Various CNN adaptations for CIFAR10

Travis Hammond (s2880024) Ivo de Jong (s3174034)

University of Groningen, Faculty of Science and Engineering

1 Introduction

A task which is commonly approached with Deep Learning due to the repeated successes is image classification. Since the features in the data may be rather complex to extract into a meaningful classification Deep Learning tends to be the go-to solution. Given this tight connection between image classification and Deep Learning, several image classification benchmarks have been developed.

1.1 Dataset, CIFAR10

One of these well established image classification benchmarks is the CIFAR10 dataset[7]. The CIFAR10 dataset contains 60000 32x32 colour images of 10 classes consisting of different animals and different vehicles. A sample from these 10 classes can found in figure 2. Previous benchmarking found an 82% test accuracy using a Convolutional Neural Network without data augmentation.

2 Method

This paper will be exploring how the use of different settings for Convolutional Neural Networks and see how they affect performance on the CIFAR10 dataset. For this we'll take a baseline architecture and model as suggested by the Keras documentation [2]. This CNN consists of two sets of two convolutions each, where each set ends with a max-pooling layer. The first set has 32 convolutions of 3x3 filters, the second has 64 of 3x3 filters. The networks ends with two fully connected layers of 512 and 10 neurons. For the baseline all connections have ReLU [1] as an activation function, except the output layer which has SoftMax [4] as an activation function. Categorical crossentropy will be used as a loss, and Adam [6] will be used as an optimizer. The dropout rate for the baseline will be set to 25% in the pooling layers and 50% between the fully connected layers.

To compare different settings we explore different activation functions to be applied in place of ReLU. We'll compare our baseline model to ones with activation functions: linear, sigmoid, SoftPlus and ELU [3].

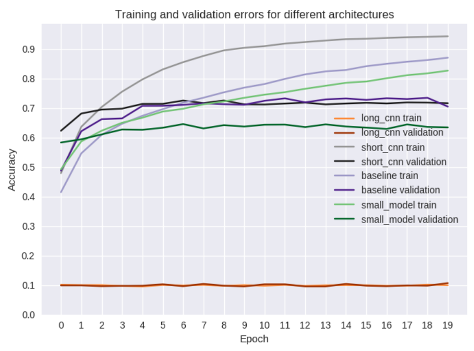
For different dropout rates we'll compare the baseline described above to a no-dropout one where there is 0% dropout and a high-dropout one where there will be a 50% dropout in the max-pooling layers and an 85% dropout between the hidden layers.

For optimizers we'll be comparing our baseline with Adam as an optimizer to ones with Stochastic Gradient Descent [8], RMSprop [5], Adadelta [9].

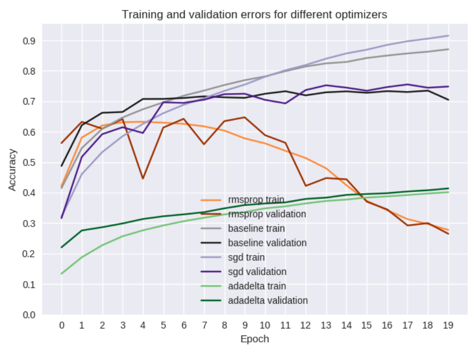
We'll also be comparing the baseline to different model architectures. The baseline will be compared to a (very) small architecture consisting of a single convolutional layer with 96 2x2 kernels, a 2x2 max-pooling layer and a fully connected output layer. We'll also compare two variations of the baseline where we have max-pooling only after all 4 convolutions, and one where we have max-pooling after each convolution. All architectures are visually demonstrated in figure 3 in the appendix.

	Best Train	Best Validation	Average Train	Average Validation
Baseline	0.87	0.73	0.74	0.70
Small architecture	0.83	0.65	0.72	0.63
Short architecture	0.94*	0.73	0.85*	**0.71***
Long architecture	0.10	0.11	0.10	0.10
SGD	0.92*	**0.76**	0.73	0.68
RMSprop	0.63	0.65	0.50	0.50
Adadelta	0.40	0.41	0.32	0.35
Linear Activation	0.64	0.57	0.61	0.49
SoftPlus Activation	0.10	0.10	0.10	0.10
Sigmoid Activation	0.10	0.11	0.10	0.10
ELU Activation	0.85	0.68	0.75*	0.61
No dropout	**0.99***	0.72	**0.92***	0.69
High dropout	0.61	0.68	0.54	0.62

