# Experiment and Comparison on Traditional CNN model

Yang Li Yingjia Gu

## Abstract

*In this paper, we develop a new model designed to do image classification and explore the effect of tuning the hyper-parameters and adopting different network structures. We build our own classifier **OurNet** that is able to correctly categorize the images into their respective classes. We also explore the different results by changing the number of layers, different optimization methods (**SGD** and **Adam**), and different pooling functions (**AvgPool** and **MaxPool**). We compare our model with the traditional model **AlexNet** and **ResNet 18** to see how different structures have an impact on the ultimate outcome.*

## 1. Introduction

### 1.1 Overview

This project is investigating CNN structures and optimization strategies to see how each influences the image classifier. There are many similar works from different people. From the piazza post of TA, we found Kuang Liu's project on CIFAR- 10. We look at the implementation of his project and the results he gets for brainstorming our projects. (Kuang Liu, [8])

### 1.2 Data Set

We are using the dataset introduced in HW5, which is CIFAR-10 ("The CIFAR-10 Dataset" [1]). This dataset is very popular since it is not too big and somewhat challenging. The dataset consists of 60000 images with 10 classes. Each has a format of 32 x 32 x 3. The last 3, corresponds to RGB channels.

### 1.3 Classes

The 10 classes are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

The classes are designed to be completely mutually exclusive: for example, neither automobile nor truck contains images of pickup trucks.

There are also big variations within the same class, for example, the bird class has different types of birds (both big and small) and the angels of the photos taken are totally different so that there might be the case that only part of the bird is shown.

We can also see that the classes are derived from all different objects —— mostly coming from two major groups: animals and transportation tools. So overall this is a great dataset for us to use in this experiment.

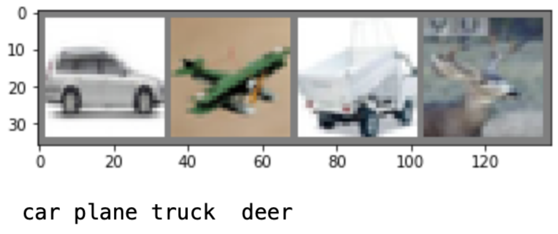


Figure 1. Example of Datas (CIFAR-10)

## 2. Tasks

### 2.1 Prediction Task

For this particular project, the task we are going to study is a very common classification task. Basically, it is taking an image as input and then returning the corresponding class. This task is extremely important because most Computer Vision areas are based on image classification. For example, self-Driving and face recognition, etc. This area is extremely valuable to study and has many real-world applications.

### 2.2 Train, Test Split

For this project, we used the default train test split by Pytorch. The training set has a size of 50000, where the testing set has a size of 10000. Normally this should be a Train, Test, Validation split. For convenience, we are not creating a validation data set. We are performing 10 epoch training for each model, which seems not causing overfitting.

### 2.3 Feature Engineering

There are two Pipelines adopted in our experiment:

1. Directly normalize on the image 32 x 32 x 3 tensors to be in a range [-1, 1]. (This Method was introduced in HW5. Will be used on Baseline Net and OurNet)
2. Change the shape of each image to 244 x 244 x 3. Then applying the normalization that was introduced from Pytorch AlexNet and ResNet 18. (Pytorch AlexNet.) This will also be used for those two structures.

## 3. Model and Method

### 3.1 Overview

We are going to use Convolutional Neural Networks for classification. CNN represents a big breakthrough in image classification and has addressed many challenging computer vision tasks.

