MLP Coursework 4 - Report

Muhammad R.B.E. Noerrahman - S1687182

Introduction

Building up from the previous effort in coursework 3, this experiment studies the architecture and implementation of Convolutional Neural Network (CNN) on the same data set as the previous experiment, CIFAR-10. Currently, the best CNN implementation on CIFAR-10 is the *Fractional Max-Pooling* method which performs at an accuracy of 96.53%. [1]. The approximated human performance accuracy is 94% [2]. From the previous experiment, an accuracy of 55.2% was achieved using only three fully connected layers and L2 regularisation. Yet, the aim of the previous experiment was to set up a baseline for this experiment, thus high performance on accuracy was not a priority.

In this experiment, only shallow (3 convolutional layers max.) CNN implemented due to constraint in computing resource and again, the objective is not to create a novel approach in CNN or produce high-performing model, but to study the implementation of the architecture, such as using a simplified implementation of *max fusion* proposed by Feichtenhofer et al. [3] and how can CNN be best implemented with performance boosting methods such as L2 regularisation and Dropout. Despite focusing on deep learning in the previous experiment, in this experiment, wide learning is also investigated through the implementation of *max fusion*. In addition, *max fusion* also allows observation into how networks could share their knowledge through combining their outputs.

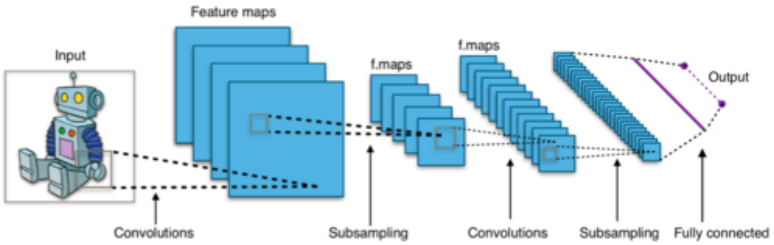

Figure 1: An example of a convolutional neural network. Source: Wikimedia Commons [4]

A simple description of CNN can be defined as a neural network which consist of layers that take in 3 dimensional tensor inputs of the image and produce features map or filters from the inputs, while a normal fully connected layer in a regular neural network (NN) only takes in input in 1 dimensional vector format and produce weights of similar shape and scale to input, due to this, fully connected layer is not ideal for classifying large images as the number of weights in the first layer would have to be as large to scale to the input *e.g.* for an RGB image of size 250×250, ideally there should be $250 \times 250 \times 3 = 187500$ neurons [6]. While in convolutional layer, the neurons are in the same dimension as the inputs, such that for a CIFAR-10 image of size, the neurons are arranged depthwise.

The outcome of this experiment is to produce a CNN model that performs better than the baseline result of the previous experiment. Some of the tests are quite similar to the previous experiment, but as convolutional layers behave differently from fully connected layer, we cannot implement the same settings to the convolutional layers. Thus, some of the tests are similar in spirit to some of the tests from previous experiment, so that optimum setting for convolutional layers can be acquired.

Convolutional Neural Network

In essence, a CNN consists of three main parts, **Convolutional Layer** (CONV), **Pooling Layer** (POOL) and **Fully-Connected Layer** (FC). These layers are usually stacked as such,

$$\text{INPUT} \rightarrow [\textbf{CONV}] \rightarrow [\textbf{POOL}] \rightarrow [\textbf{FC}] \rightarrow \text{OUTPUT}.$$

CNN comes from the biological interpretation of how complex cells in the brain react to stimulations from receptive field in the eye, such that every cell produces response when a stimulus in a certain direction of the receptive field is induced. This is the basis of how a convolutional layer behaves. While on the other hand, a pooling layer is a subsampling function that reduces the spatial dimension of the convolutional layer output.

Convolutional Layer

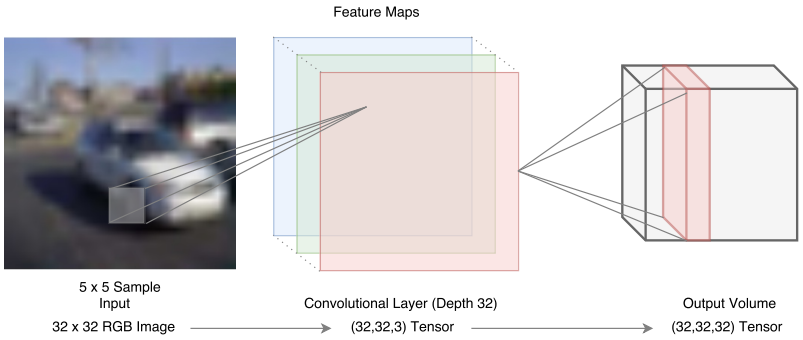

Figure 2: A convolutional layer on a sample of CIFAR-10 images.

A convolutional layer views an input in 3 dimensional depthwise perspective. In the case of an RGB image such as from CIFAR-10 as shown in Fig. 2, the depth of the image is the RGB colour channels which is 3. Similar to the learnable weight parameters in an FC layer, a CONV layer consist of small (in this experiment set to 5×5 pixels) 3 channels deep filters that are trained to be *experts* at filtering different features in images. These filters produce 2-dimensional activation features map, which in analogy to an FC layer would be a 2-dimensional weights. Each of these filters will learn to look for different features in the inputs.

A CONV layer is controlled by four hyperparameters, **depth**, **stride**, **filter size** and **padding**. Depth can be interpreted as the output dimension of the CONV layer as it corresponds to then number of learnable filters to be applied on the inputs. Stride defines how much should these filters slide during training. The size of stride is defined in pixels, in most cases this is set to 2 pixels. Filter size, as the name suggests, defines the size of these filters or in biological term the size of the *receptive field*. Padding on the other hand sets zero values around the border of the input volume, this allows control over spatial size of the output volume [6].

Pooling

The main function of a pooling layer is to reduce the spatial dimension of the output volume. A Pool layer has two parameters, **filter size** and **stride**. These parameters behaves similarly

as those in a convolutional layer. The most generally used pooling function is the max pooling, which is used in this experiment, but there are other pooling functions such as average pooling and L2-norm pooling. Fig. 4 shows how a max pooling layer acts on an output.

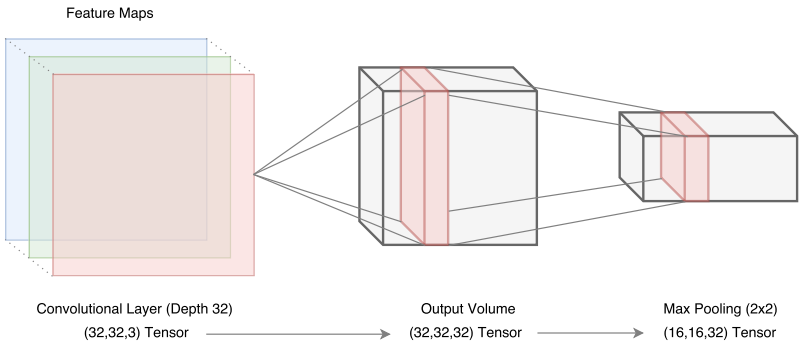

Figure 3: A 2x2 Max Pool on the output volume of the convolutional layer.

