THE UNIVERSITY OF HONG KONG

STAT4609 BIG DATA ANALYTICS

Image Classification on CIFAR-10

Tao Yufeng, Cai Yuxi, Gan Dailin, Huang Yifeng

May 3, 2021

1. Introduction

CIFAR-10 is a classical dataset for image classification. It comprises 50000 training images and 10000 testing images in ten different classes. Each image has a size of 32 by 32 pixels with three colour channels. Convolutional Neural Networks (CNN) are said to be the most suitable device to classify images , because image data consist of grid-structured input with strong spatial dependence [1]. In this project, we aim to compare different CNN models in the task of classifying CIFAR-10 images. Four well-acknowledged CNN models, namely AlexNet [Clairvoyance 2], VGG [Clairvoyance 3], ResNet [Clairvoyance 4], and DenseNet [Clairvoyance 5] are implemented, and the simplest Neural Network modelof Perceptron [Clairvoyance 1] is set as the basis of comparison to evaluate the effects of CNN. These four kinds of CNN models have their own features and advantages. Generally, models with smaller filter sizes and deeper networks would lead to better results. Hence, we plan to study their merits and drawbacks based on empirical comparison.Also, there are more than one architectural settings for VGG, ResNet, and DenseNet. We selected four different architectures for VGG and Densenet and three for ResNet to conduct an internal comparison. In total, thirteen models will be compared in our experiment. Another objective of our research project is to study the impact of optimization strategies on model training. We will compare the training results of using non-adaptive or adaptive optimizer - Stochastic Gradient Descent (SGD) and Adam - and of whether early stopping is used. Based on our experiment results by far, the optimization setting is of significant importance to the performance of CNN models. There may not be an optimal optimization setting for all models, but we would like to have a basic evaluation of these optimization devices. Thus, this group project is a comparative experiment for a better understanding of the structure and training of CNN models.

2. Models

This section introduces the background information and technical details of the models we are fitting.

2.1. Clairvoyance-1

Similar to a simple perceptron, the Clairvoyance 1 model serves as a base case for more complex models to compare with and a starting point towards multi-layer neural networks. It is composed of a single fully connected layer and a sigmoid activation function.

2.2. Clairvoyance-2

The Clairvoyance-2 model adopts the idea of AlexNet, which is the outperformer in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012 [1]. The original AlexNet architecture utilizes two parallel pipelines of processing and thus partitions the network across two GPUs. Due to the restrictions of the computing resources, the Clairvoyance-2 model only has one single processing pipeline. Nevertheless, our design of layers and the choices of hyperparameters are based on those of AlexNet with some small modifications [2]. The architecture in sequence is as follows:

1. A 2D Convolution layer (in_channels = 3, out_channels = 96, kernel_size = 11, stride = 4, pad = 4) followed by a batch normalization layer, a rectified linear unit (ReLU), and a 2D max pooling (kernel_size = 3, stride = 2).
2. A 2D Convolution layer (in_channels = 96, out_channels = 256, kernel_size = 5, stride = 1, pad = 2) followed by a batch normalization layer, a rectified linear unit (ReLU), and a 2D max pooling (kernel_size = 3, stride = 2).
3. A 2D Convolution layer (in_channels = 256, out_channels = 384, kernel_size = 3, stride = 1, pad = 1) followed by a batch normalization layer and a rectified linear unit (ReLU).
4. A 2D Convolution layer (in_channels = 384, out_channels = 384, kernel_size = 3, stride = 1, pad = 1) followed by a batch normalization layer and a rectified linear unit (ReLU).
5. A 2D Convolution layer (in_channels = 384, out_channels = 256, kernel_size = 3, stride = 1, pad = 1) followed by a batch normalization layer, a rectified linear unit (ReLU), and a 2D max pooling (kernel_size = 3, stride = 2).
6. A dropout layer (rate = 0.5) followed by a fully connected layer with 4096 neurons and a rectified linear unit (ReLU).
7. A dropout layer (rate = 0.5) followed by a fully connected layer with 4096 neurons and a rectified linear unit (ReLU).
8. A final fully connected layer, the number of out features is equal to the number of classification classes.

