolution layer, Stacked/grouped convolution layers, batch size = 512.

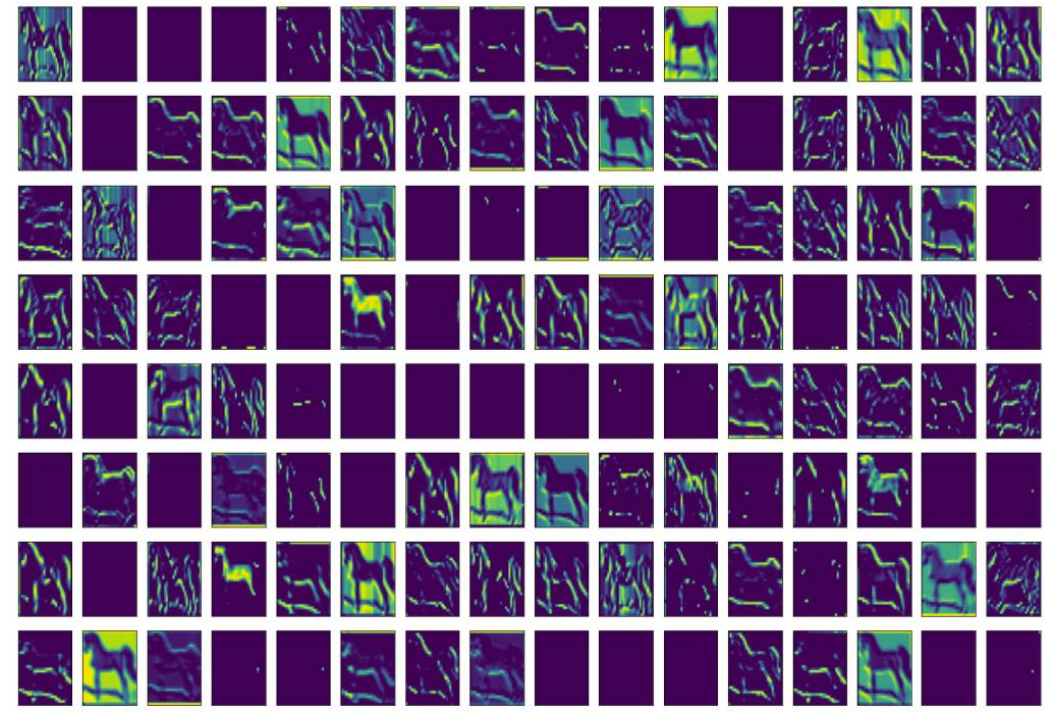
Model in experiment ten convolution layers with 128, 256, 256, 256) 512, 512, 512, 1024, 1024, 1024 (see appendix H),kernels respectively, three dense layers of 1024 nodes respectively and 10 output nodes.

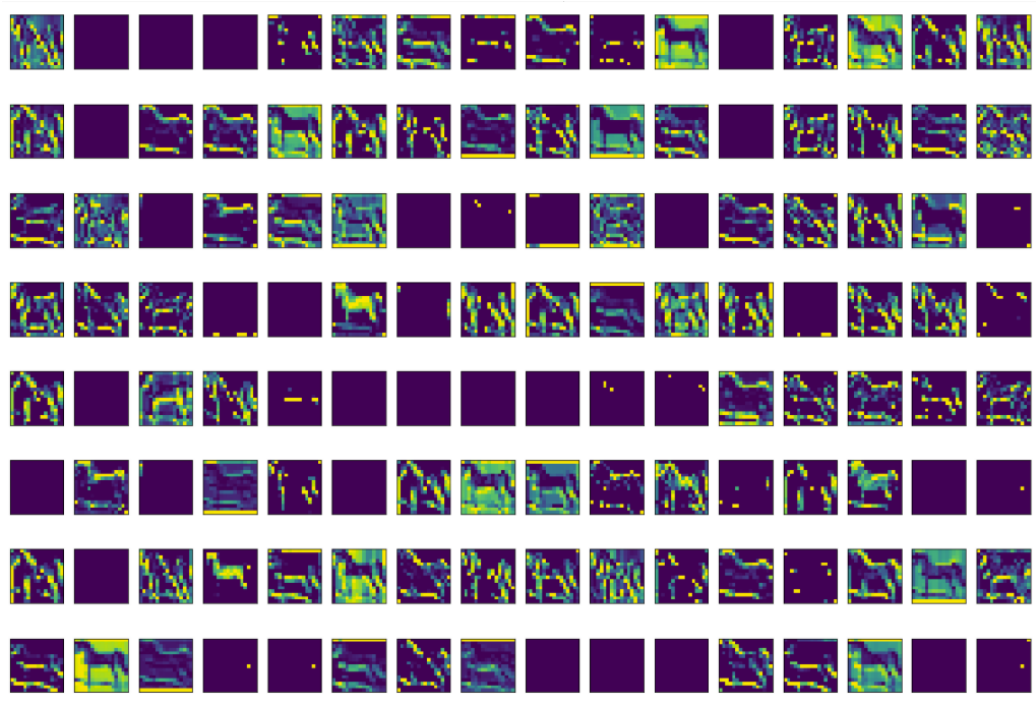
The three 256 kernel convolution layers were grouped together, the three 512 kernel convolution layers were grouped together and the three 1024 convolution layers were grouped together.

The model achieved test accuracy of 86% and a train accuracy of 96.5%. The model performance was still relatively uneven between classes; specifically, it performed poorly distinguishing between cats and dogs (See Appendix G).

Below see the features extraction of 1st convolution layer and Max pool layer respectively.

