Student Name: Lee T.

CSC2034 Major Project Report
Convolutional Neural Networks Performing on CIFAR-10 Dataset

Abstract
The overall objective was to train a Convolutional Neural Network on the CIFAR-10 dataset to the best possible standard. Whilst also modifying and testing different aspects of the CNN such as, base model, optimizer, data augmentation, and general training parameters.
In my report I tested all of these different aspects with detailed tabled results. My final model achieved a testing accuracy of 90.88% and an average testing accuracy of 90.468%. I tested a range of different predefined models, optimizers, learning rates, and general training parameters. As well as varying configurations for data augmentation. The configuration of my final model is detailed below the final table of tests.

What Was Done and How

Examining Predefined Models

I decided to use a pre-defined Convolutional Neural Network (CNN) I researched how these work and how I can utilize them. I learned that by using a pre-defined model this is, transfer learning. As the base model provided has already been built and trained, to process images and extract features. These features are inputted into the top layer which needs to be trained[1].
To examine which model would perform the best performance without modification. I decided to test each predefined model from this list provided by keras[2] that would accept images of shape (32, 32, 3). According to this book, Build Your Own Neural Network by T. Rashid[3, p. 169]. It showed, more nodes/units in a layer in some cases can lead to the model achieving a higher accuracy. Therefore, I decided while testing different pre-defined models I would also test different amounts of final layer units.

Final Layer Units	Model	Training Accuracy	Testing Accuracy	Runtime (Seconds)
1024	VGG16	72.25%	69.46%	108.51s
2048	VGG16	71.86%	71.01%	108.87s
4096	VGG16	71.59%	71.10%	109.67s
8192	VGG16	71.99%	70.36%	112.07s
16384	VGG16	70.98%	68.96%	114.63s
32768	VGG16	71.75%	69.31%	122.03s
65536	VGG16	71.45%	69.57%	131.79s
1024	ResNet50	39.70%	35.96%	142.52s
2048	ResNet50	43.57%	40.54%	142.84s
4096	ResNet50	48.65%	45.04%	145.45s
1024	ResNet50V2	18.57%	18.26%	142.76s
1024	ResNet101	41.83%	38.50%	247.52s
2048	ResNet101	45.66%	42.38%	249.82s
4096	ResNet101	51.05%	46.64%	253.45s
1024	ResNet101V2	11.85%	13.21%	249.58s

Student Name: Lee T.

1024	ResNet152	40.24%	38.51%	348.48s
1024	ResNet152V2	12.39%	12.77%	350.10s
1024	MobileNet	30.44%	30.17%	53.15s
2048	MobileNet	35.42%	35.40%	52.89s
4096	MobileNet	42.17%	40.71%	55.29s
8192	MobileNet	46.81%	45.85%	105.62s
16384	MobileNet	51.15%	48.94%	81.68s
32768	MobileNet	53.51%	51.21%	106.35s
65536	MobileNet	54.45%	52.55%	94.55s
131072	MobileNet	55.34%	53.44%	225.02s
1024	MobileNetV2	22.90%	13.60%	70.47s
1024	MobileNetV3Large	18.74%	9.72%	88.58s
1024	MobileNetV3Small	15.11%	10.45%	64.06s
1024	DenseNet121	40.19%	39.99%	149.08s
1024	DenseNet169	47.06%	47.13%	195.58s
1024	DenseNet201	48.50%	48.37%	240.74s
1024	EfficientNetB0	18.92%	19.69%	125.94s
1024	EfficientNetB1	15.00%	15.28%	172.38s
1024	EfficientNetB2	16.70%	18.76%	174.90s
1024	EfficientNetB3	19.26%	20.53%	208.00s
1024	EfficientNetB4	14.67%	15.85%	266.91s
1024	EfficientNetB5	16.70%	18.21%	350.38s
1024	EfficientNetB6	15.59%	16.00%	431.56s
1024	EfficientNetB7	17.61%	18.30%	555.80s

Predefined model test results.

