Image classification based on Convolutional neural network

Author: Kessel Zhang

Program: CIS Machine learning 5.15-7.18

Abstract

My research will train a neural network that can classify images. Different ways of training will be applied, and they will be finally compared and analyzed. The goal is to tell the features and performance of these structures, then conclude the best one of the network and decide how and why this structure works well.
The dataset I used is the CIFAR-10, it is a basic image database that has 60000 images with the size of 32*32 pixels, these images could be classified into 10 labels. Multiple strategies of training have been tested, they are respectively AlexNet, LeNet5, VGG Net. These neural network algorithms are famous since they have achieved high accuracy in competitions. However, they are not designed for training CIFAR-10. Therefore, in this research, they will be modified and tested the performance on CIFAR-10.

1. Introduction

Machine learning is a method of data analysis that automates analytical model building [1]. It involves the concept of multiple fields, such as statistics, algorithms, and linear algebra. Nowadays, machine learning has been an essential part of Artificial Intelligence, and it will be more and more important to this society in the future.
This paper will discuss image classification. Image classification is a branch of AI, it aims to identify and analyze a given picture, then label it. It could be achieved by the CNN or DNN based on Keras. In this research, Keras and TensorFlow will be used to implement the neural network. At the same time, several famous algorithms will be implemented, modified, and discussed. They are respectively LeNet-5, AlexNet, and VGG Net. These algorithms show the different structures of CNN and therefore perform various results for the CIFAR-10 database.

In this research, the work will not focus on seeking higher accuracy. Instead, how, and why these structures of CNN could be applied to this database will be analyzed. To do this, this research will show the details about how modified structures being compatible with the new database and what hyperparameters make them feasible. In the meantime, AlexNet will be picked as an example to be exploited for finding the rules and relationship between the network's hyperparameters and PEG. By analyzing these three networks, it is found that deeper networks perform better, commonly. But other parameters such as kernel size will also influence the PEG to some extent.

The database (CIFAR-10) using for this research has 60000 data, which is a huge amount of data so that data augmentation could be avoided in this case. Nevertheless, due to the complicated design of the algorithm in this research, it takes hours to process these data. In this case, training on the CPU will not give us an expected training speed. In this research, training on GPU is necessary.

2. Preparation

2.1 Choose of database

Having an excellent database is always a good start to one project. In this research, the CIFAR-10 database will be adopted.

The **CIFAR-10 dataset** (Canadian Institute For Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms [2]. It has 60000 images data, and they could be classified into 10 classes (Fig 2.1). Each image has a size of 32*32 pixels, and it has three channels (RGB). Therefore, the shape of each data frame is (32,32,3). This dataset has been split into training sets with 50000 data and test set with 10000 data.

This database has been included in Keras library. In this research, it could be imported by the following code:

