The Effect of Structure & Regularization on Model Success in Computer Vision

Abstract: The area of computer vision is one that is high risk with a potential for high reward. Decisions made on incorrect predictions by a computer present risk while the prospect of self-driving cars and instantaneous facial recognition promise reward. In this paper I conducted 12 experiments surveying the relative performance of different types of deep and convolutional neural networks. Using the CIFAR-10 dataset, the TensorFlow Keras library, and a multilayer convolutional neural network with regularization applied, I was able to reach an accuracy value of 81.39% and a loss of 0.7253.

1. Introduction

Accurate image recognition, or high-powered computer vision, has a host of applications in a variety of industries. In healthcare, where it can aid in catching tumors on x-rays, to self-driving cars, where the recognition of stop signs, traffic lights, and pedestrians is paramount. One of the most well-known and controversial may be facial recognition, which has been much sought after by law enforcement. The problem that plagues all forms of computer vision applications but is most pronounced in facial recognition and healthcare, is the importance of extreme accuracy and low to nil error rate. An incorrectly identified person in a law enforcement case could lead to an innocent person going to jail, such as in the case of Nijeer Parks, who spent 10 days in jail for a crime he did not commit (Hill, 2020). In healthcare, to base a medical decision on computer recognition alone could have dire consequences for a patient if incorrect. This is why the overarching view in the medical field is that: "The best way to think about the technology's future in medicine [...] is not as a replacement for physicians, but rather as a force-multiplier and a technological backstop that not only eases the burden on personnel at all levels, but makes them better" (Powell, 2020).

Machine learning algorithms often outperform human experts in computer vision tasks--such as in 2019 when researchers at Google revealed that AI they developed to detect lung cancer was 94% accurate, beating out radiologists and delivering fewer false positives and negatives (Powell, 2020). However, humans have less tolerance for computer error than for human error. Therefore, when building models that have direct impact on people's lives, we must strive to achieve the highest degree of accuracy and lowest degree of error.

In this research, I will be conducting experiments using CIFAR-10 data to examine the construction of convolutional neural networks (CNNs) and deep neural networks (DNNs) to see what combination of features makes a robust model with high accuracy, low error rate, and low performance time (since time is also a consideration in real world application). This will serve as an initial building block to research examining how neural networks will interact with larger and more complex forms of data and how accuracy and error rate can further be improved on those types of datasets.

2. Literature Review

2.1. Dense vs. Convolutional Neural Networks

In breaking down the differences between DNNs and CNNs, author Francois Chollet notes: "The fundamental difference between a densely connected layer and a convolution layer is this: Dense layers learn global patterns in their input feature space [...], whereas convolution layers learn local patterns" (Chollet, 2021). This leads to two compelling differences between DNNs and CNNs. The first being that once a convolutional layer learns a pattern it can recognize it elsewhere as well (also known as translation invariance). The basic idea being that once it recognizes a feature on the bottom-right of a frame, it would also be able to pick it out in the upper-left of another frame. DNNs on the other hand need to learn the pattern over and over again every time it appears in a new location. That makes CNNs more computationally efficient.

The second difference is that convolutional layers can “learn spatial hierarchies of patterns”. This means that where an initial convolutional layer will typically learn small patterns like the edges of an image, subsequent convolutional layers will build off of that learning bigger patterns made up of the ones on previous layers. This allows convolutional layers to learn more complex representations of the data (Chollet, 2021).

The strength of CNNs’ ability to identify local patterns is also mentioned in “Gradient-Based Learning Applied to Document Recognition,” as the authors point out: “[...] a deficiency of fully-connected architectures is that the topology of the input is entirely ignored. The input variables can be presented in any (fixed) order without affecting the outcome of the training. On the contrary, images [...] have a strong 2D local structure: variables (or pixels) that are spatially or temporally nearby are highly correlated. Local correlations are the reasons for the well-known advantages of extracting and combining local features before recognizing spatial or temporal objects, because configurations of neighboring variables can be classified into a small number of categories (e.g. edges, corners…)” (LeCun et al., 1998).

This paper also describes the computational cost of a typical fully-connected feed-forward network in comparison to a convolutional network. It points out--as mentioned above--that the lack of translational invariance means that even for handwriting samples that have been preprocessed and normalized, there are still always variances in style, slant, size, etc. that must be individually learned one by one, requiring a lot of processing power which was an even stronger consideration at the time of writing than it is today (LeCun et al., 1998).

2.2. Structural Elements of Convolutional Neural Networks

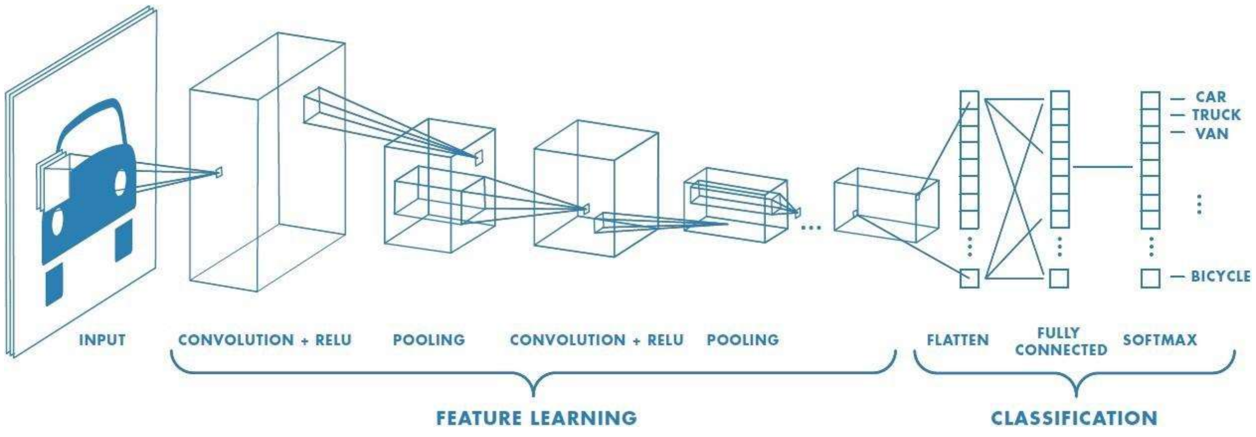
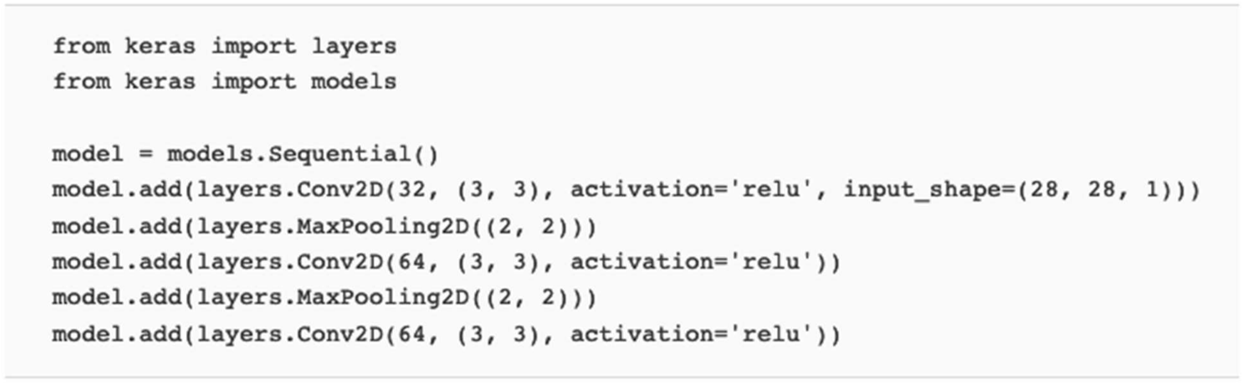


