Image classification using a Convolutional Neural Network

Arsham Atighechi **Herman Högman Ording** **Chuan Su**

Abstract

The convolutional Neural Network model has been widely adopted for image classification. In this study we performed replication experiments based on the report: *for Simplicity: The All Convolutional Net* by Springenberg et al. (2014) and achieved *85.09%* accuracy in comparison to their result *90.02%* without data augmentation. After performing a large set of experiments we discovered that a model consisting of Convolutional and Batch Normalization Layers with Adam optimizer using dropout and data augmentation tends to achieve higher accuracy. As a result, a classification accuracy of *92.68%* on the CIFAR-10 dataset is achieved in our study.

1 Introduction and related work

Image classification and object recognition is becoming increasingly important as automation becomes a more common technology. With technologies such as self driving cars becoming more developed there is a need to efficiently recognize an object in order to be able to act accordingly. This is what leads us to our problem:

- How to classify images using a simple ConvNet, while simultaneously achieve both high accuracies and efficiency?

This was done by setting up a base network from the study Springenberg et al. (2014), using their ConvNet architecture of multiple layers. This model was then continuously modified while the results were monitored. The model was then continuously modified in order to improve on the results and see how the architecture functions.

Some of the methods and models used in this report are based on the study: *for Simplicity: The All Convolutional Net* by Springenberg et al. (2014). The study concluded that it is not necessary to use a complicated model in order to achieve high rates of accuracy, nor is max-pooling and batch normalization required, which is commonly used in modern ConvNet architectures. By building a model consisting of multiple convolutional layers and data augmentation, Springenberg et al. (2014) managed to achieve an accuracy of *95.59%* on the CIFAR-10 dataset. It is important to note that this result was achieved using a very specific model that is not the focus of this study. Instead this study focuses on their model without large-scale data-augmentation which achieved an accuracy of 92.75% on the CIFAR-10 dataset.

1.1 Learning rate & Cyclical learning rate

Learning rate has always been a very important factor in successfully training a neural network. Previously, the state-of-the-art networks used methods such as momentum with a schedule that changes the learning rate over time. Smith (2015), however, experimented with a learning rate that would oscillate between a minimum and maximum value as training continued. In their paper, Smith discovered that using this type of oscillating behaviour resulted with an increase in efficiency. Without using the cyclical learning rates networks would often require a longer amounts of epochs before reaching a high validation accuracy. For instance, with a fixed learning rate, Smith (2015) reached an accuracy of 81.4% after 70,000 iterations. However, using the cyclical learning rates, the same result was achieved after only 25,000 iterations.

Cyclical learning rate implements a schedule that updates the learning rate over time. An upper and lower bound is set before-hand. With increasing epochs the learning rate will continuously oscillate within the range of the upper and lower bound. One of the implementation of cyclical learning rate includes the option to perform a decrease over time of the upper bound learning rate available to the model during training. Each cycle, the upper bound of the learning rate is decreased by a factor of 1/2. This method is referred to as *Triangular2* and is being used in most of our experiments involving cyclical learning rate. It is preferred due to the way it quickly enables the gradients to find their local minimum and once they are there the learning rate slows down allowing them to settle. Smith (2015)

1.2 Batch normalization

Batch Normalization helps reduce model over-fitting and thus is able to achieve higher classification accuracy. The usage of batch normalization causes the activation layers standardization to become much more stabilized and as a result, less regularization is needed. Ioffe and Szegedy (2015)

1.3 Data augmentation

Classification performed by a convolutional neural network should be invariant to image object translation, viewpoint, size or illumination. The training dataset may contain certain patterns that the model should not rely on to classify the images. This could lead to the model relying on certain patterns to classify an image while in reality those patterns are not related to the object being classified, such as orientation or scale. This is called over-fitting and is essentially the premise of data augmentation, which is used to increase the variety of data seen during training to achieve higher classification accuracy. In the study *Fractional Max-Pooling (Graham, 2015)* a test error of 3.82 % was achieved using data augmentation compared to a test error of 5.61% in traditional MP2 pooling for the CASIA-OLHWDB dataset which is a standard Chinese handwriting dataset. A recent Kaggle competition relating to CIFAR-10 was won with a test error of 4.47% using training data augmentation, which even surpassed human performance.

