Assignment 2

Nguyen Phuc Thai
a1846505

1. Introduction

This report aims to explore Convolutional neural network (CNN) application in image classification. CNN has been widely used in this task over the last decade, with the development of complex and deep network such as Alexnet [9], ZF net [15], VGG net [13], Googlenet [14] and Resnet [**?**]. And the application of the network has been expanded since to more complex applications, for instance, image detection or image segmentation. However, the aim of this report is restricted to exploring CNN and how to improve a CNN model, hence, the task used on CNN will also be restricted to image classification.

The dataset used for this project will be CIFAR10, which contains 50,000 training images, 10,000 testing images of ,all images of size 32x32x3, 10 different types of object [16]. This dataset is considerably smaller than the main dataset used in the mentioned CNN structures above, namely the Imagenet dataset, which contains over one million training images of size 227x227x3 with about 1000 classes [6]

The aim of the project is to explore various aspect of training and improving a CNN for image classification on the CIFAR10 dataset, , including architecture, batch normalization, padding image, optimization strategy, regularization techniques including drop out and L2 normalization and learning rate scheduling. An explanation of each of these features, how they could improve the model and what experimentation was done with them will be included in section 2 and 3 of this report, respectively. The implemented code to build the model will be included in section 4. The best developed model from the project will also be compared with a network created following a similar style of the VGG network [13]. For a quick comparison, the model inspired by VGG achieved an accuracy at about 82%, while the developed model achieved an overall accuracy of about 79%.

2. Methodology

Developing a CNN in this project will follow a step-by-step process and refining each features individually, rather than looping through multiple possible choices of features and choosing the best one. This is due to the computational power and time required to process and train each convolutional network, which make tuning multiple types of parameter at once very time-consuming. On the other hand, training a network in this manner facilitate a better understanding of how each feature affects the model's performance.

The overall approach to building a CNN for classification on CIFAR10 is first to consider whether Batch normalization and image padding should be applied in CNN, then continue with expanding the structure of the network to explore how it could improve the model. As increasing network complexity improves the validation accuracy, it also shows behaviour of overfitting, which we attempted to address via regularization methods and creating a learning rate schedule. After arriving at a good model structure, a different optimizer is tested to see if it improves performance. Then the best model following this procedure will be compared with a model built based on VGG net

2.1. Convolution, Pooling and Activation function

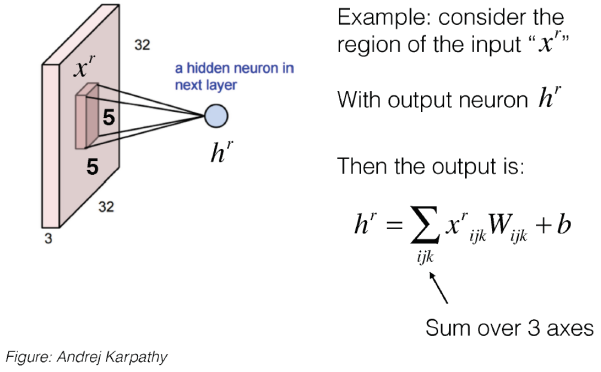

Figure 1. Convolution

Figure 1 (adapted from [1]) demonstrate the work of a Convolutional filter, in essence given a filter, the centre of it will be slid over every pixel of an image, except for pixel around the edge where if a filter is applied, part of the filter will not be on top of any pixel. At each pixel, a calculation

is done via the formula in the image, where x^r_{ijk} is the image's pixel will be multiplied by W_{ijk} which is the weight from the filter that is placed on the pixel. However, filtering the image this way means that at the next layer, width and height will be reduced compare to the previous. The issue could be alleviated by zero padding, or applying filter on the image's edge, but the weight outside of the image pixel will be multiplied by zero, or increasing image size, but with zero pixels, to preserve layer size and information. b in the image is the bias. The purpose of convoluting is to help the model to be able to enhance the signal of the small part of the original image in later layers [5]. For this project the filter used on most networks is a 3x3 filter with stride one.