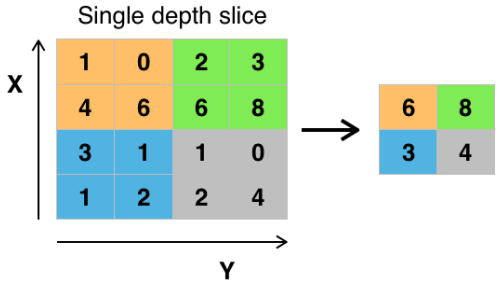

Figure 4: How a 2x2 max pooling with stride of 2 downsample an output. Source: Wikimedia Commons [5]

Local Response Normalisation

Local Response Normalisation (LRN) was brought forward in 2012 by Krizhesvsky et al. as a method to aid generalisation [7]. Although as suggested in the paper, that input normalisation is not explicitly required for ReLUs to prevent oversaturation, there is still a slight advantage in using normalisation. Thus why LRN is implemented in the experiment.

LRN is defined as,

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} \left(a^j_{x,y}\right)^2\right)^\beta}$$

where $a^i_{x,y}$ is the output of a unit or neuron computed by applying filter or kernel i at position (x, y), N represents the number of filters in that layer. $b^i_{x,y}$ is the *response-normalised* output calculated by sum over n adjacent filter maps at the same spatial position. While k, n, α and β are hyperparameters of the function. In TensorFlow, this normalisation is built into the library as **tf.nn.lrn**, eliminating the need to define a new function.

Maximum Fusion of Convolutional Layer Output Volumes

In 2016, Feichtenhofer et al. put forward a novel approach in CNN, named *Convolutional Two-Stream Network Fusion* [3]. This approach was implemented for video action recognition. One of the methods of fusing two convolutional networks is the *max fusion*, where it takes the maximum of two feature maps, defined as,

$$y_{i,j,d}^{max} = max\left\{x_{i,j,d}^{a}, x_{i,j,d}^{b}\right\}$$

where $x^{a,b}$ are the output volumes of the CNNs and $y_{i,j,d}^{max}$ is the spatially fused output volume. In the paper, the two CNNs are trained on two different representations of the inputs, where one takes in temporal stream of the data, the other takes in the spatial stream of the data.

In this experiment, the implementation if max fusion is simplified into two CONV layers, with different representations of the inputs, where one takes in the RGB images of the inputs, the other takes in the grayscale images of the inputs. The implementation in TensorFlow is simply done by placing a **tf.maximum()** after these layers to combine their output volumes by taking the maximum. The intuition behind applying this is that shapes and features are easier to recognise in grayscale rather than in colour. Thus combining these two feature maps, a more robust fused feature maps can be produced. On the other hand, in a simplified method this is an investigation into whether knowledge can be shared between two different neural networks to produce better understanding.

Beside testing how max fusion could improve learning, a less traditional approach to data augmentation is taken in the max fusion test, where in one of the models, the grayscale CONV layer is fed lower contrast images. In this approach, data augmentation is not simply added to the input batch, but a different network simply learns from an augmented data and share its knowledge with another network which learns from the standard inputs.

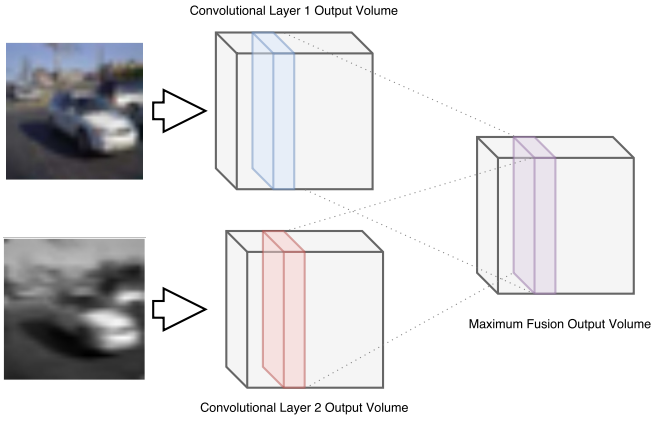

Figure 5: Max fusion implementation in the experiment.

Method

The experiment utilises TensorFlow, a high-level ML Python library, as used in the previous experiment. To keep a uniformity between this experiment and the previous, the same training batch size of 128 and validation batch of 50 are used. Although in this experiment, training epochs

are varied, since some model requires far more than 20 epochs to train, thus the number of epochs used are either 20, 50, 100 or otherwise stated.

In the last experiment, ADAM was set to be the default learning rate scheduler, which in this experiment was proven to be less efficient, thus in most of the tests Momentum Optimizer (MomOpt) is used as a test will prove that momentum learning schedule optimizer is more efficient at maximising accuracy. MomOpt's initial learning rate is set to 0.01 with momentum of 0.9. The built in ReLU activation was proven to work rather well in the previous experiment, but has proven otherwise for a convolutional layer. The Leaky ReLU implementation from the previous experiment [9] produces better accuracy than ReLU. On the other hand, beside from regularisation methods and hyperparamters searching, a rather interesting neural network architecture approach is tested, where the outputs of two convolutional layers, where one takes in RGB images and the other takes grayscale version of the same images are combined, by taking the maximum values from both of the feature maps. For the fully connected layer(s) of the model, ReLU is chosen to be the default activation function for all tests, as previous experiment have shown ReLU to be effective for fully connected layers. The default hyperparameters setting for the convolutional layer are **strides** = 2, **filter size** = 5 x 5 and **padding** is set to 'SAME' (padding of [2,4]). The pooling operation used in the experiment is the Max Pooling operation with filter size of 2×2 and stride of 2. From the previous experiment on FC layers, the best performing dimension for a single layer was a layer of 1024 neurons, this is set as the default number of neurons for the FC layer.

On top of these, Dropout and L2 regularisation are also applied. In some cases, the Dropout method can be rather aggressive at dumping neurons, this is due to the constraint of number of epochs the test could be run to produce significant results, as for time efficiency reason. During training, the model is tested on validation set at every 5 epochs. Regularisers are implemented using Tensorflow built in functions, such as **tf.nn.dropout** for Dropout and **tf.nn.l2_loss** for L2 regularisation. Different **keep probability** of Dropout are implemented, with the standard, general case value of **0.75** in some tests and a rather more aggressive keep probability of only **0.45** in other tests. L2 **beta** hyperparameter is kept constant at 0.001 on all tests. Local Response Normalisation (LRN) is implemented for every convolutional layer with the default hyperparameters of **depth radius** = 4, **bias** = 1.0, **alpha** = 0.001/9.0 and **beta** = 0.75.