from keras.datasets import cifar10

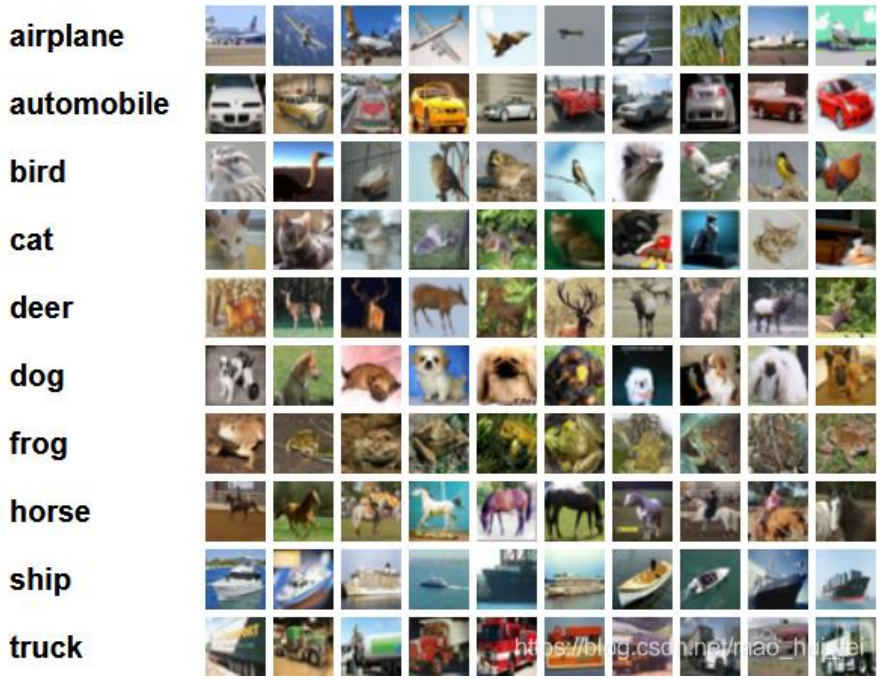

Figure 2.1 Example of images in CIFAR-10

2.2 Running on GPU

Training a neural network is a process of calculating and updating node weights. Therefore, the efficiency of our training is highly associated with the speed of the machine. Commonly, TensorFlow will automatically use the CPU (central processing unit) of a computer to train the network. It is enough for a lightweight project and a simple database. However, for a relatively complicated deep learning project, the CPU (graphic processing unit) computation capability will be an issue that slows down the project's progress.

Hence, before implement the network, the configuration for using the GPU should be prepared. Following the official document provided by TensorFlow, the following program should be installed:

Python	3.7.0
Tensorflow-gpu	2.5.0
CUDA	11.2
cuDNN	8.1
Graphic Driver	471.11

Both CPU and GPU have the ability to compute. Nevertheless, GPU is much more adept at parallel calculation. The main reason is that common GPU has only several cores, but GPU may have thousands of cores that could be used for calculation. And the neural network is embarrassingly parallel [3], so the training process (especially deep learning) is always effective on GPU.

The graphic card used in this research is a GTX3070, its computation capability is 8.4. That is fast enough for this project. After using the GPU, the training speed has been improved dozens of times. By contrast, for the same iteration, GPU use only 12s and CPU takes more than 6 minutes (Fig 2.2).

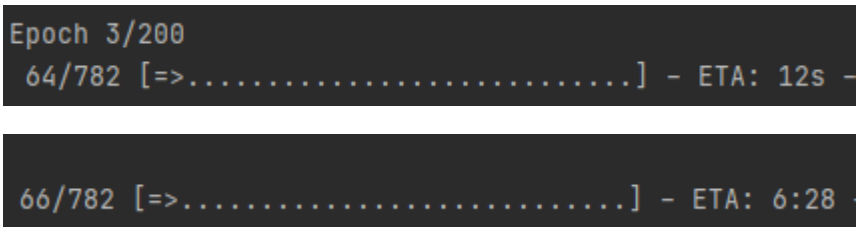

Figure 2.2 Speed comparison of training with GPU and CPU

2.3 Comparation between CNN and DNN

Before the implementation of the neural network, the structure of the network used should be decided.

In this research, DNN has been considered as one way to implement image recognition. However, DNN and CNN have different structures, so they perform differently in this research.