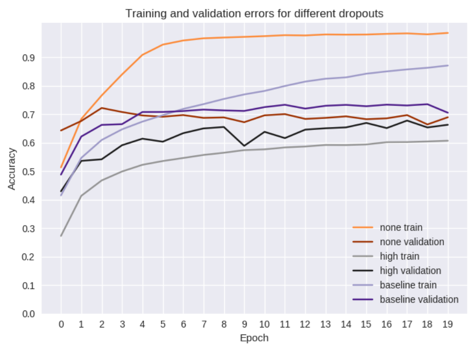
Table 1: Accuracy's of different CNN settings. Bold values denote the highest accuracy in a measure. A * indicates an accuracy higher than the baseline in that measure.



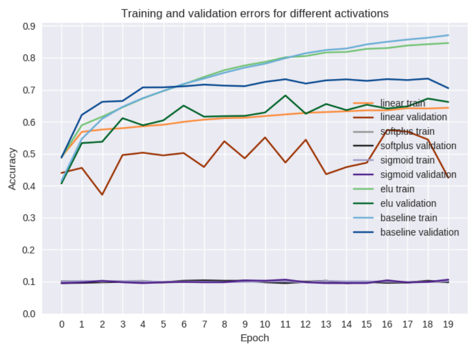
(a) Architectures



(b) Optimizers



(c) Dropouts



(d) Activations

Figure 1: Training and validation accuracy's for different parameters across 20 epochs. Enlarged versions are available in figure 4 in the appendix.

3 Results

3.1 Architectures

We can see in Figure 1a, as well as in table 1 that the long CNN architecture has an accuracy of 10%, which is equivalent to random guessing. This may be because all the max-pooling layers also have a dropout rate of 25%. Having too many of such layers may make the chance for a feature to pass through too small to train the model.

The small model with only a single convolutional layer has a low comparatively low training accuracy because it is not able to extract more complex features. On the bright side, this also decreases the number of parameters, making it less prone to overfitting.

The short architecture shows a very high training accuracy, which indicates overfitting. The lower number of Max Pooling with Dropout layers decreases regularization, allowing for more overfitting. In this case the validation accuracy was not negatively affected by this.

3.2 Optimizers

Here we see the Stochastic Gradient Descent perform better than the Adam baseline in terms of peak validation accuracy. Note that we cannot conclude here that SGD model would have a 76% accuracy on new data, because it is to some extent ”trained” on the validation data. It may be the case that the default parameters for Adam are not optimal in this case, allowing SGD to outperform it.

RMSprop shows a decrease in testing accuracy after several epochs as seen in 1b. This may be because the decay rate ρ is set to a value that allows the accumulated squared gradient to become too big.

Adadelta performed consistently worse than both Adam and SGD. Adadelta is considered a robust adaptation on AdaGrad, but that may not make it the best specialist.

3.3 Dropout

The comparisons of different dropout levels are fairly straightforward. No dropout results in a high training accuracy and a low validation accuracy, a typical sign of overfitting due to a lack of regularization and a resulting model that is too complex. High dropout leads to worse performance overall because of a too weak model underfitting on the data, no longer fully capable of generalizing as much as the baseline which performs the best.

In this case the high dropout is very high, and the no dropout is very low. This makes it inevitable that the baseline with a more middle-of-the-road dropout rate will perform better. There will likely be some different dropout rates that may further improve performance.

3.4 Activation functions

Across the different activation functions, the ReLU baseline performs best.

The poor performance on sigmoid was to be expected. Since the initial weights were not set specifically within reasonable bounds for the sigmoid function it is very possible that certain large or small values are at the ”plateau” parts of the sigmoid curve, rendering these connections ineffective.

The sigmoid and softplus activation functions performs no better than chance and do not manage to improve. This may be because the initial weight values are set outside of the effective curve of the sigmoidal functions. This is likely due to over saturation and exploding or vanishing gradients.

Somewhat surprisingly, the softplus activation function is also not able gain any traction. This is somewhat surprising as it is essentially similar to ReLU, but with a smoother curve around 0. It would be expected that performance would be similar to ReLU, though with more computational costs.

The linear model performs surprisingly well, despite the fact that consecutive applications of linear functions express a linear function as well. This should end up expressing as a linear regression adaptation, which seems to give a surprsingly sufficient performance.

