Student Number: 183708

1. Approach

This report details the design and implementation of a system for multi-class image classification using a deep neural network architecture. The data for this assignment is the CIFAR-10 image dataset. This is a balanced dataset comprised of small 32×32 RGB images, each belonging exclusively to one of the ten classes. Examples are displayed in Appendix A, Figure 12.

Convolutional neural networks (CNNs) are standard in image recognition tasks and have achieved success in classification challenges such as on the CIFAR-10 dataset [15, 19, 10]. These networks cover a range of smaller two-layer networks to much larger networks with a depth of 100 or even 1000 layers. CNN's suitability for such tasks is due to their ability to learn characteristics from the data without supervision and grasp complex spatial dependencies within an image. For this project, we begin with a smaller-sized network with only two convolutional layers and a fully connected (dense) output layer. This provides the first baseline; the second baseline expands on this by increasing the depth of the network by four additional convolutional layers. This second baseline uses inspiration from VGG network architecture [27] and is for the purposes of demonstrating the effects deeper networks and the issues faced with adding such complexity. The baselines implemented here make use of three different layers: convolutional, dense and pooling layers. Both baselines also use cross-entropy loss due to the small number of classes in CIFAR-10; this loss function has been shown to perform well in such a classification task [35]. Only minor pre-processing is applied to the CIFAR-10 image data: inputs RGB values are normalized to be between $[0, 1]$. It is worth noting that in the case of image data this is not strictly required as the scale of pixels values will be relatively equal - between $[0, 255]$ - however, this can have an effect of allowing the network to converge faster and be more stable so this pre-processing is applied.

The convolutional layer applies a filter across a multi-dimensional array (an image in this case) where each input results in an activation. This repeated action creates a feature map and indicates the strength of an input's detected features as well as its location. The baselines implemented here and their extensions use small kernel sizes (the size of the grid of pixels to convolve) and initially small filter sizes (the number of convolutions to create the feature map). The filter sizes scale upwards with each convolutional block whereas kernel size is kept at 3×3 - a receptive field sufficient enough to capture features to the left and right, and up and down [27].

Each neuron in the dense layer takes input from neurons in the previous layer and connects to all neurons in the following layer. In the first baseline the dense layer is used only as an output layer. This uses a softmax activation function to translate the would-be outputs into a probability distribution suitable for multi-class classification.

The operations in the pooling layer typically take a maximum, a sum, or an average over a local area to create invariance to the small transformations which result from the sliding convolutions. This preserves the important details in the joint feature representations and discards the less important details [2]. By reducing the dimensionality of the feature maps, and so the number of parameters required to train, pooling also has the effect of decreasing the computational complexity required for the network to learn. The implementations covered in this project use max-pooling with a kernel size of 2×2.

2. Methodology

After normalizing the image data and one-hot encoding the labels, we split the CIFAR-10 supplied training data into training and validation partitions using Sklearn's train-test-split function [25]. This has the option of passing an additional *statify* parameter as to also keep a class balance. To fully gauge the effects of trialing different network layouts and layers, as well as tuning the hyper-parameters of the network, we experiment with the different features one-by-one. All following networks are implemented using TensorFlow's Keras API.

The first of the features tested was batch normalization (BN) [16]. BN is implemented as a layer within the network and solves the issue of the distribution of a layer's inputs changing as the previous layer's parameters update during training - known as *Internal Covariate Shift* [16]. BN mitigates this by reparameterizing the network to standardize the activations of each input variable, which in turn scales each layer's output [9]. The effect of this, is that training is stabilized, the time taken to train is reduced, and it has a mild regularizing effect on the network [21]. In the case of this project, BN was introduced to reduce the overfitting seen in Baselines-1 and 2 but also to provide grounds for testing out the effect of applying the BN layer before and after the activation functions of the trainable layers. The original paper details applying BN in-between the linear and non-linear layers to achieve the desired stable distribution however we also experiment with applying the BN layer after the activation and with different activations, such as hyperbolic tangent (tanh) and sigmoid (more on this later). Varying values for training batch size are also experimented in tandem to BN; this is to explore the effect of smaller batch sizes on the BN's statistics estimations, which can diminish the regularizing effect and cause the model to un-

derfit [30].