Fig 1: (Saha, 2018)

The typical structure of a CNN includes convolutional layers with activations followed by pooling layers before the model gets flattened and passed to one or more fully connected layers before being outputted.



```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Fig 2: (Chollet, 2021)

In the above example, Conv2D layers and MaxPooling layers are alternated. The Conv2D layers have the kernel size (3,3) and the MaxPooling layers have a filter size of 2x2. The activation 'relu' is used in each of the Conv2D layers. The precedent for relu being used goes back to the reasoning given in "ImageNet Classification with Deep Convolutional Neural Networks": "Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units" (Krizhevsky et al., 2017). The filter size used for MaxPooling layers is almost always (2,2) which effectively reduces the feature map by ½ on each dimension (and therefore ¼ total). The Conv2D layers also have 32, 64, and 64 filters each, in the above example. The number of filters represents the depth of the output feature map (Chollet, 2021).

The top example represents a very basic version of a CNN but there are many constructions of varying complexity. A particularly deep CNN, for example, is the Microsoft ResNet from 2015--a 152 layer network with an error rate of 3.6% on the ImageNet data. On the other hand, one of the things about CNNs that has spurred their popularity is how simple they can be while still achieving strong performance metrics, as in the case of VGG Net, which possesses only 19 layers and achieved a 7.3% error rate on ImageNet data (Deshpande, 2016).

2.3. Regularization Methods

I will be examining and contrasting three types of regularization in this paper: L1 regularization, L2 regularization, and batch normalization.

L1 and L2 regularization are variations on the same theme. They are weight regularization techniques that act by adding a weight penalty to the loss function. Simply put, L1 penalizes the absolute value of the weights and L2 penalizes the squared value of the weights (Jaiswal et al., 2018). The way this affects the fit of the model is that with L1, parameters deemed unimportant are pushed towards zero and in L2 they are pushed towards small values. This means that features that are not important will have either zero or very little effect on the output. By de-prioritizing some features, these forms of regularization reduce the complexity of the model, aiding in the fight against overfitting (Pykes, 2021).

Batch normalization is a mechanism that was first introduced in 2015 as outlined in the paper “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift” (Ioffe et al., 2015). As the title implies, the problem that batch normalization was invented to fix is something called internal covariate shift. What the paper describes as internal covariate shift is the phenomenon by which “the distribution of each layer’s inputs changes during training, as the parameters of the previous layers change" (Ioffe et al., 2015). This can potentially cause problems within a model because the changes in the distributions means that the surrounding layers constantly need