2.3. Clairvoyance-3

Leveraging the design of VGG [3], the Clairvoyance-3 model reduces the filter size and increases the depth of the network. With small 3×3 filters and 16-19 weight layers, it retains its simplicity by taking only pooling layers and fully connected layers as the other components. The following image 1 by Davi Frossard [4] shows the macrostructure of VGG16, which the Clairvoyance-3 model resembles closely. Similar architectures of different depths are tried out and presented in figure 2. Specifically, we vary the number of stacking layers as Simonyan et al. [3] proposed to compare the performance.

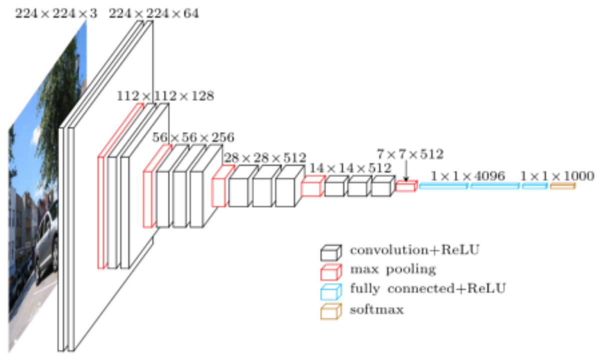
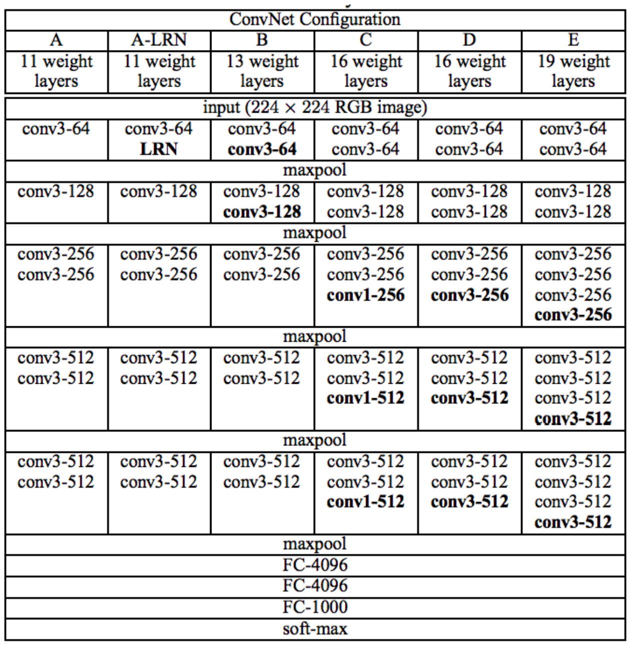


Figure 1. Macrostructure of VGG16 by Davi Frossard



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2. Experiment Architecture of VGGNet (Simonyan et al., 2014)

2.4. Clairvoyance-4

Our Clairvoyance-4 model borrows the architecture of ResNet proposed by He et al (2015) [5]. It utilizes skip connections and enables the layer to learn residual functions with reference to the layer inputs. There are two types of building blocks, the basic block for models with fewer layers and the bottleneck block for the deeper models. A basic block consists of two convolutional layers with 3×3 kernel size and same number of filters. The inputs of the first layer are added to the outputs of the second layer via the identity mapping. In contrast, the bottleneck block is composed of a stack of 3 layers, which are 1×1, 3×3, and 1×1 convolutions. The aim of using the 1×1 convolution layers is to reduce and restore the dimensions, enabling the 3×3 layer to perform with smaller input and output dimensions. The identity shortcut is also applied in the bottleneck block design, where the inputs of the first 1×1 layer are added to the outputs of the last 1×1 layer by mapping before ReLU activation. The skip connection technique is proved to solve the degradation problem effectively, making the architecture to enjoy accuracy gains by largely increasing the depth. Besides, the identity shortcut connections would not add additional parameters, leaving the model with the same computational complexity. The model design in our project is shown as follows:

- 18-layer model:
  - CONV layer (3×3, nfilters = 64, stride=1, padding=1)
  - Max Pool (3×3, stride=2, padding=1)
  - Basic Block (3×3, nfilters=64, padding=1) ×2
  - Basic Block (3×3, nfilters=128, padding=1) ×2
  - Basic Block (3×3, nfilters=256, padding=1) ×2
  - Basic Block (3×3, nfilters=512, padding=1) ×2
  - Average Pool
  - Fully connected layer with the dimension of outputs equal to the number of classes

The architecture of the 34-layer model is similar to that of the 18-layer model, with only the number of basic blocks changed. It consists of 3 basic blocks with size 64, 4 with size 128, 6 with size 256, and 3 with size 512.

- 50-layer model:
  - CONV layer (3×3, nfilters = 64, stride=1, padding=1)
  - Max Pool (3×3, stride=2, padding=1)
  - Bottleneck Block ([1×1, 64], [3×3, 64], [1×1, 256]) ×3
  - Bottleneck Block ([1×1, 128], [3×3, 128], [1×1, 512]) ×4
  - Bottleneck Block ([1×1, 256], [3×3, 256], [1×1, 1024]) ×6
  - Bottleneck Block ([1×1, 512], [3×3, 512], [1×1, 2048]) ×3
  - Average Pool
  - Fully connected layer with the dimension of outputs equal to the number of classes

2.5. Clairvoyance-5

Our Clairvoyance-5 model adopts the architecture of DenseNet proposed by Huang et al (2017) [6]. Based on previous observations on other convolutional networks, like ResNet [5], shorter connections between layers close to the input and output could provide deeper and more accurate model structures. DenseNet connects each layer to every other layer in a feed-forward fashion. The structures of DenseNet contain dense blocks with multiple dense layers, transition layers and fully connected layers. Taking DenseNet121 as an example, it has 4 dense blocks with each block containing 6, 12, 24, 16 layers respectively. A transition layer is put between each block and hence there are 3 transition layers in DenseNet121. For a typical layer in a dense block, it contains two convolutional layers with kernel sizes 1 and 3 respectively. The brief architecture of DenseNet121 in sequence is as follows:

- Convolutional layer with kernel size 7×7 and stride size 2
- Max pooling layer with kernel size 3×3 and stride size 2
- Dense Block 1 with 6 layers described above
- Transition Layer1 with one convolutional layer with kernel size 1×1, and an average pooling layer with kernel size 2×2
- Dense Block 2 with 12 layers described above
- Transition Layer2 with one convolutional layer with kernel size 1×1, and an average pooling layer with kernel size 2×2
- Dense Block 3 with 24 layers described above
- Transition Layer3 with one convolutional layer with kernel size 1×1, and an average pooling layer with kernel size 2×2
- Dense Block 4 with 24 layers described above
- Classification layer with a 7×7 average pooling layer and one fully connected layer with softmax.

Next, we further compared three models from DenseNet. DenseNet121, DenseNet169 and DenseNet201 share the same basic structure with one convolutional layer at the very beginning, four dense blocks with three transition layers and a fully connected layer. The differences are that the three models have different numbers of dense layers in each dense block. DenseNet121 has 6, 12, 24, 16 dense layers for dense blocks 1, 2, 3, and 4 respectively. DenseNet169 has 6, 12, 36, 24 dense layers and DenseNet201 has 6, 12, 48, 32 dense layers. Therefore, the three methods have 121, 169 and 201 layers respectively.

3. Data Preprocessing and Optimization

Preprocessing steps include data transformation and data loading. For data transformation, we first resize the input data to be $3\times224\times224$ (channel$\times$height$\times$width) to accommodate the AlexNet architecture for Model Clairvoyance-2 [7]. We then perform data normalization using the mean and standard deviation of the corresponding dataset. That is, we calculate the means and standard deviations of the training set and the test set of CIFAR-10 respectively before normalization. In data loading, we randomly divide the CIFAR-10 training dataset of size $50k$ into a training set of size $44k$ and a validation set of $6k$. The training batch size is set to be 32, and the validation and testing batch sizes are both 100. A preview of a batch of input training data is to ensure the proceeding of following steps.