The next setting explored is the heuristic for weights initialization. This defines the initial values for the network's parameters prior to their training. In the past these values have been randomly sampled from Gaussian or uniform distributions [9] however, more modern approaches are decided by the choice of activation function. Following sigmoid or tanh, generally a form of *Glorot* (Xavier) initialization is used [8] whereas for networks that use the rectified linear activation function (ReLU), *He* initialization is preferred [11]. For a fun sanity check we also demonstrate the effects of initializing the weights to zero and forcing the symmetry of hidden units sharing identical input, output weights [1]. Due to empirical success, *He* initialization and a batch size of 256 are included in baseline-2 going forward.

At this point, having explored the aforementioned parameters, the baseline-2 model still showed signs of overfitting - potentially due to its larger size and the comparatively small dataset. Thus, the next setting tested was introducing dropout layers. These layers remove or "dropout" outputs from the previous layer [28] and has the effect of making the training process noisier by probabilistically forcing nodes to co-adapt and correct the mistakes passed up from the layers prior. High dropout rates on the final dense layers have led to success on CIFAR-10 [14], however, as the dense layer within the baseline is also the output layer, we solely experiment with dropout applied after the convolutional layers. Both fixed rates and increasingly large dropout rates per layer are tested.

Another technique to reduce overfitting is weight decay [20]. This technique adds a penalty which penalizes oversized weights. This serves the purpose of avoiding creating a model with large variance and small bias, i.e. the model is less sensitive to statistical noise, as is seen with networks with large weights [26]. Techniques for doing so are *L1*, *L2*, and *L1 L2*, which are the sum of absolute weights, the sum of the squared weights, and the sum of both. The networks implemented in this project only apply kernel regularizers (as opposed to bias regularizers) and we test the regularizing effect of weight decay at different orders of magnitude both with and without dropout layers.

The next hyper-parameter setting explored is one that dictates how the weighted sum of a layer's input is transformed into an output: the activation function. The experiments thus far have used Rectified Linear Activation (ReLU) [24]. This retains some of the easily-optimizable properties of linear activation functions [9] and solves some of the vanishing gradient issues [4] caused by using hyperbolic tangent or sigmoid in deeper networks. The activation function experiments in this project test ReLU, Leaky ReLU (this introduces a positive gradient for inactive nodes) [23], sigmoid, and tanh. These four activations are applied in the convolutional layers of the baseline networks and are tested in the context of: before a batch normalization layer, with no BN, after a BN layer.

Learning rate is the final parameter tested; this is often called the most important hyper-parameter [9] as this controls how well a network can adapt to a problem - too small a rate and training will be stunted whereas too large a rate can make cause convergence to suboptimal solutions. Learning rate is explored in the context of different optimization algorithms which offer adaptive [18, 32, 7] and non-adaptive learning rates. After testing a range of these algorithms at their default settings - bar RMSprop [13] and stochastic gradient descent, which are tested with momentum and momentum and Nesterov momentum [29] respectively - we explore a range of initial learning rate values for the best performing algorithms.

Finally, to combat the overfitting that was still present across many of the previous experiments, we introduce data augmentation. Increasing the size of the training set is a way to improve a model's performance and its generalization ability [22]. The previously mentioned models [20, 19] in Section 1 enjoyed image recognition success using a variety of data augmentation techniques. This project uses techniques from Tensorflow's data augmentation library and in addition, implements functionality which introduces *cutout* [5] and *mixup* [33]; these two techniques have been shown to improve performance and generalization in image recognition tasks such as CIFAR-10.