to adapt, causing a lower learning rate. Batch normalization helps fix this issue by: "Standardizing the activations of the prior layer [which] means that assumptions the subsequent layer makes about the spread and distribution of inputs during the weight update will not change, at least not dramatically. This has the effect of stabilizing and speeding-up the training process of deep neural networks"(Brownlee, 2019). The paper also notes that Dropout may not be required for regularization where Batch Normalization is being used. Using Batch Normalization, the authors were able to achieve 4.9% top-5 error rate on ImageNet classification (Ioffe et al., 2015). For reference, AlexNet had a top-5 error rate of 15.3% (Krizhevsky et al., 2017).

3. Methods

I will be constructing deep and convolutional neural networks for this research and making changes to the hyperparameters in order to observe the effect of those changes on the performance of the models. Specifically, the experiments will involve examining the differences between DNN and CNN models, and the effect of regularization methods and other alterations to the complexity of the models.

As in previous experimentation, I am utilizing Google Colaboratory with Python 3, Tensorflow and Keras. I will also utilize NumPy, Matplotlib, and Pandas. Tensorflow and Keras are used for model construction and some data preprocessing while the others will be used for calculations, plotting, and other functions.

I will visualize my results using tables comparing the performance of my experiments, broken up by section (The complete table can be found in the appendix), plots comparing the accuracy and loss performance over the course of the epochs for each experiment, and confusion matrices for several of the experiments. These visuals will aid in illustrating how the models are performing in comparison to one another and give insight into what they are doing.

For this paper I will be using the CIFAR-10 dataset. This is a dataset of 60,000 32x32 color images that fall into 10 classes. The classes include: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset has 5 training batches and 1 test batch. Each training batch is made up of 5,000 images from each class while the test batch has 1,000 images randomly selected from each of the classes (Krizhevsky et al., 2009). Since I am interested in beginning with a smaller dataset and then subsequently moving to large and/or more complex ones, CIFAR-10 makes perfect sense as there is also a CIFAR-100 dataset that has 100 categories and I could see that being the next logical step after completion of this experimentation.

4. Results

4.1. Experiments 1-4, 12

In experiments 1-4 I first sought to establish a baseline between 4 basic varieties of neural networks--two that are DNN and two that are CNN. For these experiments I kept the epochs at 50 and the batch size at 100. For the optimizer I used Adam and for the activation and loss functions I used relu/softmax and sparse categorical cross-entropy, respectively. Data was split into train, validation, and test sets. 10,000 were reserved for test, 47,000 for train and 3,000 were split off of the training set into a validation set.