1.4 Batch size

According to Keskar et al. (2016) larger batch sizes can be more efficient computationally, however, this could also lead to a worse performance. This is due to smaller batch sizes sometimes causing the gradients to go "in the wrong way" - causing the weights to jump out of a bad local minima. Keskar et al. (2016)

1.5 Dropout

Dropout is a regularization technique that reduces over-fitting in neural networks. The term dropout refers to dropping output units (both hidden and visible), which causes the the model to not rely on only one feature and the weights to be spread out. Dropout is similar to the approach of L2 regularization in reducing over-fitting. Sullivan (2019)

1.6 Optimizers

The study by Springenberg et al. (2014) used a modified SGD optimizer with a learning rate 0.01 with a fixed multiplier of 0.1 after 200, 250 and 300s epochs respectively and momentum of 0.9. According to lecture notes by Sullivan (2019) adaptive learning-rate methods are generally easier to implement than normal SGD since less fine-tuning of the learning rates are required. There are other optimizers than SGD available, for example ADAM optimizer which is an adaptive learning rate optimizer which computes different learning rates for the parameters it optimizes based on how they behave. It is well suited for large datasets, and has empirically shown to produce good results according to Kingma and Ba (2014).

1.7 Data

For this project, the CIFAR-10 dataset from Toronto University was used, which consists of 10 different labeled classes with a total of 60,000 picture samples. In this project 50,000 samples were used to train the network and the remaining 10,000 images were used for testing. Additionally, each image data was preprocessed to have have values between 0 and 1 or preprocessed to have zero mean and then normalized based on this mean. Springenberg et al. (2014)

This dataset is commonly adopted to train various neural networks for image classification. The current state-of-the-art neural networks have been able to achieve accuracies as high as *99.0%* using different methods such as pipeline parallelism, automatic data augmentation, fractional max-pooling, among others. One of the best performing results on the CIFAR-10 dataset is held by Devries and Taylor (2017). In this study the idea is to use standard data augmentation together with cutout data augmentation to reduce the over-fitting that generally happens in convolutional neural networks. This has proven to work well for them, reaching accuracies as high as *97.69%* on the CIFAR-10 dataset.

Most state-of-the-art results seem to be focusing on reducing over-fitting by improving regularization. For instance the automatic data augmentation focuses on an algorithm that tries out multiple different augmentation policies and then applies the most appropriate one. Cubuk et al. (2018)

2 Methods

In order to build up a base for our model, we looked at the base model presented by Springenberg et al. (2014). The model is based on a simple ConvNet architecture with added dropout as well as a layer for global averaging as presented in Table 3. Our base model does not include any batch normalization in order to stay true to the experiments made by Springenberg et al. (2014), which did not use batch normalization. However, this model was only used as a base, and changes were made in order to apply different methods. The base model uses the Stochastic Gradient Decent optimizer, with set weight decay factors of 0.001 on each layer.

In our experiments this model was adjusted and different concepts were introduced in order to improve both accuracy and efficiency. The adjustments we made included the application of batch normalization, standardization as well as data augmentation. While the study of Springenberg et al. (2014) did include some data augmentation in the form of whitening and contrast normalization it was not clear exactly how they did it. Because of this uncertainty that type of data augmentation was not included in our base model. Furthermore, cyclical learning rate was implemented as developed by Smith (2015). Additionally, when performing experiments on our base model from table 3 the ADAM optimizer was used so as to not have to fine-tune the learning rate as much as would be needed when using SGD.

When implementing cyclical learning rate our initial upper and lower bounds for learning rate were based on the results discovered by Smith (2015) with base learning rate of 0.001 and a max learning rate of 0.006. These learning rates were altered during our experiments. The code used to implement the cyclical learning rate was taken from Kenstler (2019) and each update step corresponds to a new batch handled by the model. The other method of learning rate adjustments applied was learning rate decay with a fixed schedule as seen in Springenberg et al. (2014). This method had initial learning rate of 0.01 and after epoch 200, 250 and 300 the learning rate was set to reduce by a factor 0.1.

Finally, in order to further push the accuracy slightly higher, data augmentation was introduced. The data augmentation of choice was online augmentation, which translates to continuous data augmentation during training. Whereas offline data augmentation is done prior to the training. Both online and offline augmentation were tested in our experiments, however offline augmentation in a large dataset, such as CIFAR-10, proved to be somewhat inefficient. Using offline data-augmentation required a lot more computational power which would add too much computational time. Therefore, offline data-augmentation was not implemented in the final model, instead online data augmentation was used, and the images were slightly altered as they were fed into the ConvNet. The settings for the data augmentation was random rotation by ± 20 degrees, random translations horizontally and vertically by values up to 20% of the total length in each direction. Finally, a 50% chance to horizontally flip the image.

