Report of CS4487 Project: An Application of Image Classification on CIFAR-10 by Using Machine Learning Approach

Wenbo Yang (55203027) Deheng Zhang (55199998)
City University of Hong Kong
{wenboyang3-c, dehezhang2-c}@my.cityu.edu.hk

16 December 2019

1 Abstract

Image classification is an interesting and hot topic in the field of computer vision. Instead of writing a program to assign the class label explicitly, data-driven approach has a better performance. In this report, we analyze the nature and difficulty of image classification and conduct experiments on the data set CIFAR-10 [1] to try different models, which suggests that the convolutional network ([2] and [3] especially) has the best performance for image classification problem. Besides, we also adjust the hyper-parameters such as batch size and weight penalty, which suggest that the proper selection of hyper-parameters can increase the accuracy of prediction significantly. Finally, we achieve 95.95% validation accuracy on the local machine and 94.90% test accuracy on the Kaggle contest by using WideResNet-20 [4].

2 Introduction

Machine learning is a field of study that gives the computer the ability to learn without explicitly programmed. A more professional definition is: a computer program learns from experience E with respect to some task T, measured by the probability of an accurate estimation of P (Mitchell, 1997). Machine learning can be divided into classification problems and regression problems according to the expected output. Deep learning is part of a broader family of machine learning methods based on artificial neural networks with so many architectures such as deep neural networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition. [5]

In this project, we applied machine learning approach to solve the image classification problem in the field of computer vision. The data set is CIFAR-10, which contains 60,000 32-by-32 color (RGB) photographs of objects from 10 classes (plane, car, bird, cat, deer, dog, frog, horse, ship, truck). There are 50,000 training images and 10,000 testing images. The Figure: 1 are the classes in the data set.

This problem is an image classification problem, which is defined as given a feature vector from an image $\mathbf{x} \in \mathbb{R}^n$ that describes an object that belongs to one of $|\mathcal{Y}|$ classes from the set $\mathcal{Y}$, predict which class the object belongs to. The performance of the model (classifier) is measured by the accuracy equation (1) of

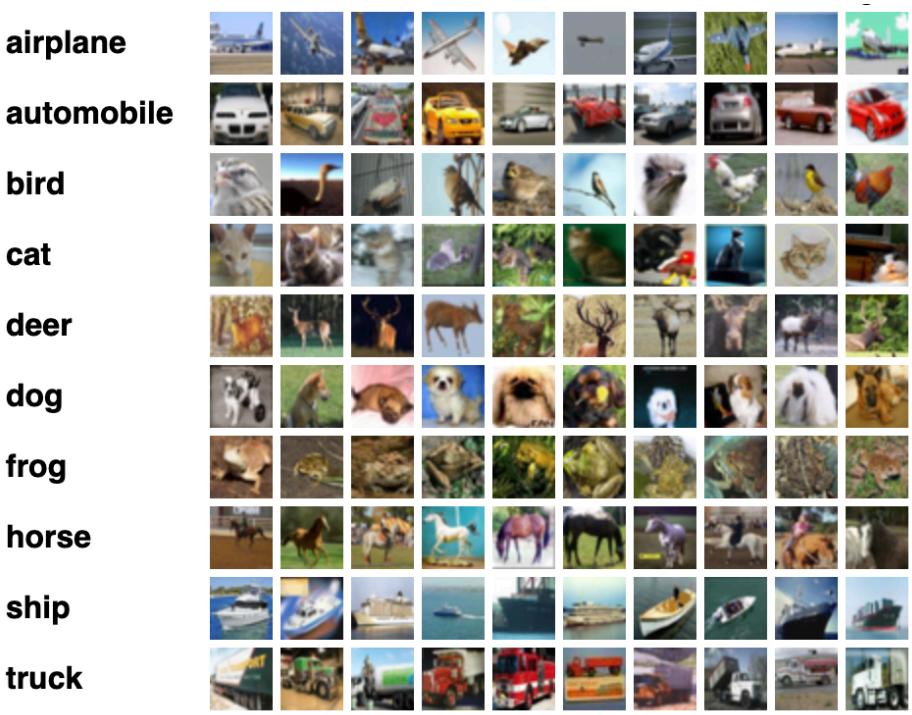

Figure 1: A image to show part of images in 10 classes of CIFAR-10 [1].