Experiment	1	2	3	4	12
Model Type	DNN	DNN	CNN	CNN	CNN
# of Layers	2 Dense	3 Dense	2 Conv 2 Pool 1 Dense	3 Conv 3 Pool 1 Dense	3 Conv 3 Pool 1 Dense

Regularization	N/A	N/A	2 Dropout (0.3)	3 Dropout (0.3)	N/A
Test Accuracy	50.27%	45.63%	72.77%	79.10%	72.23%
Test Loss	1.5008	4.5419	1.5441	0.7784	2.5361
Processing Time (s)	134.16	188.13	382.93	443.08	443.06
Parameters	428,938	563,082	3,841,930	2,269,578	2,269,578

Fig 3: Results of Experiments 1-4, 12

The first clear difference between DNNs and CNNs is the processing time. The CNNs took more than twice as long as DNNs. The second difference is that they achieved accuracy between 22.5-33.5% higher than the DNNs.

The next thing that was interesting in my results were the differences between the loss values. In order to better understand what was going on there I took a look at the performance metrics (Appendix A).

Specifically, between experiment 1 and 2, there was an initially unexpected difference in that experiment 2 performed worse on accuracy and loss. However, when comparing the performance metrics, it looks like some overfitting is happening, given that the training set achieved extremely high accuracy but there is a huge gap between that and the validation accuracy, which is really low in comparison. I think given that I used no regularization on the DNNs that result makes sense because with experiment 2, adding that additional layer which also has a higher number of units (I used 128/256 for experiment 1, and 128/256/512 for experiment 2) is creating a complex model that is fitting well to the training data and not generalizing as well to the validation data or test data. For experiments 3 and 4 I used Dropouts after each of the MaxPool layers because I expected a high level of overfitting given what I had read in the course of my literature review. Just to be sure, I also ran experiment 12 with exactly the same structure as experiment 4 but without Dropouts. As expected, the overfitting was almost immediate and that model scored lower than experiment 4.

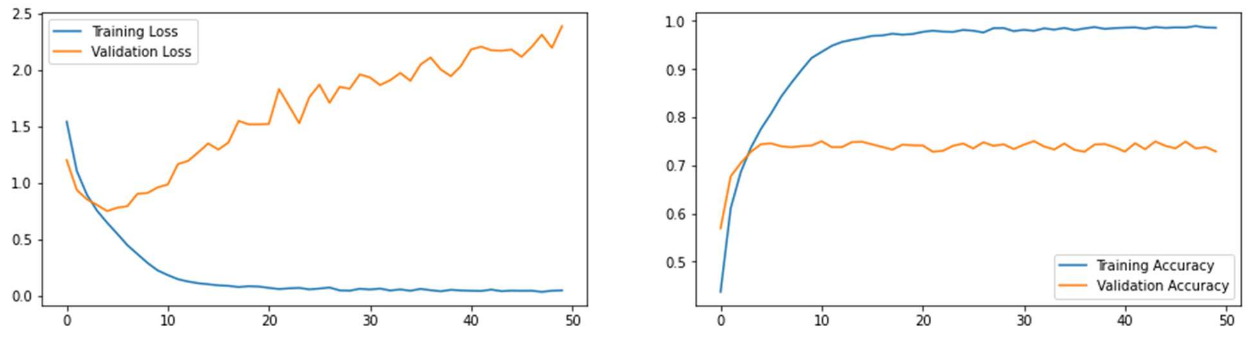


Fig 4: Performance Metrics, Exp. 12

Since Experiment 4 performed the best so far, I decided to use its structure as a jumping off point for my experiments in 5-10. We can see from the results that 3 Conv/Pool layers with Dropouts between works better than all previous experiments. So in Experiments 5 and 6 I sought to examine what additional layers would do--both convolutional as well as fully connected layers.

4.2. Experiments 5-6

For experiment 5 I increased the Conv2D layers to 4 and the MaxPool layers were put at 2. I did it this way in order to roughly mimic the structure I had seen used in VGG Net, the simple yet high performing model from University of Oxford researchers in 2014 (Deshpande, 2016). In that case they used 2 convolutional layers before one MaxPool instead of the structure I had used previously which was 1 MaxPool for every convolutional layer. In experiment 6 I did the same but increased the number of Dense layers to 2 without adding any additional regularization. Similar to experiment 4 I included Dropouts after each MaxPool, therefore only having 2.