For the DNN structure program, three hidden layers and one output layer were used (Fig 2.3.1). This kind of model could provide test accuracy of 0.9 in the MNIST database, however, it has only 0.44 accuracy in this research.

```
network = Sequential()
network.add(Dense(32, activation='relu'))
network.add(Dense(64, activation='relu'))
network.add(Dense(128, activation='relu'))
network.add(Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 2.3.1 The structure of DNN in this research

Since CIFAR-10 has more complex images, the simple DNN structure is not enough in this case (Fig 2.3.2). DNN is getting higher accuracy by calculating and updating the node weights, but when the input images become sophisticated (RGB and different shapes). It could not perfectly identify the feature between different images. Therefore, CNN is proposed to solve this sort of image problem. Since CNN, instead, will carefully extract features from not only one dimension but both the height and width of an image.

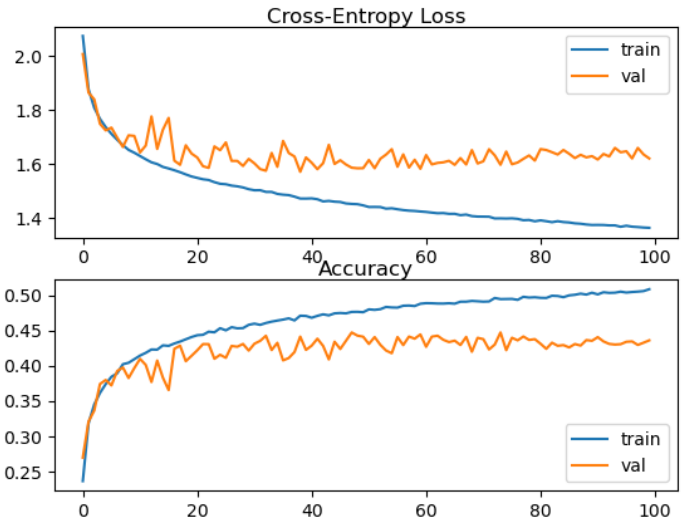

Figure 2.3.2 The performance of DNN on CIFAR-10

3. AlexNet

3.1 Introduction to AlexNet

AlexNet is a famous neural network architecture that was mainly designed by Alex Krizhevsky[4]. It was well known for competing in the ImageNet Large Scale Visual Recognition Challenge in 2012 and get the champion. The network achieves a 15.3% error rate, which is an amazing rate since that was 10.8 percentage points lower than the runner-up.

The concept of AlexNet is that the depth of a model is extremely important for performance. Therefore, it was written in CUDA, so that it could be running on GPU to get a higher training speed. AlexNet is not the first network that is training on GPU, while it brings huge effect in this field. It activates more research than training on GPU and was considered as one of the most important articles in the field of image recognition.

The typical AlexNet (Fig 3.1.1) has eleven layers, they are 5 convolutional layers, 3 max-pooling layers, and 3 fully connected layers.