3. Results

To build a grounds for comparison, we first test the two baseline models. These were trained on an $80/20$ split of the training data (training, validation) and had callbacks monitoring validation loss that would halt training if no improvement was seen for three epochs. The full architecture for each is included in Appendix A, Figures 9, 10. These models performed as expected, with Baseline-2 outperforming Baseline-1 due to the added complexity. Figures 1 and 2 show each model's performance during training. Both experience dramatic overfitting after only a small number of epochs although Baseline-2 does perform better.

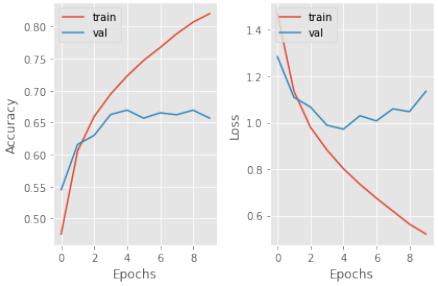

Figure 1. Baseline-1 training metrics

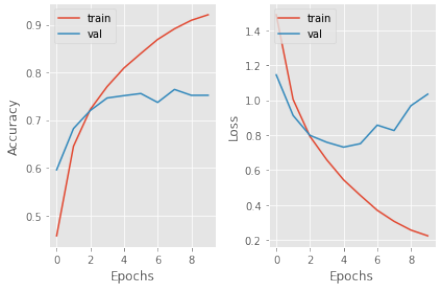

Figure 2. Baseline-2 training metrics

Baselines-1 and 2's performance on the 10,000 CIFAR-10 test images was evaluated in terms of accuracy: they scored 64.89% and 73.67% respectively and a confusion matrix displaying Baseline-2's predictions versus an image's actual label can be found in Appendix A, Figure 11. From this confusion matrix, we can see that Baseline-2 over predicts images belonging to the cat class when instead they are dogs, and it also shows difficulty in distinguishing ships and airplanes - perhaps due to their similarity in being metal objects surrounded by an often blue background. These models' convolutional layers use ReLU activation functions, are padded to preserve image size, and after each max-pooling layer the number of filters in a convolutional layer is increased to the next power of two.

Batch Normalization is then introduced to combat this overfitting. Both baselines were tested with BN before and after the activation functions and both baselines performed better with batch normalization before the activations - as stated in the original publication [16], applying before the non-linearity is where BN is most likely to result in a stable distribution. The difference in validation accuracy and validation loss when applying BN at different points is shown in Figure 3. This image shows, in terms of validation performance, there was no significant material difference between BN before activation and BN after. However, we observe in terms of the validation loss that applying BN after appeared resulted in validation loss increasing whereas applying it before continued to noisily decrease.

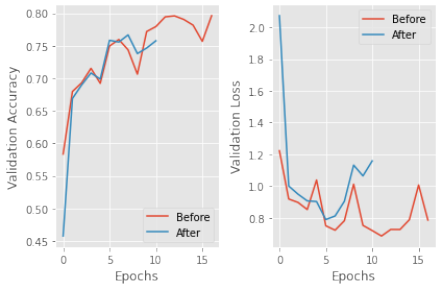

Figure 3. Baseline-2: validation loss and validation accuracy with BN applied before and after activation functions

Maintaining applying BN before the activation function for Baseline-2 and testing different settings for batch size results in Figure 4. This shows validation accuracy with changing values for batch size (for loss see Appendix A Figure 13). A batch size of 256 seemingly outperformed the others in terms of validation accuracy as well as greatly reduced the time taken to train when compared to TensorFlow's default batch size of 32. At batch size 256 we also observe the validation accuracy plateauing, which can be a sign that the model's optimizer requires review.