To briefly explain how CNN works, images can normally be considered a high dimensionality matrix with each entry being considered as a pixel. CNN's multiple convolutional filters scan through the feature matrix, achieving the dimensionality reduction and feature extraction. Therefore, CNN's effectiveness in reducing feature numbers without the cost of losing on the model quality suits the purpose of image classification. Also, by pooling and larger strides, CNN has resistance to noise. (Mishra, Prafful [2] )

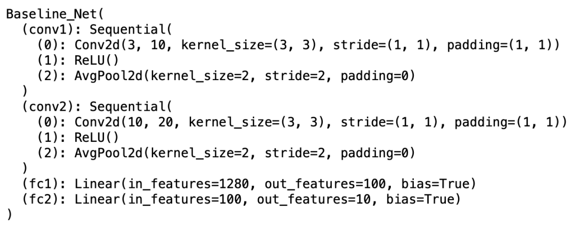
CNN is not only being used in computer vision research, but also being used as a feature extraction layer on other areas like Natural Language Processing. So it is extremely important to understand how this method works.

### 3.2 Baseline

We are going to directly use the model we used in HW 5. This can be considered as a rather shallow neural network compared to the recent models.

It consists of two convolution layers with kernel size 3, stride 1, and padding 1. It follows with ReLu as an activation function and an AvgPool2d. In the end, there are two fully connected layers for classification.

We can see that this model is very simple and light. At the same time, this is the very first model we implemented during this class, so we want to set this as a baseline to see if other models can outperform this model.



```
Baseline_Net(
  (conv1): Sequential(
    (0): Conv2d(3, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (conv2): Sequential(
    (0): Conv2d(10, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (fc1): Linear(in_features=1280, out_features=100, bias=True)
  (fc2): Linear(in_features=100, out_features=10, bias=True)
)
```

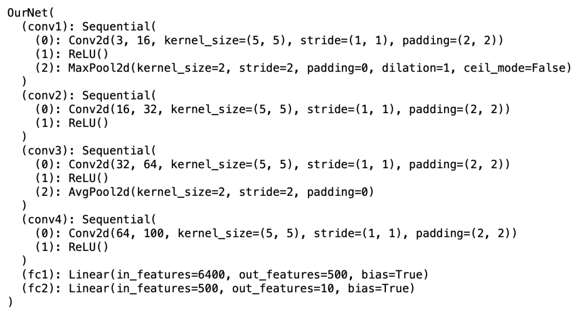
Figure 2. Baseline model Structure (HW5)

### 3.3 OurNet

For this model, we implemented it ourselves. It is developed on top of the baseline model. The main goal here for us to beat the baseline model. During the creation process, we have tried out different ways to stack layers and have explored with different activation functions. The model we create is somewhat lighter than AlexNet.

The model consists of 4 convolution layers (kernel size 5, stride 1, padding 2) and 2 fully connected layers.

The new thing here is instead of doing a pooling layer after each convolution layer, We apply a Maxpool after the first convolution layer, and for the third layer, we apply the Avgpool. It is later shown that it works well in this dataset.



```
OurNet(
  (conv1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
  )
  (conv3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (conv4): Sequential(
    (0): Conv2d(64, 100, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
  )
  (fc1): Linear(in_features=6400, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=10, bias=True)
)
```

Figure 3. OurNet Structure

### 3.4 AlexNet

AlexNet is developed by Alex Krizhevsky. It was a very deep network back in 2012.

The model has 5 convolution layers. The pooling layers are all overlap max pooling, which helped reduce the error-rate compared to traditional max pooling. The activation function is Relu. It also contains some dropout layers to deal with overfitting and fully connected layers for classification. (Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. [3])

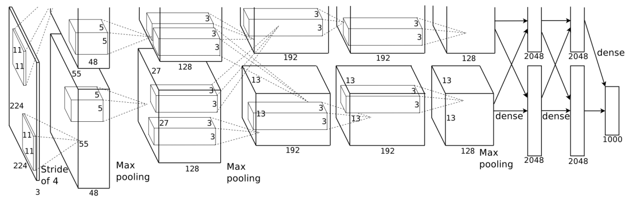


Figure 4. AlexNet Structure (Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. [3])

### 3.5 ResNet-18

ResNet is developed by Kaiming He. It is a simple and clean framework to train very deep nets.