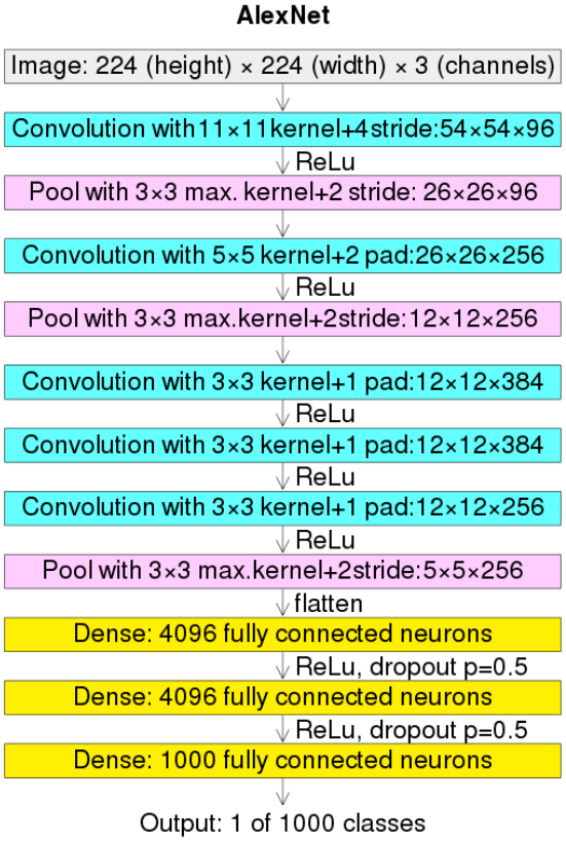

Figure 3.1.1 The architecture of AlexNet for ImageNet [4]

In the picture above (Fig 3.1.2), the structure of AlexNet could be observed. Before building the model, several features of AlexNet should be noticed.

1. The relu activation function was used for each convolutional layer.
2. The regularization method of the dropout was implemented among the fully connected neurons.
3. The input size for typical AlexNet is 256*256.
4. data augmentation trick will increase the accuracy by an extra 1 percent.

Before thinking about how to improve the accuracy, let's take a quick look at the performance when the original architecture is directly applied to the new database (only the input is modified, otherwise the program will not compile).

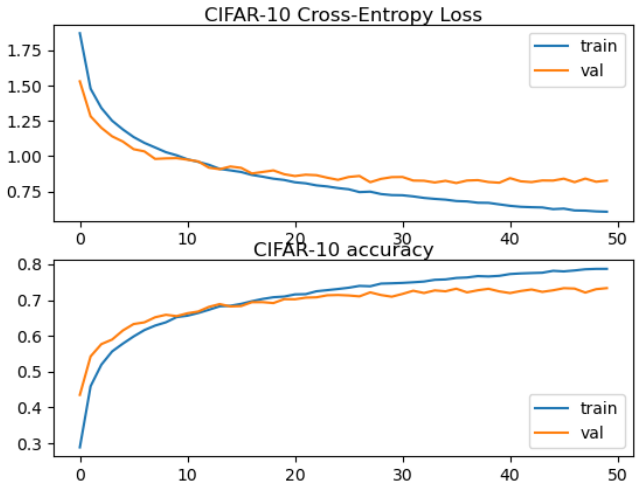

Figure 3.1.2 The performance of original structure on CIFAR-10

A concise conclusion: The performance does not achieve the expectation. The test accuracy is only about 0.7.

3.2 Update on the AlexNet Structure

Since the AlexNet has large input and corresponding kernel sizes. Such a structure fits 224*224 images well, but it is not compatible with a database with a smaller size of images. Nevertheless, the hyperparameters of each layer could be modified so that the new model could adapt to the CIFAR-10 database. In this section, some feasible methods that could improve the accuracy will be studied.

Compared to the original model, several changes have been made.

1. The input size must be edited to (32,32,3). Without a correct input size, the program will collapse.
2. The kernel size in the first layer changes from 11*11 to 3*3. A smaller filter size will provide the CNN more details about images.
3. To avoid unstable variables, all paddings are set to “same”.
4. To avoid overfitting and to speed up the training process, the number of filters decrease (include both convolutional layers and fully connected layers).

After several tests and updates, the final model gives a structure like this (Fig 3.2).