The first model I tested VGG16 achieved a testing accuracy of 69.46%. I then increased the number of units in the top layer for which my next test yielded a higher accuracy of 71.01%. Once again, I doubled the number of units from 2048 to 4096 in which this test yielded a 0.09% increase. For my next 4 tests I doubled the number of units to ensure that 4096 units was the optimum value. From these four tests I found that 4096 was the optimum value. I then moved onto testing other models. While testing all other predefined models I tested them only with 1024 units to start with. If the model had a high enough accuracy on the first test, I would increase the number of units by doubling the previous value. This was to examine whether there was a predefined model that could achieve more accuracy than VGG16. If the model scored considerably lower than VGG16 I would move onto testing the next model.

While testing the MobileNet model I found the runtime around ~50% faster than the VGG16 model with 1024 units. Based on this I decided to test MobileNet with more units for the final layer. However, I found the runtime for MobileNet became considerably higher than VGG16 whilst achieving a lower accuracy.

Student Name: Lee T.

Based on my tests an increase in units in the top layer usually yields an increase in accuracy. I found VGG16 with 4096 final layer units to be the best model for this dataset. As this model performed the best, I decided to carry on testing other aspects with this configuration.

Examining Optimizers

According to this book on neural networks[3, p. 74] (NNs) when the NN predicts incorrectly during training an error or loss is calculated. The weights that determine the classification based on input are updated by this loss. According to this article[4] Optimizers determine how the weights are updated. Importantly "optimizers are related to model accuracy, a key component". Therefore, I decided to test all optimizers provided by keras[5]. To examine which one performs the best on the CIFAR-10 dataset. Furthermore, while testing different optimizers I also decided that based on this book[3, p.254], as the weights are updated by a proportion of the error determined by the learning rate. I would also modify and test different learning rates for the best optimizers. As adjusting the learning rate can lead to a possible increase in accuracy.

Optimizer	Learning Rate	Training Accuracy	Testing Accuracy	Runtime (Seconds)
Adadelta	0.001	71.59%	71.10%	109.67s
Adam	0.001	91.20%	78.74%	175.27s
SGD	0.01	86.31%	81.30%	174.60s
RMSProp	0.001	10.12%	10.00%	173.02s
Adagrad	0.001	81.81%	77.50%	177.43s
Adamax	0.001	96.54%	83.26%	176.84s
Nadam	0.001	10.03%	10.00%	185.78s
Ftrl	0.001	9.84%	10.00%	171.98s
SGD	0.10	10.10%	10.00%	163.68s
SGD	0.10	9.89%	10.00%	163.56S
SGD	0.05	87.22%	82.46%	171.96s
SGD	0.025	88.82%	83.38%	174.14s
SGD	0.0125	87.89%	78.44%	174.68s
Adamax	0.010	41.33%	42.16%	170.10s
Adamax	0.005	10.10%	10.00%	166.45s
Adamax	0.0025	9.67%	10.00%	167.10s

My first 7 tests are for each optimizer with their default values. From these 7 tests I found SGD and Adamax to achieve the highest testing accuracies, 81.30% and 83.26% respectively. The next 8 tests I modified and tested different learning rates with these two optimizers. When testing different

learning rates, I started with a value ten times greater than the default. When adjusting the learning rate for SGD the first two tests indicated a higher value degrades the quality of the model. I decided to half the value from 0.10 to 0.05 which achieved a higher accuracy. This indicated a smaller value may increase the accuracy. Based on this I halved the learning rate again until the quality of the model no longer improved. From these tests I found that using SGD with a learning rate of 0.025 performed better than Adamax with default values.

As SGD with a learning rate of 0.025 and Adamax with default values achieve similar results. I decided to test both optimizers a repeated number of times to examine which achieves a higher accuracy on average.