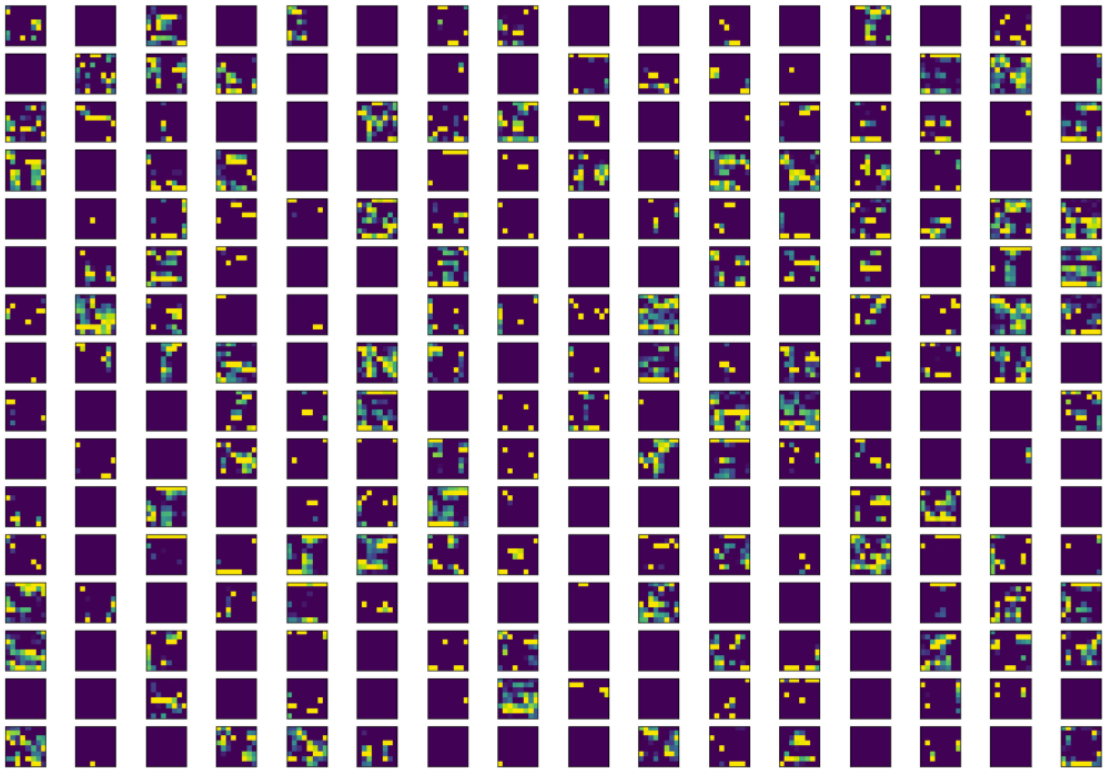
This layer is training on an image of the horse and the below image shows the 128 filter (8 rows and 16 columns) capturing the edges (mostly horizontal edges) of the horse's image. We can also observe the dropout filers.

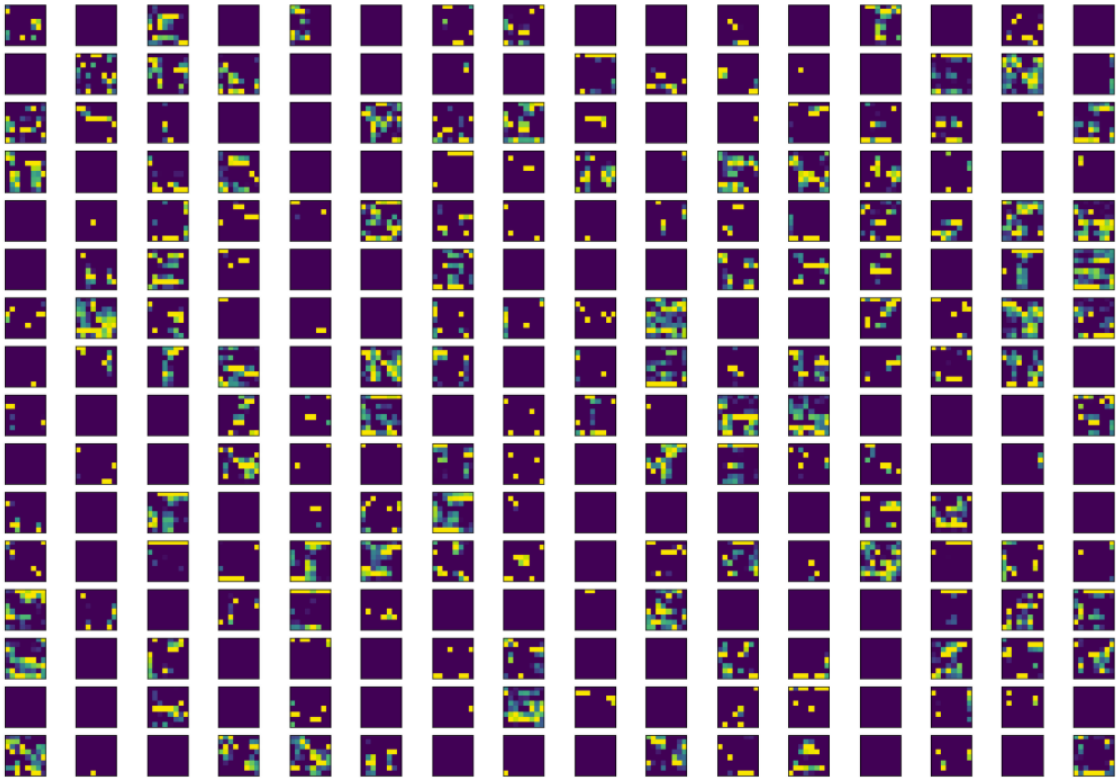




Below see the features extraction of 2nd (256, 256,256) stacked and Max pool layer respectively.