The basic structure of ResNet is really clear. It has many residual learning blocks. The 18 is the number of residual learning blocks in the network. Each layer contains three components: two 3×3 convolutional networks, batch-normalization, and ReLU. Residual learning is applied for every two layers. ReLU is used after addition if there is a shortcut combination. (“ Residual Networks (ResNet).”[4], He Kaiming [5])

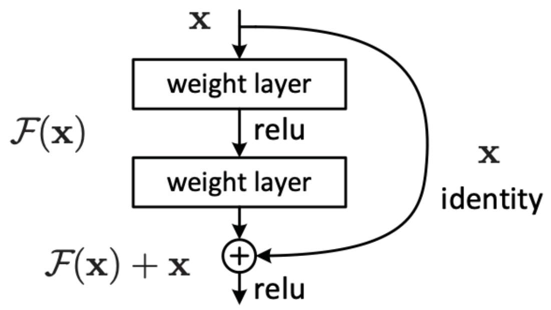

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 5. Example of Residual Learning Block (He Kaiming [5])

This structure is very useful to preserve the information from the previous layer and pass it into the next layer. It is being proved to be working in image classification tasks.

And we can also notice this model is built on residual learning blocks. So in the original paper, they also introduced a more stack version of this model, like ResNet 152.

## 4. Experiments

### 4.1. Overview

There are 3 sets of experiments we perform. We compare different optimization techniques on 4 models selected. And the last experiment is testing the usefulness of pre-trained models.

### 4.2 Experiment 1

We are using the data to have the mini-batch size as 4, with 10 epoch for each model. Most importantly, we used **SGD** (Stochastic Gradient Descent) with a learning rate equals to 0.001 and momentum equals 0.9. Then we are also checking the accuracy at epoch 5 for each model to look at the optimizer performance.

For clarification, We didn't use pre-trained AlexNet and ResNet in order to compare the actual usefulness of the structure.

Results can be found in Table 1.

### 4.3 Experiment 2

In this experiment, we are using **Adam (**Kingma, Diederik P., and Jimmy Ba. [9])with a learning rate of 0.0001 as an optimizer. Except this, all settings remain the same.

Results can be found in Table 2.

### 4.4 Experiment 3

In this experiment, we are using Adam with a learning rate of 0.0001 as an optimizer. We only perform training on two best models in this project, AlexNet and ResNet 18. This time we are actually using the pre-trained version of those two models, which are directly from Pytorch. (Pytorch AlexNet and ResNet. [6][7])

For this experiment, we are mainly discussing the differences between pre-trained model and non-pre-trained model. How much the accuracy will increase with pre-training. This is important to actually understand how the pre-train will work in real world applications.

For faster training, the mini-batch size here is 16.

Results can be found in Table 3.

## 5. Results

### 5.1 Overview

The evaluation metric we use here is accuracy. Since this is a classification task, it is natural to adopt this metric. So the results are shown below with tables. For AlexNet and ResNet 18, "NP" means Non-Pre-Trained and "P" means Pre-Trained model. "Acc" stands for accuracy.

### 5.2 Experiment 1 result

| Model | Acc at 5 epoch | Acc at 10 epoch |
|---|---|---|
| Baseline | 48% | 66% |
| OurNet | 51% | 71% |
| AlexNet (NP) | 52% | 78% |
| ResNet 18 (NP) | **55%** | **80%** |

Table 1. Experiment 1 results (SGD Optimizer)

According to the Table 1 above, we can see some patterns. The deeper the model is, the better it performs. For AlexNet and ResNet-18, they maintain relatively high accuracy when having the 5 epochs and 10 epochs. For OurNet, it is between the Baseline model and AlexNet. For SGD, in this experiment we find ResNet 18 has the best accuracy, and AlexNet is 2% below ResNet 18.

### 5.3 Experiment 2 result

| Model | Acc at 5 epoch | Acc at 10 epoch |
|---|---|---|
| Baseline | 45% | 62% |
| OurNet | 54% | 72% |
| AlexNet (NP) | 61% | 77% |
| ResNet 18 (NP) | **63%** | **83%** |

Table 2. Experiment 2 results (Adam Optimizer)

For this set of experiments, we can notice that the accuracies at 10th epoch are pretty similar. ResNet 18 has a higher accuracy after using Adam as an Optimizer.

