# COMP-1801 Machine Learning

Michael Briggs
001002629
*Data Science MSc*
*University of Greenwich*
London, Greenwich
mb5878n@gre.ac.uk

***Abstract*—This document contains an investigation into machine learning, the pipeline required for data to be processed successfully, observations of different model structure and how results were used to improve the accuracy of a model. By learning techniques demonstrated during lectures throughout this module and then applying them to a dataset, ideally with an outcome that has accuracy better than a random guess.**

***Index Terms*—Machine Learning, Neural Network, Deep Neural Network, Convolutional Neural Network, Artificial Intelligence, CIFAR-10**

## I. Introduction

Computer vision and image classification is a highly complex subject and difficult to achieve with high accuracy, however this subset of Machine Learning (ML) is becoming more relevant in the modern world from facial recognition software to self-driving vehicles.

The ability for humans to recognise objects comes by learning by repetition and is something that is done every day, however a computer does not know what an object is so cannot differentiate between any objects without first being trained to do so. Using the CIFAR-10 dataset [1] which has ten classes of object and 60000 total images to train and validate a model with an accuracy of 50% as a minimum goal.

Before training can begin, the data pipeline must be correct, as without a properly functioning pipeline training will not proceed as intended, so it is vitally important that rather than focusing on achieving a state-of-the-art result on this dataset, the focus is turned to how data is transformed and fed to the model before training.

## II. Related work

Computer vision is becoming more prominent in the modern world, from phone facial recognition to self driving vehicles. Nvidia proved that using a single camera and behavioural cloning, they could train a self driving vehicle [2] with a fairly simple Convolutional Neural Network (CNN), replicating the actions that had been shown during the training process. Recent work on activation functions shows that by using Swish over ReLu improves cutting edge networks [3], questions still need to be answered on its effectiveness on smaller neural networks. Gavali also used the cifar dataset using Graphics Processing Unit (GPU) technology [4] which is becoming more common with Google Colabs offering free usage of GPU hardware on their platform. Agarap also experimented with model architectures, comparing softmax and support vector machine (SVM) but did not yield any conclusive results to confidently say which was better for the problem space [5].

LeCun et al states, *"shallow classifiers require a good feature extractor"* [6] when they discuss the advantages that deep learning posses over conventional methods of image classification.

## III. Dataset preparation

### A. Loading Data

Tensorflow [7] has dataset APIs which allows importing many datasets simple, for this case it is the CIFAR-10 dataset. By assigning the dataset to a dictionary variable this allows access to all 60000 images and labels that are included with the dataset. By saving the dictionary using the python package pickle, this also removes the need to consistently download the dataset every time a new session is run. A simple function that checks for a .pkl file with the name *cifar10_data* and if this does not exist it gets the data and saves it. The dataset is split into four on initial download, with the training images consisting of 50000 images with a shape of 32x32x3 as seen in Table I. This means that the image is 32 pixels high and 32 pixels wide with a depth of 3, a single layer for each colour Red, Green and Blue. This gives each image a total of 3,072 pixels with the simple calculation of $height * width * depth$. The labels for the training set is a flat array of 50000 entries where the index of each entry corresponds to the same index in the training image array. Testing images and labels are set out in the same manner as the testing images and labels. Fig1 and Fig2 show how these are distributed across the ten classes. These initial steps confirms that the raw data is loaded and ready to be processed, further to this Fig3 shows a five

| Dataset | Shape | Min/Max |
|---|---|---|
| Training Images | 50000, 32, 32, 3 | 0 to 255 |
| Training Labels | 50000, 1 | 0 to 9 |
| Testing Images | 10000, 32, 32, 3 | 0 to 255 |
| Testing Labels | 10000, 1 | 0 to 9 |

TABLE I
ORIGINAL DATASET PROPERTIES.

randomly picked images with a numerical label for each class resulting in 50 random images from the training dataset.

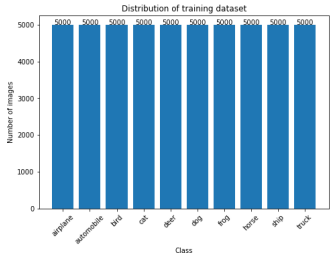


Fig. 1. Distribution of CIFAR-10 training set.