This layer was training on an image of the horse and the below image shows the 256 filters (16 rows and 16 columns). This layer is capturing the features of the previous convolution layer and therefore we cannot see the outline of the horse anymore.





Key takeaway from experiment 10

A complex stacked CNN showed some modest improvement in model performance but there was still high variance.

In the next model, I augmented the images to address overfitting, which should have enabled better accuracy by removing noise and keeping the invariant features from the image.

Experiment 11

Optimizer = 'Adam', batch normalization and dropout (0.5) on Convolution and Dense layer, padding = same on Convolution layer, Stacked/grouped convolution layers, Image augmentation, batch size = 64.

Model in experiment had ten convolution layers with 128, 256, 256, 256) 512, 512, 512, 1024, 1024, 1024 (see appendix H)kernels respectively, three dense layers of 1024 nodes respectively and 10 output nodes.

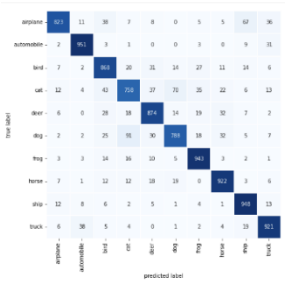
The three 256 kernel convolution layers were grouped together, the three 512 kernel convolution layers were grouped together and the three 1024 convolution layers were grouped together,

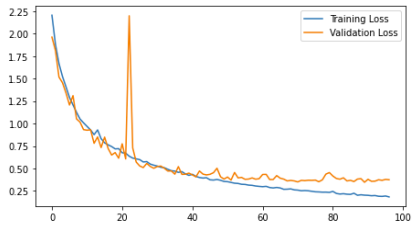
Model achieved test accuracy of 88% and a train accuracy of 93%. The model performance was uneven between classes; it also performed poorly distinguishing between cats and dogs.

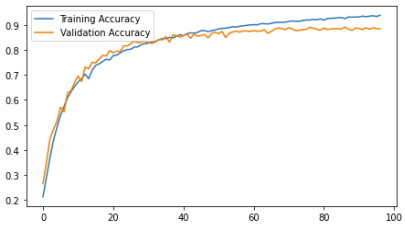
Classification Report

	precision	recall	f1-score	support
0	0.94	0.82	0.88	1000
1	0.93	0.95	0.94	1000
2	0.83	0.87	0.85	1000
3	0.82	0.76	0.79	1000
4	0.86	0.87	0.87	1000
5	0.86	0.79	0.82	1000
6	0.89	0.94	0.92	1000
7	0.89	0.92	0.91	1000
8	0.88	0.95	0.91	1000
9	0.89	0.92	0.90	1000
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000

Accuracy Score: 0.8796
Root Mean Square Error: 1.5002999700059985







Key takeaway from experiment 11

This complex stacked CNN with image augmentation was the best performing model among my experiments. The model still was not able to accurately discriminate between dogs and cats, but intuitively this makes sense as the images of cats and dogs at the edges and outlines (features) of the two animals can be very similar, which makes it harder for convolution and deep layers to distinguish between the two. This issue can potentially be solved by having more training samples or/and by better image augmentation technique.

6. Conclusions

Experiments	Model Type	Regularization Hyperparameters	Optimizer	Batch Size	Test Accuracy %	Validation Accuracy%	Train Accuracy%	Validation Loss	Training Loss	Training Time
Exp 11	CNN 10 layer (128, 256 x 3 stacked, 512 x 3 stacked, 1024 x 3 stacked) + 3 Dense layer (1024, 1024,1024)	L2, Batch Normalization, Dropout, Padding, Image Augmentation	Adam	64	88	89.16	93	0.35	0.22	3686s
Exp 10	CNN 10 layer (128, 256 x 3 stacked, 512 x 3 stacked, 1024 x 3 stacked) + 3 Dense layer (1024, 1024,1024)	L2, Batch Normalization, Dropout, Padding	Adam	512	86	85.88	96.5	0.69	0.02	1332s
Exp 9	CNN 4 layer (128, 256, 512, 1024) + 2 Dense layer (256, 512)	L2, Batch Normalization, Dropout, Padding	Adam	512	82	84.18	99.82	0.73	0.006	233s
Exp 7	CNN 3 layer (128, 256, 512)+ 1 Dense layer 384	L2, Batch Normalization, Dropout	RMSprop	512	81	81.82	99.8	0.74	0.1	241s
Exp 8	CNN 4 layer (128, 256, 512, 1024)+ 2 Dense layer (256, 512)	L2, Batch Normalization, Dropout, Padding	RMSprop	512	79	82	99.7	0.74	0.106	298s
Exp 3	CNN 2 layer (64, 128) + 1 Dense layer 384	-	RMSprop	512	73	72.9	100	1.87	0	59s
Exp 4	CNN 3 Layer (64,128,256)+ 1 Dense layer 384	-	RMSprop	512	73	73.12	99.94	1.74	0.004	34s
Exp 2	MLP 3 layer (1024 ,516, 256)	-	RMSprop	512	51	49.7	80.6	1.81	0.55	33s
Exp 1	MLP 2 layer (512,512)	-	RMSprop	512	49	48	69.8	1.6	0.85	28s
Exp 5	MLP 5 layer (512 x 5)	L2, Batch Normalization	Adam	256	47	45.74	63	1.53	0.006	21s
Exp 6	MLP 5 Layer (512 x 5	L2, Batch Normalization, Dropout	Adam	256	44	43.37	46	1.7	0.072	76s