For hyperparameter selection, it is necessary to introduce our efforts in optimization. At the early stage of our group project, we did not plan to compare the effect of different optimizers, simply trying to select an optimizer that would lead to a desirable result. Based on the training results of some intermediate experiments, we changed the optimizer from the Adam optimizer to SGD algorithm and replaced the scheduler StepLR with ReduceLROnPlateau; the latter will adjust the learning rate according to the training loss per epoch. The training results for Model Clairvoyance-1 and Model Clairvoyance-2 are then both significantly improved from around 10% to over 50%. However, we later found that choice of schedulers and the value of hyperparameters will affect the performance of these two optimizers in an uncertain way. Also, according to the paper titled The Marginal Value of Adaptive Gradient Methods in Machine Learning, adaptive methods like Adam perform worse than non-adaptive methods in general [8]. Third, the initialization of the learning rate and the decay scheme is significant to the performance of the Adam optimizer. This heavily relies on manual setting. Hence, using Adam is highly demanding in terms of experience and knowledge, although the adjustment of learning rate is automatic. Thus, based on the above three reasons, we decide to compare Adam and SGD with the same hyperparameter setting (the initial learning rate = 0.001, momentum = 0.9, and the weight decay rate = 0.0005). As for the scheduler, we found that adjusting the learning rate every epoch might not work better, so we opt for ReduceLROnPlateau that changes the learning rate only if it is necessary.

To optimize the training result, we also implement early stopping to avoid overfitting in the training process, which can increase training efficiency as well. To evaluate the effect of early stopping, the tool is used in the training of Model Clairvoyance 1 and 2, in which the computation load is not very heavy, and the number of epochs is set as 100. Due to the limited computation power, we only run 10 epochs in the training of the remaining eleven models.

Our primary objective here is to compare the efficiency of Adam and SGD. That is, in a 10-epoch experiment, we are interested in which optimizer will achieve a better training result.

4. Experimental Setting

Since we have several models and optimizers with different hyperparameters, multiple experiments are conducted for two main purposes: 1. searching for the best combination as our final model; 2. comparing the efficiency of different optimizers for a given model. The efficiency here means the training results after a given number of epochs: the lower the average loss and the higher the accuracy & F1 score on the test set, the more efficient the optimizer. To increase comparability, we fix the hyperparameters for the two optimizers as follows:

- Adam with lr=0.001, wd = 0.0005
- Momentum based SGD with lr=0.001, wd = 0.0005, momentum = 0.9

Each model is trained by these two optimizers for 10 epochs, except Clairvoyance-1 and Clairvoyance-2. Because of their relatively simple architectures, our computational resources can afford to run them for 100 epochs. To reduce unnecessary running time, however, early stopping with patience = 7 is imposed on these two models. The final results are presented in the next section.

5. Results

In this section, we provide the results in two different perspectives. Firstly, we compare the effects of early stopping with 100 epochs in two simple models, C-1-Perceptron and C-2-AlexNet with two different optimizers, Adam and SGD. Due to limited computational resources provided by Google Colab, we cannot finish 100 epochs on C-3-VGG, C-4-ResNet, and C-5-DenseNet, which we treat as a limitation. Secondly, we compare the effects of different structures of the same model. For instance, with a fixed optimizer, we compare the performance of C-3-VGG11, C-3-VGG13, C-3-VGG16, and C-3-VGG19.