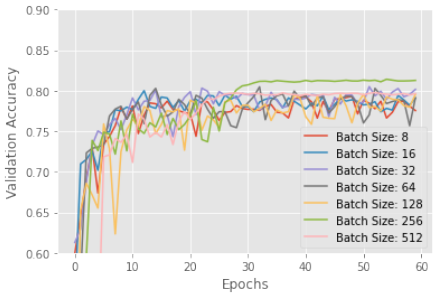

Figure 4. Baseline-2's validation accuracy, with BN applied before activation, on differing batch sizes

Figure 5 shows the effects on validation accuracy of different weight initialzations. As previous research had suggested [11], *He* initialization worked best for this model due to its use of ReLU. This image has *zero* initialization clipped as this setting does not train - the full image is found in Appendix A, Figure 14. Going forward, *He* initialization and batch size 256 are applied to all experiments.

The first dropout tests used a constant dropout factor of 0.2 on each convolutional layer. After twenty epochs, Baseline-2 had only started to mildly overfit and it achieved a 79.26% test accuracy score. Increasing dropout factor to a constant 0.5 had a negative effect and dropped the test accuracy score to 75.54% after twenty epochs. This may have been due to the model not yet converging and so epochs were increased to 50 and this was tested once again. The validation loss callback halted training after 43 epochs however, it had now reached a test accuracy of 80.33%. One last dropout experiment was performed. This time with increas-

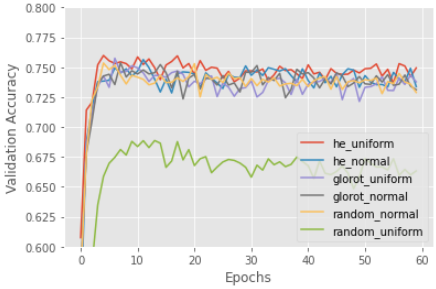

Figure 5. Trainable layer's weight initializations

ing values for each block of convolutional layer in Baseline-2: 0.2, 0.5, 0.7. This model scored 81.03% even though callbacks halted training after 32 epochs.

Whilst dropout had seemingly solved some of the issues faced with regards to overfitting, weight decay and sometimes work in balance with dropout to provide further regularization (although they may also work against each other [12]). Figure 6 compares varying amount of weight regularization applied to each trainable layer in the network; tested are L1, L2, and L1_L2. This image shows fairly similar performance in the 1×10^{-5} to 1×10^{-3} range. Past this L1 and L1_L2 drop off however, L2 regularization remains fairly well performing.

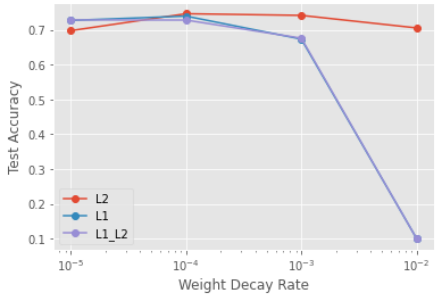

Figure 6. Weight decay over varying magnitudes, plotted on a logarithmic scale

The next feature tested was the choice of activation function in our trainable convolutional layers. The project had so far made use of ReLU - judging from Figure 7, this appeared to be a well suited choice. This image also shows us that for all except the hyperbolic tangent function, BN after the activation performed nearly as well if not better than BN before. Interestingly the sigmoidal function did perform slightly better when BN layer was placed after activation.

We next examine the effects of the learning rate in context of optimization algorithms. A total of nine were initially tested (Appendix A, Figure 15) and the top four best performing were explored in greater depths by varying the initial value for learning rate. Figure 8 displays this on a logarithmic scale. Perhaps due to their similarity, RMSprop and Adam's performances show no great material difference. Stochastic gradient descent and Adamax both perform better at large initial learning rates than the other two and the default learning rate for Adam (1×10^{-3}) appears well suited.