The results of the 11 experiments show that CNN performs better than MLP because CNN can extract spatial relationships between pixels with the filters which makes them translation invariant and therefore, they perform better on test data as compared to MLP. It is clear from the experiments that CNN is more complex than MLP and requires more computational resources to train a model. The experiments also prove that adding layers in a CNN model improves accuracy; however, the improvements are modest (diminishing rate of return) and deep models tend to overfit and take more time to train. It also faces the issue of exploding gradient, and class imbalances. The best model as shown in Experiment 11 still showed relatively low accuracy in classifying dogs and cats.

Adding layers was not enough; we need to use regularization techniques to reduce the variance in the CNN model. I found that batch normalization and image augmentation not only reduced variance but helped the model accuracy. Image augmentation is key in finding the final performance improvements in a CNN model. With the right image augmentation, we can remove the noise from the image and provide the invariant features to the model.

The best model was loosely based on the VGG 16 architecture of grouped/stacked Convolution layers and utilized image augmentation which improved model performance to 88% accuracy and average f1 score of 0.88.

In conclusion, I would recommend using image augmentation with deep CNN architecture for image classification as demonstrated in experiment 11, which was the best performing model.

References:

Blahnik, Vladan and Schindelbeck, Oliver. "Smartphone imaging technology and its applications" Advanced Optical Technologies, vol. 10, no. 3, 2021, pp. 145-232. https://doi.org/10.1515/aot-2021-0023

Joshi S., Verma D.K., Saxena G., Paraye A. (2019) Issues in Training a Convolutional Neural Network Model for Image Classification. In: Singh M., Gupta P., Tyagi V., Flusser J., Ören T., Kashyap R. (eds) Advances in Computing and Data Sciences. ICACDS 2019. Communications in Computer and Information Science, vol 1046. Springer, Singapore. https://doi.org/10.1007/978-981-13-9942-8_27

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: Deep Residual Learning for Image Recognition, Dec 2015, DOI: https://arxiv.org/abs/1512.03385

Hernández-García, A., König, P., & Kietzmann, T.C. (2019). Learning robust visual representations using data augmentation invariance. ArXiv, abs/1906.04547.

Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60 (2019). https://doi.org/10.1186/s40537-019-0197-0

Boesch, G. (2021). Vision Transformers (ViT) in Image Recognition – 2021 Guide https://viso.ai/deep-learning/vision-transformer-vit/

Frahcois, C. (2018). Deep Learning with Python. Shelter Island, New York: Manning Publications Co.

Gelìron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems* (2nd ed.). O'Reilly.

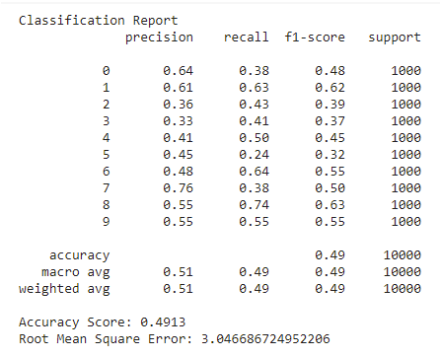
Ye, Andre. 2020. "*Batch Normalization: The Greatest Breakthrough in Deep Learning*- Python." *towards data science*. May 27, 2020. https://towardsdatascience.com/batch-normalization-the-greatest-breakthrough-in-deep-learning-77e64909d81d#:~:text=Batch%20normalization%20helps%20make%20sure,Solving%20the%20exploding%20gradient%20problem.

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556.

Tolstikhin, I.O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Keysers, D., Uszkoreit, J., Lucic, M., & Dosovitskiy, A. (2021). MLP-Mixer: An all-MLP Architecture for Vision. ArXiv, abs/2105.01601.

Appendix A

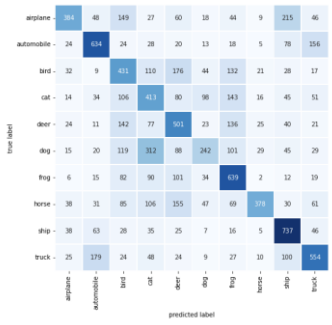
Experiment 1 result



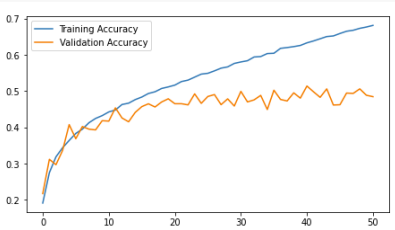
Classification Report

	precision	recall	f1-score	support
0	0.64	0.38	0.48	1000
1	0.61	0.63	0.62	1000
2	0.36	0.43	0.39	1000
3	0.33	0.41	0.37	1000
4	0.41	0.50	0.45	1000
5	0.45	0.24	0.32	1000
6	0.48	0.64	0.55	1000
7	0.76	0.38	0.50	1000
8	0.55	0.74	0.63	1000
9	0.55	0.55	0.55	1000
accuracy			0.49	10000
macro avg	0.51	0.49	0.49	10000
weighted avg	0.51	0.49	0.49	10000