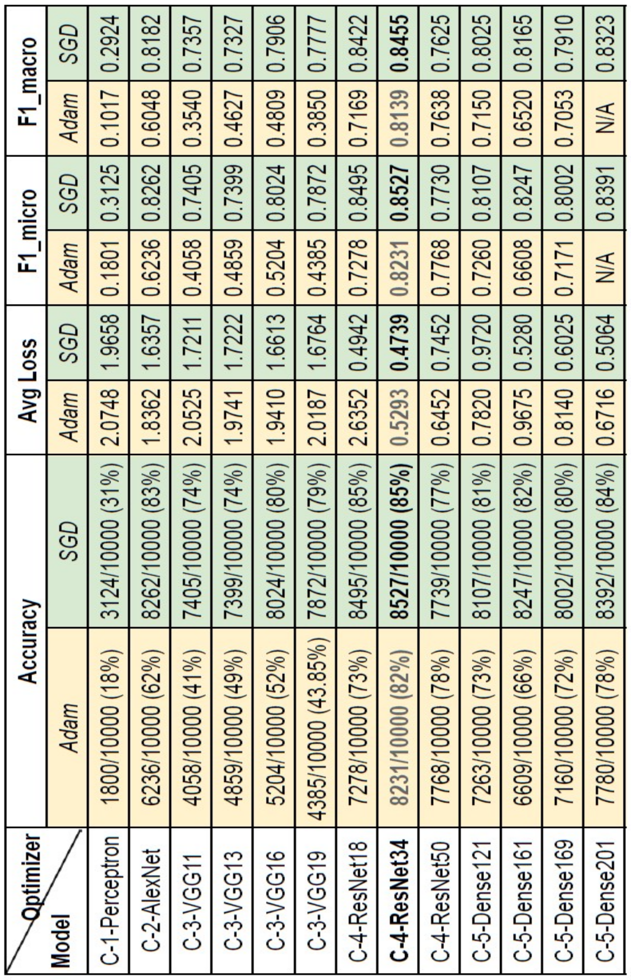
For the first part, we compare C-1-Perceptron and C-2-AlexNet with 100 epochs and early stopping. C-1-Perceptron with Adam optimizer stops with 17 epochs while C-1-Perceptron with SGD optimizer stops with 100 epochs. C-2-AlexNet with Adam optimizer stops with 30 epochs while C-2-AlexNet with SGD optimizer stops with 42 epochs. C-2-AlexNet may converge faster than C-1-Perceptron based on the stopping epochs. Overall speaking, C-2-AlexNet has smaller average loss in test datasets, higher accuracy, larger F1 score micro and F1 score macro in both optimizer settings than C-1-Perceptron. The results are expected since C-1-Perceptron is too simple to integrate complicated image information and to capture local structures. Considering the effects of optimizers, we notice that models with optimizer SGD always provide smaller average loss, higher accuracy, larger F1 score micro and F1 score macro. Therefore, the optimizer SGD may be a better choice for both simple models.

Next we consider the effects of different structures of the same with different optimizer settings. For C-3-VGG, we consider four different structures which are C-3-VGG11, C-3-VGG13, C-3-VGG16 and C-3-VGG19, with the depth increasing accordingly. For the effects of optimizers, models with optimizer SGD always provide smaller average loss on test datasets, higher accuracy, and larger F1 score micro and F1 score macro. Therefore, the optimizer SCG may be a better choice for VGG. Then, we consider the depth of C-3-VGG. As the depth increases from 11 to 16, and then to 19 under both optimizer settings, the average loss first decreases and then increases; the accuracy score, F1 score micro, and F1 score macro first increases and then experiences a drop at depth 19. Therefore, the best results are achieved at the depth 16, which is coherent to the results described in the original paper. Therefore, in C-3-VGG models, the recommended setting is at depth 16 and deeper depth may be harmful to the performance.

For C-4-ResNet, we consider three structures which are C-4-ResNet18, C-4-ResNet34, and C-4-ResNet50. We actually tried to implement C-4-ResNet101 and C-4-ResNet152, but in vain due to limited memory provided by Colab. Therefore, we only include above three C-4-ResNet models. For the effects of optimizers, models with optimizer SGD always provide smaller average loss on test datasets, higher accuracy, and larger F1 score micro and F1 score macro. For the effect of depth, as the depth increases from 18 to 50, the average loss decreases, and the accuracy score, F1 score micro, and F1 score macro increases. It is noteworthy that the increment becomes smaller as the depth increases using SGD optimizer. Hence, a good result may be generated with a relatively shallow setting with SGD optimizer for C-4-RenNet model.