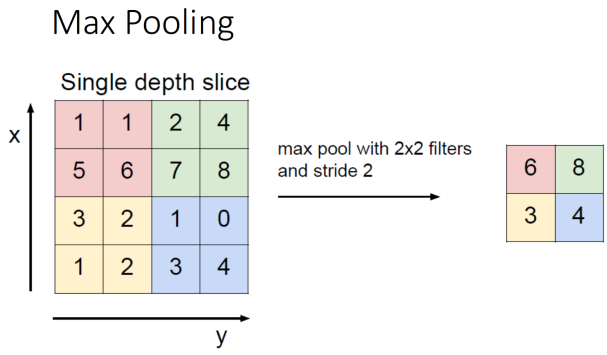

Figure 2. max pooling

The pooling layer (example in image 2 adapted from [1]), in the CNN is another important part of the network that helps the CNN compressed the information flowing through the network, hence reduce parameters and computation cost [5]. Aside from max pooling, there are also average and min pooling. For this project, 2x2 max-pooling layer with stride 2 is applied in CNN, following the steps of other networks build for the similar problems such as VGG and GoogLeNet [13], [14].

Another feature applied to all model for this project is the ReLu activation function, $f(x) = max(0, x)$ for its simplicity to compute and is among one of the activation function that can help with improving convergence speed [2]

2.2. Batch Normalization

Because of the need for multiple layer, the input distribution within a network may differs which could destabilize training process , and this problem could be addressed via Batch Normalization [12]. The idea is similar to data standardization and is done via the following algorithm from [12] for each minibatch:

$$\mu_\beta = \frac{1}{m}\sum_i^m x_i$$

$$\sigma^2_\beta = \frac{1}{m}\sum_i^m (x_i - \mu_\beta)^2$$

$$\widehat{x_i} = \frac{x_i - \mu_\beta}{\sqrt{\sigma^2_\beta + \epsilon}}$$

$$y_i = \gamma\widehat{x_i} + \beta$$

Where:

- β represents minibatch
- μ_β and σ_β are the mean and standard deviation of the minipatch, respectively.
- x_i and $\widehat{x_i}$ is the input and standardized input according to the minibatch
- ϵ is the small error to avoid division by 0
- y_i is the result of the batch normalization, and it requires an extra step because we are not normalize for each batch, but for the whole training data

In pytorch, the normalization is added via BatchNorm2d.

2.3. Network architecture

There are empirical evidence suggesting that increasing the depth of the network could improve the model. For instance, earlier works with AlexNet and ZF Net [15] uses model with 8 layers ([9], [15]) , while VGG and GoogleNet increases the layer depth to 19 and 22, respectively ([13], [14]), and continuing the trend of deeper layer works better. Although ResNet further shows further improvement experiment a network containing 152 layers, it was also shown in the development of ResNet that just adding plain layers along is not enough to improve the model [7].

For this project, networks with different number of layer will also be tested to see how they affect performance, but the depth of the network will be trimmed compared to other well-known architecture. We will also start testing with smaller network and then increase the depth of the layer to explore whether it could improve performance. The overall structure of the networks tested in this project will comprised of a series of convolutional and max pooling layers to process the input image before being put through fully connected layers to produce the output.

2.4. Training process of a network

The 50000 observations of CIFAR10 will be separated into a training and a validation set, where the training set

will have 45000 observation and the validation set contains 5000 observations. The training process is done by passing every training data through the network in minibatches, which was set to 32 for this project.

After every training is passed through and output computed (1 epoch), a average training loss and accuracy (on both train and validation set) is computed before backpropagation and the process restarts again. Throughout this project each model will be trained in 30 epochs. To assist with computation efficiency, early stopping will also be implemented, where if the validation accuracy does not improve after a certain number of epoch, the process will be stopped.

The loss function used for the the training procedure will be the cross entropy loss function, which for 1 single observation, takes the form of: $-log\left(\frac{e^{sy_i}}{\sum e^{s_j}}\right)$ where s_{y_i} is the output for the true class, and e^{s_j} is the output for all 10 classes. The losses for the whole training and testing data will be the average losses for all the observation. As for the fraction inside the log, it is converting the continuous output into probability, and the classes with the highest probability will be the output class from the model to be compare with the true class.