Accuracy Score: 0.4913
Root Mean Square Error: 3.046686724952206



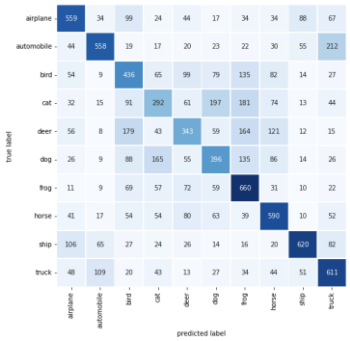
Experiment 2 result

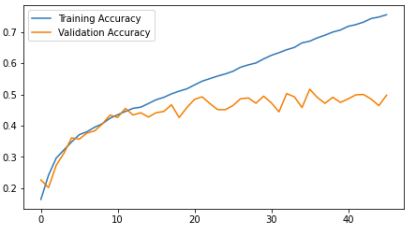


Classification Report

	precision	recall	f1-score	support
0	0.57	0.56	0.57	1000
1	0.67	0.56	0.61	1000
2	0.40	0.44	0.42	1000
3	0.37	0.29	0.33	1000
4	0.42	0.34	0.38	1000
5	0.42	0.40	0.41	1000
6	0.46	0.66	0.55	1000
7	0.53	0.59	0.56	1000
8	0.70	0.62	0.66	1000
9	0.53	0.61	0.57	1000
accuracy			0.51	10000
macro avg	0.51	0.51	0.50	10000
weighted avg	0.51	0.51	0.50	10000

Accuracy Score: 0.5065
Root Mean Square Error: 3.0167697956589263





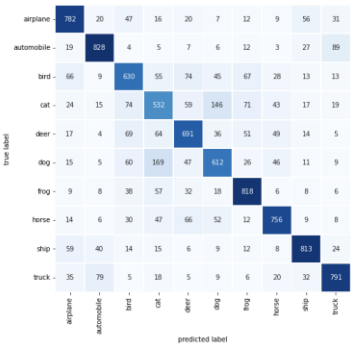
Appendix B

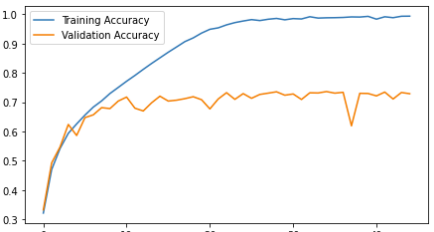
Experiment 3 result

Classification Report

	precision	recall	f1-score	support
0	0.75	0.78	0.77	1000
1	0.82	0.83	0.82	1000
2	0.65	0.63	0.64	1000
3	0.54	0.53	0.54	1000
4	0.69	0.69	0.69	1000
5	0.65	0.61	0.63	1000
6	0.75	0.82	0.78	1000
7	0.78	0.76	0.77	1000
8	0.81	0.81	0.81	1000
9	0.79	0.79	0.79	1000
accuracy			0.73	10000
macro avg	0.72	0.73	0.72	10000
weighted avg	0.72	0.73	0.72	10000

Accuracy Score: 0.7253
Root Mean Square Error: 2.187715703650728



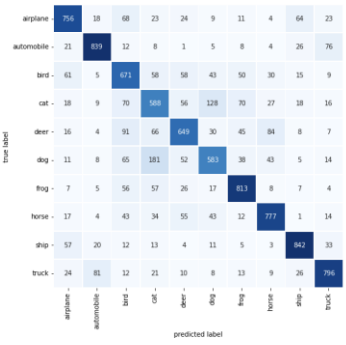


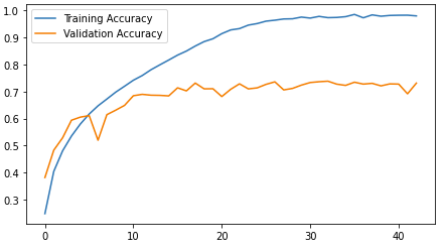
Experiment 4 result

Classification Report

	precision	recall	f1-score	support
0	0.77	0.76	0.76	1000
1	0.84	0.84	0.84	1000
2	0.61	0.67	0.64	1000
3	0.56	0.59	0.57	1000
4	0.69	0.65	0.67	1000
5	0.66	0.58	0.62	1000
6	0.76	0.81	0.79	1000
7	0.79	0.78	0.78	1000
8	0.83	0.84	0.84	1000
9	0.80	0.80	0.80	1000
accuracy			0.73	10000
macro avg	0.73	0.73	0.73	10000
weighted avg	0.73	0.73	0.73	10000

Accuracy Score: 0.7314
Root Mean Square Error: 2.117829077144801





Appendix C

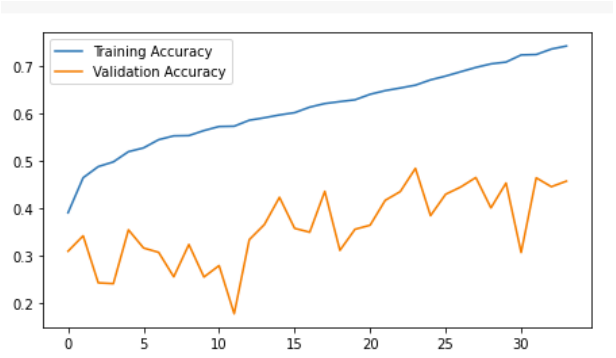
Experiment 5 result

```
Classification Report
              precision    recall  f1-score   support

           0       0.49      0.53      0.51      1000
           1       0.51      0.72      0.60      1000
           2       0.34      0.35      0.34      1000
           3       0.36      0.27      0.31      1000
           4       0.44      0.37      0.40      1000
           5       0.45      0.36      0.40      1000
           6       0.48      0.61      0.54      1000
           7       0.53      0.47      0.50      1000
           8       0.67      0.43      0.52      1000
           9       0.44      0.58      0.50      1000

    accuracy                           0.47     10000
   macro avg       0.47      0.47      0.46     10000
weighted avg       0.47      0.47      0.46     10000

Accuracy Score: 0.4686
Root Mean Square Error: 3.278719262151
```