```
model = Sequential()
model.add(Conv2D(28*2, (3, 3), input_shape=(32,32,3),strides=(1, 1), padding='same',
          activation='relu')) # valid

model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same'))

model.add(Conv2D(64*2, (5, 5), strides=(1, 1), padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same'))

model.add(Conv2D(96*2, (3, 3), strides=(1, 1), padding='same',
          activation='relu'))

model.add(Conv2D(96*2, (3, 3), strides=(1, 1), padding='same',
          activation='relu'))

model.add(Conv2D(64*2, (3, 3), strides=(1, 1), padding='same',
          activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same'))
model.add(Flatten())
model.add(Dense(1024*2, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1024*2, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

Figure 3.2 The structure of the new model

3.3 Result and conclusion

From the chart below (Fig 3.3), it is observed that the test accuracy could reach 90% after 100 epochs. And there is a slight jump inaccuracy at epoch 100, the reason is that a callback had been set in the model, that could control the learning rate during the training. When the epoch is over 100, the learning rate will be reduced from 0.01 to 0.001. In that case, a lower learning rate provides lower changes in the process of gradient descent, so that a global minimum is more possible to be found.

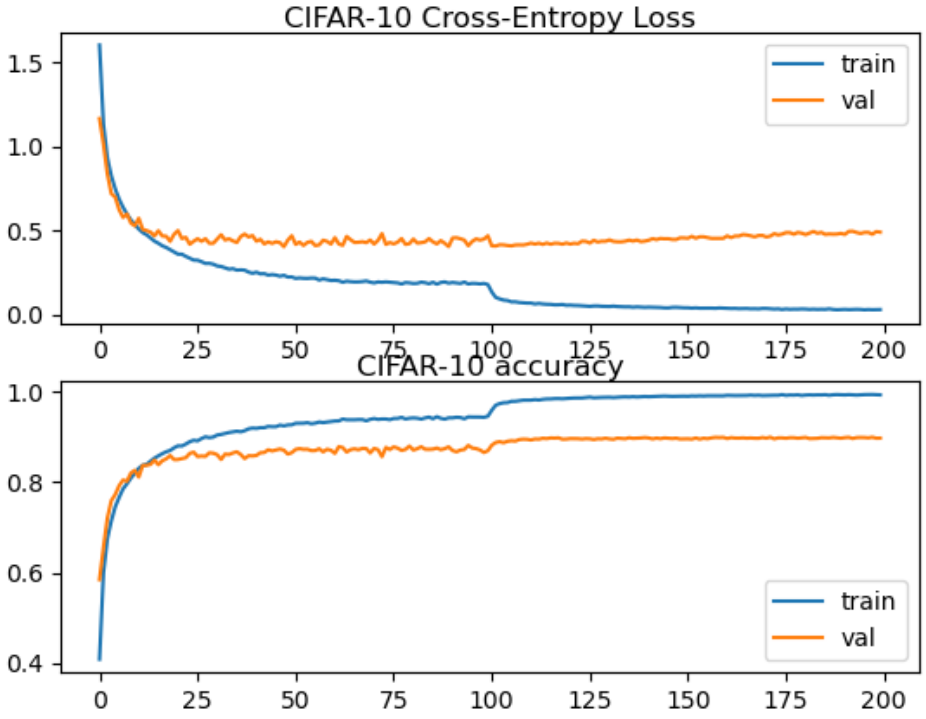

Figure 3.3 The performance of this model

4. LeNet-5

4.1 Introduction to LeNet-5

LeNet-5 is a typical and old neural network architecture. That was firstly raised in 1998 by LeCun. It is not a complex model compared to nowadays neural network structure, but it could be considered as the step-stone of CNN.

The LeNet-5 was designed for the MNIST database, it performs well for this kind of uncomplex database. For MNIST, PEG could be lower than 0.01 when the LeNet-5 was applied.

The LeNet-5 structure is not hard to understand (Fig 4.1.1), so after simply modifying the input size (form 28*28 to 3 channel 32*32), the first trial could be started.

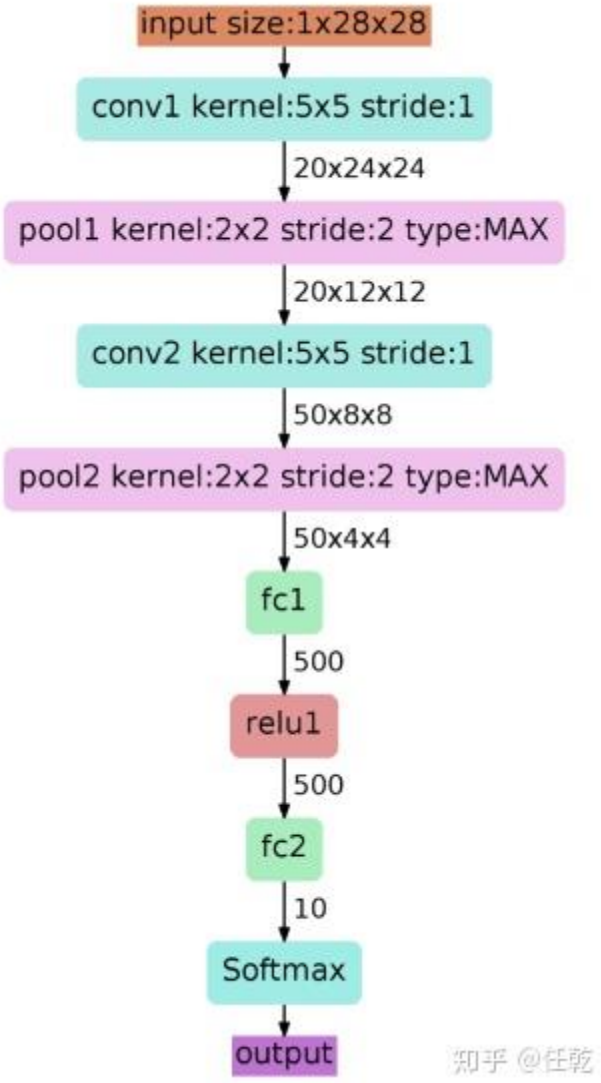

Figure 4.1.1 The architecture of LeNet-5 for MNIST [5]

4.2 Performance of original LeNet-5 model on CIFAR-10

For the first experiment, all hyperparameters remain the same as the original LeNet-5. It is not hard to observe that strong overfitting happens to this database (Fig 4.2.1). The chart illustrates that overfitting happens after the 10^{th} epoch, the training accuracy keeps going up (almost reach 0.9), but the validation accuracy remains around 0.6.