As for backpropagation, a simple illustration of the process is shown as follow:

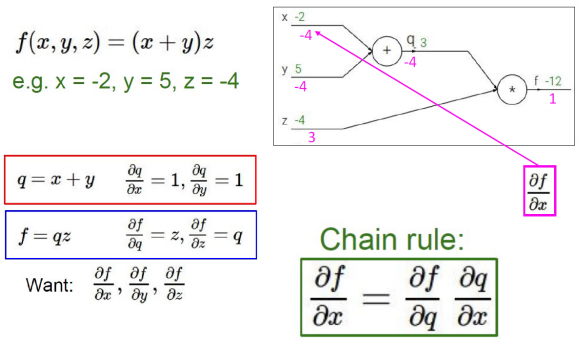

Figure 3. backpropagation example

Figure 3 (adapted from [3]) demonstrate how a backpropagation works, which for the neural networks means from a computed loss, compute the contribution of the weights from previous layers to see how they contribute to the losses, and the computation is done via differentiation. After the contribution is learned, the weights will be updated such that they can improve the loss via different optimization strategies.The CNN in this report utilize two strategies to optimize loss, one is the Stochastic Gradient Descent (SGD), and the other is a proposed improvement on gradient descent, Adam (Adaptive moment estimation), which incorpoporate the first and second moments of the gradient to update the weights [8]. Both Adam and SGD uses the same hyperparameter, learning rate, to determine the rate of updating weight.

2.5. Regularization and learning rate scheduling

To control overfitting, regularization techniques, namely lasso regularization and dropout. For the former, this involves controlling the weight updating process to manage how the weight is updated, and is controlled in pytorch via the parameter weight_decay.

As for dropout, this method involves dropping some of nodes of the network layer, which introduce noises in the training process, or in the model itself, which makes the resulting final prediction a combination of multiple different models, hence avoid one single model to overlearn on the data [11]. The regularization is controlled by the probability that each single node in the fully connected layer being dropped, with higher probability leading to greater chance of having networks that differs more. In this project, this is implemented only in the fully connected layer of the network, mimicking the use of this process in the VGG network [13],

Another aspect applied to CNN in this project is learning rate scheduling, which helps the model settles to a minimum if the process already arrived at a minimum and does not seem to improve anymore [4]. Two types of scheduler were used, one is reducing the learning rate if the loss does not decrease after a certain number of epoch, and the other is reducing the learning rate after a set number of epochs. For both methods, learning rate is reduced by a factor of 10

3. Experimental analysis

In this section, we will use validation accuracy as the main comparison factors between models. Other metrics such as validation loss, training accuracy and training loss will also be looked at but as an indicator for overfitting. All results mentioned in this section can be found in the coding implementation, which is accessible from the linked provided in section 4

3.1. Baseline approach

The baseline model for this project uses the convolutional layers from the LeNet-5 [10] which is one of the earlier approach to building a CNN. The convolutional layers contains one convolution layer of depth, followed by max pooling, one layer of depth 16 and one max pooling before going into the fully connected layer with 100 nodes, followed by the output layer. All convolution filter is of size 5x5 with no padding. During training, this model achieves validation accuracy of about 62.6%. This value is achieved with SGD optimization strategy, with 0.001 learning rate and momentum 0.09, and the learning is stopped if the validation accuracy does not improved after 6 epochs.

3.2. Batch Normalization and padding

Using the same configuration as the baseline, batch normalization is added to the convolution layer, before the activation function. It is found that, while batch normalization did not improve results significantly (best validation accuracy is about 64%), it helps the model converge to this level of accuracy quicker, which is around 10 epochs compared to around 20 epochs.

Similarly, applying padding to the baseline alone also helps the model converge after around 10 epochs, possibly due to the ability to preserve information between convolution layers, and this also achieved accuracy of about 63.8%. It should be noted that from this point onwards, the filter will now have dimensions 3x3 with stride one. Combining both padding and normalization also improves convergence speed of the model (after about 7 epochs), but not the accuracy, as the best accuracy achieved with both is 64%. This is illustrated in figure 4.