The Exponential Linear Unit and Rectified Linear Unit perform the best, with ReLU performing even better on the validation data. It makes sense that they perform similarly, as their activation functions are similar for values larger than 0. This difference in performance may be because ReLU is able to completely avoid the vanishing gradient problem, while ELU is not.

4 Conclusion

We can conclude that the baseline provided by Keras is very appropriate. There may be some room for improvement by using SGD as an optimizer, or by slightly tweaking the dropout rate. Convergence could be quicker with a smaller of pooling layers, but there is insufficient evidence for this.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018. cite arxiv:1803.08375Comment: 7 pages, 11 figures, 9 tables.

[2] François Chollet et al. Keras. https://keras.io, 2015.

[3] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[4] Kaibo Duan, S. Sathiya Keerthi, Wei Chu, Shirish Krishnaj Shevade, and Aun Neow Poo. Multi-category classification by soft-max combination of binary classifiers. In *Proceedings of the 4th International Conference on Multiple Classifier Systems*, MCS'03, page 125–134, Berlin, Heidelberg, 2003. Springer-Verlag.

[5] Geoffrey Hinton, Butusg Srivastava, and Kevin Swersky.

[6] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[7] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[8] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[9] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

A Data

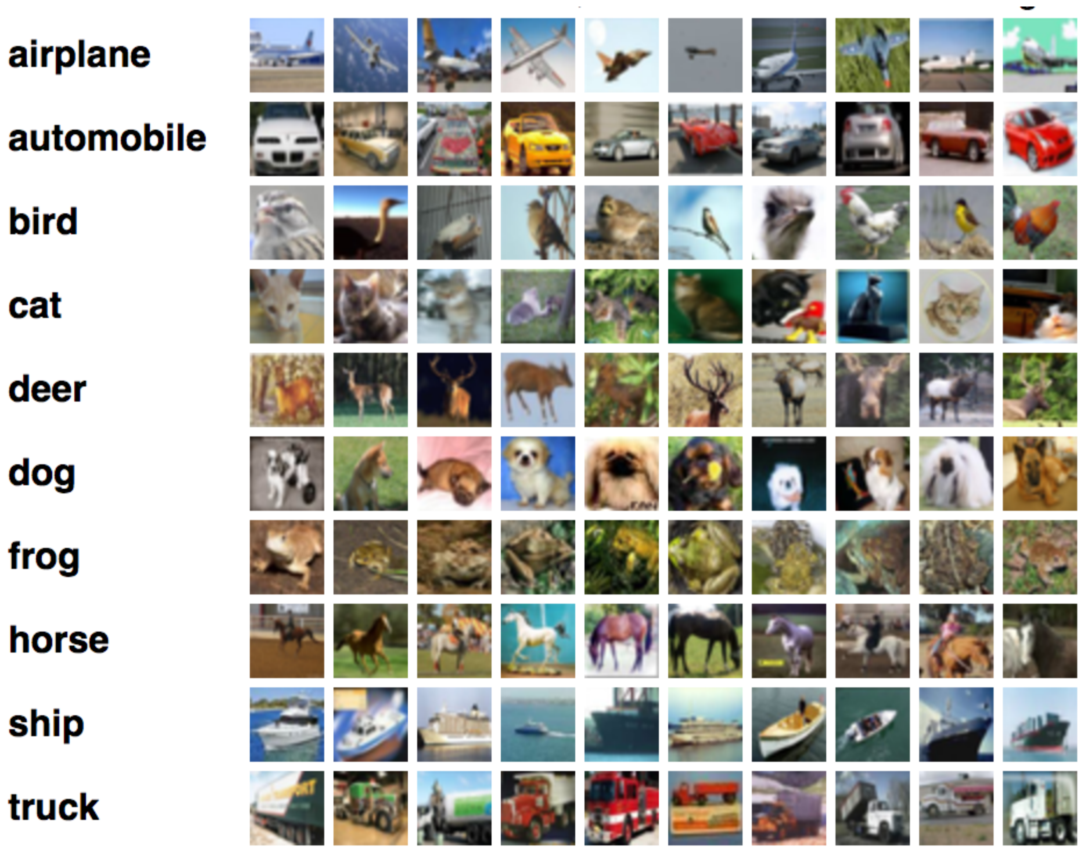
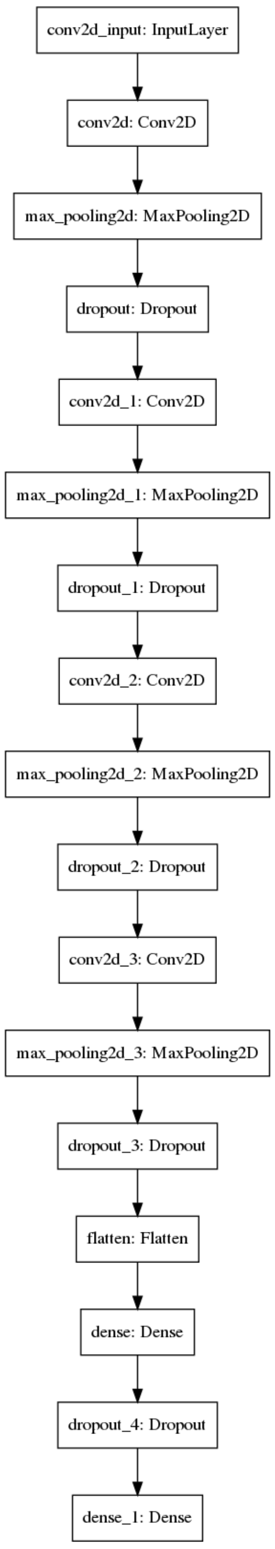
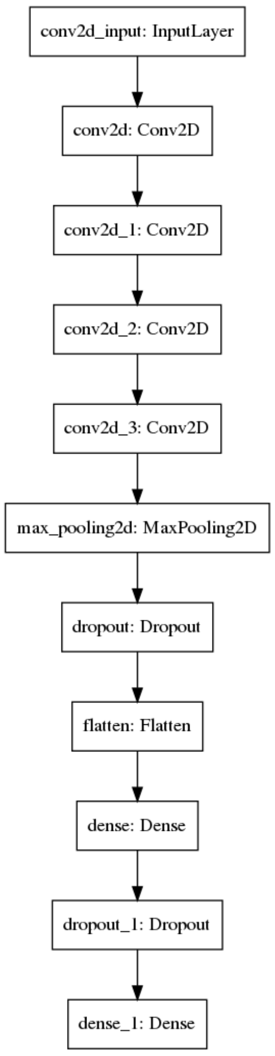