We also performed experiments with different amounts of dropout to see how it would affect the regularization of the model.

Table 1: Base Convnet model used for experiments, "All-cnn-c"

Layer	Name
1	Input 32 x 32 image with 20% dropout chance
2	3 x 3 Conv Layer with filter size 96, ReLU
3	3 x 3 Conv Layer with filter size 96, ReLU
4	3 x 3 Conv Layer with filter size 96, ReLU with stride 2 and 50% dropout chance
5	3 x 3 Conv Layer with filter size 192, ReLU
6	3 x 3 Conv Layer with filter size 192, ReLU
7	3 x 3 Conv Layer with filter size 192, ReLU with stride 2 and 50% dropout chance
8	3 x 3 Conv Layer with filter size 192, ReLU
9	1 x 1 Conv Layer with filter size 192, ReLU
10	1 x 1 Conv Layer with filter size 10, ReLU
11	Global averaging over 6 x 6 spatial dimensions using 2D global averaging with input shape (6, 6, 10)
12	Flattening the data into a fully connected 10-way softmax
	All the trainable weights in the base model have weight decay of 0.001

3 Experiments

The experiments shown in this section were made using the CIFAR-10 dataset as mentioned in section 1.2. Not all experiments are included in this section. There were a lot of minor adjustments and testing but not all of them had relevant outcomes and therefore not presented. In Table 2, a short description of the different experiments performed is presented, the models used in these experiments all have between *1-1.4* million trainable parameters.

3.1 Result

Table 2, row 1 and 2 presents the results of our replication experiments for the study *for Simplicity: The All Convolutional Net* by Springenberg et al. (2014) *Table 3* presents the experiment results based on our model with *Adam* Optimizer.

Table 2: All-CNN-C Replication With SGD Optimizer Experiment Result

#	DA[1]	LR[2] + M[3]	BS[4]	Dropout	Averaging	BN[5]	Epochs	WD[6]	Accuracy
1	-	0.01[7]+0.9	100	0.2,0.5,0.5	6x6x10	-	350	1e-3	85.06%
2	-	0.01 + 0.9	100	0.2,0.5,0.5	6x6x10	-	350	1e-3	85.27%
3	x	0.01 + 0.9	100	0.2,0.5,0.5	6x6x10	-	350	1e-3	85.95%
4	-	0.01 - 0.06 + 0.9	32	0.2,0.5,0.5	6x6x10	-	350	1e-3	86.73%

Table 3: All-CNN-C With Adam Optimizer Experiment Result

#	DA[1]	LR[2]	BS[4]	Dropout	Averaging	BN[5]	Epochs	WD[6]	Accuracy
5	x	0.01	32	3 x 0.2	x	x	350	-	92.68%
6	x	1e-3 - 6e-3	32	3 x 0.2	x	x	350	-	91.7%
7	x	1e-3 - 6e-3	100	3 x 0.2	-	x	350	1e-5	91.64%
8	x	1e-3 - 6e-3	100	3 x 0.2	-	x	300	1e-5	91.36%
9	x	1e-3 - 6e-3	100	3 x 0.2	-	x	100	-	90.45%
10	x	1e-3 - 6e-3	32	-	x	x	100	-	90.07%
11	x	1e-3 - 6e-3	100	3 x 0.2	-	x	100	-	89.99%
12	-	1e-3[7]	32	3 x 0.2	-	x	350	-	87.07%
13	x	1e-3 - 6e-3	32	-	-	x	25	-	86.22%
14	x	1e-3 - 6e-3	100	3 x 0.2	-	x	25	-	87.36%
15	x[8]	0.001 - 0.006	100	3 x 0.2	-	x	100	2e-3	60%

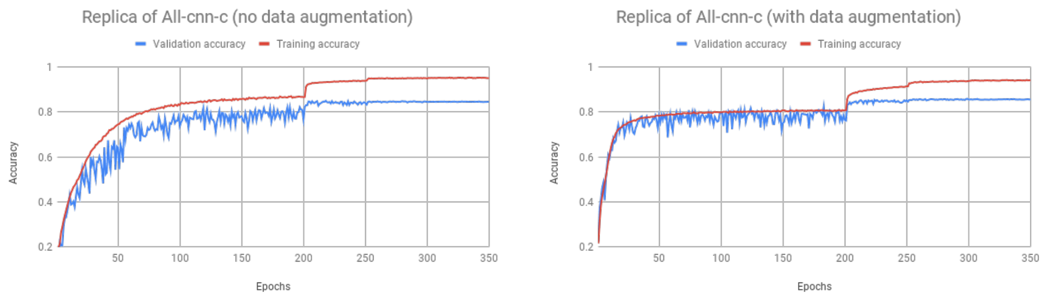


Figure 1: Left: Highest achieved result in replication experiments: without data augmentation (Experiment #1). Right: with data augmentation (Experiment #3)