Experiment	4	5	6
Model Type	CNN	CNN	CNN
# of Layers	3 Conv 3 Pool 1 Dense	4 Conv 2 Pool 1 Dense	4 Conv 2 Pool 2 Dense
Regularization	3 Dropout (0.3)	2 Dropout (0.3)	2 Dropout (0.3)
Test Accuracy	79.10%	79.37%	79.34%
Test Loss	0.7784	0.7616	0.8522
Processing Time (s)	443.08	324.36	260.24
Parameters	2,269,578	3,498,250	3,595,530

Fig 5: Results of Experiments 4-6

As you can see from the table, both models performed quite well--accuracy was slightly better than experiment 4 but loss in the case of experiment 6 was a little higher.

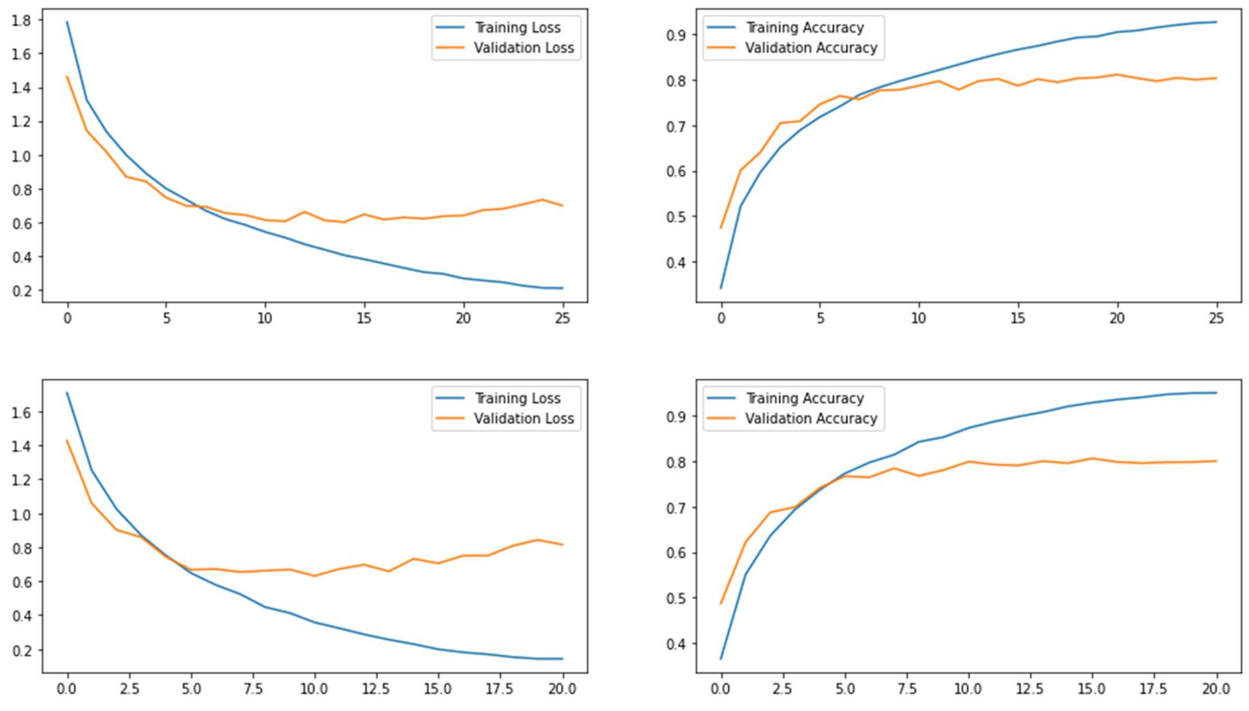


Fig 6: Performance Metrics Exp. 5 (Top) & 6 (Bottom)

In both cases, it can be seen that the models begin overfitting pretty quickly. Noting the relative number of parameters we can see that 5 and 6 both have over 1 million more parameters than experiment 4. Remembering that a model with a higher number of parameters is more prone to overfitting, these results are somewhat expected. Given what an issue overfitting presents, the next several experiments focus on different strategies to combat it.

4.3. Experiments 7-9

In experiments 7-9, as previously mentioned, the goal was to test out regularization methods against the best performing model that resulted from experiments 1-4. In this case, the 3-Layer CNN. The three methods I tested out were L1 regularization, L2 regularization, and batch normalization.

Typically between L1 and L2 regularization the one that I have seen used is L2, not L1, which mostly just gets a passing mention. I decided to test both out of curiosity at the difference in their performance. What I initially found when testing L1 on my convolutional layers was that the model performed abysmally. It was unable to get above 10% accuracy on either the training or test data. Confused, I looked into it and found that the default value for L1/L2 regularizers in keras is 0.01. However as Jason Brownlee points out: “Weight regularization does not seem widely used in CNN models [...] L2 weight regularization with very small regularization hyperparameters such as (e.g. 0.0005 or 5 x 10^−4) may be a good starting point”(Brownlee, 2020). Indeed in “ImageNet Classification with Deep Convolutional Neural Networks” the authors mention using weight decay of 0.0005 and noting its usefulness as a regularizer and as a way to reduce the model’s error rate (Krizhevsky et al., 2017).

With this information, I made a change to the default value on both experiments 7 and 8 to 0.0005 and found that the accuracy was vastly improved. The error rates were still very high compared to experiment 9 so I'm interested in conducting further experimentation just with L1 and L2 to see what changes could significantly improve those statistics, if any.

Experiment	7	8	9
Model Type	CNN	CNN	CNN
# of Layers	3 Conv 3 Pool 1 Dense	3 Conv 3 Pool 1 Dense	3 Conv 3 Pool 1 Dense
Regularization	L1 (0.0005)	L2 (0.0005)	3 Dropout (0.3) Batch Norm on Dense
Test Accuracy	71.04%	73.25%	80.60%
Test Loss	1.2761	1.3709	0.6543
Processing Time (s)	378.72	178.51	443.12
Parameters	2,269,578	2,269,578	2,271,114

Fig 7: Results of Experiments 7-9