On the hardware side, the experiment is run fully on an Amazon Web Service (AWS) GPU Compute p2.xlarge instance, which runs on an NVIDIA K80 GPU, 61 GB RAM and 4 virtual CPUs of 2.7 GHz each

Results - Single Layer Convolutional

Similar to the previous experiment, to find the optimum setting for the CNN, the most effective activation functions, dimensionality and learning schedule for the convolutional layer must first be obtained. Thus, the main objective of the first few tests are to find which settings and functions are to be used as the default in the rest of the experiment. The later tests are mostly on observing regularisation methods, hyperparameters testing and architectural implementation of the CNN. For the next four tests on single convolutional layer, the following model is used.

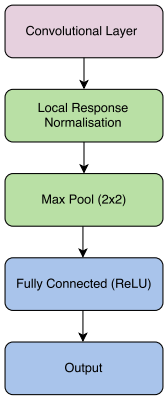

Figure 6: Single convolutional layer model.

Activation Functions

In the previous experiment, ReLU was proven to perform well with FC layers. Yet, following the claim made by Xu et al. [8], the Leaky ReLU (a type of Parametric ReLU) activation should perform better than ReLU in a CIFAR-10 classification using CNN. In 2015, He et al. [9] surpassed human human-level performance on ImageNet using a CNN which implements Parametric ReLU. Thus, Leaky ReLU (LReLU) should overcome standard ReLU performance in this experiment. The objectives of this test are to observe how different activation functions affect a CONV layer and to find the best performing activation function for the rest of the tests.

ReLU is defined as

$$f\left(x_i\right) = max\left(x_i, 0\right),$$

where x_i is the input value. LReLU is defined as ,

$$f(x_i) = \begin{cases} x_i & x_i \geq 0 \\ \frac{x_i}{\alpha_i} & x_i < 0 \end{cases}$$

where $\alpha_i \in (1, +\infty)$. Xu et al. [8] suggests that α_i to be set at a smaller value of 5.5 rather than the original value of 100, put forward by Maas et al. in 2013 [10]. In this experiment, α_i is set to the suggested value of 5.5. Another Parametric ReLU implementation, Random ReLU [8], from the previous experiments is also tested, but the focus are on ReLU and LReLU.

LReLU can be implemented rather simple in TensorFlow as shown,

$$\textbf{out = tf.maximum(x*alpha, x)}.$$

where **x** is the input from the layer.

The test consists of 20 epochs of training and utilises ADAM optimiser with the default learning rate of 0.0001. The dimension of the CONV layer is set to be 64. The activation function of the FC layer is defaulted to ReLU. LRN and Max Pooling of [2 x 2] are also implemented.

In accordance to the hypothesis made earlier, LReLU does perform better than ReLU in a CONV layer as shown in Table 1 and also from the graphs in Fig. . Indeed, that simplified implementation of Random LReLU surpasses the performance of LReLU, but due to the fact that it is stochastic method, perhaps the performance might not be reflected as well later in the experiment, thus for uniformity, LReLU is chosen as the default activation function of the CONV

Activations	Accuracy(Train)	Accuracy(Valid)
ReLU	0.979	0.583
LReLU	0.982	0.589
RLReLU	0.977	0.595
Sigmoid	0.559	0.532
Tanh	0.976	0.591

Table 1: Accuracy after 20 epochs training with different dimensions.

layer(s). Although, in all cases except Sigmoid, it is clear that heavy overfitting is occurring as shown in the validation error graphs. This is one of the reasons why a significantly small keep probability is used in the later tests.

Depth/Number of Filters

The aims of the following test are to observe how number of filters (**depth**) of a CONV layer affects its performance and thus find the most effective depth for the CONV layers for later tests. Depth affects the size of the output of the CONV layer, thus it also affects the size of the computation in the layers. As for the input volume of an RGB image of $[32 \times 32 \times 3]$, a CONV layer of 16 filters will produce an output volume $[32 \times 32 \times 16]$.

Indeed, with larger number of filters, the model would produce much more detailed feature maps, but this also has to reflect with the spatial size of the images, such that it would be almost a waste to use 128 filters/ depth for 32×32 images. The economy between time and accuracy must also be met. Larger depth will produce higher accuracy, but it will also be time consuming. This test aims to find a depth that is both time efficient yet still produces significant accuracy. The same model as the previous test is implemented, with LReLU as activation function.

Depth	Accuracy(Train)	Accuracy(Valid)
16	0.993	0.595
32	0.994	0.598
64	0.984	0.607
128	0.983	0.614

Table 2: Accuracy of model after 20 epoch training with different depths.

As stated earlier, larger depth does indeed produce more accurate results, but it also does take significantly more time to complete an epoch, as depth of 128 takes almost 2 mins to complete an epoch. It would be far too timely to use this much depth. From Table 2, the most reasonable choices for the size of depth for the experiment is either 32 or 64. Time-wise, the time per epoch for 64 is not significantly larger than 32, yet it does produce higher accuracy and less overfitting than depth of 32. Also, with the hardware provided, perhaps it would interesting to exploit the computational power slightly more by using slightly larger depth. The depth of 64 will be used as the default depth of CONV layer in the experiment.

Learning Rate Scheduler/Optimiser

As choices of activation function and depth of CONV layer have been established, the next logical step is to find the optimum learning rate scheduler for the model. The two main features that are

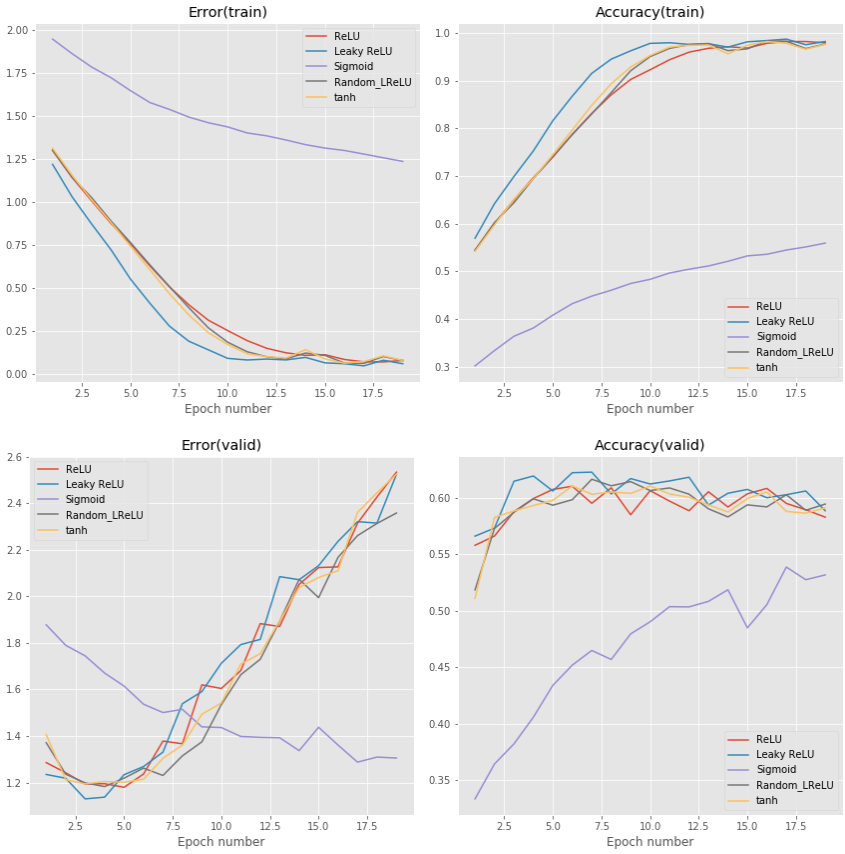

Figure 7: Accuracy and errors from training and validation of activations test.

desired from the scheduler are, rapid convergence (as it is costly to run long test of many epochs on AWS and also is time-inefficient) and best accuracy.