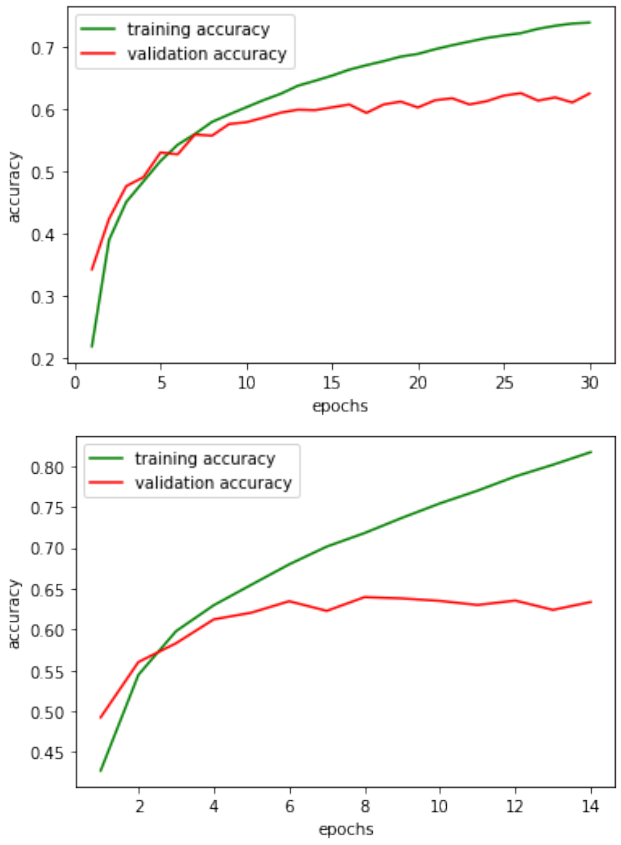

Figure 4. Baseline (top) vs baseline with batch norm and padding (bottom) accuracy learning curve

3.3. Altering Network structure

Aside from increasing the depth of the network, the depth of each convolutional layer is also increased in the developing the model. The result from these experiment in terms of validation accuracy is shown in the following (conv denotes convolution layers, and FC denotes fully connected layers

Structure	Validation Accuracy
Conv:6-16, FC: 100	64%
Conv:16-32, FC: 100	66.3%
Conv:32-64, FC: 100	69.2%
Conv:32-64-128, FC: 100	70.7%
Conv:32-64-128, FC: 300-100	73.5%

Table 1. Results from increasing depth of network

Consistent with the previous studies and experiment with CNN mentioned before, increasing the depth of the model tend to improve its predictive power, and this is also true with increasing the depth within the convolution layer as seen in the first three structure in table 1, and increasing the fully connected layer as observed in the last structure.

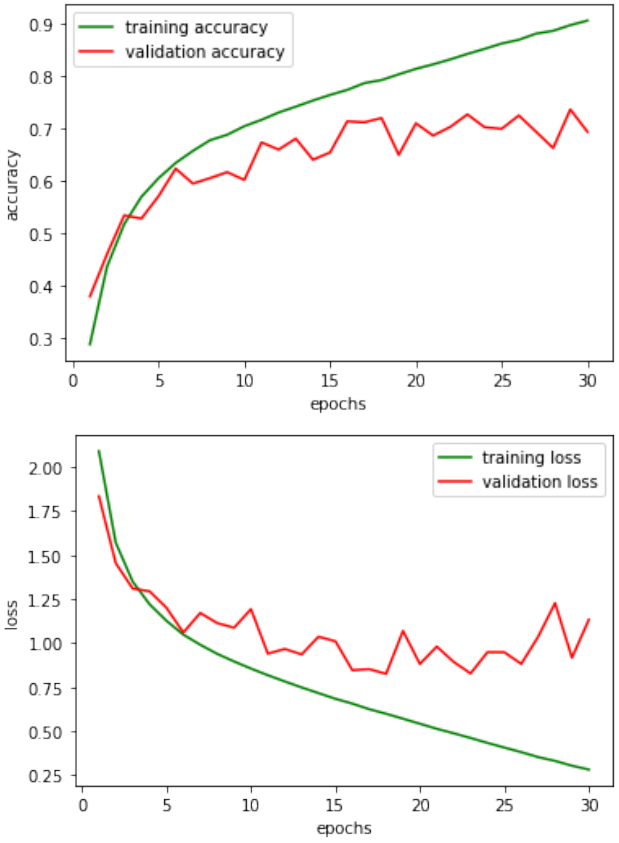

Figure 5. five-layer model learning curves

However, the performance gap between training and validation also widens as the network structure complexity increases, and this was also considerably noticeable from the model with the learning curve shown at the bottom of figure

4. This signals that the model is overfitting and to the training data and cannot replicate its performance on the training data to other unseen data. This is even more observable on the loss learning curve of the model. As shown in figure 5, towards the end of the training process, the validation loss increases, while the training loss keep on decreasing.