For experiment 9 I also used the structure of 3 Conv2D, 3 MaxPool, and 3 Dropouts with 1 Dense layer. The only addition here was the BatchNorm between the Dense layer and the output. Batch normalization performed extremely well, finally cracking into the 80% range on accuracy while keeping an acceptably low loss metric. In terms of processing time, it took around the same amount of time that many of the other models had taken, roughly 7 minutes.

It should be noted that for experiments 5-11, I implemented EarlyStopping on several models, out of fear of overutilizing my GPU allocation. This is why the processing time for experiment 8, for example, is so low because it stopped improving very soon into its run.

4.4. Experiment 10-11

In the original paper proposing the Batch Normalization concept, the author noted that Dropouts may not be necessary where Batch Normalization is being used (Ioffe et al., 2015).

With that in mind I decided to try replacing the Dropouts in my model for experiment 10 with BatchNorm instead and see how it performs. I returned to the structure of experiment 6, and placed the BatchNorm layers between the Conv2D layers as well as the 2 Dense layers, thus giving my model a total of 3 BatchNorm layers.

Experiment	10	11
Model Type	CNN	CNN
# of Layers	4 Conv 2 Pool 2 Dense	5 Conv 3 Pool 2 Dense
Regularization	Mult. BatchNorm	Dropout +BatchNorm
Test Accuracy	72.37%	81.39%
Test Loss	1.9923	0.7253
Processing Time (s)	560.02	620.96
Parameters	2,517,770	2,517,258

Fig 8: Results of Experiments 10-11

Given how much the original paper hyped the effectiveness of Batch Normalization I was a little disappointed to see that it really is not a full replacement for Dropout layers. Looking at the performance metrics, the training and validation diverge and stay pretty far apart from almost the beginning of the run. Upon further research on my part, I was able to find an article from more recently that confirmed that: "[Batch Normalization] can act as a regularizer, which can help overcome overfitting and help learn better. However, the noise added is quite small. Thus, it generally is not enough to properly regularize on its own and is normally used along with Dropout" (Riva, 2021). With this information, I decided to take what I had previously noticed, which is that Dropouts work well on convolutional layers and Batch Normalization works well on fully connected layers and combine the two for experiment 11. For good measure, I also opted to try deepening the model a little past previous experiments, giving it 5 convolutional layers, 3 pooling layers, and 2 fully-connected layers.

This model performed better than all my previous models by almost a full 1%, confirming that the combination of Dropout and Batch Normalization is a good one for preventing overfitting while still achieving high accuracy and low error. Performance time was longer, however, than my previous experiments and I expect that in further experimentation if I was to add to this model it would increase the performance time further. However, 10 minutes is still well within an acceptable realm of performance time for the real-world application of most industries.