The two optimisers that are tested are Momentum Optimiser (MomOpt) and ADAM. ADAM is chosen to represent adaptive gradient descent algorithms, as in previous test, ADAM compares well to other adaptive algorithms such as ADAGrad and ADADelta. Momentum is chosen out of the simplicity of its algorithm. Thus, producing a comparable result that shows whether more novel adaptive algorithms or simple adaptive algorithm such a Momentum would suffice for a shallow CNN model trained for a short number of epochs.

MomOpt's initial learning rate is set to 0.01 with momentum of 0.9. ADAM is initialised according to the default setting with learning rate of 0.001, $\beta_{\mathbf{1}} = 0.9$, $\beta_{\mathbf{2}} = 0.999$ and $\epsilon = 1e-08$.

Optimiser	Accuracy(Train)	Accuracy(Valid)
ADAM	0.864	0.632
MomOpt	0.999	0.659

Table 3: Accuracy of model after 20 epoch training with different optimisers.

From Fig. 9 and Table 3, it is very clear that MomOpt can give the fastest convergence and highest accuracy, with the caveat of overfitting. Although problematic, overfitting can be tackled using regularisation, in which will be investigated in the next test.

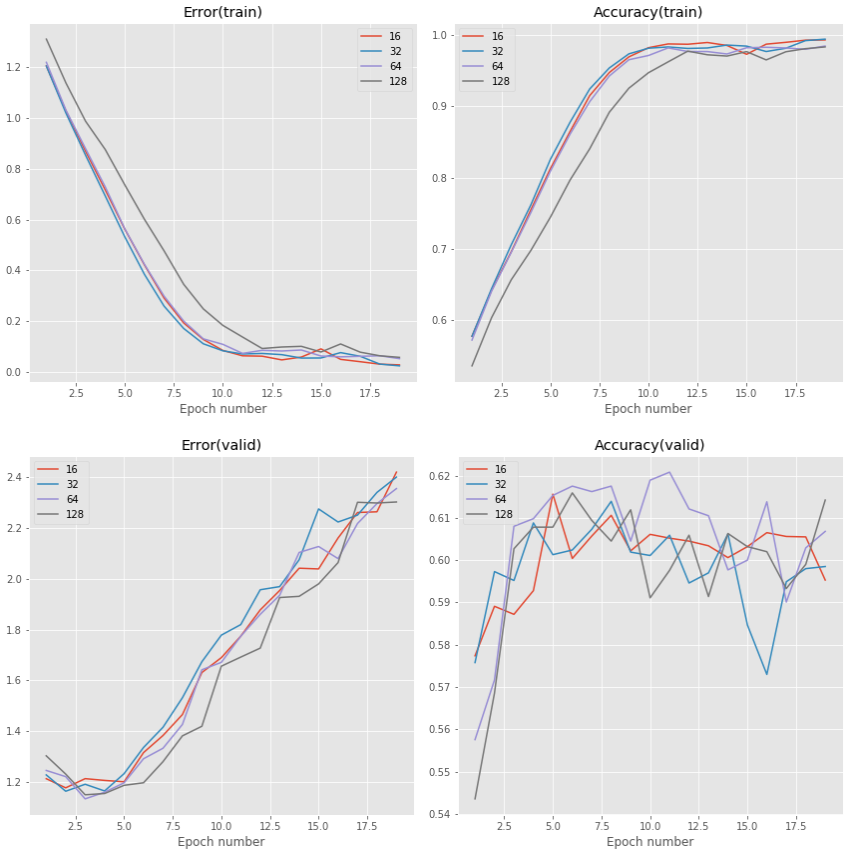

Figure 8: Accuracy and errors from training and validation of depths test.

Regularisation

In this test, two regularisations are implemented, L2 and Dropout. In the first test, only Dropout is applied to the model, while in the second both are applied. The tests are done to observe whether Dropout would suffice to regularise the model or whether another regularisation is required.

Although Srivastava et al. [11] suggest that a Dropout on a convolutional layer should have a keep probability of > 0.70, this would take a longer time to produce more significant result. In this test, a rather more extreme value is used, as the keep probability is set to 0.45 on all layers.

In the previous baseline experiments, L2 on its own works the best, achieving the highest performance out of all the tests with a 55.2% accuracy.

Regularisation	Accuracy(Train)	Accuracy(Valid)
Dropout	0.720	0.653
Dropout + L2	0.722	0.663

Table 4: Accuracy of two hidden layers model after 20 epoch training with different number of neurons.

Referring to both Fig. 9 and 10, it can be observed that Dropout solves the large overfitting of the MomOpt on the model. While in Fig. 11, L2 minimises this overfitting further and boosts accuracy from 0.653 to 0.663. This reflects the result from the previous baseline experiment, that simply by using L2, the best performance was achieved.