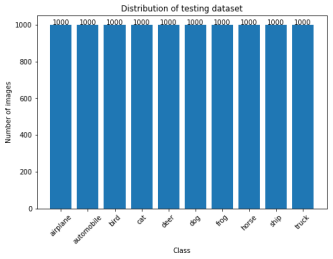


Fig. 2. Distribution of CIFAR-10 testing set.


Fig. 3. Visualisation of the dataset.

### B. Normalisation

With the dataset ready at the start of the pipeline, normalisation can be applied to both sets of training and testing images. As pixel data is currently an integer between 0 and 255 representing the colour of that pixel in a matrix which is fine for representing an image on a screen but the scale is large and can cause the neural network to train slowly. To counter this issue, normalising the image data to scales between 0 and 1 is the next step.

$$x = \frac{x - x[min]}{x[max] - x[min]} \tag{1}$$

By implementing equation 1, the image matrix is scaled to a float between 0 and 1 without loosing the information contained.

The label arrays are also transformed by applying the one-hot-encoding technique, allowing usage of a binary approach for categorising labels. Both the training and testing label data is transformed from a single column array to a binary matrix with the same information contained within, but better suited for machine learning categorical data processing.

### C. Compute Properties

With normalisation and one-hot-encoding applied to both training and testing datasets, computing all properties will make sure that all the correct transformations have taken place on the data before training the model begins. Transformations that have been applied up to this point to the cifar-10 dataset can be seen in Table II With the confirmation of transformation being applied, plotting the data set to check the distribution is even across classes. A dataset that has a heavy skew towards a single class could lead to a model that is bias towards this particular class when classifying data. But Krizhevsky *et al*(2010) has already expressed the dataset composition [1], checking the datasets samples per class will confirm this as seen in Figures 1 and 2.

| Dataset | Shape | Min/Max |
|---|---|---|
| Training Images | 50000, 32, 32, 3 | 0.0 to 1.0 |
| Training Labels | 50000, 10 | 0.0 to 1.0 |
| Testing Images | 10000, 32, 32, 3 | 0.0 to 1.0 |
| Testing Labels | 10000, 10 | 0.0 to 1.0 |

TABLE II
DATASET PROPERTIES AFTER TRANSFORMATIONS.

## IV. Proposal

Images contained within cifar-10 are 32x32x3 in shape, this means that are are a total of 3072 pixels per image which would be cumbersome and slow to train. To combat this, convolutional layers paired with pooling layers to extract features from the image before passing over to the fully connected layers has been implemented, turning the model into a CNN similar to LeNet-5 [8] architecture but with more convolution layers due to higher complexity of images used.

### A. Tensorflow Functional API

Tensorflow [7] allows three different method to model building. Sequential models are built layers by layer, easy to implement but lack functionality. Functional API models work in much the same way as Sequential models, but can have a higher degree of functionality allowing more complex models and model branching to be easily implemented. The

final architecture is model sub-classing, this is highly complex and allows for full control over the layers in a model, making full customisation possible with the cost of high complexity.

As this is is an exercise in learning, the Functional API will be implemented, but not with branching or multi input output models in mind as cifar-10 does not require this level of complexity, but rather a exercise in different methods to build models.

### B. Convolutional Layers

Convolution layers allow feature extraction from images that are being passed through, specifying a number of feature maps that each layer will extract from the image giving the convolutional layers the prominent roll of finding features that are important to learn. Kernel height and width is expressed here as an integer, giving the convolution a box to pass over the image and gather features. Activation layers will include Rectified Linear Activation Function (ReLu) which Krizhevsky et al (2017) [9] describe as computationally fast, denoted as:

$$A(x) = max(0, x)$$

Using padding function allows the output to be the same size as the input, by implementing "same" keyword lets convolutions pass over the image fully and extracting data in the image without changing the dimensions of the matrix.

### C. MaxPooling Layers

Max pooling allows the reduction of the image size by a specified high ad width. Down-sampling the image matrix within the window by taking the maximum value, which can speed up the training time Max-pooling which Nagi et al state *"leads to faster convergence rate by selecting superior invariant features which improves generalization performance."* [10]

### D. Batch Normalisation

Batch normalisation layers keeps the mean output close to 0 while standard deviation is kept close to 1. Implementing this *"prevents small changes to the parameters from amplifying into larger and suboptimal changes in activations in gradients"* [11] Utilising batch normalisation should improve model training and reduce model overfitting

### E. Fully Connected Layers

Fully connected layers are made up of artificial neurons which have outputs weighted sum of the neuron's passed thought the activation function.