Optimizer	Learning Rate	Training Accuracy	Testing Accuracy	Runtime (Seconds)
Adamax	0.001	96.54%	83.26%	176.84s
Adamax	0.001	97.24%	84.71%	420.82s
Adamax	0.001	96.93%	83.08%	216.97s
Adamax	0.001	96.99%	84.42%	180.24s
Adamax	0.001	96.51%	84.37%	181.26s
SGD	0.025	88.82%	83.38%	174.14s
SGD	0.025	72.81%	83.33%	179.45s
SGD	0.025	91.28%	81.05%	179.25s
SGD	0.025	90.72%	79.52%	178.92s
SGD	0.025	89.15%	84.68%	178.83s
Adamax Avg. Training Accuracy: 83.968%			**SGD Avg. Training Accuracy:** 82.392%	

From my test results I found that using Adamax on average achieved a higher accuracy of 83.968%. Whereas using SGD achieved an average of 82.392%, 1.576% less than using Adamax. Based on current and previous results I decided to carry on testing models with both optimizers. As the average results are quite close in difference.

Examining General Training Parameters

According to this article[6] the batch size is the number of training examples inputted before the weights are updated. Batch sizes are often tested in powers of 2s. As the CNN has only been trained with a batch size of 256, I decided to test other powers of two to find the optimum value.

Optimizer	Batch Size	Epochs	Training Accuracy	Testing Accuracy	Runtime (Seconds)
SGD	512	10	84.03%	74.13%	159.84s
SGD	256	10	97.24%	84.71%	420.82s

Student Name: Lee T.

SGD	128	10	97.05%	85.44%	252.52s
SGD	64	10	97.99%	85.23%	284.92s
SGD	32	10	98.25%	85.59%	435.59s
Adamax	512	10	93.61%	83.41%	174.27s
Adamax	256	10	97.24%	84.71%	420.82s
Adamax	128	10	97.42%	84.91%	223.60s
Adamax	64	10	97.95%	85.34%	297.17s
Adamax	32	10	98.18%	85.33%	469.14s

Based on these results Adamax performs the best on average with different batch sizes. However, SGD outperformed all other SGD tests and Adamax tests with a batch size of 32. Which achieved an accuracy of 85.59%. Whereas the highest accuracy achieved by Adamax with a batch size of 64 was 85.34%. These two results show a trade-off of 138.42 seconds for 0.25% more accuracy.

According to the same article[6] I learned a single epoch is training a model on all examples in the training dataset, once. Epochs can be increased to repeat the number of times a model is trained on the training set of examples. This can lead to an increase in accuracy.

Optimizer	**Batch Size**	**Epochs**	**Training Accuracy**	**Testing Accuracy**	**Runtime (Seconds)**
Adamax	64	10	97.95%	85.34%	297.17s
Adamax	64	15	98.80%	85.54%	430.17s
Adamax	64	17	99.27%	85.76%	497.25s
Adamax	64	20	99.45%	85.07%	588.85s
SGD	32	10	98.25%	85.59%	435.59s
SGD	32	15	99.08%	86.44%	665.54s
SGD	32	17	99.15%	86.59%	766.68s
SGD	32	20	99.65%	87.15%	865.03s
SGD	32	100	100.00%	88.61%	4272.14s
SGD	32	50	100.00%	88.54%	2126.08s

Models with different optimizers tested with different amounts of epochs.

At first, I attempted to test an epoch value of 20. However, at the time Google Collab would crash halfway through epoch 18. I decided to test an epoch value of 17 and then 15 to measure the difference. For the Adamax optimizer, 7 more epochs increased the accuracy by 0.42%. Whereas 5 more epochs yielded a 0.20% accuracy increase. (The next day Google Collab stopped crashing I was

able to test 20 epochs). However, Adamax with 10 more epochs decreased the accuracy of the model by 0.27%. Based on this decrease in accuracy I decided to move onto the other optimizer. Stochastic gradient descent (SGD) with a previously proven effective learning rate of 0.025. I tested the model with 15, 17 and 20 epochs at first, all of which yielded an increase in accuracy. Based on these positive results I then tested 50 and 100 epochs. Which both achieved a higher accuracy. However, 50 epochs compared to 100 shows a trade-off of 2,146.06 seconds for 0.07% accuracy. When using Stochastic Gradient Descent and 100 epochs achieved the highest recorded accuracy I decided to carry on testing with this optimizer and not Adamax.