5. Conclusions

In this paper I have compared the structure of several different deep and convolutional neural networks. I explored the different strategies to combat overfitting, which is one of the major challenges with convolutional neural networks. Finally, I arrived on a model structure that combined two different types of regularization, to good results.

My recommendation is to take the best performing model from this research and use it as a starting point for model construction with a more complex dataset such as CIFAR-100. Once there is a reasonably high level of confidence as to its performance, the accuracy rate and error rate should be compared to the rate of human accuracy and error in the area of industry application. If the model's error rate falls below that of human error, then it is acceptable to begin preliminary usage in said industry. Wide-scale adoption of the technology should not take place until preliminary runs yield results in line with expected high accuracy and low error.

6. Citations

Brownlee, J. (2019, December 3). *A Gentle Introduction to Batch Normalization for Deep Neural Networks*. Machine Learning Mastery. https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.

Brownlee, J. (2020, August 25). How to Use Weight Decay to Reduce Overfitting of Neural Network in Keras. Machine Learning Mastery. https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-learning-with-weight-regularization/.

Chen, B. (2020, July 26). Batch Normalization in practice: an example with Keras and TensorFlow 2.0. Medium. https://towardsdatascience.com/batch-normalization-in-practice-an-example-with-keras-and-tensorflow-2-0-b1ec28bde96f.

Chollet, F. (2021). Chapter 5: Deep Learning For Computer Vision. In Deep Learning with Python (pp. N-A). essay, O'Reilly Media.

Deshpande, A. (2016, August 24). The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3). The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3) – Adit Deshpande – Engineering at Forward | UCLA CS '19. https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html.

Hill, K. (2020, December 29). Another Arrest, and Jail Time, Due to a Bad Facial Recognition Match. The New York Times. https://www.nytimes.com/2020/12/29/technology/facial-recognition-misidentify-jail.html.

Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR.

Jaiswal, S., Mehta, A., & Nandi, G. C. (2018). Investigation on the Effect of L1 an L2 Regularization on Image Features Extracted Using Restricted Boltzmann Machine. 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS). https://doi.org/10.1109/iccons.2018.8663071

Krizhevsky, A., Nair, V., & Hinton, G. (2009). The CIFAR-10 dataset. CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84–90. https://doi.org/10.1145/3065386

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324. https://doi.org/10.1109/5.726791

Powell, A. (2020, December 4). Risks and benefits of an AI revolution in medicine. Harvard Gazette. https://news.harvard.edu/gazette/story/2020/11/risks-and-benefits-of-an-ai-revolution-in-medicine/.

Pykes, K. (2021, July 19). Fighting Overfitting with L1 or L2 Regularization - Which One Is Better? neptune.ai. https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization.