For C-5-DenseNet, we consider four structures which are C-5-DenseNet121, C-5-DenseNet161, C-5-DenseNet169, C-5-DenseNet201 with depth increasing accordingly. For the effects of optimizers, models C-5-DenseNet161, C-5-DenseNet169, and C-5-DenseNet20 with optimizer SGD always provide smaller average loss on test datasets, higher accuracy, and larger F1 score micro and F1 score macro. For C-5-DenseNet121, optimizer Adam provides smaller average loss than optimizer SGD, but for the other measurements, optimizer SGD shows better results. As the depth increases from 161 to 201, the average loss decreases, and the accuracy score, F1 score micro, and F1 score macro increases. But the decrement and increment become smaller as the depth increases. One abnormal situation is that as the depth increases from 121 to 161 with optimizer Adam, the loss increases. Based on above findings, for C-5-DenseNet, the recommended settings may be using optimizer SGD with depth 201 if computational resources allow.

The predictive test set performance by different models is summarized as follows 3. Please refer to the notebook for training visualization and class-wise accuracy.



Optimizer / Model	Accuracy		Avg Loss		F1_micro		F1_macro	
	Adam	SGD	Adam	SGD	Adam	SGD	Adam	SGD
C-1-Perceptron	1800/10000 (18%)	3124/10000 (31%)	2.0748	1.9658	0.1801	0.3125	0.1017	0.2924
C-2-AlexNet	6236/10000 (62%)	8262/10000 (83%)	1.8362	1.6357	0.6236	0.8262	0.6048	0.8182
C-3-VGG11	4058/10000 (41%)	7405/10000 (74%)	2.0525	1.7211	0.4058	0.7405	0.3540	0.7357
C-3-VGG13	4859/10000 (49%)	7399/10000 (74%)	1.9741	1.7222	0.4859	0.7399	0.4627	0.7327
C-3-VGG16	5204/10000 (52%)	8024/10000 (80%)	1.9410	1.6613	0.5204	0.8024	0.4809	0.7906
C-3-VGG19	4385/10000 (43.85%)	7872/10000 (79%)	2.0187	1.6764	0.4385	0.7872	0.3850	0.7777
C-4-ResNet18	7278/10000 (73%)	8495/10000 (85%)	2.6352	0.4942	0.7278	0.8495	0.7169	0.8422
C-4-ResNet34	8231/10000 (82%)	**8527/10000 (85%)**	0.5293	**0.4739**	0.8231	**0.8527**	0.8139	**0.8455**
C-4-ResNet50	7768/10000 (78%)	7739/10000 (77%)	0.6452	0.7452	0.7768	0.7730	0.7638	0.7625
C-5-Dense121	7263/10000 (73%)	8107/10000 (81%)	0.7820	0.9720	0.7260	0.8107	0.7150	0.8025
C-5-Dense161	6609/10000 (66%)	8247/10000 (82%)	0.9675	0.5280	0.6608	0.8247	0.6520	0.8165
C-5-Dense169	7160/10000 (72%)	8002/10000 (80%)	0.8140	0.6025	0.7171	0.8002	0.7053	0.7910
C-5-Dense201	7780/10000 (78%)	8392/10000 (84%)	0.6716	0.5064	N/A	0.8391	N/A	0.8323

Figure 3. Model Performance Summary

6. Conclusion

In this project, we use the CIFAR-10 dataset as a testing ground for models with different complexities. Recall that we have built five types of models, which are C1 (simple perceptron), C2 (AlexNet-like), C3 (VGG-like), C4 (ResNet-like), and C5 (DenseNet-like). We experimented with these models starting from the simplest one, and encountered low accuracies at the initial stage. Trained with few epochs, the simple models such as C1 and C2 simply categorized most of the test images to the same class. This motivated us to build more sophisticated models, and apply a better optimizer. Besides, we have also investigated how to decide the training epoch by employing the early stopping technique. Eventually, we discovered that C-4-ResNet34 with SGD yields the best performance, with up to 85% when tested on the CIFAR-10 dataset. The following paragraphs give a short summary of our findings.