3.4. Altering learning rate and regularization

The last two structures from table 1 was also tested with an increased constant learning rate explore how it affects results, and validation accuracy shows an improved performance for both. On the four-layer network, best validation accuracy improves to 74.4%, and on the five-layer network, results increases to 75.5%. However, alongside providing an improved accuracy, the higher learning also makes the model overfit more to the trainning data, such as the deeper model achieved a near perfect training accuracy (99.3%) using the higher learning rate, as shown in figure 6:

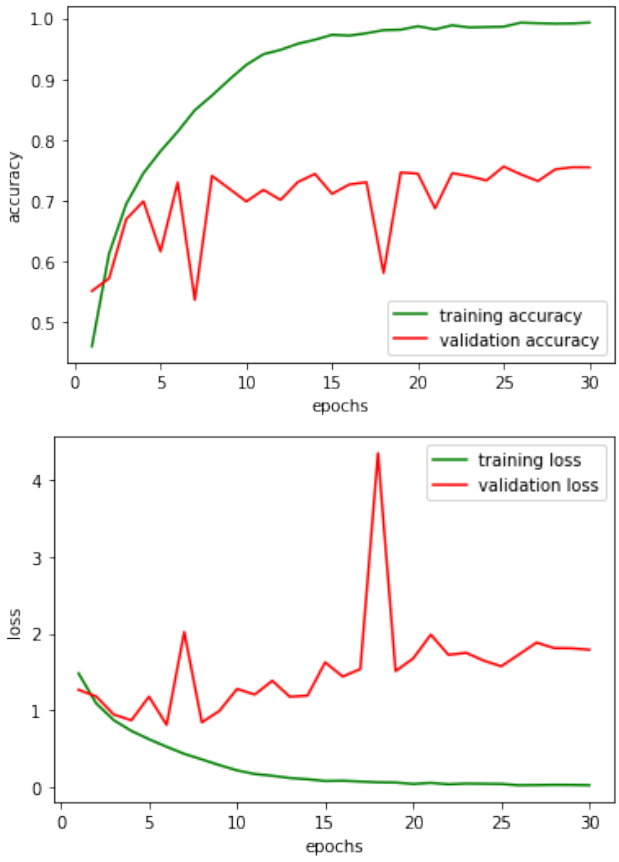

Figure 6. five-layer model learning curves with learning 0.01

Comparing to the validation accuracy curves in figure 5, the validation accuracy for the increased learning rate reaches the converging point relatively quicker, but also shows more volatility. Furthermore, on the loss curves of figure 5 and 6, validation loss learning process with the higher learning rate also start increasing within much fewer epochs, and it actually surpassed the initial validation loss.

With the models starting to overfit, while almost fitting perfectly on the training data, regularization is then applied to them to help the model generalize better. The regularizer is used only on the five-layer model to improve its performance. The first tested regularizer is the lasso, with weight decay factor set to 1e-5 for the lower learning rate, and 1e-4 for the higher learning rate. However, there are no significant changes observed on both, as the resulting learning curves for both are very similar to that in figure 5 and 6, and best validation accuracy reduces for the 0.001 learning rate (72% compared to 73.5%) and slightly improves for the 0.01 learning rate (75.8% compared to 75.5%)

To add an extra layer of regularization, dropout is then added to the network configuration, and implement during the fully connected layer, and is implemented alongside the Lasso. The dropout probability was set to 0.2. Best validation accuracy for each model again shows noo significant changes (72.5% and 76.2% for 0.001 and 0.01 learning rate, respectively), and the loss curve for the lower learning rate also shows no difference. But the ending validation for 0.01 learning rate does not surpass its value at initial epoch anymore, which signals that regularization did work, although the increasing trends at the ending epochs remained.

3.5. Learning rate scheduling

For this project, the types of scheduling implemented is reducing learning rate during training. Because of the nature of the schedule, 0.01 is the only starting point learning rate tested here as 0.001 will be incorporated in latter epochs by the implementation of scheduling. Two general rules for reducing learning rate applied was reducinig learning rate when validation loss stop decreasing (reduce on plateau), or reduce it after a fixed number of epochs have passed (reduce by step). This is done alongside with both types of regularization mentioned above, lasso with weight decay set to 1e-4 and drop out with probability 0.2. The first tested schedule is reducing on plateau, with the length of the plateau set to 5 epochs. And because of the varying learning rate, the model is also modified to only stop if no improvement is observed after 10 epochs.