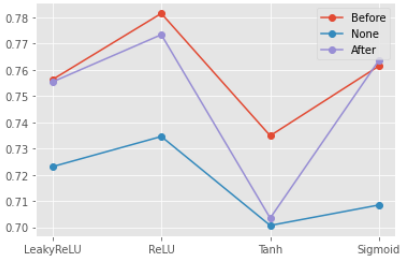

Figure 7. Comparison of activation functions tested alongside batch normalization - evaluated by test accuracy score

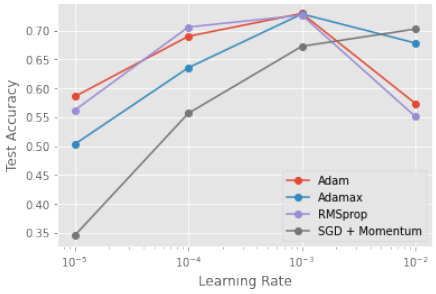

Figure 8. Comparison of optimizer algorithms with varying initial learning rates

Now that these hyper-parameters have been explored, we may combine some reasonable features, such as incrementally increasing dropout rates per convolutional layer, L2 weight decay (1×10^{-4}), and *He* weight initialization, into one model. This model uses the Adam optimizer with an initial learning rate of 1×10^{-3}, a batch size of 256 for training, and ReLU activation functions in the convolutional layers. This model scores 83.7% on the CIFAR-10 test set. Applying data augmentation surprisingly did not increase test accuracy or demonstrate considerably less overfitting. Applying width and height shifts, horizontal flips, and zooms yielded an accuracy of 82.2%. More recent techniques such as mixup [34] and cutout [6] performed similarly: 83.7% and 81.3% respectively. Examples of these techniques are shown in Appendix A Figures 16, 17. When combined together test set accuracy dropped to 78.6%.

4. Discussion

In this paper we have shown the effects of some of the hyper-parameters and techniques available for image classification using convolutional neural networks. Upon reflection, this project focuses on classification accuracy as the evaluation metric whereas it may have been interesting to perform error-analysis on the correct and incorrect classifications; this could have expanded on the confusion matrix shown and used class activation maps [36] to identify the areas the model was underperforming in and why this was so. Exploring data augmentation in more depth, using GridMask [3] or CutMix [31], may also have led to some interesting results as they expand on the regional dropout techniques implemented in this paper. Future works might look more in-depth at the relationship between the placement of the batch normalization layer and the activation function or attempted learning rate optimizer algorithm schedulers, which would alternate the choice of algorithm during training [17] once a condition is met.

References

[1] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012. 2

[2] Y.-L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010. 1

[3] P. Chen, S. Liu, H. Zhao, and J. Jia. Gridmask data augmentation. *CoRR*, abs/2001.04086, 2020. 4

[4] G. E. Dahl, T. N. Sainath, and G. E. Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013. 2

[5] T. Devries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. 2

[6] T. Devries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. 4

[7] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 2

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. 2

[9] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 1, 2

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. 1

[11] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. 2, 3

[12] D. P. Helmbold and P. M. Long. Dropout versus weight decay for deep networks. *CoRR*, abs/1602.04484, 2016. 4

[13] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning (lecture 6a). 2

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. 2

[15] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. 2018. 1

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. 1, 3

[17] N. S. Keskar and R. Socher. Improving generalization performance by switching from adam to SGD. *CoRR*, abs/1712.07628, 2017. 4

[18] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[19] A. Krizhevsky. Convolutional deep belief networks on cifar-10. 05 2012. 1, 2

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 2

[21] P. Luo, X. Wang, W. Shao, and Z. Peng. Towards understanding regularization in batch normalization. *CoRR*, abs/1809.00846, 2018. 1

[22] J.-J. Lv, X.-H. Shao, J.-S. Huang, X.-D. Zhou, and X. Zhou. Data augmentation for face recognition. *Neurocomputing*, 230:184–196, 2017. 2

[23] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013. 2

[24] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. 2

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 1

[26] R. Reed. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks (A Bradford Book)*. A Bradford Book, feb 1999. 2

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2015. 1

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. 2

[29] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. 2

[30] H. Yong, J. Huang, D. Meng, X. Hua, and L. Zhang. Momentum batch normalization for deep learning with small batch size. In *European Conference on Computer Vision*, pages 224–240. Springer, 2020. 2

[31] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. *CoRR*, abs/1905.04899, 2019. 4

[32] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. 2

[33] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017. 2

[34] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017. 4

[35] Z. Zhang and M. R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *CoRR*, abs/1805.07836, 2018. 1

[36] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 4

A. Appendix

Baselines-1 and -2's layout is shown below.