Examining Data Augmentation

From this article[7] I learned data augmentation is the process of synthesizing new data. This is a solution to the diversification of data and amount of data. Augmented data is modified existing data. When testing data augmentation, I decided to decrease batch size and epochs. This is to save time in order to collect more results.

Batch Size	Epochs	Data Augmentation	Training Accuracy	Testing Accuracy	Runtime (Seconds)
256	10	False	90.92%	81.70%	59.20s
256	10	False	91.58%	78.54%	58.67s
256	10	False	91.24%	77.58%	58.88s
256	10	False	89.69%	84.26%	58.72s
256	10	False	90.19%	84.37%	58.67s
Average Testing Accuracy: 81.29%					

Table displays results for average testing accuracy without data augmentation.

Feature wise Center	Featurewise STD Normalization	Rotation Range	Width Shift Range	Height Shift Range	Horizontal Flip	Testing Accuracy	Runtime (Seconds)
True	True	20	0.2	0.2	True	55.75%	216.21s
False	True	20	0.2	0.2	True	58.76%	212.19s
False	*False*	20	0.2	0.2	True	82.51%	201.79s
False	False	20	0.2	0.2	*False*	81.76%	224.74s
False	False	*360*	0.2	0.2	True	70.76%	203.44s

Student Name: Lee T.

False	False	*180*	0.2	0.2	True	68.28%	247.35s
False	False	*90*	0.2	0.2	True	75.71%	227.60s
False	False	*5*	0.2	0.2	True	83.91%	228.49s
False	False	*10*	0.2	0.2	True	82.46%	228.62s
False	False	*15*	0.2	0.2	True	83.05%	278.32s
False	False	5	*0.3*	*0.3*	True	82.82%	255.58s
False	False	5	*0.1*	*0.1*	True	85.20%	251.17s

Results for testing data augmentation – italic fields show difference from previous test.

The first test showed me the initial values for data augmentation are suboptimal. As with data augmentation the quality of the model has degraded. I decided to test data augmentation by modifying each value one by one. My second and third test showed me that Featurewise center and STD normalization decreases the accuracy of the model. My fourth test revealed horizontal flipping is beneficial as without it the model's quality degrades. I then conducted three tests with a higher rotation range. These tests indicated a high rotation range degrades the quality of the model. Based on this I tested lower rotation ranges (lower than provided value of 20). My tests with lower range values achieved more accuracy. As a rotation range of 5 achieved the highest accuracy, I moved onto testing whether a bigger or smaller height and width shift is achieves more or less accuracy. My last two tests show that a smaller height and width shift is beneficial as the model achieved the highest accuracy with these values.

Final Model Tests

Predefined Model	**Final Layer Units**	**Learning Rate**	**Optimizer**	**Batch Size**	**Epochs**	**Data Augmentation**	**Training Accuracy**	**Testing Accuracy**	**Runtime (Seconds)**
VGG16	4096	0.025	Stochastic Gradient Descent	32	100	True	99.60%	90.14%	3342.37s
VGG16	4096	0.025	Stochastic Gradient Descent	32	100	True	99.49%	90.88%	3310.34s
VGG16	4096	0.025	Stochastic Gradient Descent	32	100	True	99.57%	90.23%	3322.85s
VGG16	4096	0.025	Stochastic Gradient Descent	32	100	True	99.47%	90.75%	3336.94s

Student Name: Lee T.