Figure 7 illustrates the learning curve for the first tested scheduler, and in this run, learning rate is reduced four times. While overfitting is still apparent, the increasing trend of the validation loss was cut off towards the end thanks to decreasing learning rate. Furthermore, this model also provides an improved best validation accuracy of 77.9%. It should also be noted that this at the ending epochs, validation accuracy is much flatter comparing to earlier models without learning rate scheduling, showing the effect of the schedule to help models ”settle” to a minimum. Considering both the overfitting aspect and accu-

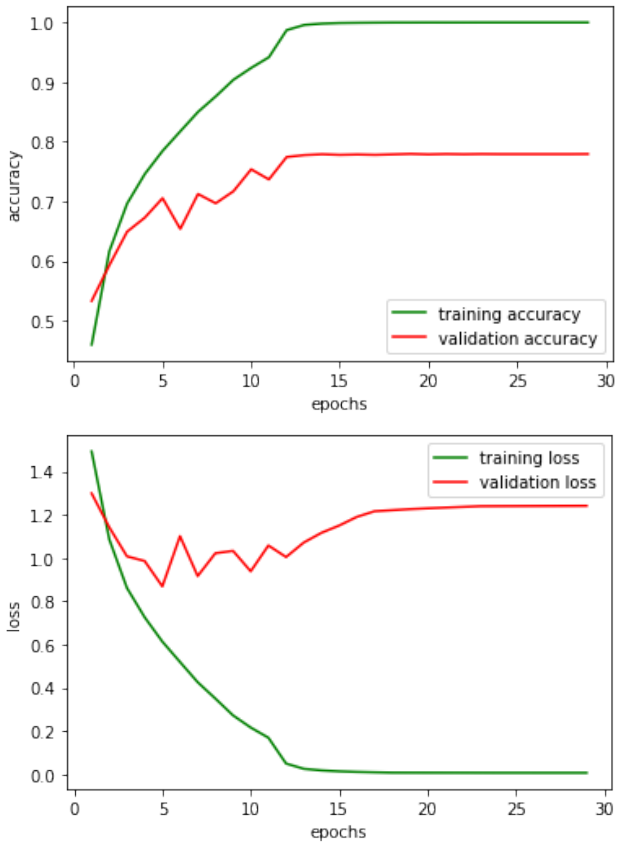

Figure 7. learning curve with Reduce on plateau

racy, this is an improvement compared to previous model. After this trial, a model using reduce on plateau with four no-improvement epochs and a model with reducing learning after seven epochs were also tested. However, they both return very similar results to this model in terms of minimizing overfitting and improving accuracy (best validation accuracy were 77.1% and 77.4% respectively).

3.6. Adam optimization strategy

The last tested model features in training feature in CNN for this project is to implement a different optimization strategy, the Adam optimizer. The first trial for Adam uses a scheduler with reduce on plateau of length five (same as the best model mentioned so far), dropout probability 0.2, but with learning 0.001 and weight decay 1e-5. This model returns the best validation accuracy of 77.7%, which is worse than the best model using SGD. However, its loss learning curve in figure 8 shows it overfit much more, with ending validation loss more than doubles ending training loss.

Due to overfitting, stronger regularization is then applied to this model. The first trial was to increase weight decay factor to 1e-4, and increase dropout probability to 0.3, learning rate scheduler was also changed to decrease after 6

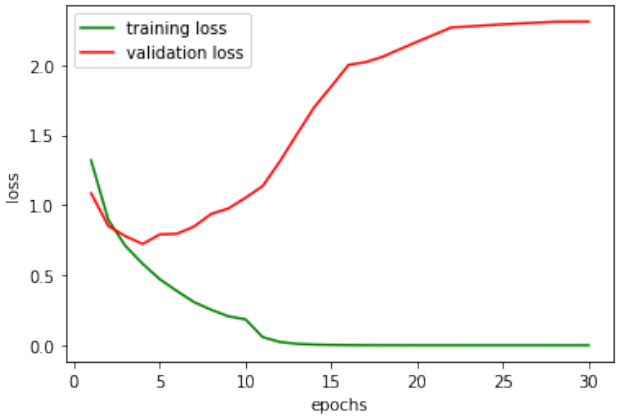

Figure 8. Loss learning curve of the first model using Adam

epochs (as in the loss curve, the validation loss increase after about 5 epochs, but the learning curve is first reduced after 10 epochs). This helps the model achieve best validation accuracy of 78.6%, yet, this model's ending validation loss is still greater than its initial loss. Hence, further changes are implemented, which are increasing the lasso strength to 1e-3, keeping dropout probability the same and switch back to reduce on plateau, but making the reduction more aggressive (after three epochs without loss reduction).