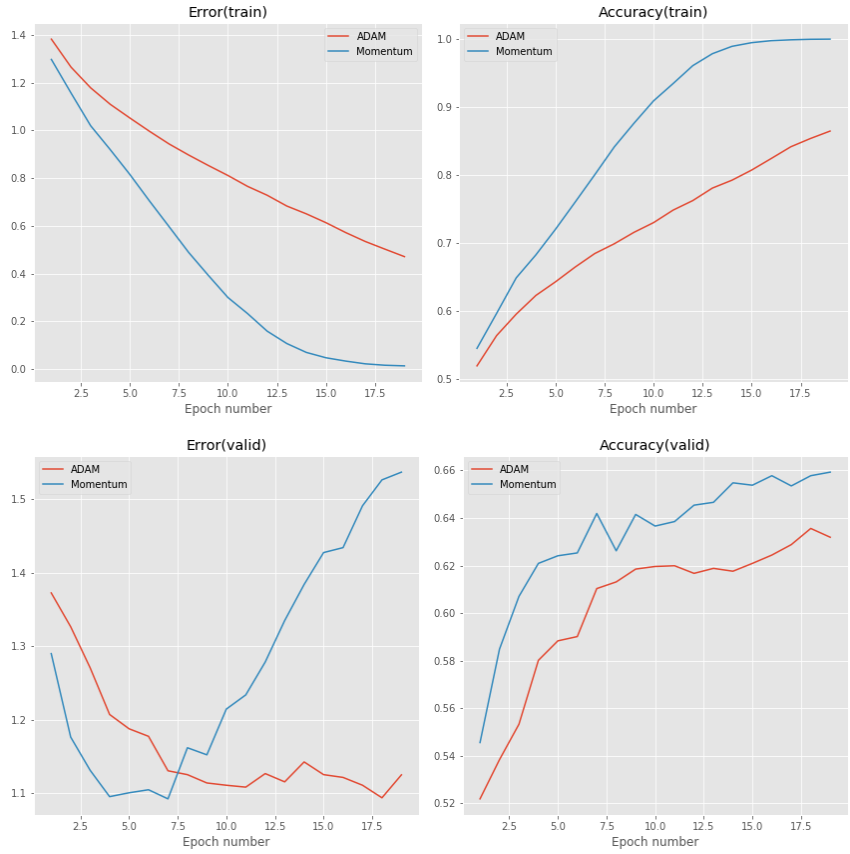

Figure 9: Accuracy and errors from training and validation of optimisers test.

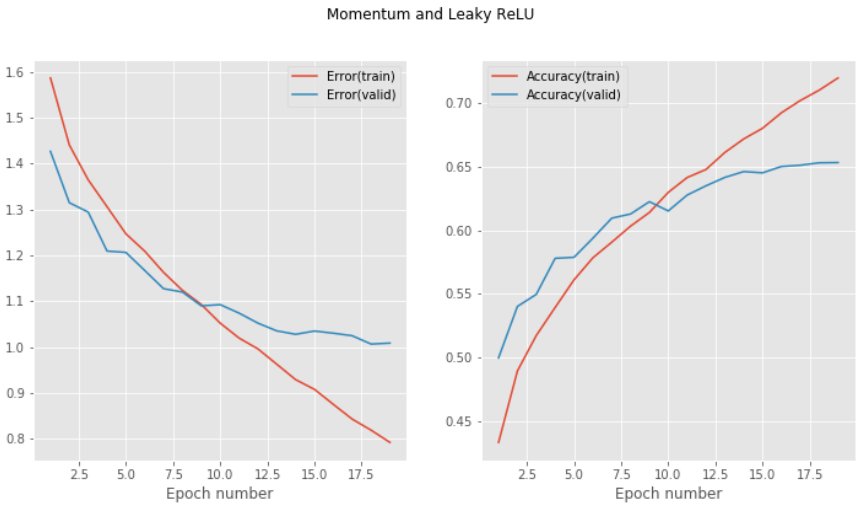

Figure 10: Error and accuracy of model regularised using only Dropout.

Results - Multiple Layer Convolutional

The previous tests on single convolutional layer serves as the basis to build larger CNN. The approach of this part experiment are to study how Dropout is best implemented in a shallow CNN

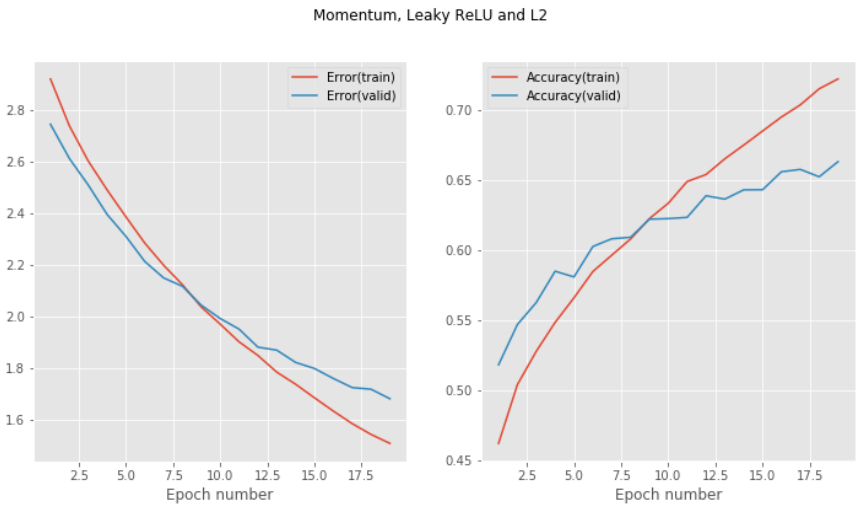

Figure 11: Error and accuracy of model regularised using Dropout and L2.

and how fusing two convolutional layers, creating a wide and shallow network can improve the performance of a CNN. In the baseline experiment, the best performing model consisted of three FC layers, while in this experiment, we shall observe whether lesser layers of CONV can produce better results.

Implementing Dropout in a Shallow CNN

As suggested in the earlier test, the current implementation of Dropout is not in accordance to the suggested setting given by Srivastava et al. [11]. Srivastava et al. given that a keep probability of 0.75 should be implemented on the CONV layers and a keep probability of 0.5 on the FC layers for best performance, yet this was performed for a training with large number of epochs, which is not ideal for this experiment. On the other hand, a CONV layer already has a small number of kernels, in comparison to the size of weights given by an FC layer, perhaps its best to keep all these kernels intact and only apply Dropout on the FC layer.

In this test, a two CONV layers model is used, implemented as shown below in Fig. 12 and the training epoch is set to 100.

Dropout Implementation	Max Accuracy(Train)	Max Accuracy(Valid)
Keep Prob = 0.45 on All Layers	0.775	0.696
Keep Prob = 0.75 (CONV) and 0.50 (FC) [11]	0.925	0.625
Keep Prob = 0.45 (FC Only)	0.984	0.653

Table 5: Accuracy after 100 epochs training with different implementation of Dropout.

As shown in Table 5, the current implementation of Dropout outperforms the other implementation for short-epochs training. In Fig. 13, 14 and 15, the other implementations also seem to overfit significantly more than the current implementation. This also proves that it is possible to outperform FC deep learning model with lesser layers of CONV layer.

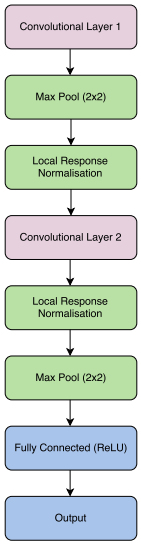

Figure 12: Two convolutional layers model.