When the number of training epochs and the optimizer used are fixed, the models' performances generally improve as the model's complexity and depth increase from C1 to C5, which is consistent with our expectations. The performance of C1 is significantly poorer than those of other models. It indicates that a naive model like the simple perceptron is incapable of handling the image classification task because a single layer of hidden neurons fails to learn sophisticated spatial features from images. C4 and C5 have comparable performances: the accuracy of them is up to around 85% and the two models are superior to C2 and C3. Therefore, we conclude that the ResNet-like and the DenseNet-like models are the optimal candidates for the image classification in this project. Further, we have compared C4 and C5, and found that they share some common properties. Both architectures are considerably deep, empowering the high-level feature extraction. They both employ connections between layers to alleviate the degradation problem and make models easier to optimize. Besides, different from the VGG, ResNet and DenseNet tend to use small kernel sizes for the convolutional layers. This helps reduce memory usage and accelerate computation.

It is noteworthy that, within each type of model, the performance of the architecture does not necessarily improve as the number of layers increases. For the model C3, the 16-layer architecture has made more accurate predictions than the 19-layer one. The degradation problem may account for the phenomenon. For C4 and C5 where the connections are employed, we can observe the trend that the model can gain additional accuracies by growing deeper.

Apart from the design of model architecture, the settings for the training process also play an unignorable role in determining the model's performance on the actual classification task. In terms of the selection of the optimizer, our experiments echoed the conclusion drawn by the previous paper The Marginal Value of Adaptive Gradient Methods in Machine Learning [8]. We have compared the Adam and the momentum stochastic gradient descent optimization algorithms under similar hyperparameter settings. The SGD optimizer is constantly more efficient than the other one, when applied to train our five models within the fixed number of epochs. According to the paper, we may attribute the worse performance of the Adam optimizer to its adaptive nature and its high sensitivity to the hyperparameter choices. As for the decision on the number of training epochs, we have found that it is optimal to choose the value by the early stopping technique automatically if there are no limitations for computation resources. For example, in the initial stage where we trained the model with 5 epochs, the C2 model only yielded an accuracy of 10%, which was no better than a random guess. However, after applying the early stopping algorithm, we discovered that C2's performance was significantly boosted.

In conclusion, through this project, we have learnt that there are several key factors that determine whether a model can obtain a satisfactory result in the image classification task. We have to carefully design the architecture so that the model can capture high-level features and provide a suitable representation for the image. We shall also keep in mind that the training settings, such as the selection of the optimizer and the number of training epochs, could also impact the model's performance.

7. Limitation

We consider the following possible limitations in our implementations. Firstly, due to limited computational resources, we only consider 10 epochs for C-3-VGG, C-4-ResNet and C-5-DenseNet without early stopping. Therefore, it would be hard to conclude that those models converge with limited epochs, and our comparison may be biased since some models may require longer time to provide convergent results. Secondly, although we compare two different optimizers, we only use default parameters and we do not provide parameter searching in our projects. Some models may be sensitive to the setting of parameters and hence cannot perform well. Besides, we do not provide the results of C-4-ResNet101 and C-4-ResNet152 due to limited memory capacity, and hence we cannot provide a comprehensive comparison for C-4-ResNet models.

As for further approaches to improve model performance or interpretability, ablation studies can be conducted, in which model structures are adjusted and more detailed settings could be compared. Besides, visualization of hidden layers can help to explore whether the convolutional neural networks capture the distinguishable characteristics when performing classification tasks, and in turn to help us adjust the architecture design.

References

[1] Aggarwal, C. C. (2019). *Neural networks and deep learning: A textbook*. New York: Springer. 1

[2] PyTorch. (n.d.). Retrieved March 31, 2021, from https://pytorch.org/hub/pytorch_vision_alexnet/ 1

[3] Simonyan, K., Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. 1, 2

[4] Frossard, D. (2016, June 17). VGG in TensorFlow. Retrieved April 06, 2021, from https://www.cs.toronto.edu/~frossard/post/vgg16/ 1

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2016.90 2, 3

[6] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708). 3

[7] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 1097-1105. 3

[8] Wilson, A., Roelofs, R., Stern, M., Srebro, N., & Recht, B. (2018). The marginal value of adaptive gradient methods in machine learning. *arXiv preprint arXiv:1705.08292*. 3, 5