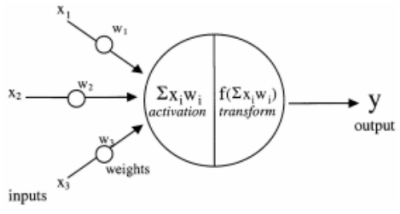


Fig. 4. *"Typical biological inspired neuron"* Nwankpa (2018) [12]

$$y = Activation((X_1 * W_1) + (X_1 * W_1)...(X_n * W_n))$$

The final output layer has ten layers to match the number of classes that cifar-10 contains, with a softmax activation function denoted:

$$\sigma(\overrightarrow{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where $\overrightarrow{z}$ denotes input vector, $z_i$ are values contained within input vector, $e^{z_i}$ applies the standard exponential function to each element, and the term $\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ is normalisation that makes sure that the output range is between $0, 1$ while $K$ in the number of classes.

### F. Data Generator

By implementing a data generator this allows images to be manipulated by increasing or decreasing brightness, random rotations, width and or height shifts and flipping the image horizontally. This is done as a brighter image of a dog is still a dog, but to a CNN it is a completely different image as the values have been changed. By implementing this type of generator, it will increase training time but will also decrease overfitting of models and allow better generalisation.

### G. Adam Optimiser

Adaptive Moment Estimation (Adam) [13] uses both adaptive learning rates and momentum to update individual weights which in turn helps speed up convergence, there are many different optimisers that can be used, but Adam is an algorithm that updates using the first and second order of momentum which rapidly converges but is computationally expensive to run.

### H. Categorical Cross Entropy

As the label for the training and testing set have been one-hot-encoded with more then one label, categorical cross entropy matches this pattern. By calculating the loss of predicted against actual labels loss functions evaluate model performance during training, higher loss means that the predicted values are incorrect therefore we require a low loss output.

### I. Model

With a brief explanation of components that make up this model the summary of the model can be seen in Table V, using six convolutional layers with six batch normalisation layers after each convolution, with three max pooling layers placed every other convolution to extract features. The flatten layer then transforms the image matrix to a one dimensional array and fed into the first fully connected dense layer with an output dense layer with ten neurons.

## V. Evaluation

Evaluating the model performance not using the data generator and with the same model and parameters training over 50 epochs has been performed.

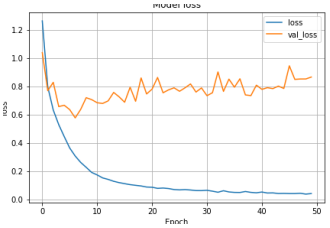


Fig. 5. Overfitted Network Loss

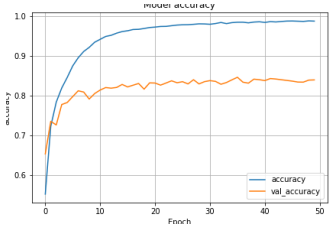


Fig. 6. Overfitted Network Accuracy

Figures 5 and 6 show a clear indication of model over fitting with loss values getting larger as training progresses, this is due to not using the data generator. With training and testing accuracy having a clear gap also points to over fitting to data.

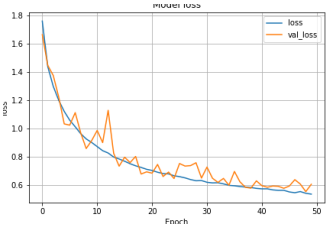


Fig. 7. Well Fitted Network Loss

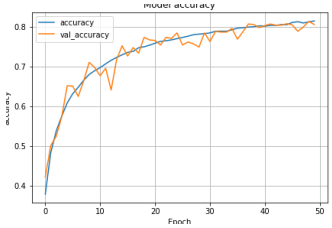


Fig. 8. Well Fitted Network Accuracy

Figures 7 and 8 show the validation loss and accuracy following the training curves closely, although the validation values show spikes in values. This also shows that 50 epochs was not enough for convergence as training values for both loss and accuracy have not flattened out.