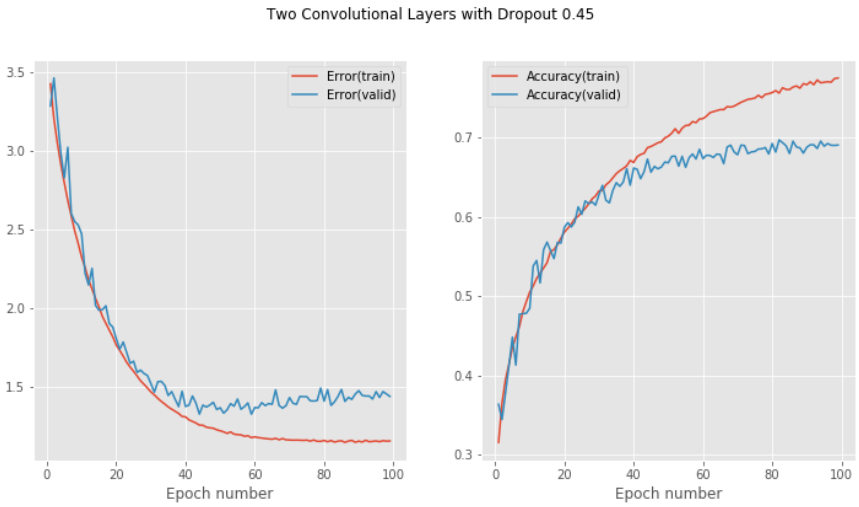

Figure 13: Error and accuracy of model with Dropout implementation of Keep Prob = 0.45 on all layers.

Wide CNN: Fusing the Output Volume of Two Convolutional Layers

Using the *max fusion* technique described earlier, in combination with all the best performing models from previous tests, in this test the best implementation of max fusion is investigated. The following architecture shown in Fig. 17 is implemented.

As stated before, this test inquires the implementation of wide learning and whether it can perform as well as a deep learning model, while also studying the notion of knowledge sharing in a CNN. In this test two different approaches are taken, where in one model, the grayscale CONV layer takes in just grayscale images, and in the other, the grayscale CONV takes in grayscale images of lower contrast (set to 0.5 lower than mean contrast of the image). The grayscale conversion is applied using **tf.images.rgb_to_grayscale()**. The motivation behind shifting the

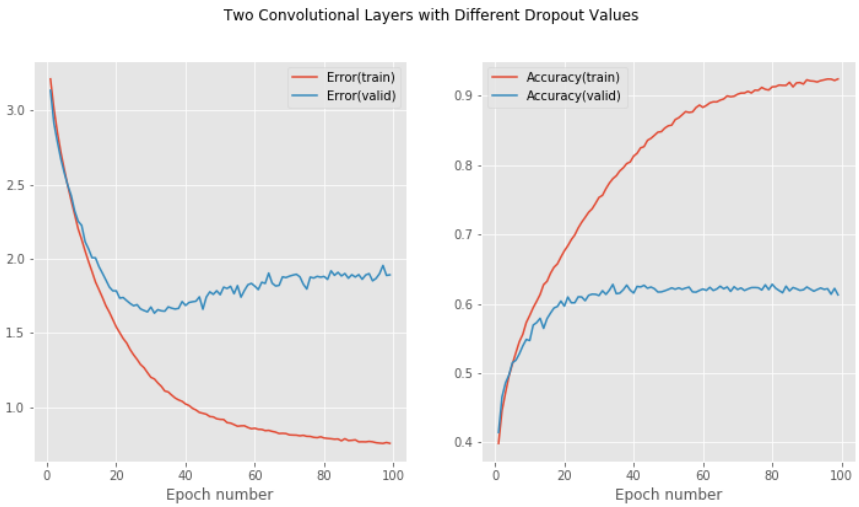

Figure 14: Error and accuracy of model with Dropout implementation of Keep Prob = 0.75 (CONV) and 0.50 (FC).

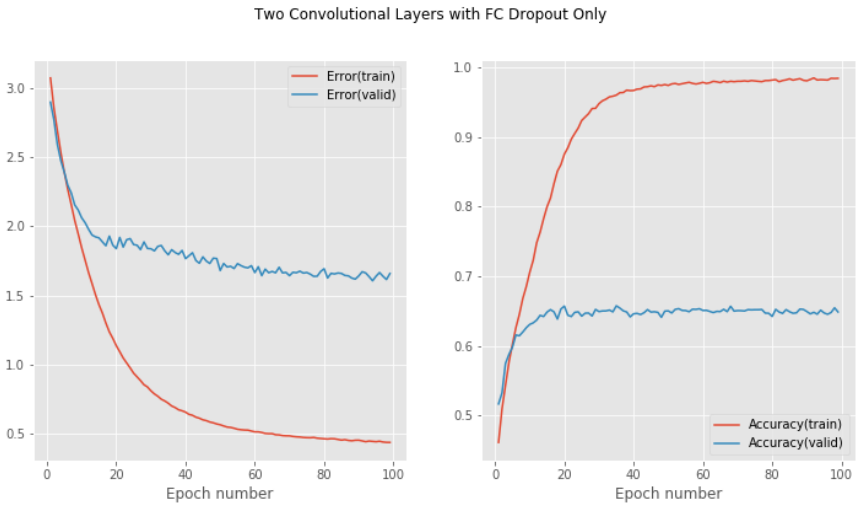

Figure 15: Error and accuracy of model with Dropout implementation of Keep Prob = 0.45 (FC Only).

contrast of the images is to include data augmentation in the test, but not in the traditional sense where augmented data are included in the batch, but that a different network only learns from an augmented set and shares its knowledge with another network.

Example of the input images are shown in Fig. 18. The CONV layers are both set to the depth of 64. The same regularisation method in the previous tests are implemented in this test. Training epoch is set to 100. Different Dropout keep probability are tested.

Reflecting on the result of the previous test, Dropout keep probability of 0.45 on all layers produces the best result for both cases. While the standard RGB + grayscale max fusion network performs rather well, by producing the best accuracy and least overfitting, as expected. The RGB

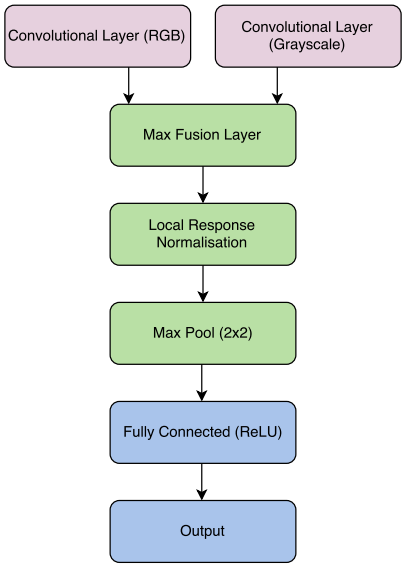

Figure 16: Max fusion architecture.

Figure 17: Left: RGB input. Middle: Grayscale input. Right: Grayscale input with low contrast.

Max Fusion (Keep Prob)	Max Accuracy(Train)	Max Accuracy(Valid)
RGB + Grayscale (0.45)	0.911	0.694
RGB + Grayscale (0.70)	0.978	0.690
RGB + (Grayscale + Low Contrast) (0.45)	0.900	0.689
RGB + (Grayscale + Low Contrast) (0.70)	0.980	0.691

Table 6: Accuracy after 100 epochs training with different implementation of Dropout.

+ grayscale max fusion network with augmented contrast does not perform as well expected, the most likely reason for this is that feature maps of output volumes from each of the layers do not conform to one another.