prediction on the test set,

$$Acc(f, \mathcal{D}_{test}) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}[y^{(i)} = f(\mathbf{x}^{(i)})] \tag{1}$$

For image classification in this project, non-parametric approaches such as K-Nearest-Neighbor is not suitable. Because the data set is too large to be stored, and it takes much time for predictions. Besides, the features for images are so complex that cannot be described by low-dimension distance. Therefore, we decided to use parametric approaches such as naive Bayes, support vector machine, convolutional neural network, VGG network, and ResNet. Besides, we tried to use pre-processing methods such as fast Fourier transform, normalization, principal component analysis, and other physical operations on the training image. To further analyse the data set, we also visualized distributions of features for images and averages of images.

3 Challenges

Although it is a trivial task for a human to label an image, the task is hard for computers. Because computers cannot understand the features of an image efficiently and accurately. Besides, there are a lot of extreme situations for image classification:

3.1 Perspective and Viewpoint

The size and features of an object are different for different distance and viewpoint of the observer (camera), which increases the number of profile types that need to be learned by the computer.

3.2 State and Environment

The states of the observed object are diverse and changing. For example, the size of an animal might be influenced by its age and its posture when being photographed. Besides, there might be some other objects in front of the observed object, which increase the difficulty to recognise the correct object. The environment also affects the prediction accuracy, because the brightness of the light and the background colour will make the object challenging to be discovered.

3.3 Variation and Similarity

Though all categories in CIFAR-10 are properly defined, images showing objects from some different categories in CIFAR-10 are very similar, for example e bird-plane, cat-dog and truck-car pairs, as shown in Figure: 2. Some of them are very difficult to distinguish, even by an adult human.

Figure 2: A image in the category 'plane' in CIFAR-10, which looks like a bird

3.4 Time and Memory Limit

By using deep neural networks, the issues mentioned above can be partially solved. However, DNN introduces a new problem - the time and memory limit problem. DNN uses a considerable amount of matrix computation, which is not efficient for CPUs to operate. GPUs can operate the DNN calculation in a relatively fast speed, but due to the memory limit for our GPUs, it is hard to train DNN with a large number of layers or to increase the batch size of the data loader when predicting.

In the last few models, it takes around 1.5 to 10 minutes for our GPUs to go through each epoch of training. At the end of the training, it takes 2 epochs for every 0.2% of testing accuracy improvement. Unlike final year students who have access to machines from CSLab with advanced graphic cards, we only have older mid-level consumer-level GPUs. This has significantly affected our progress. We tried to use the server of CSLab, but it is not easy to get out of the queue of using GPUs.

Considering that batch-normalisation is used in our project, the training batch size needs to be reasonably large, for example, 128 or 256. After testing, we find that it is important to keep a large testing batch size to get a higher score. However, this is a burden on our graphics cards, as we only have 4 GiB of graphic memory.