VGG16	4096	0.025	Stochastic Gradient Descent	32	100	True	99.56%	90.34%	3518.74s
Highest Testing Accuracy: 90.88% **Average Testing Accuracy:** 90.468%									
VGG16	4096	0.025	Stochastic Gradient Descent	32	50	True	99.09%	89.70%	1696.03s

Final test results with final configuration for my Convolutional Neural Network.

Note, runtimes are much faster as I am using Google CoLab Pro. For my final results I tested 50 and 100 epochs, as the latter achieved a higher accuracy, I decided to do 5 tests to collect the average and best result.

Final Artefact

The final artefact's base model is VGG16 without the top layer but instead a single dense layer with 4096 units. The optimizer used is Stochastic Gradient Descent with a modified learning rate of 0.025. The model is trained with a batch size of 32 for 100 epochs. Data augmentation is enabled with modified variables with only a rotation range of 5, height and width shift of 0.1 and horizontal flipping enabled. From five tests of this final model the highest accuracy it achieved was 90.88% and an average accuracy of 90.468%. The final artefact configuration is also detailed in the above table. With data augmentation values in the previous table, last row.

Conclusions, Evaluation and Future Work

Summary and Evaluation

The first task description states to "evaluate a variety of well-known CNN architectures". I performed tests for each predefined model provided by keras that was compatible with the CIFAR-10 dataset. I also modified and tested the final layer units. This was in line with the first task and overall objective of training a CNN to perform as well as possible on the provided dataset. As from my range of tests I was able to determine the best predefined model to use. As well as slightly improve the accuracy of the best model for the task. I then completed the second task, "evaluate a variety of solvers", as I tested all optimizers provided by keras. From these tests I determined the two best optimizers. I also modified and tested different learning rates to improve the two best optimizers. In order to meet the overall objective of training the CNN to the best possible standard. For tasks 3 and 4 "evaluate the use of image augmentation" and "[Evaluate] General training parameters". I tested a range of different batch sizes and epochs with the two best optimizers. From which I found the optimum values and better optimizer for this dataset. I then modified and tested a range of different configurations for data augmentation to find the best configuration. I finally applied all of the best values from each different aspect, to build the final model. I believe I met the task requirements as I modified and tested a range of different variables for each task to find the best possible value. This was also in line with overall objective. As a result, the final model achieved an average accuracy of 90.468% while the highest accuracy was 90.88%.

Personal Reflection

I found this project a fun learning curve as I started by researching convolutional neural networks. Specifically, I learned the underlying structures and processes involved, such as how each different

type of layer contributes to the overall model. I learned that convolutional layers filter an input[8]. Which calculates a collection of values (feature map) indicating how strongly a feature is detected. A pooling layer reduces the spatial size of the feature map by only taking the highest values in a section from the feature map[9]. The output from the pooling layer is fed into a standard neural network (NN), dense layer(s). From which the final layer outputs a classification based on the input and weights.

From my research and testing I learned a variety of hyper parameters and their contributions to the overall model. Such as, the batch size determines how many training examples are inputted before the weights are updated. The epochs determine the number of times the model is trained on the training dataset. I learned the weights of the model are a large apart of determining the classification. I also learned the optimizer determines how the weights are updated. As well as how the learning rate affects the proportion of loss used to update the weights.

At first, I found it quite hard to plan out how I was going to test each aspect. However, over time and re-reading the specification I was able to plan out how to test each aspect. Also at first, reading and understanding the code was quite a challenge. However, through research and carefully analysing the code, over time I was able to understand and modify the code quite easily for tests.

Personally, I have strongly developed my understanding of Convolutional Neural Networks. Furthermore, I have developed my skill in fine tuning machine learning models. As from this experience I can repeat certain processes for my own side projects. Which can aid me further in developing my own portfolio for potential employers.