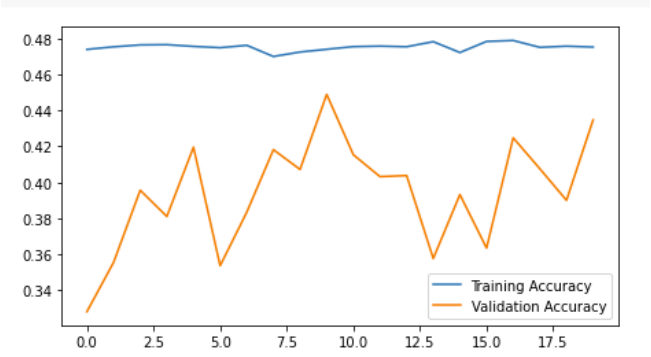
Experiment 6 Result

```
Classification Report
              precision    recall  f1-score   support

           0       0.63      0.32      0.42      1000
           1       0.58      0.60      0.59      1000
           2       0.28      0.47      0.35      1000
           3       0.28      0.27      0.28      1000
           4       0.43      0.36      0.39      1000
           5       0.35      0.45      0.39      1000
           6       0.43      0.65      0.52      1000
           7       0.53      0.49      0.51      1000
           8       0.60      0.56      0.58      1000
           9       0.71      0.26      0.38      1000

    accuracy                           0.44     10000
   macro avg       0.48      0.44      0.44     10000
weighted avg       0.48      0.44      0.44     10000

Accuracy Score: 0.4423
Root Mean Square Error: 3.0306764921383476
```



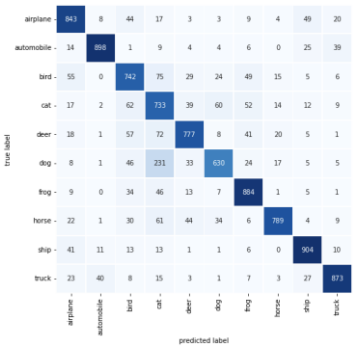
Appendix D

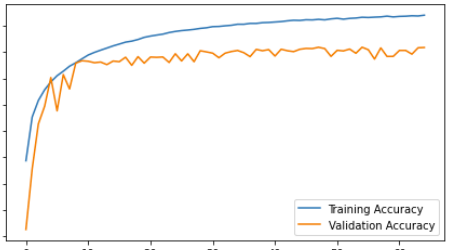
Experiment 7 result

Classification Report

	precision	recall	f1-score	support
0	0.80	0.84	0.82	1000
1	0.93	0.90	0.92	1000
2	0.72	0.74	0.73	1000
3	0.58	0.73	0.65	1000
4	0.82	0.78	0.80	1000
5	0.82	0.63	0.71	1000
6	0.82	0.88	0.85	1000
7	0.91	0.79	0.85	1000
8	0.87	0.90	0.89	1000
9	0.90	0.87	0.88	1000
accuracy			0.81	10000
macro avg	0.82	0.81	0.81	10000
weighted avg	0.82	0.81	0.81	10000

Accuracy Score: 0.8073
Root Mean Square Error: 1.7619875141441836





Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 128)	3584
max_pooling2d (MaxPooling2D)	(None, 15, 15, 128)	0
dropout (Dropout)	(None, 15, 15, 128)	0
conv2d_1 (Conv2D)	(None, 13, 13, 256)	295168
max_pooling2d_1 (MaxPooling2	(None, 6, 6, 256)	0
dropout_1 (Dropout)	(None, 6, 6, 256)	0
conv2d_2 (Conv2D)	(None, 4, 4, 512)	1180160
max_pooling2d_2 (MaxPooling2	(None, 2, 2, 512)	0
dropout_2 (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 384)	786816
batch_normalization (BatchNo	(None, 384)	1536
dropout_3 (Dropout)	(None, 384)	0
dense_1 (Dense)	(None, 10)	3850