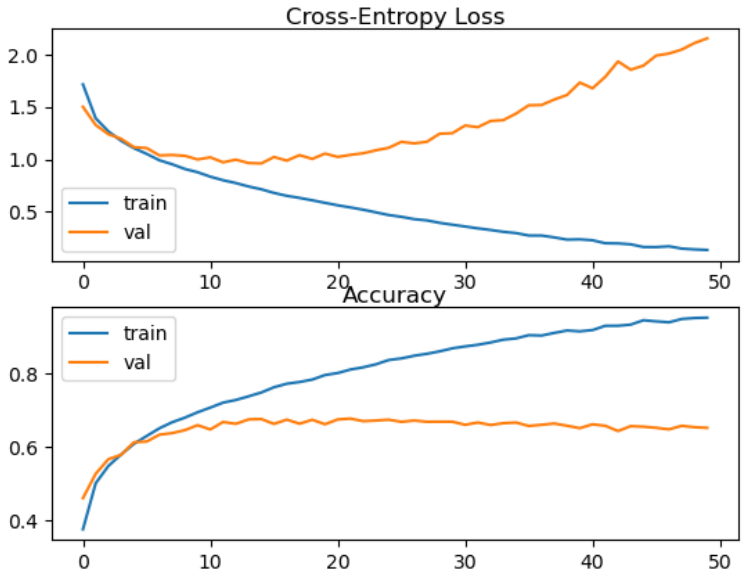

Figure 4.2.1 The performance of LeNet-5 (first trial)

From this result, it could be concluded that the LeNet-5 network works perfectly for its MNIST but when it comes to a more complex situation, it could not show the same performance as before.

4.3 The idea about improving accuracy on LeNet-5

The most important thing is enhancing the depth of such a network. A deeper network will give higher accuracy. After that, some tricks should be considered to solve the overfitting problem in this model. From the picture, it is found that severe overfitting happens after the 10^{th} epoch, after that, the test accuracy keeps going down. This means that without regularization or data augmentation, the overfitting problem will destroy the whole model.

5. VGG Net

5.1 Introduction to VGG Net

VGG Net is a deep convolutional neural network developed by the Visual Geometry Group from Oxford University, and researchers from Google DeepMind Company. It gets the runner-up in the competition of ILSVRC 2014. The VGG Net develops based on AlexNet and again proved the importance of depth in a neural network.

The VGG Net has multiple configurations, this chart illustrates the difference between different configurations in structure (Fig 5.1). Commonly, the 16 weights layers VGG Net will be called “vgg16”. With the same rationale, the 19 weight layers are the “vgg19”. And most of the research prefers to use vgg16 and vgg19 as an example to demonstrate this model. In this research, vgg19 will be adopted.

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					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figure 5.1 The configurations of different VGG Net architecture [6]

5.2 Performance of original VGG Net model on CIFAR-10

By observing the model structure of the vgg19, it was found that for each filter, VGG Net uses the (3,3) as the kernel size with stride 1. This means that the filter size is small enough to traverse the whole image, no matter what the size of the image is. Unlike the AlexNet, which uses a large kernel size for 224*224 images so that the kernel size must be decreased for smaller input. In this case, the vgg19 could directly be implemented without changing any hyperparameters except the input size.

Then we could quickly get the resulting chart below (Fig 5.2). It illustrates the accuracy of 0.8 for 50 epochs. It is much better than our first trial of AlexNet (70% accuracy) due to vgg19's high compatibility. With more tricks such as data augmentation or regularization, its accuracy could be further improved.