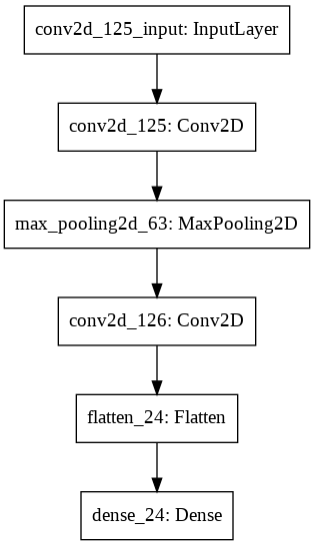

Figure 9. Baseline-1 Network Architecture

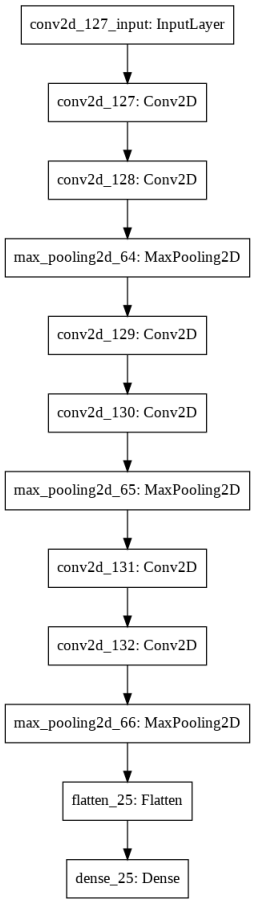

Figure 10. Baseline-2 Network Architecture

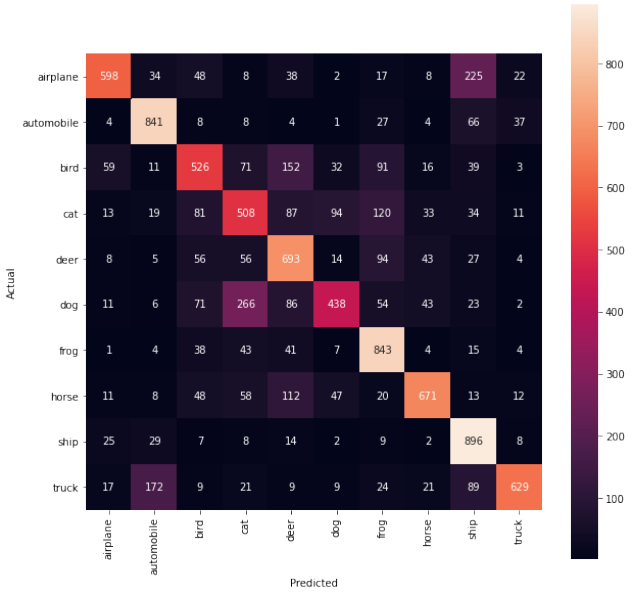

Figure 11. Confusion Matrix of Baseline-2's predictions versus actual labels per class

Figure 12. Examples from the CIFAR-10 Dataset

Baseline-2, batch normalization applied before activation functions, comparing validation loss with different values for batch size.

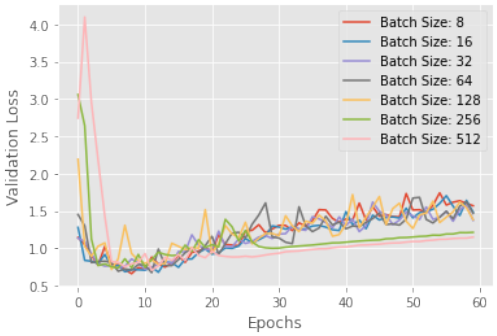

Figure 13. Baseline-2's validation loss, with BN applied before activation, on differing batch sizes

Weight initialization experiment with *zero* initialization included.