Total params: 2,271,114
Trainable params: 2,270,346
Non-trainable params: 768

Appendix E

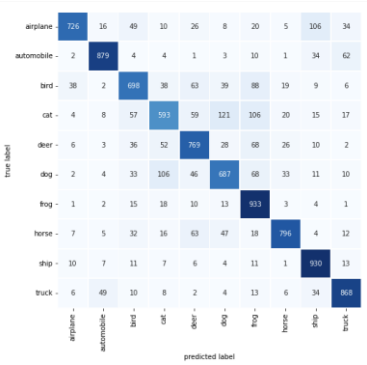
Experiment 8

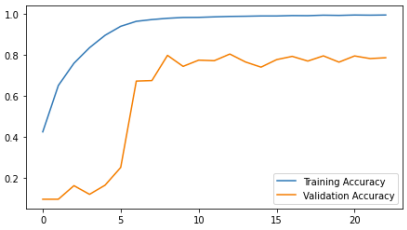
```
Classification Report
              precision    recall  f1-score   support

           0       0.91      0.73      0.81      1000
           1       0.90      0.88      0.89      1000
           2       0.74      0.70      0.72      1000
           3       0.70      0.59      0.64      1000
           4       0.74      0.77      0.75      1000
           5       0.72      0.69      0.70      1000
           6       0.70      0.93      0.80      1000
           7       0.87      0.80      0.83      1000
           8       0.80      0.93      0.86      1000
           9       0.85      0.87      0.86      1000

    accuracy                           0.79     10000
   macro avg       0.79      0.79      0.79     10000
weighted avg       0.79      0.79      0.79     10000

Accuracy Score: 0.7879
Root Mean Square Error: 1.8829232591903473
```






```
Model: "sequential_41"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_147 (Conv2D)          (None, 32, 32, 128)       3584
_________________________________________________________________
max_pooling2d_147 (MaxPoolin (None, 16, 16, 128)       0
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_8 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_148 (Conv2D)          (None, 16, 16, 256)       295168
_________________________________________________________________
max_pooling2d_148 (MaxPoolin (None, 8, 8, 256)         0
_________________________________________________________________
dropout_9 (Dropout)          (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_9 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_149 (Conv2D)          (None, 8, 8, 512)         1180160
_________________________________________________________________
max_pooling2d_149 (MaxPoolin (None, 4, 4, 512)         0
_________________________________________________________________
dropout_10 (Dropout)         (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_10 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
conv2d_150 (Conv2D)          (None, 4, 4, 1024)        4719616
_________________________________________________________________
max_pooling2d_150 (MaxPoolin (None, 2, 2, 1024)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 2, 2, 1024)        0
_________________________________________________________________
batch_normalization_11 (Batc (None, 2, 2, 1024)        4096
_________________________________________________________________
flatten_35 (Flatten)         (None, 4096)              0
_________________________________________________________________
dense_73 (Dense)             (None, 256)               1048832
_________________________________________________________________
batch_normalization_12 (Batc (None, 256)               1024
_________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_74 (Dense)             (None, 512)               131584
_________________________________________________________________
batch_normalization_13 (Batc (None, 512)               2048
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_75 (Dense)             (None, 10)                5130
=================================================================
Total params: 7,394,826
Trainable params: 7,389,450
Non-trainable params: 5,376
_________________________________________________________________
```

Appendix F

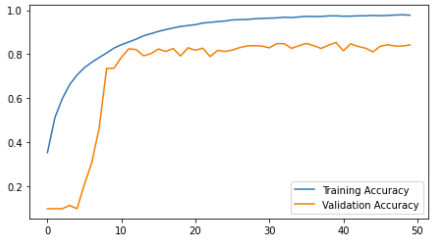
Experiment 9

```
Classification Report
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1000
           1       0.96      0.87      0.91      1000
           2       0.84      0.69      0.76      1000
           3       0.71      0.65      0.68      1000
           4       0.70      0.90      0.79      1000
           5       0.81      0.68      0.74      1000
           6       0.85      0.90      0.87      1000
           7       0.86      0.90      0.88      1000
           8       0.89      0.93      0.91      1000
           9       0.88      0.91      0.89      1000

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.83     10000
weighted avg       0.83      0.83      0.83     10000

Accuracy Score: 0.8309
Root Mean Square Error: 1.6363984844774209
```




```
Model: "sequential_42"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_151 (Conv2D)          (None, 32, 32, 128)       3584
_________________________________________________________________
max_pooling2d_151 (MaxPoolin (None, 16, 16, 128)       0
_________________________________________________________________
dropout_14 (Dropout)         (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_14 (Batc (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_152 (Conv2D)          (None, 16, 16, 256)       295168
_________________________________________________________________
max_pooling2d_152 (MaxPoolin (None, 8, 8, 256)         0
_________________________________________________________________
dropout_15 (Dropout)         (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_15 (Batc (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_153 (Conv2D)          (None, 8, 8, 512)         1180160
_________________________________________________________________
max_pooling2d_153 (MaxPoolin (None, 4, 4, 512)         0
_________________________________________________________________
dropout_16 (Dropout)         (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_16 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
conv2d_154 (Conv2D)          (None, 4, 4, 1024)        4719616
_________________________________________________________________
max_pooling2d_154 (MaxPoolin (None, 2, 2, 1024)        0
_________________________________________________________________
dropout_17 (Dropout)         (None, 2, 2, 1024)        0
_________________________________________________________________
batch_normalization_17 (Batc (None, 2, 2, 1024)        4096
_________________________________________________________________
flatten_36 (Flatten)         (None, 4096)              0
_________________________________________________________________
dense_76 (Dense)             (None, 256)               1048832
_________________________________________________________________
batch_normalization_18 (Batc (None, 256)               1024
_________________________________________________________________
dropout_18 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_77 (Dense)             (None, 512)               131584
_________________________________________________________________
batch_normalization_19 (Batc (None, 512)               2048
_________________________________________________________________
dropout_19 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_78 (Dense)             (None, 10)                5130
=================================================================
Total params: 7,394,826
Trainable params: 7,389,450
Non-trainable params: 5,376
```