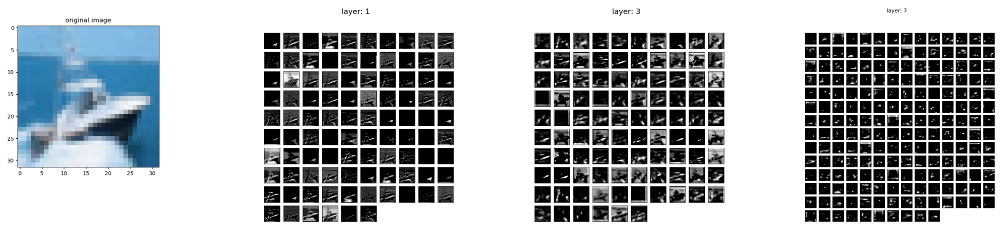
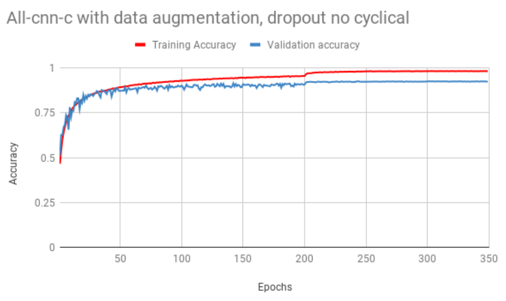


Figure 2: Visualization of the layer output for our replica of the All-cnn-c model (Experiment #1). From left to right: Original image, first convnet layer, third convnet layer, sixth convnet layer

3.2 Discussion

To evaluate the performance to be improved from applying data augmentation, we performed two experiments with the same model, optimizer and hyper-parameters to the CIFAR-10 training dataset either with data augmentation or without.

[1]Data augmentation
[2]Learning rate / Cyclical Learning rate
[3]Batch size
[4]Batch normalization
[5]Weight decay
[6]Momentum
[7]Learning rate 0.01 with a fixed multiplier of 0.1 after 200, 250 and 300s epochs respectively.
[8]Also included 50% chance to vertically flip the image.



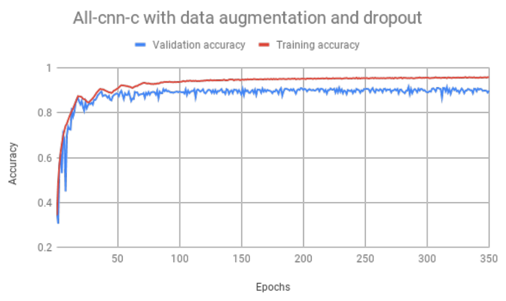


Figure 3: Highest achieved result with our network, uses data augmentation and dropout Left: without cyclical learning rate (Experiment #5), Right: with cyclical learning rate (Experiment #7)

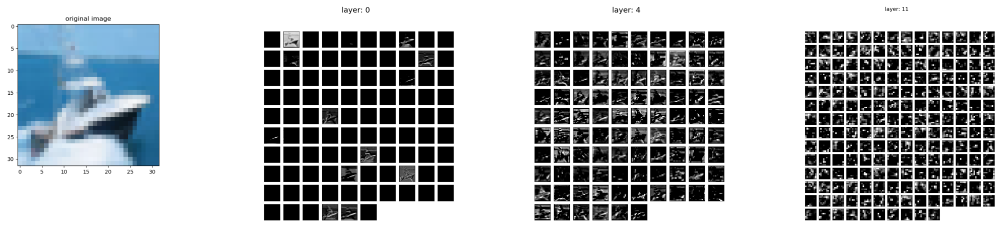


Figure 4: Visualization of the layer output for the best performing model (Experiment #5). From left to right: Original image, first convnet layer, third convnet layer, sixth convnet layer

A test error of 7.53% is achieved in comparison to 11.2% without data augmentation. It is remarkable that under-fitting can occur when combined with 'too much' dropout or training data augmentation (Graham, 2015). We experienced under-fitting issues that only achieves 60% test accuracy when applying too much training data augmentation in our experiment along with dropout and weight decay.