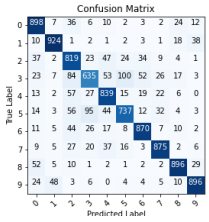


Fig. 9. Overfitted Confusion Matrix

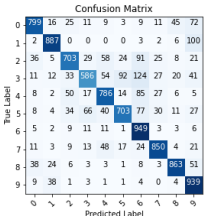


Fig. 10. Well Fitted Confusion Matrix

Using a confusion matrix we can visualise where these models have issues categorising images, both training methods had issues categorising cats correctly while using the data generator, predicting automobiles as trucks happened 100 times. Although not strictly correct in this case, a truck is a subclass of automobile. While not using data generation, this model categorised dogs as cats 100 times.

This issue with using a data generator without memory reply can been seen in Table III. With each step only taking 4 milliseconds when not generating data as training is happening compared to 7 milliseconds as observed during these tests.

| Data Generator | Time Per Step |
|---|---|
| No | 4ms/step |
| Yes | 7ms/step |

TABLE III
Training Time Per Step (ms)

This could be improved with the use of a parallel generator filling a memory buffer with augmented images while training on a batch taken from the buffer opposed to augmenting images as they are needed.

| Data Generator | Loss | Accuracy |
|---|---|---|
| No | 0.868155300617218 | 0.8389000296592712 |
| Yes | 0.6042004823684692 | 0.8065000176429749 |

TABLE IV
Evaluation Scores

Although the accuracy scores seen in Table IV allude to a better model with higher accuracy by not using a image augmentation technique, observed here is a high loss value pointing towards a model that is over fit and will not generalise well. With image augmentation, we have a lower loss value and a slightly lower accuracy value. With more training these values could possibly improve further using image augmentation.

## VI. Future Work

With a keen interests in computer vision, self driving vehicles and lessons learnt during this module, the plan is to incorporate both of these subject into the up coming Masters Dissertation. Implementing these techniques and applying them to AWS Deepracer, extending this work carried out during this module is implementation of reinforcement learning algorithms to make decisions based on maximum reward values in a given step. With the problem space being more complex in Deepracer than this experiment, but the basic structure of data pipeline does not change and is still a key requirement to making a successful model.

## References


[1] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," accessed on 18-11-2020. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[2] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[3] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017.

[4] P. Gavali and J. S. Banu, "Chapter 6 - deep convolutional neural network for image classification on cuda platform," in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, A. K. Sangaiah, Ed. Academic Press, 2019, pp. 99 – 122. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128167182000130

[5] A. F. Agarap, "An architecture combining convolutional neural network (cnn) and support vector machine (svm) for image classification," 2019.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org, Accessed on 02-11-2020. [Online]. Available: https://www.tensorflow.org/

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017, accessed on 8-11-2020.

[10] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, 2011, pp. 342–347, accessed on 10-11-2020.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[12] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018, accessed on 10-11-2020.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

## APPENDIX

### A. Model

| Layer (type) | Output Shape | Param |
|---|---|---|
| Input Layer | [(None, 32,32,3)] | 0 |
| conv2d | [(None, 32,32,3)] | 896 |
| batch_normalization | (None, 32, 32, 32) | 128 |
| conv2d_1 (Conv2D) | (None, 32, 32, 32) | 9248 |
| batch_normalization_1 | (None, 32, 32, 32) | 128 |
| max_pooling2d | (None, 16, 16, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 16, 16, 64) | 18496 |
| batch_normalization_2 | (None, 16, 16, 64) | 256 |
| conv2d_3 (Conv2D) | (None, 16, 16, 64) | 36928 |
| batch_normalization_3 | (None, 16, 16, 64) | 256 |
| max_pooling2d_1 | (None, 8, 8, 64) | 0 |
| conv2d_4 (Conv2D) | (None, 8, 8, 128) | 73856 |
| batch_normalization_4 | (None, 8, 8, 128) | 512 |
| conv2d_5 (Conv2D) | (None, 8, 8, 128) | 147584 |
| batch_normalization_5 | (None, 8, 8, 128) | 512 |
| max_pooling2d_2 | (None, 4, 4, 128) | 0 |
| flatten (Flatten) | (None, 2048) | 0 |
| dropout (Dropout) | (None, 2048) | 0 |
| dense (Dense) | (None, 1024) | 2098176 |
| dropout_1 (Dropout) | (None, 1024) | 0 |
| dense_1 (Dense) | (None, 10) | 10250 |
| Total params: | 2,397,226 | |
| Trainable params: | 2,396,330 | |
| Non-Trainable params: | 896 | |

TABLE V
MODEL SUMMARY

## LIST OF FIGURES



## LIST OF TABLES