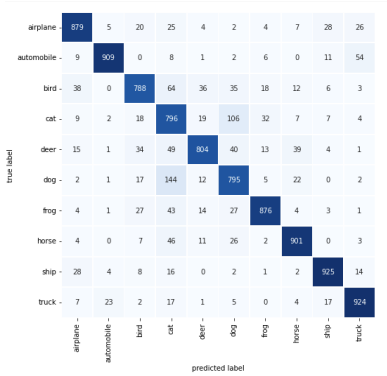
Appendix G

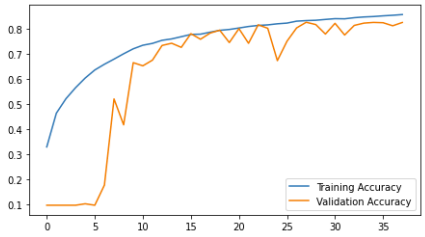
Experiment 10

Classification Report

	precision	recall	f1-score	support
0	0.88	0.88	0.88	1000
1	0.96	0.91	0.93	1000
2	0.86	0.79	0.82	1000
3	0.66	0.80	0.72	1000
4	0.89	0.80	0.85	1000
5	0.76	0.80	0.78	1000
6	0.92	0.88	0.90	1000
7	0.90	0.90	0.90	1000
8	0.92	0.93	0.92	1000
9	0.90	0.92	0.91	1000
accuracy			0.86	10000
macro avg	0.87	0.86	0.86	10000
weighted avg	0.87	0.86	0.86	10000

Accuracy Score: 0.8597
Root Mean Square Error: 1.4781745499094483





Model: "sequential_38"

Layer (type)	Output Shape	Param #
conv2d_94 (Conv2D)	(None, 32, 32, 128)	3584
max_pooling2d_64 (MaxPooling	(None, 16, 16, 128)	0
dropout_96 (Dropout)	(None, 16, 16, 128)	0
batch_normalization_154 (Bat	(None, 16, 16, 128)	512
conv2d_95 (Conv2D)	(None, 16, 16, 256)	295168
conv2d_96 (Conv2D)	(None, 16, 16, 256)	590080
conv2d_97 (Conv2D)	(None, 16, 16, 256)	590080
max_pooling2d_65 (MaxPooling	(None, 8, 8, 256)	0
dropout_97 (Dropout)	(None, 8, 8, 256)	0
batch_normalization_155 (Bat	(None, 8, 8, 256)	1024
conv2d_98 (Conv2D)	(None, 8, 8, 512)	1180160
conv2d_99 (Conv2D)	(None, 8, 8, 512)	2359808
conv2d_100 (Conv2D)	(None, 8, 8, 512)	2359808
max_pooling2d_66 (MaxPooling	(None, 4, 4, 512)	0
dropout_98 (Dropout)	(None, 4, 4, 512)	0
batch_normalization_156 (Bat	(None, 4, 4, 512)	2048
conv2d_101 (Conv2D)	(None, 4, 4, 1024)	4719616

```
conv2d_102 (Conv2D)          (None, 4, 4, 1024)        9438208
_________________________________________________________________
conv2d_103 (Conv2D)          (None, 4, 4, 1024)        9438208
_________________________________________________________________
max_pooling2d_67 (MaxPooling (None, 2, 2, 1024)        0
_________________________________________________________________
dropout_99 (Dropout)         (None, 2, 2, 1024)        0
_________________________________________________________________
batch_normalization_157 (Bat (None, 2, 2, 1024)        4096
_________________________________________________________________
flatten_29 (Flatten)         (None, 4096)              0
_________________________________________________________________
dense_133 (Dense)            (None, 1024)              4195328
_________________________________________________________________
dropout_100 (Dropout)        (None, 1024)              0
_________________________________________________________________
batch_normalization_158 (Bat (None, 1024)              4096
_________________________________________________________________
dense_134 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dropout_101 (Dropout)        (None, 1024)              0
_________________________________________________________________
batch_normalization_159 (Bat (None, 1024)              4096
_________________________________________________________________
dense_135 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dropout_102 (Dropout)        (None, 1024)              0
_________________________________________________________________
batch_normalization_160 (Bat (None, 1024)              4096
_________________________________________________________________
dense_136 (Dense)            (None, 10)                10250
=================================================================
Total params: 37,299,466
Trainable params: 37,289,482
Non-trainable params: 9,984
```

Appendix H

Experiment 11

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 128)       3584
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 128)       0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization (BatchNo (None, 16, 16, 128)       512
```

conv2d_1 (Conv2D)	(None, 16, 16, 256)	295168
conv2d_2 (Conv2D)	(None, 16, 16, 256)	590080
conv2d_3 (Conv2D)	(None, 16, 16, 256)	590080
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 256)	0
dropout_1 (Dropout)	(None, 8, 8, 256)	0
batch_normalization_1 (Batch	(None, 8, 8, 256)	1024
conv2d_4 (Conv2D)	(None, 8, 8, 512)	1180160
conv2d_5 (Conv2D)	(None, 8, 8, 512)	2359808
conv2d_6 (Conv2D)	(None, 8, 8, 512)	2359808
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 512)	0
dropout_2 (Dropout)	(None, 4, 4, 512)	0
batch_normalization_2 (Batch	(None, 4, 4, 512)	2048
conv2d_7 (Conv2D)	(None, 4, 4, 1024)	4719616
conv2d_8 (Conv2D)	(None, 4, 4, 1024)	9438208
conv2d_9 (Conv2D)	(None, 4, 4, 1024)	9438208
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 1024)	0
dropout_3 (Dropout)	(None, 2, 2, 1024)	0
batch_normalization_3 (Batch	(None, 2, 2, 1024)	4096
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 1024)	4195328
dropout_4 (Dropout)	(None, 1024)	0
batch_normalization_4 (Batch	(None, 1024)	4096
dense_1 (Dense)	(None, 1024)	1049600
dropout_5 (Dropout)	(None, 1024)	0
batch_normalization_5 (Batch	(None, 1024)	4096
dense_2 (Dense)	(None, 1024)	1049600
dropout_6 (Dropout)	(None, 1024)	0

```
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10250
=================================================================
Total params: 37,299,466
Trainable params: 37,289,482
Non-trainable params: 9,984
```