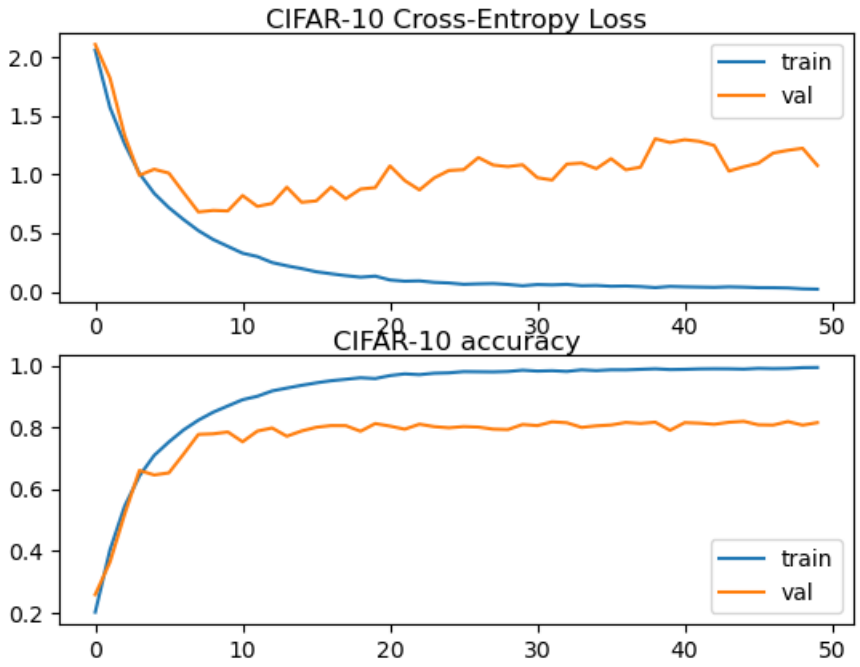

Figure 5.2 The configurations of different VGG Net architecture

5.3 The idea about improving accuracy on VGG Net

On the VGG Net, overfitting happens but it is not the most important problem. The performance of VGG Net is fine but its accuracy should be increased. A good choice is to use data augmentation.

6. Conclusion

In this research, during the preparation process. The DNN structure is proven to be unsuitable for a complex image since it does not care about the width and height of an image. CNN is commonly a better choice for such an image classification problem.

When a networks' structure goes too complex, the training process would usually take many hours. Some structure that owns higher performance may probably take more than one days. To run networks on GPU with an Nvidia graphic card, tensorflow-gpu, CUDA,cuDNN should be installed. After comparing the speed, the result is that programs running on GPU are about 30 times fast than running on CPU.

Among these three CNN, AlexNet performs best. It performs much better than LeNet-5, which proves that deep neural networks will show better performances. However, at the same time, VGG Net has also a deep structure but its accuracy is lower than AlexNet. By analyzing such a situation, it could be concluded that other parameters such as the dropout method and l2 regularization have also an impact on the performance of accuracy.

During the training of AlexNet, a slight jump of accuracy was observed. That is because the learning rate changes. This means that when a network is trained deeply, the learning rate should be diminished.

7 Reference

[1] Machine Learning: What it is and why it matters. (2021). Retrieved 21 July 2021, from https://www.sas.com/en_us/insights/analytics/machine-learning.html#:~:text=Machine%20learning%20is%20a%20method,decisions%20with%20minimal%20human%20intervention.

[2] CIFAR-10 - Wikipedia. (2021). Retrieved 21 July 2021, from https://en.wikipedia.org/wiki/CIFAR-10

[3] Why deep learning and neural network need GPU. (2021). Retrieved 21 July 2021, from https://zhuanlan.zhihu.com/p/106669828

[4] AlexNet - Wikipedia. (2021). Retrieved 21 July 2021, from https://en.wikipedia.org/wiki/AlexNet

[5] Deep learning | image classification: LeNet-5. (2021). Retrieved 21 July 2021, from https://zhuanlan.zhihu.com/p/74176427

[6] [convolutional neural network] VGGNet analysis and implementation. (2021). Retrieved 21 July 2021, from https://blog.csdn.net/qq_39071739/article/details/103971737