Figure 9 presents the learning process of this final model. The model's ending validation loss is not higher than its initial value as in figure 8 showing that regularization is effective, although it still overfit the training data. Looking at the accuracy curve, the resulting best validation accuracy of the model is also closer to 80% (79.6% best validation accuracy), which means this model is the best model so far developed in this project.

3.7. Comparison to a network following VGG's structure

A model that follows closely VGG's architecture is implemented in this project to compare with the best model developed. This VGG-style model will also has the structure of a stack of convolutional layer followed before pooling layer to increase the model's receptive field [13]. In particular, the stack of convolutional layer implemented will have 2 convolution layer, and a total of three stacks will be used (which is similar to the three-convolution-layer structure of the model developed, each convolution layer in each stack has depth of 32-64-128) followed by a layer of 2 fully connected layers (with 256 and 128 nodes respectively) before the output layer. The VGG-style network will also use the same specification of the best model, which includes dropout in the fully connected layer (dropout probability of 0.2), weight decay of 1e-3 and reduce on plateau with length of three epochs. It should be noted that this model is not as deep as the actual VGG, and its specifications set to mimic

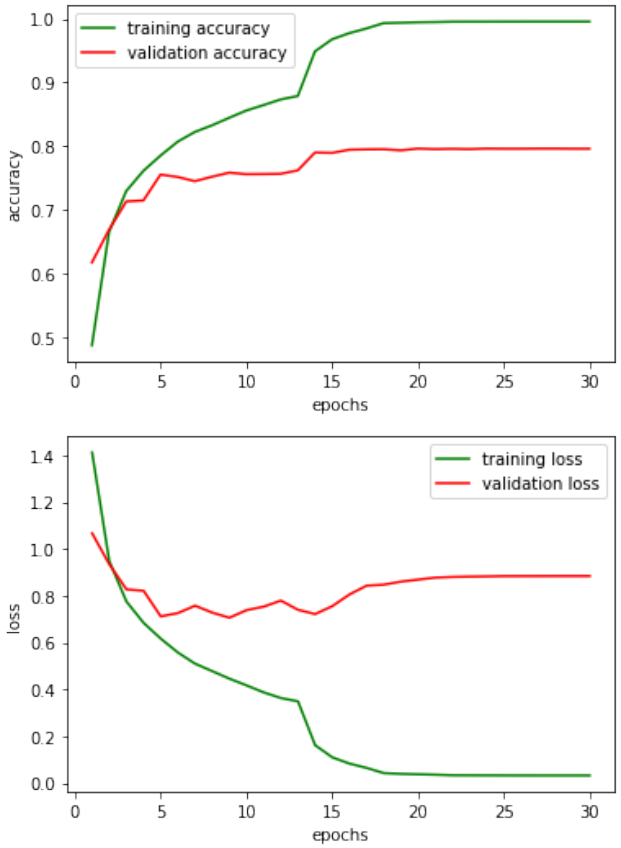

Figure 9. best model learning curve

the models in this project rather than just the original VGG.

The overall shape of this model's learning curve is similar to that of the best model, yet this VGG-style network achieved an improved validation accuracy of 82.1%, nearly 3% better than the model this project developed.

3.8. On the test dataset

The result of on the test data is presented in table 2, and similar to validation result, the VGG-style network also outperform the project's model. The results for both model does not differs to much compared to the validation results, in fact, both model achieved slightly better overall accuracy (79.85% for the project's model test accuracy compared to 79.6% validation accuracy, and 82.33% test accuracy for the VGG-style network, compared to 82.1& validation accuracy).

Interestingly, both models' performance for each individual class in the dataset also seems to have a strong correlation in terms of how well they did for each individual classes. Both did worst at recognizing cats, while did best as recognizing automobile. Furthermore, the ranking of how well the model predict for each individual classes are also largely similar, showing the similarities in their classification pattern.