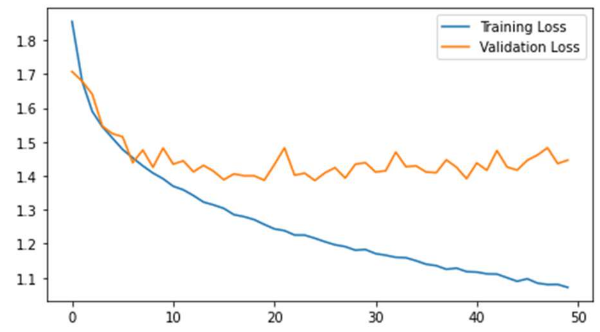
Riva, M. (2021, March 15). Batch Normalization in Convolutional Neural Networks. Baeldung on Computer Science. https://www.baeldung.com/cs/batch-normalization-cnn.

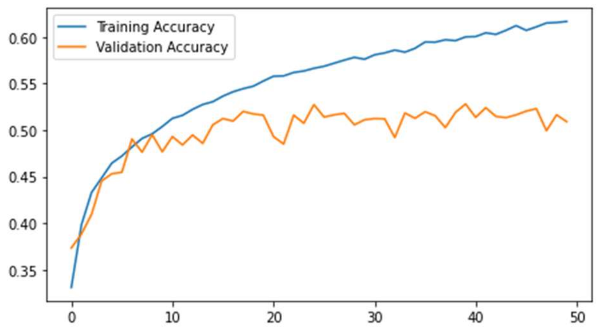
Saha, S. (2018, December 17). A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

7. Appendices

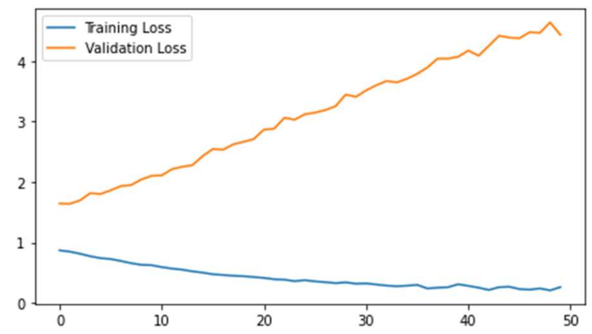
Appendix A

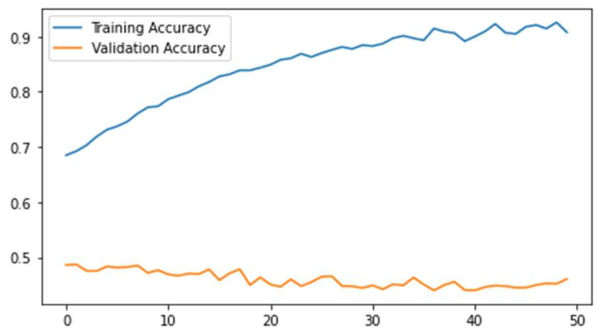
Performance Metrics:
Exp. 1:



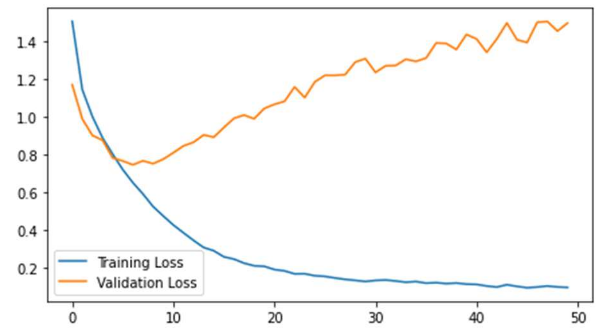


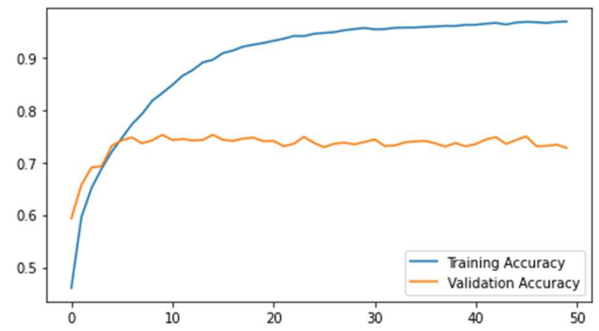
Exp. 2:



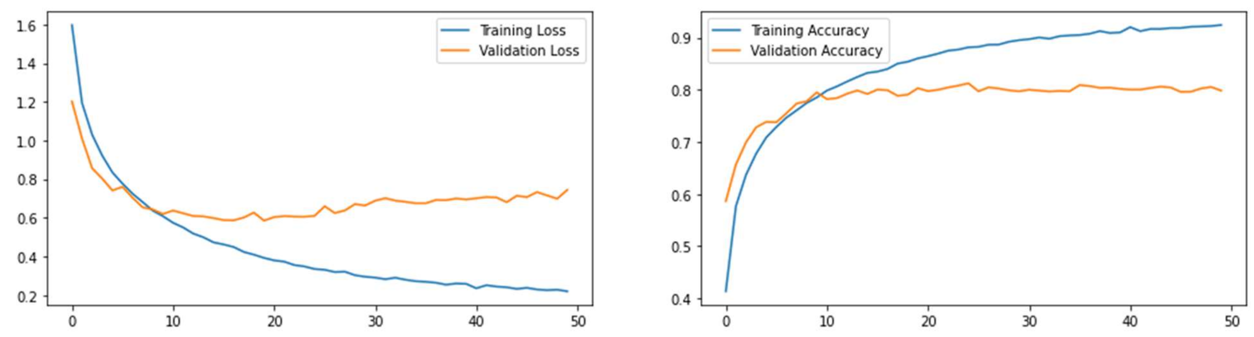


Exp. 3:

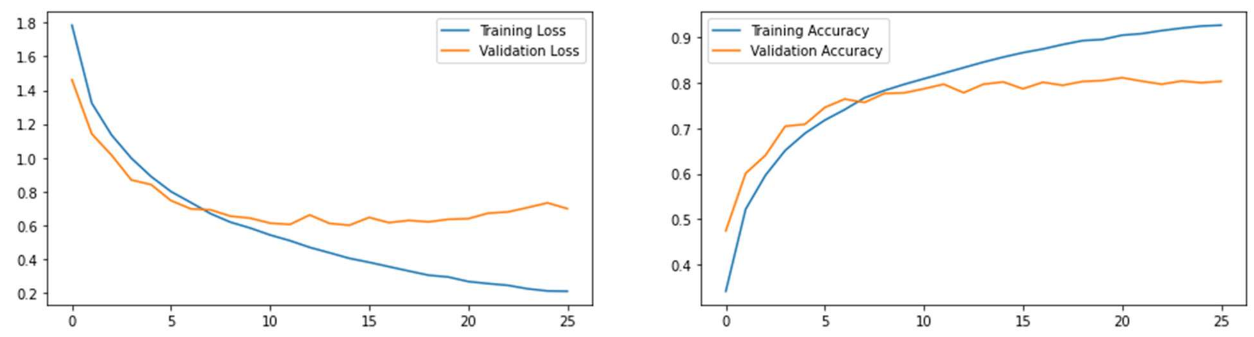




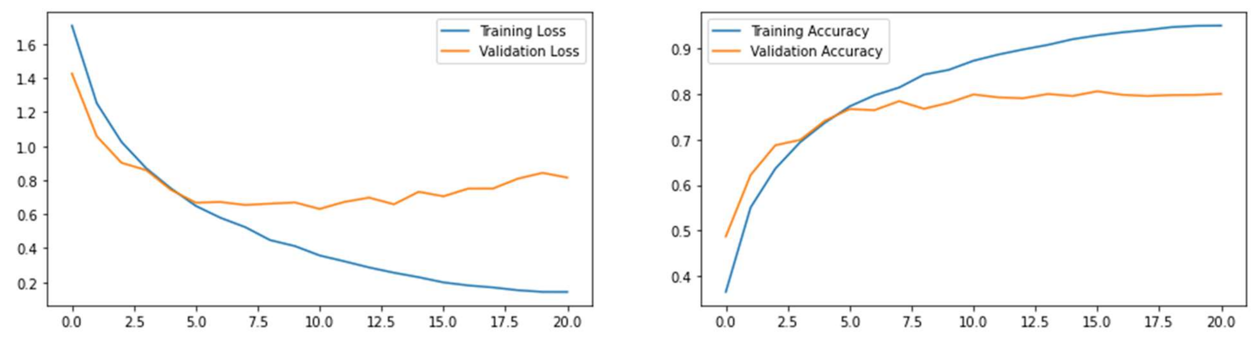
Exp. 4:



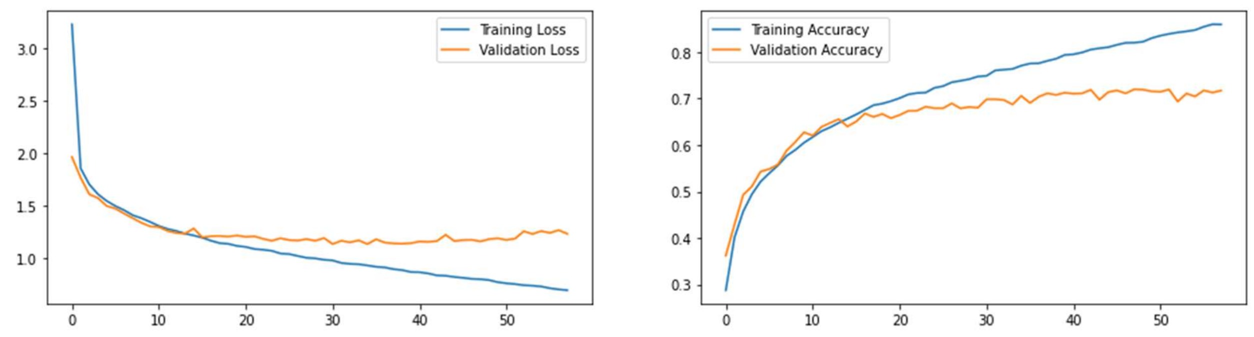
Exp. 5:



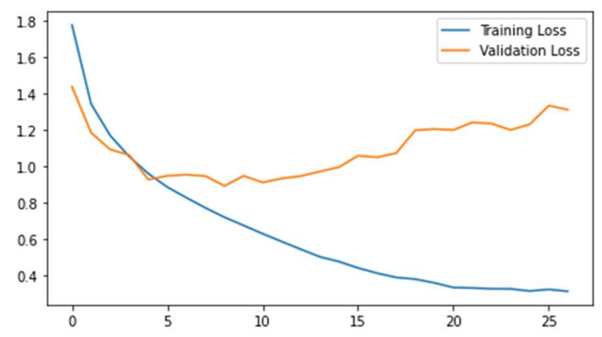
Exp. 6:

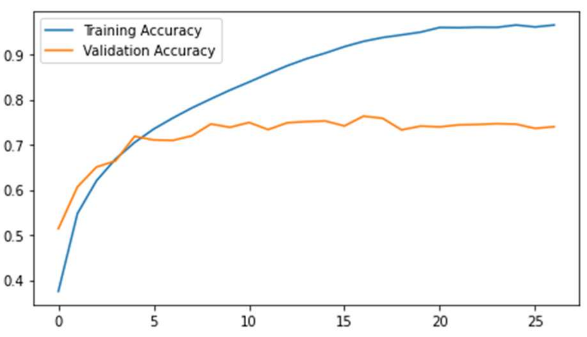


Exp. 7:

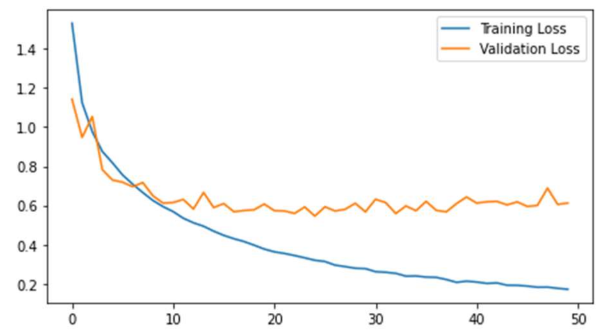


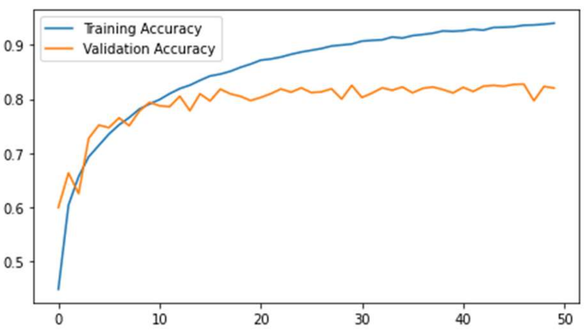
Exp. 8:



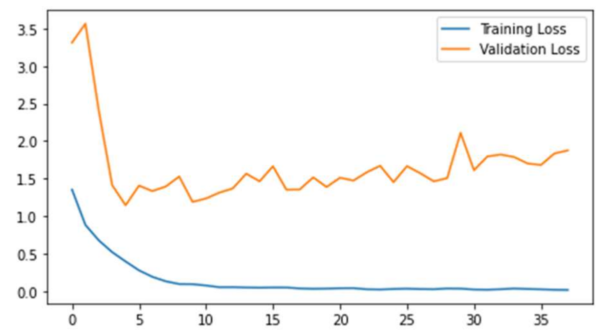


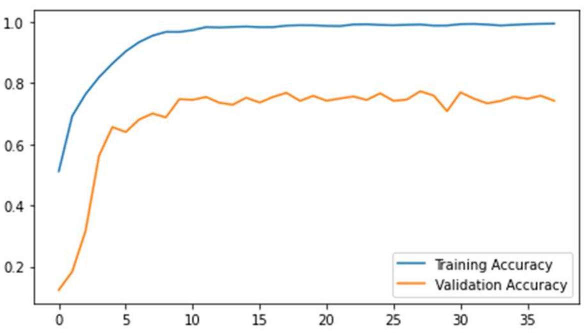
Exp. 9:



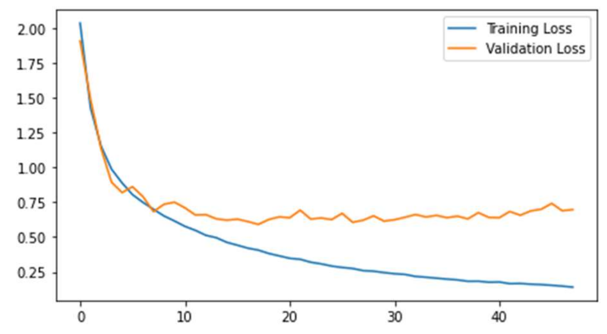


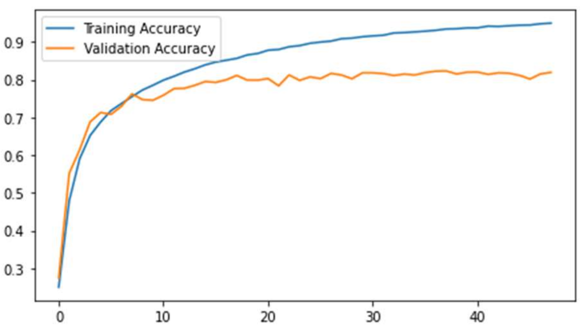
Exp. 10:



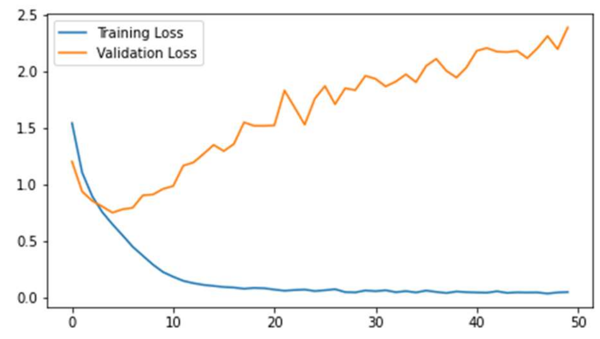


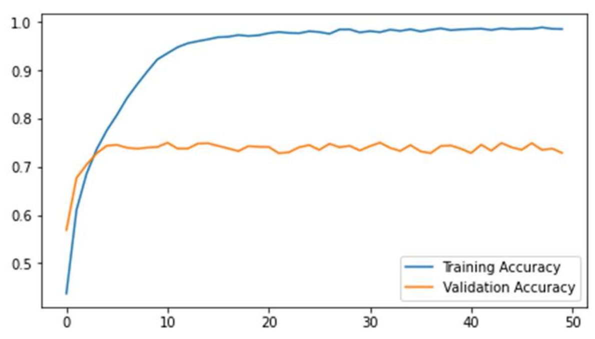
Exp. 11:





Exp. 12:

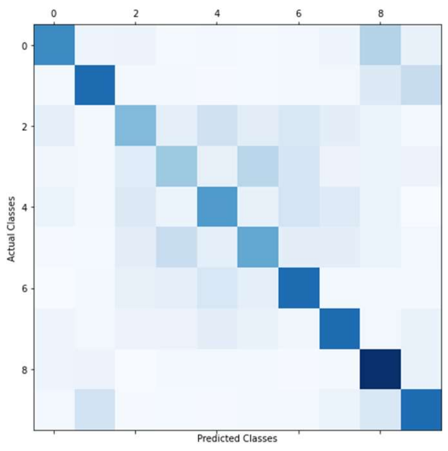




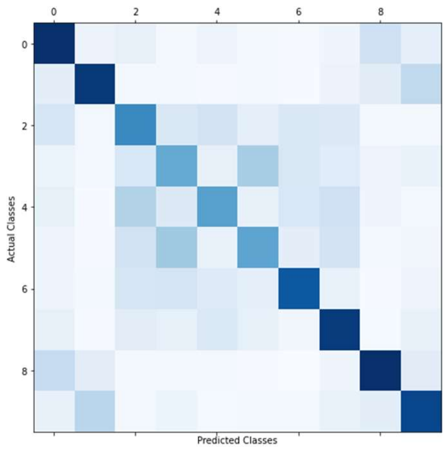
Appendix B

Confusion Matrices:

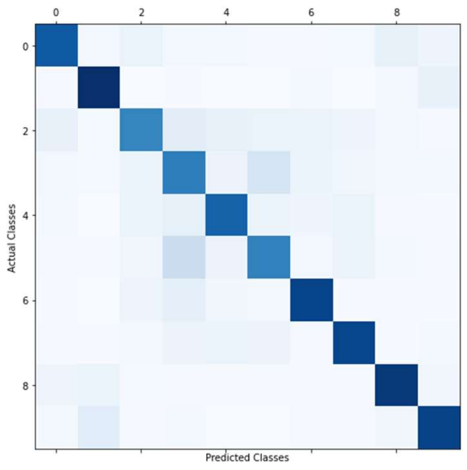
Exp. 1:



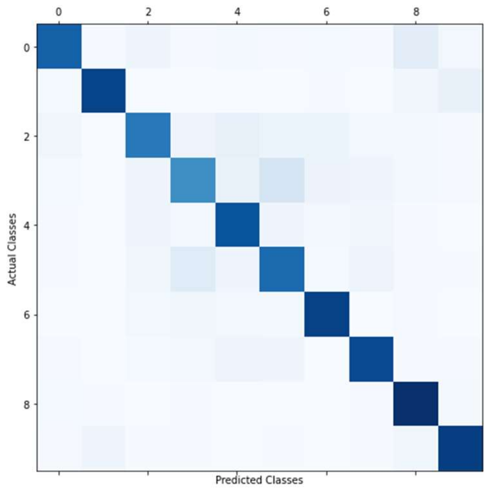
Exp. 2:



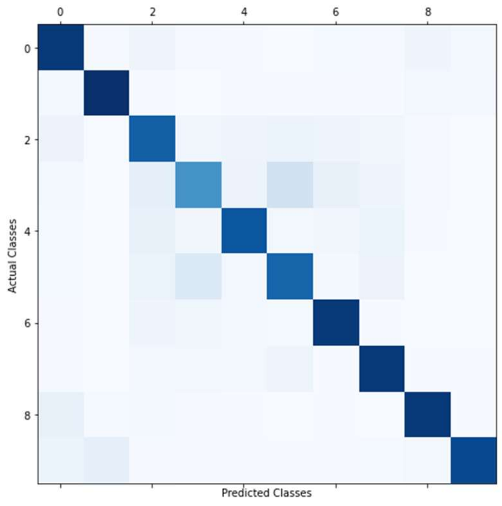
Exp. 3:



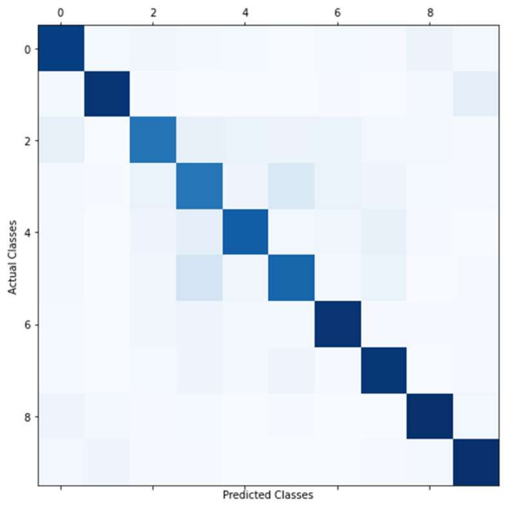
Exp. 4:



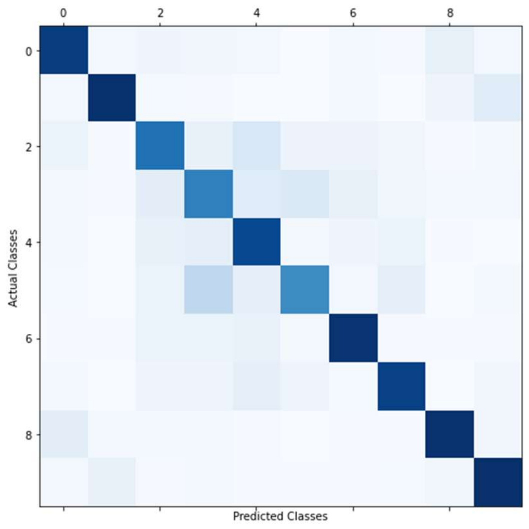
Exp. 5:



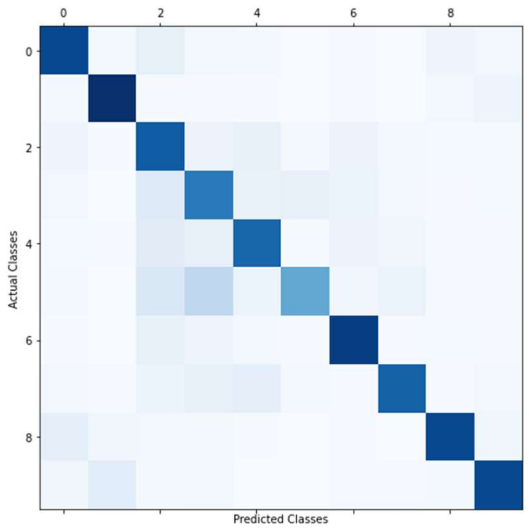
Exp. 6:



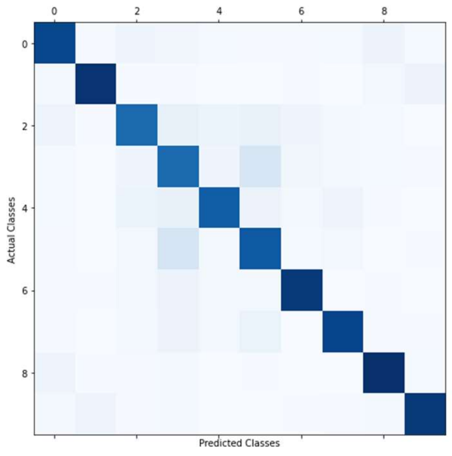
Exp. 7:



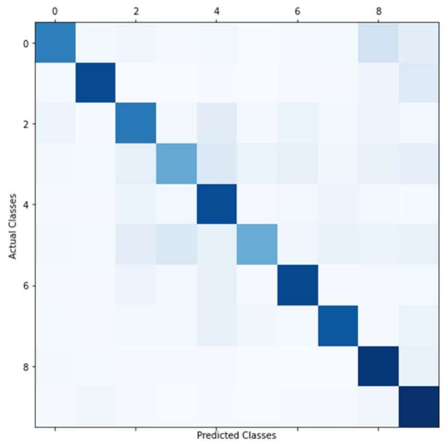
Exp. 8:



Exp. 9:

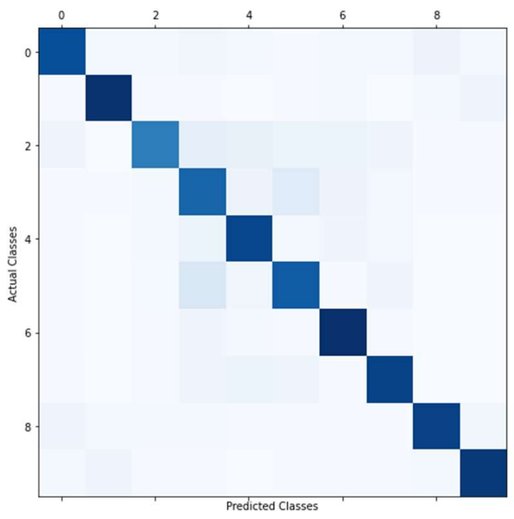


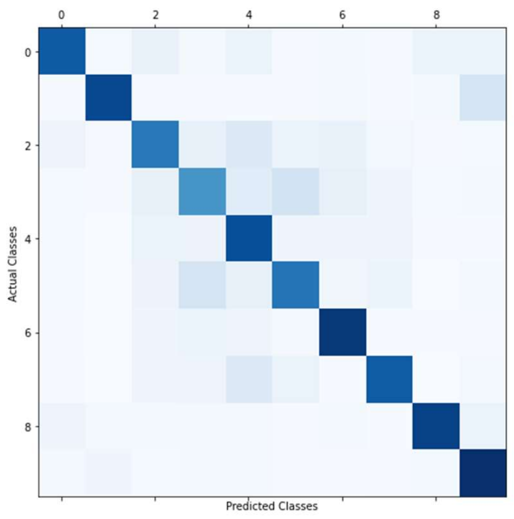
Exp. 10:



Exp. 11:

Exp. 12:





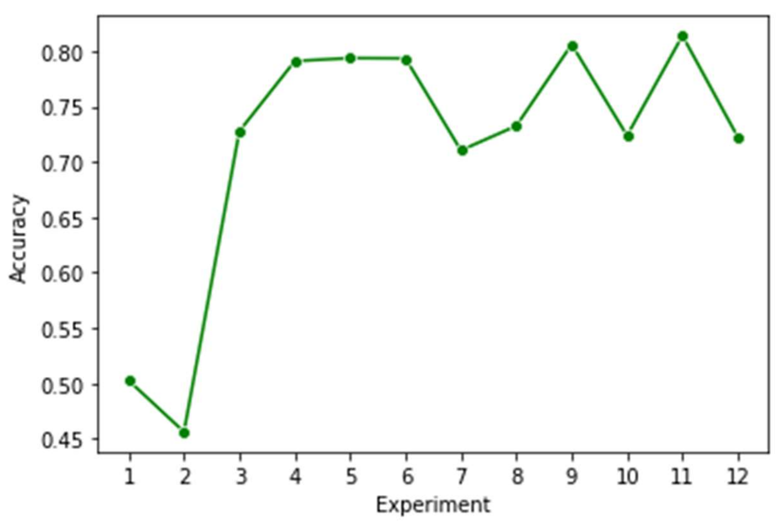
Appendix C

Table of Experimental Results:

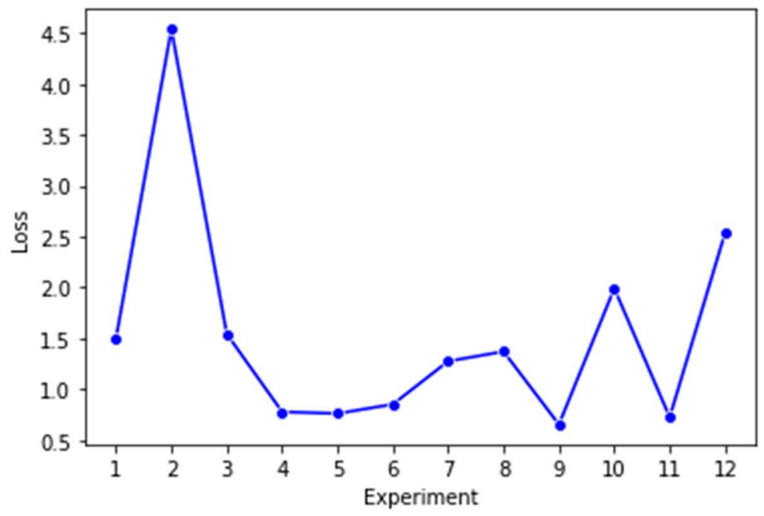
Experiment	1	2	3	4	5	6	7	8	9	10	11	12
Model Type	DNN	DNN	CNN	CNN	CNN	CNN	CNN	CNN	CNN	CNN	CNN	CNN
# of Layers	2 Dense	3 Dense	2 Conv 2 Pool 1 Dense	3 Conv 3 Pool 1 Dense	4 Conv 2 Pool 1 Dense	4 Conv 2 Pool 2 Dense	3 Conv 3 Pool 1 Dense	3 Conv 3 Pool 1 Dense	3 Conv 3 Pool 1 Dense	4 Conv 2 Pool 2 Dense	5 Conv 3 Pool 2 Dense	3 Conv 3 Pool 1 Dense
Regularization	N/A	N/A	2 Dropout (0.3)	3 Dropout (0.3)	2 Dropout (0.3)	2 Dropout (0.3)	L1 (0.0005)	L2 (0.0005)	3 Dropout (0.3) Batch Norm on Dense	Mult. BatchNorm	Dropout +Batch Norm	N/A
Test Accuracy	50.27%	45.63%	72.77%	79.10%	79.37%	79.34%	71.04%	73.25%	80.60%	72.37%	81.39%	72.23%
Test Loss	1.5008	4.5419	1.5441	0.7784	0.7616	0.8522	1.2761	1.3709	0.6543	1.9923	0.7253	2.5361
Processing Time (s)	134.16	188.13	382.93	443.08	324.36	260.24	378.72	178.51	443.12	560.02	620.96	443.06
Parameters	428,938	563,082	3,841,930	2,269,578	3,498,250	3,595,530	2,269,578	2,269,578	2,271,114	2,517,770	2,517,258	2,269,578

Appendix D

Plotted Accuracy, All Models:



Plotted Loss, All Models:


Loss
Experiment