Conclusions

From my first set tests with different predefined models, I was able to conclude that VGG16 was the best model suited for this dataset. I also noticed that more units/neurons in a dense layer can increase the accuracy however too many neurons can also decrease the accuracy. I concluded that more neurons do not always increase accuracy.
From my second set test of different optimizers, I was able to conclude Adamax and Stochastic Gradient Descent (SGD) are best suited for this model and dataset. I also concluded that adjusting the learning rate to find the optimum value is important. As from my tests on learning rate SGD became the most accurate, which was further improved later on.
From set of tests on batch size and epochs, I noticed as the batch size decreased the accuracy and runtime increased. From these results I concluded there was a trade-off of time for accuracy when using different batch sizes. I also concluded that the greater number of epochs the more likely it is for the model to increase in accuracy.
From all of my tests I have concluded that when designing and training a machine learning model. Most if not all hyperparameters should be modified and tested to find their optimum value. This is to design the best possible model, that can achieve a high accuracy.

Suggestions for future work

I am personally interested in unsupervised learning in which a machine learning model learns by taking its own actions and learning from them. I have already read and ran a project in which a model learns to path find[10]. However, I would like to develop my skill in this area further. Such as for a side project I would like to train a model to play a simple game. I would then, like to develop a collection of similar projects to further build up my knowledge.

Student Name: Lee T.

I am also interested in Natural Language Processing (NLP). So far, I have investigated projects such as an NLP model being able to detect sarcasm, as well as bad and good reviews for movies[11]. I have also researched some of the processes involved such as tokenization, and sequencing. However, I would personally like to study and build my own NLP models, something similar to classifying text as a certain category. I find this area interesting as the overall goal of NLP has a lot of potential for real world application.

References

[1]J. Brownlee, "A Gentle Introduction to Transfer Learning for Deep Learning", Machine Learning Mastery, 2019. [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deeplearning/ [Accessed: 17- May- 2021].

[2]K. Team, "Keras documentation: Keras Applications", Keras.io. [Online]. Available: https://keras.io/api/applications/ [Accessed: 17- May- 2021].

[3]T. Rashid, Make your own neural network. 2016, p. 169, 74, 254.

[4]"Introduction to Optimizers", Algorithmia Blog, 2018. [Online]. Available: https://algorithmia.com/blog/introduction-to-optimizers [Accessed: 17- May- 2021].

[5] K. Team, "Keras documentation: Optimizers", Keras.io. [Online]. Available: https://keras.io/api/optimizers/ [Accessed: 17- May- 2021].

[6] J. Brownlee, "Difference Between a Batch and an Epoch in a Neural Network", Machine Learning Mastery, 2019. [Online]. Available: https://machinelearningmastery.com/difference-between-abatch-and-an-epoch/ [Accessed: 17- May- 2021].

[7]Great Learning Team, "What is Data Augmentation & how it works?", GreatLearning Blog: Free Resources what Matters to shape your Career!, 2020. [Online]. Available: https://www.mygreatlearning.com/blog/understanding-data-augmentation/ [Accessed: 17- May- 2021].

[8] J. Brownlee, "How Do Convolutional Layers Work in Deep Learning Neural Networks?", Machine Learning Mastery, 2020. [Online]. Available: https://machinelearningmastery.com/convolutionallayers-for-deep-learning-neural-networks/ [Accessed: 17- May- 2021].

[9] H. Pokharna, "The best explanation of Convolutional Neural Networks on the Internet!", Medium, 2021. [Online]. Available: https://medium.com/technologymadeeasy/the-best-explanation-ofconvolutional-neural-networks-on-the-internet-fbb8b1ad5df8#:~:text=A%20pooling%20layer%20is%20another,in%20pooling%20is%20max%20pooling [Accessed: 17- May- 2021].

[10]S, Raschka, Python Machine Learning. 2015, p. 977.

[11]J. Loy, Neural Network Projects with Python. 2019, p. 189.