The biggest takeaway from doing these changes is the importance of certain areas of the model. For instance, during our experiments, when we removed layers for batch normalization the accuracy would drop significantly. Even when we attempted to replicate the model of Springenberg et al. (2014), the results dropped as we removed the batch normalization. Additionally, we noticed that some form of regularization is required. Removing the dropout layers increased the amount of over-fitting, while having too many dropout layers lead to under-fitting.

We also experimented with using Cyclical Learning Rate with lower bounds *0.001 - 0.006* compared to the learning rate of *0.01* used in the study Springenberg et al. (2014) and only achieved an accuracy of *80%* in epoch *200*, which is much lower when compared to higher learning rate values. It is obvious to see that the learning rate plays a key role in terms of training efficiency.

We found that overusing dropout regularization may result in a higher probability of under-fitting in image classification. We experimented with applying fraction of dropout *0.5* on input layer and resulted in lower accuracy (under-fitting) in comparison to the approach of applying *0.0* dropout (no dropout at all) on input layer.

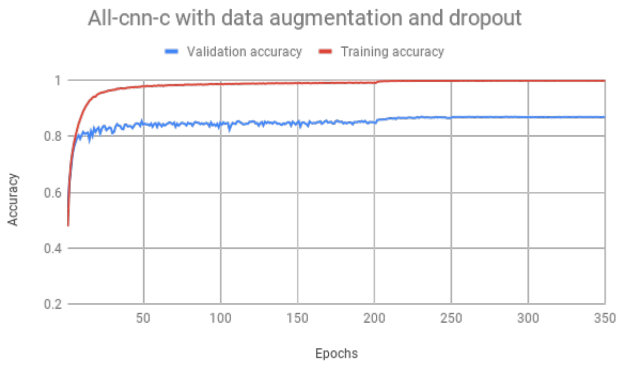


Figure 5: Results from having a bad choices of learning rate, causing the results to not improve (Experiment #12)

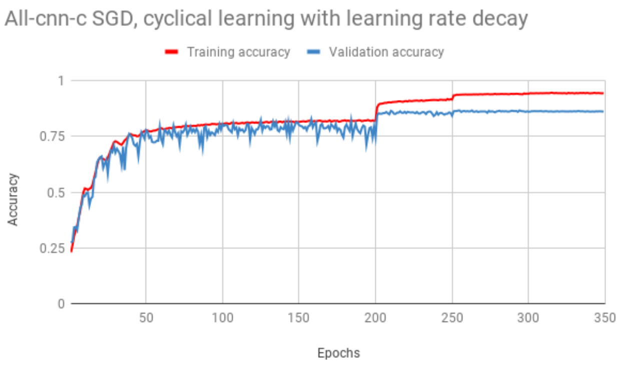


Figure 6: Merging both scheduled learning rate decay and cyclical learning rate without data augmentation and batch normalization (Experiment #4)

3.3 Common failures

The most common failure was that the models experienced over-fitting or under-fitting, which resulted in a low validation accuracy. We mainly attribute this to incorrect usage of hyper-parameters or too heavy usage of regularization methods and it took some experimenting to find good values. Another reason this happens could be due to the learning rate being too low, causing the gradients to get stuck in local minmas and therefore not being able to climb out of the local minimas, as shown in Figure 5.

4 Conclusion

We did not manage to recreate the results achieved in Springenberg et al. (2014). Their SGD experiments reached *90.72%* and our experiments with their SGD model only reached *85.95%* classification accuracy. We believe that our implementation was a correct replica in all aspects except for the data augmentation, which we were not able to recreate due to computational requirements on CPU which were not feasible. Looking back at the result one thing that stands out is how important regularization is for avoiding over-fitting when having so many trainable parameters. With roughly *1.3* million trainable parameters it is possible to create a model that perfectly predicts every item in the training data but has very poor generalizability and as such is not very good at predicting data which it has not encountered before. This can be shown when comparing Experiment #1 to Experiment #12, where the one with data augmentation performed significantly better.

This lack of data augmentation would then explain why our replica was not able to achieve as high of an accuracy as Springenberg et al. (2014) did. Also worth noting is that we did manage to reach roughly the same accuracy as them with a slightly modified version of their base model in Table 3 experiment number 5, we believe this to be due to that experiment having good data augmentation.