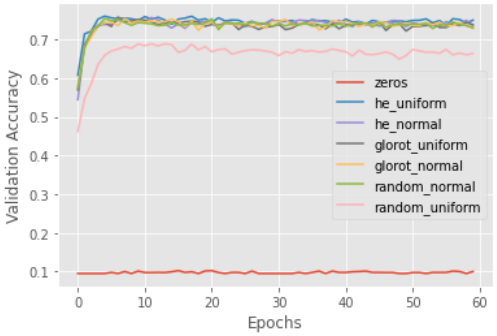

Figure 14. Weight initialization experiment in full

All optimizer algorithms tested in this project.

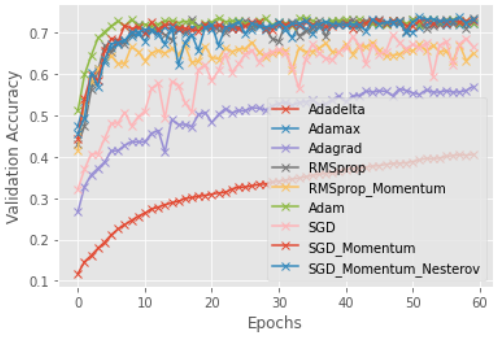

Figure 15. Optimizer algorithms' validation accuracy scores over increasing epochs

Example of mixup data augmentation.

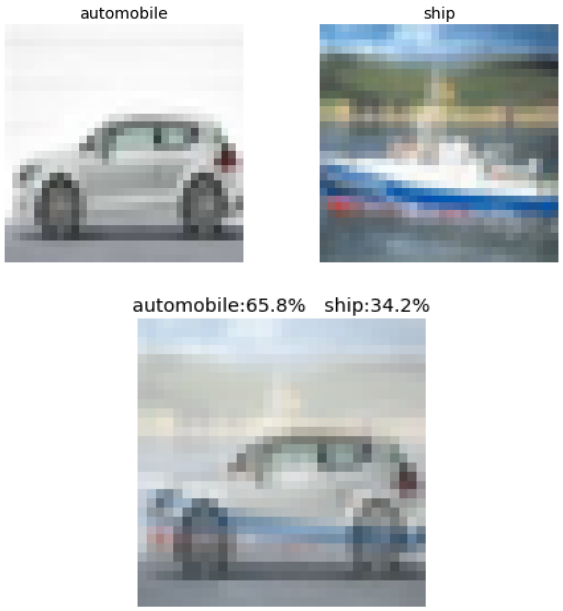

Figure 16. Mixup of images from the automobile and ship classes

Example of an image after Cutout data augmentation.

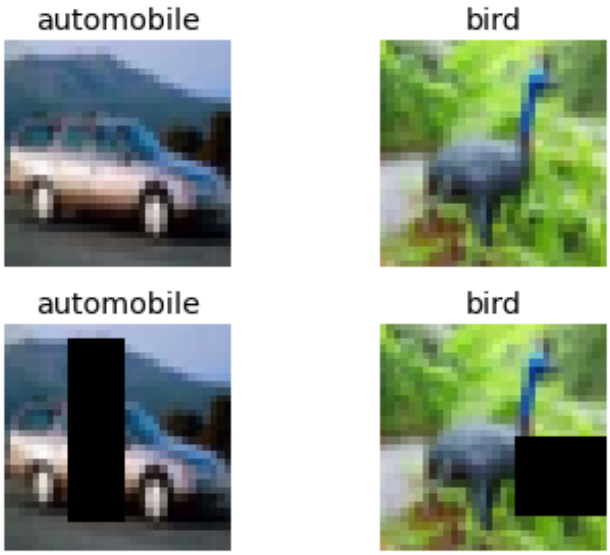

Figure 17. Regional dropout: Cutout on two images