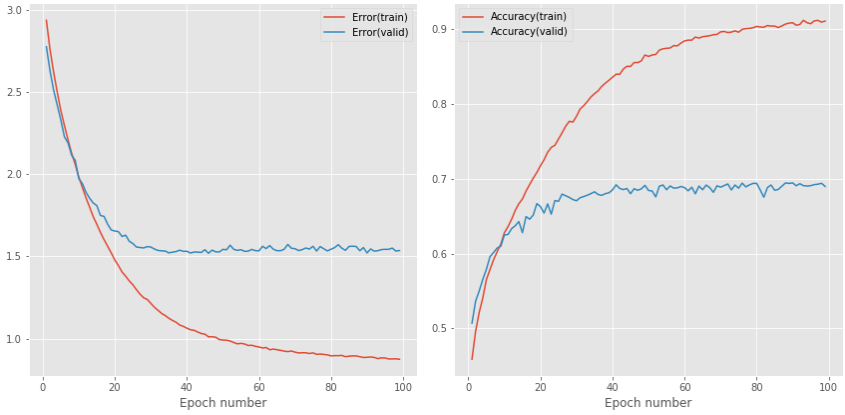

Figure 18: Max fusion grayscale with keep prob = 0.45.

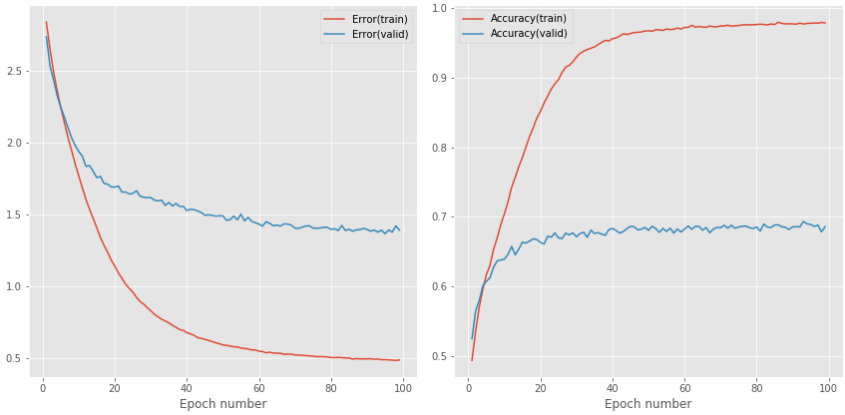

Figure 19: Max fusion grayscale with keep prob = 0.70.

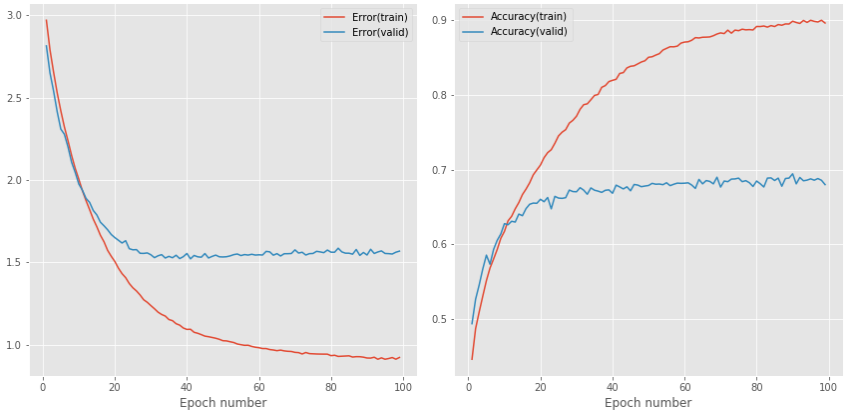

Figure 20: Max fusion grayscale and low contrast with keep prob = 0.45.

Shallow and Wide CNN: Max Fusion Network with a Convolutional Layer

Carrying on with the result from previous test, in this test both the notions of shallow CNN and wide learning are combined to see whether max-fusion layer combined with a CONV layer can

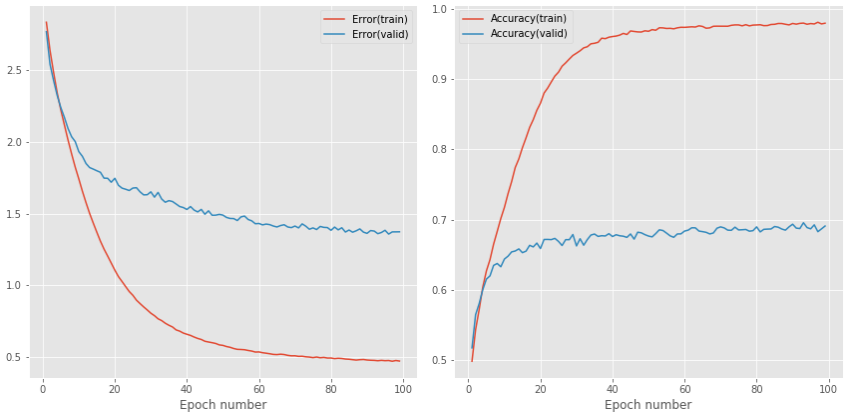

Figure 21: Max fusion grayscale and low contrast with keep prob = 0.70.

outperform the two layer CNN model proposed earlier. For ease, this model shall be called the **MaxCNN** model. The architecture of the MaxCNN is quite simple, as shown in Fig. 22

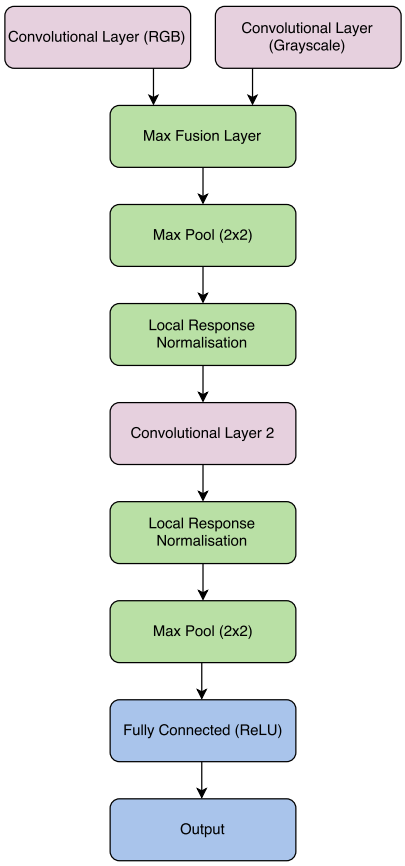

Figure 22: MaxCNN architecture.

At first, MaxCNN was trained for 100 epoch, but overfitting was found to be quite minimal, thus to see investigate how much overfitting will occur with longer epoch, MaxCNN is also trained for 200 epochs. MaxCNN utilises the same regularisation method established in the earlier test.

Model (Epochs)	Max Accuracy(Train)	Max Accuracy(Valid)
MaxCNN (100)	0.770	0.696
MaxCNN (200)	0.806	0.704

Table 7: MaxCNN accuracy on training and validation sets.