Regarding cyclical learning rate we expected to see significant changes between models using it and models without it. We did see slightly higher performance with cyclical learning rate in certain cases, however this was not always the case. For instance, in the instance when using Adam optimizer we did not notice any significant changes between using cyclical learning rate and without. In the case of SGD it did seem to have a small increase in efficiency.

4.1 Future improvements

There are some parameters that we did not have time or opportunity to test around with too much. The activation function used is one of those - we only used the ReLU activation function. If this work was to be improved upon in the future one thing that would be good to look at would be how other activation functions perform. Figure 6 shows a combination of multiple learning rate strategies, however without data augmentation. This model performed very well and could use further improvements using data augmentation

References

Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V. and Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data, *CoRR* **abs/1805.09501**.

Devries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout, *CoRR* **abs/1708.04552**.
URL: *http://arxiv.org/abs/1708.04552*

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift, *ICML*.

Kenstler, B. (2019). Cyclical learning rate.
URL: *https://github.com/bckenstler/CLR*

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M. and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima, *CoRR* **abs/1609.04836**.
URL: *http://arxiv.org/abs/1609.04836*

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Smith, L. N. (2015). Cyclical learning rates for training neural networks, *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)* pp. 464–472.

Springenberg, J. T., Dosovitskiy, A., Brox, T. and Riedmiller, M. A. (2014). Striving for simplicity: The all convolutional net, *CoRR* **abs/1412.6806**.

Sullivan, J. (2019). Lecture notes in dd2424.

Appendix

Individual Study Evaluation

Arsham Atighechi

I have gotten a better understanding of how to structure and set up models using the Tensorflow Keras backend, as well as the perks in using modern GPUs for computations as compared to use CPUs. Additionally, I have received further insight in the workings of data augmentation, why it is important, how it affects the results (over-fitting vs under-fitting) as well as the effect it has on the code itself, in terms of efficiency and accuracy. Finally, I have received a much better understanding for the 2 optimizers, Adaptive Moment Estimation (Adam) and Stochastic Gradient Descent (SGD).

This knowledge was achieved by delving into the tenserflow and keras API, reading articles regarding the subject, watching youtube guides. Most importantly testing my way forward while coding, this way I was able to find out what works and does not work, however the Keras API is very well documented and that helped out a lot. Additionally the Arxiv site was a great site to go deeper into a specific topic in order to be able to learn further. The evidence for this would be find in the multiple codes written and posted in our git repository. Finally, this report should also serve as some proof as to what I have learned.

Herman Högman Ording

While working on this project I feel like I have become decently proficient in how to use Tensorflow with Keras backend for implementing a sequential model using Python. This proficiency comes from extensive trial & error when implementing our models and includes using callbacks for saving data and/or changing parameters during runtime, as well as how to set it up to make it actually work. Evidence of this can be found in the code written for the project.

I have also gotten a better understanding of how regularization can be used and how it should not be used. This knowledge was acquired by reading lecture notes, different guides online and most importantly trying different settings in our experiments. When launching different experiments testing different regularization methods - like dropout, data augmentation, and weight decay - I feel like I learned how they can be implemented and I also discovered what happens when you implement too much regularization. Evidence of this are the different experiments we performed were we encountered both overfitting and underfitting as well as the fact that we managed to find a good middle-ground with good performance.

The third major thing I feel I have learned during this project is a deeper understanding for how cyclical learning works and how to implement it in a model not really built for it (i.e. implementing it into Tensorflow Keras using callbacks and calculating how many epochs would correspond to a cycle etc.) This was done by finding a Github repository where they had created an implementation for using cyclical learning. I then studied how this worked and figured out how it's learning rate decay could be adapted to our project in a way that actually enables us to get the full advantage of ending the training when the learning rate is at its lowest - something that did not happen with the default settings. Evidence of this is in the implementation of the cyclical learning rate class as well as how it is used.

Chuan Su

In this project, I have gained a good understanding of ConNet Model architecture, ConvNet, Pooling and Full Connected Layers through reading lecture notes and standford online videos as well as group discussions. I also experienced tensorflow and keras the modern Deep Learning frameworks and learned how to build Sequential Keras Model with deep layers. The Tensorflow documentation, Stackoverflow as well as group mates helped a lot. Moreover I have learned the application of Data Augmentation to increase modal classification invariance as well as the use of Dropout layer for reducing overfitting. I mainly get to know this through reading papers and online tutorials as well as experiments performed in this project.