Figure 2: The 10 different classes in the CIFAR10 dataset, with some corresponding samples

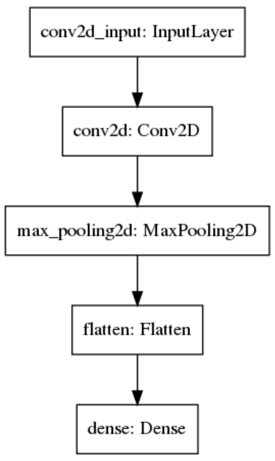
B Architectures



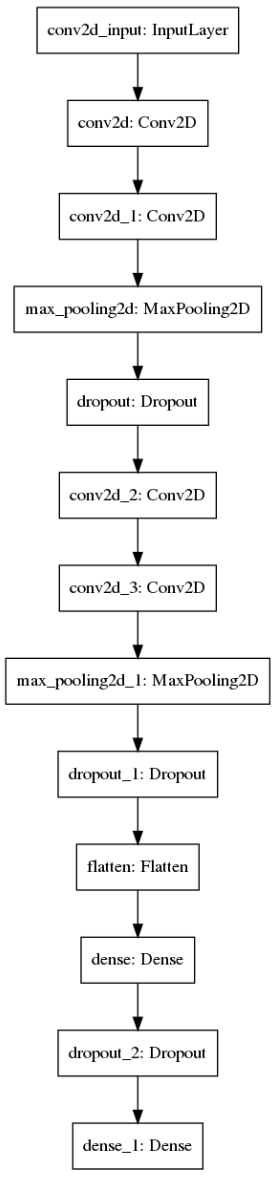
(a) long_cnn



(b) short_cnn



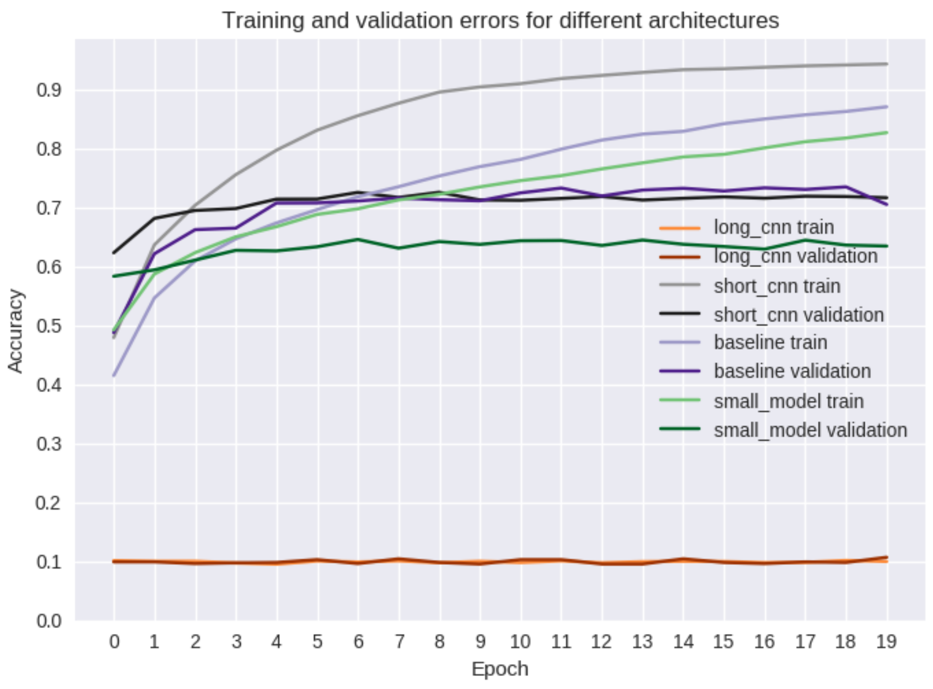
(c) small_model



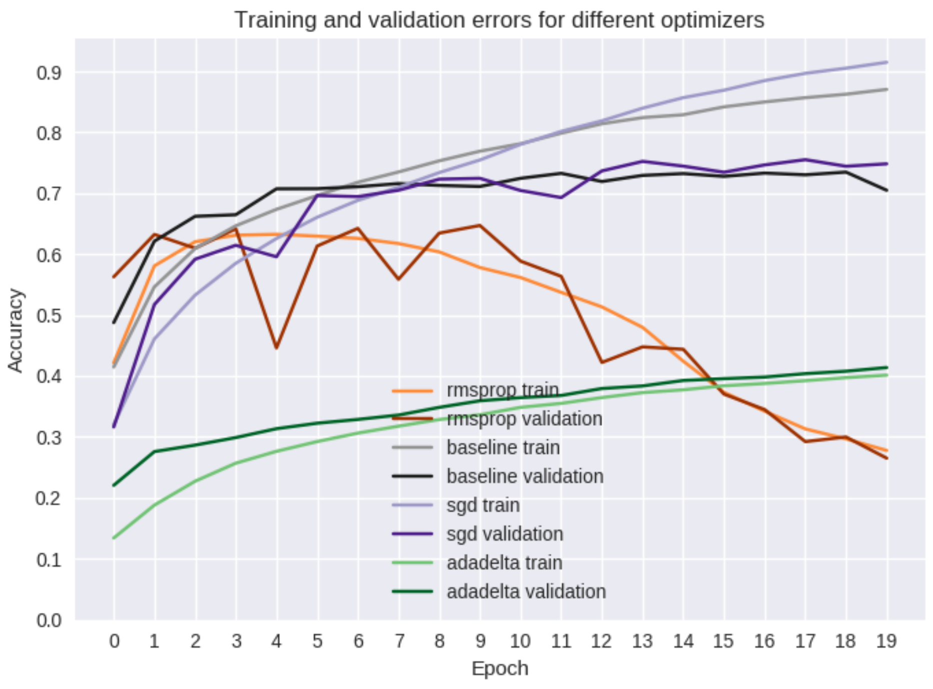