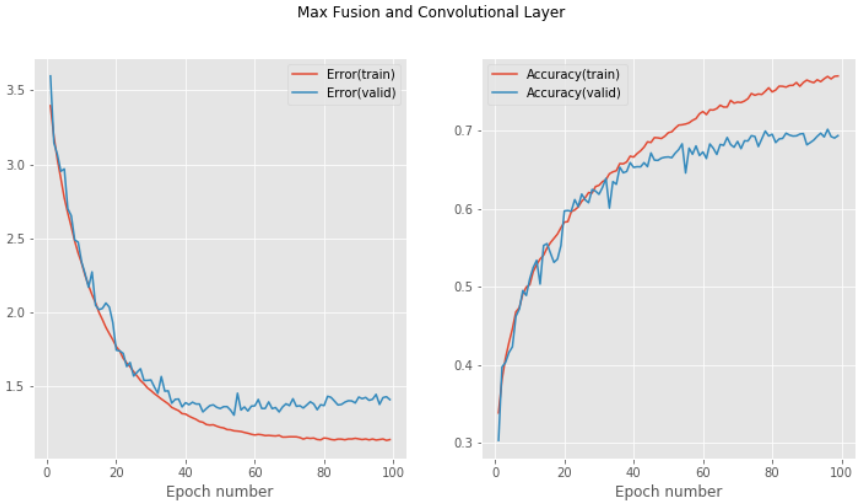

Figure 23: MaxCNN after 100 epochs.

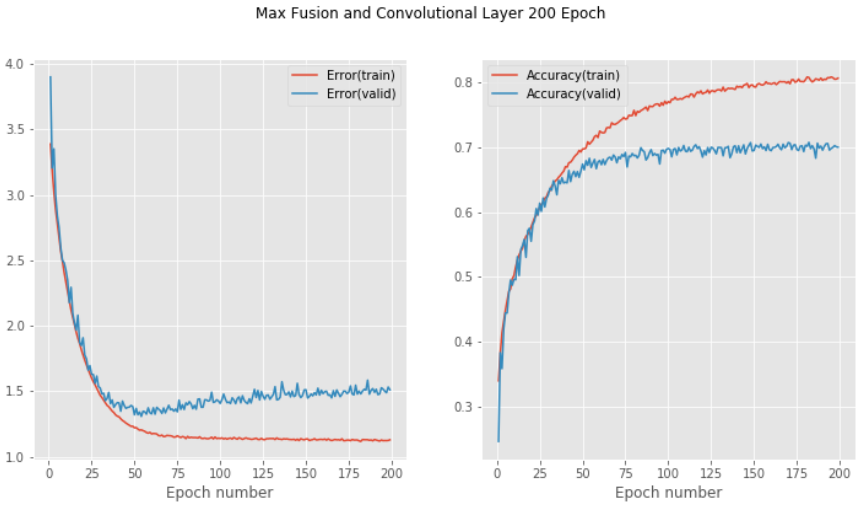

Figure 24: MaxCNN after 200 epochs.

As shown in Table 7, after 200 epochs, MaxCNN achieves a maximum accuracy of 70.4% on validation. Referring to Fig. 24, even after 200 epochs, MaxCNN still overfits significantly far lesser than some of the models from previous tests, which are only trained for 20-100 epochs, this shows that beside boosting performance, max fusion layer also functions as a regulariser to the model by keeping the model in steady generalisation.

Conculsion

With MaxCNN, the main objective of implementing a CNN model that can outperform the baseline model of the previous experiment has been achieved. As shown in Table 8, MaxCNN have managed to both outperform baseline accuracy while also minimising overfitting. Some of the regularisation

Model	Max Accuracy(Train)	Max Accuracy(Valid)
Baseline	0.947	0.552
MaxCNN	0.806	0.704

Table 8: MaxCNN in comparison to best performing baseline model.

methods tested in the baseline experiments that have been carried on to this experiment still proves to perform as well as their baseline performances. Yet, there are interesting alternative implementations of these methods that could have been tested as well, such as adaptive Dropout [12]. While data augmentation is generally a good approach to improving the performance of a neural network, it was decided that the experiment would look more into knowledge sharing instead, through layers fusing. Since faster convergence was expected in this experiment, perhaps some of the learning rate or learning method applied would be far too drastic for models that are trained for longer epochs.

On the other hand, while the implementation of the basic CNN in this experiment performs rather well, investigating into different pooling functions could have been an interesting subject to include, yet this was decided not to be done, as max pooling is widely used in CNN models and other approaches to pooling seem to underperform in comparison to max pooling [6]. The novel approach of *max fusion* in MaxCNN have also proven to be beneficial in boosting the performance of the CNN tested in this experiment. Yet it is still an overly simplified implementation of the original technique proposed by Feichtenhofer et al. thus is due to restrain in computational cost of using AWS, as creating two multiple layers CNN and then fusing their output to be processed through another CNN would mean running three different CNNs, this will prove to be time consuming and cost inefficient.

Yet, the experiment itself is successful in producing a good performing, short-trained, shallow CNN that is aided by wide learning. The experiment also shows that sharing knowledge between layers at the same layer level helps the model from overfitting and allows the model to learn better. In the future it would be interesting to study a larger scale implementation of MaxCNN by applying more layers in the max fusion layer phase and also after, while also implementing more novel regularisation approaches.

References

[1] Graham, B. *Fractional Max-Pooling*, e-print: arXiv:1412.6071v4, (2015)

[2] Karpathy, A. *Lessons learned from manually classifying CIFAR-10*, `http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/`, (2011)

[3] Feichtenhofer, C., Pinz, A., Zisserman, A. *Convolutional Two-Stream Network Fusion for Video Action Recognition*, e-print: arXiv:1604.06573, (2014)

[4] *Typical CNN Example* `https://commons.wikimedia.org/wiki/File%3ATypical_cnn.png`, (2015)

[5] *Max Pooling* `https://commons.wikimedia.org/wiki/File%3AMax_pooling.png`, (2015)

[6] *CS231n: Convolutional Neural Networks for Visual Recognition*, Stanford University, `http://cs231n.github.io/convolutional-networks/`, (2016)

[7] Krizhevsky, A., Sutskever, Hinton, G., *ImageNet Classification with Deep Convolutional Neural Networks*, Advances in Neural Information Processing Systems 25 (NIPS 2012), (2012)

[8] Xu, B., Wang, N., Chen, T., Li, M. *Empirical Evaluation of Rectified Activations in Convolution Network*, e-print:arXiv:1505.00853, (2015)

[9] He, K., Zhang, X., Ren, S., Sun, J. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, e-print:arXiv:1502.01852v1, (2015)

[10] Maas, A., Hannun, A., Ng, A., *Rectifier Nonlinearities Improve Neural Network Acoustic Models*, e-print: `http://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf`, (2013)

[11] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research 15, e-print: `http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf`, (2014)

[12] Ba, L., Frey, B., *Adaptive dropout for training deep neural networks*, Advances in Neural Information Processing Systems 26 (NIPS 2013), (2013)