We do notice some interesting things though. If we look at the accuracy at 5th epoch, the deeper models have a much higher accuracy than SGD gives. This is probably because of the adaptiveness of Adam.

For the 10th epoch, AlexNet has a 9% increase in accuracy with Adam. We do see that Adam will be a better optimizer to choose in some sense.

We didn't time the training process. However, since Adam involves more calculations for gradients, the training time is a little bit longer for models. There will be a trade off for those two methods.

In our opinion, Adam is a better optimizer to choose if there are enough computing resources.

### 5.4 Experiment 3 result

| Model | Acc at 10 epoch |
|---|---|
| AlexNet (P) | 86% |
| ResNet 18 (P) | **92%** |

Table 3. Experiment 3 results

From table 3, we notice the pre-trained model is really powerful. We know that it is novel to use a pre-trained model. However, it is still good for newcomers to try it out and see the differences.

For both models, there is a 9% increase in accuracy. This transfer learning technique is definitely something we can explore and experiment on.

### 5.5 Finding and Thoughts

We will not discuss this as an experiment but it is still worth talking about. The mini-batch size plays an important role in the training process.

For a bigger batch size, we noticed the speed of convergence is much slower, and it may converge to a bad minimum. If the batch size is really small, the training process is longer due to updating the parameter more. So there is some trade off going on.

We have discussed the mini-batch size problem with Kening Zhang. Many researchers focus more on the structure and they probably don't pay much attention to this problem. However, in our opinion, it is still worth noticing.

## 6. Conclusion

In this detailed and well designed project, we have tried different layers, activation functions, and structures of CNN. We have a rough idea of how things work and how we should implement them.

We proposed our own Model OurNet with the highest accuracy is 72 percent. Although it beats our baseline, it didn't perform as well as AlexNet in our experiment. The best result we achieved was 92% accuracy with the ResNet 18 Pre-trained model. This is the similar result in Kuang Liu's project. (Kuang Liu [8].)

Also, we have discussed the different performance caused by two different optimizers (SGD and Adam). Adam is performing better at the early epochs, and at the end they are roughly the same. However, Adam is more computationally expensive.

At the very end, we briefly discussed the finding based on mini-batch size. The larger batch size will help with training speed, but sometimes will hurt the performance. There is a tradeoff between performance and speed. This is something we consider very important but hasn't been commonly discussed.

Overall this is a very comprehensive experiment for newcomers. We explored the basic CNN and optimizers. It will help us to understand how things work, and give us some hands on experience with deep neural networks.

## 7. Acknowledgements

We would like to give special thanks to Prof. Zhuowen Tu who has taught many interesting topics and gone over the details of Neural Networks. He also helps us to brainstorm and gives useful suggestions and links on creating our own project. We would also like to thank Kening Zhang, our teaching assistant for all the technical support and discussion on batch size differences.

## 8. References

[1] “The CIFAR-10 Dataset.” CIFAR-10 and CIFAR-100 Datasets, www.cs.toronto.edu/~kriz/cifar.html.

[2] Mishra, Prafful. “Why Are Convolutional Neural Networks Good for Image Classification?” Medium, Data Driven Investor, 20 July 2019, medium.com/datadriveninvestor/why-are-convolutional-neural-networks-good-for-image-classification-146ec6e865e8.

[3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[4]“ Residual Networks (ResNet).” Residual Networks (ResNet) - Dive into Deep Learning , d2l.ai/chapter_convolutional-modern/resnet.html.

[5] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[6] Pytorch AlexNet. https://pytorch.org/hub/pytorch_vision_alexnet/

[7] Pytorch ResNet. https://pytorch.org/hub/pytorch_vision_resnet/

[8] Kuang Liu, pytorch-cifar. https://github.com/kuangliu/pytorch-cifar

[9] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).