(d) baseline

Figure 3: Different CNN Architectures

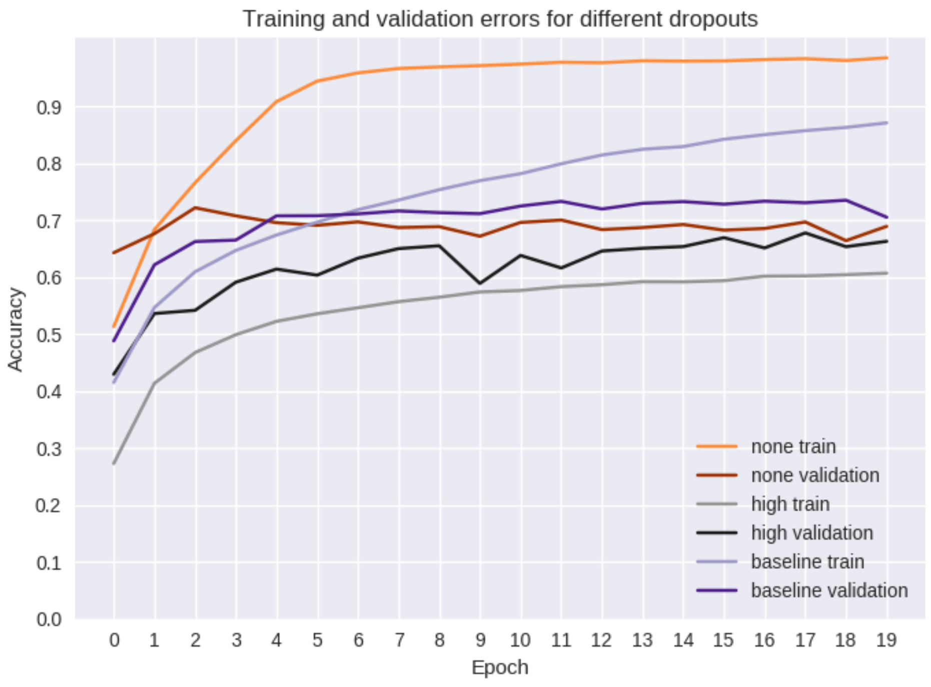
C Enlarged graphs



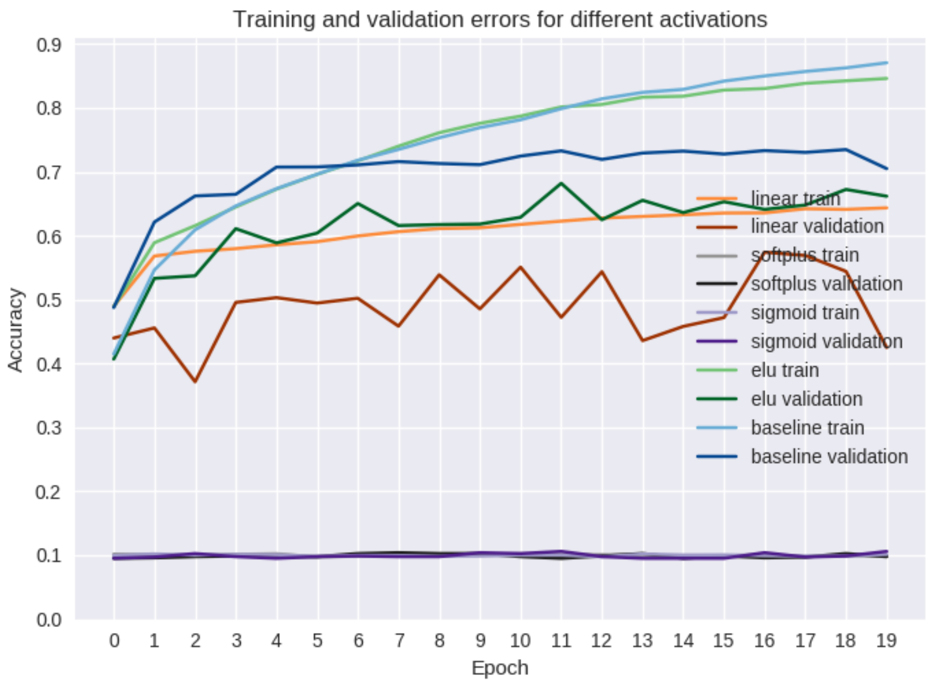
(a) Architectures



(b) Optimizers



(c) Dropout rates



(d) Activation functions

Figure 4: Training and validation accuracy's for different parameters across 20 epochs.