Class	Project's model (ranking)	VGG-style model (ranking)
Airplane	84% (5)	86.2% (5)
Automobile	88.6% (1)	92% (1)
Bird	71% (8)	73.5% (9)
Cat	64.2% (10)	67.3% (10)
Deer	78.5% (7)	79.6% (7)
Dog	70.9% (9)	74.4% (8)
Frog	84.4% (4)	86.5% (4)
Horse	83.2% (6)	85% (6)
Ship	88.2% (2)	89% (3)
Truck	85.5% (3)	89.8% (2)
Overall	79.85%	82.33%

Table 2. Performance on the test set

4. Implementation code

Codes for this project can be accessed from `https://github.com/phucthai67/COMP-SCI-7138-assignment-2.git`

5. Conclusion, Limitation and Future testing

This report descrbes the process of designing and experimenting different aspect of trainnig a Convolutional Neural Network for image classification. Some major experimentation implemented in this project include testing effect of image padding to preserve information passing through the network, batch normalization efffect, network architecture and depth effect on model performance, regularizations method including dropout and lasso and testing whether adam optimizer improves the model compared to SGD. The project also explore and compare the model developed with a network with architecture inspired by VGG, a well-known CNN for image classification.

Further testing for CNN be implemented for future project from the results obtained in this one. This can include further exploring the Adam optimizer strategy, such as altering learning rate rather than using pytorch's default 0.001, or testing the effect on the performance of activation function that are developed from ReLu (such as ELu or Leaky ReLu). More testing and alteration of the VGG-inspired network could also be studied, as only one VGG network were tested in this project or increasing regularization. Or more aggressive learning rate scheduler (reducing learning rate more frequently) to eliminate overfitting problem as the issue was lessened but not totally eliminated in this project. Should a more powerful computational power be available, a deeper network closer to that of VGG or other types of network could also be tested to see if it can

further improve performance, or architecture and design of the Resnet could be explore which involves skip connections between layer within a very deep network [7]

References

[1] Dick, A & Shi, J 2022, Convolutional Neural Network, Part 2' lecture notes distributed in COMP SCI 7318 University of Adelaide, on 19 September 2022.

[2] Dick, A & Shi, J 2022, Training deep Neural Network, Part 3' lecture notes distributed in COMP SCI 7318 University of Adelaide, on 19 September 2022.

[3] Dick, A & Shi, J 2022, Training deep Neural Network, Part 2' lecture notes distributed in COMP SCI 7318 University of Adelaide, on 19 September 2022.

[4] Géron, A 2019,'Training models' in *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, 2nd edn, O'Reilly Media, Incorporated, Sevastopol

[5] Géron, A 2019,'Deep computer vision using convolutional neural network' in *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, 2nd edn, O'Reilly Media, Incorporated, Sevastopol

[6] Image-Net, n.d *Download ImageNet Data*, viewed 20 October 2022, `https://www.image-net.org/download.php`

[7] Kaiming, H, Xiangyu, Z, Shaoqing, R & Jian, S, 2016, 'Deep Residual Learning for Image Recognition', in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, pp. 770–778.

[8] Kingma, DP & Ba, J 2014, 'Adam: A Method for Stochastic Optimization'.

[9] Krizhevsky, A., Sutskever, I., & Hinton, G,E, 2012 "ImageNet classification with deep convolutional neural networks" NIPS, pp. 1106–1114.

[10] Lecun, Y, Bottou, L, Bengio, Y & Haffner, P 1998, 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324.

[11] Nitish, S, Geoffrey, H, Alex K, Ilya S & Ruslan, S 2014 "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958.

[12] Sergey, I & Christian, S, 2015, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *Proceedings of the 32nd International Conference on Machine Learning*, pp. 448–456.

[13] Simonyan, K & Zisserman, A 2014, 'Very Deep Convolutional Networks for Large-Scale Image Recognition' Cornell University, DOI: 10.48550/ARXIV.1409.1556.

[14] Szegedy, C, Liu, W, Jia, Y, Sermanet, P, Reed, S, Anguelov, D, Erhan, D, Vanhoucke, V, Rabinovich, A, 2014 "Going Deeper with Convolutions", Cornell University, DOI: 10.48550/ARXIV.1409.4842

[15] Zeiler, M,D & Fergus, R 2014, 'Visualizing and Understanding Convolutional Networks', in Computer Vision – ECCV 2014, vol. 8689, Springer International Publishing, Cham, no. 1, pp. 818–833

[16] The University of Toronto, n.d *CIFAR10 and CIFAR100 datasets*, The University of Toronto, viewed 20 October 2022, `https://www.cs.toronto.edu/~kriz/cifar.html`