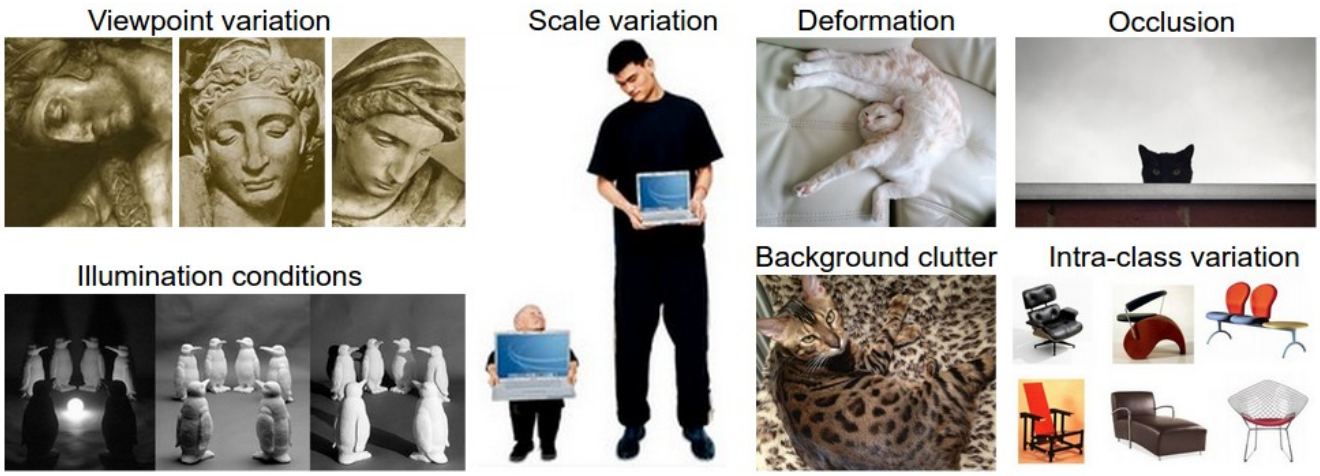

Figure 3: Some examples of first three limitations [6].

```
    542         for hook in self._forward_hooks.values():
    543             hook_result = hook(self, input, result)

D:\documents\Lecture_Note\cs4487\project\zdh\models\wide_resnet.py in forward(self, x)
     42             out = self.relu(self.bn2(self.conv1(x_out)))
     43             if self.droprate > 0:
---> 44                 out = F.dropout(out, p=self.droprate, training=self.training)
     45             out = self.conv2(out)
     46         else:

D:\Anaconda\lib\site-packages\torch\nn\functional.py in dropout(input, p, training, inplace)
    805     return (_VF.dropout_(input, p, training)
    806             if inplace
--> 807             else _VF.dropout(input, p, training))
    808
    809

RuntimeError: CUDA out of memory. Tried to allocate 20.00 MiB (GPU 0; 4.00 GiB total capacity; 2.86 GiB already allocated; 1
6.80 MiB free; 80.12 MiB cached)
```

Figure 4: GPU out of memory during training a neural network.

4 Experiments and Research

4.1 Data Analysis

4.1.1 RGB Distribution

As shown in Figure 5, we have visualized the distribution for each one of the RGB channels. The mean and standard deviation of each channel is calculated during the process, they are $\mu = (0.4914, 0.4822, 0.4465)$, $\sigma = (0.2023, 0.1994, 0.2010)$.

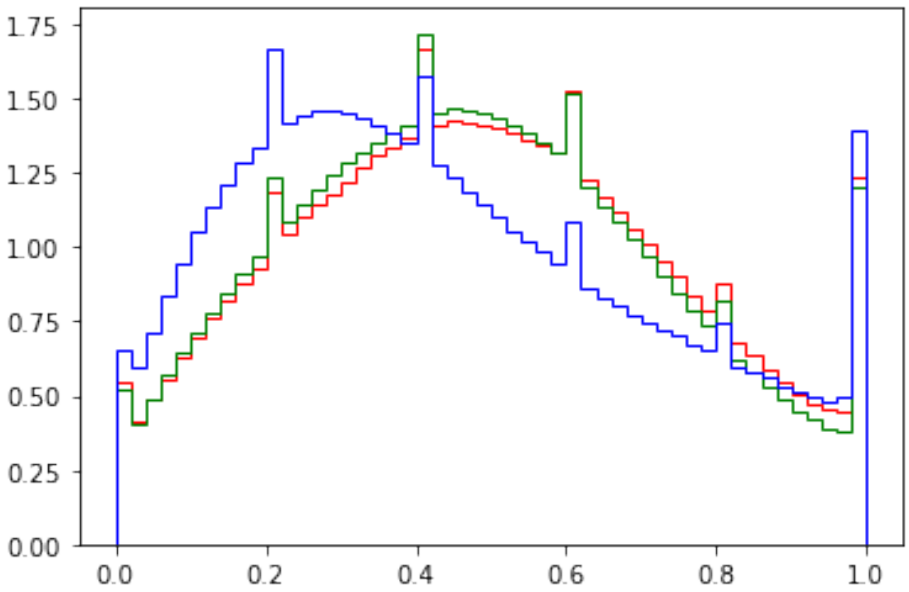

Figure 5: Distribution of the red, green and blue value of pixels in the training images.

4.1.2 Mean Value Visualization

As shown in Figure 6, we have visualized the mean value of the training set. Most part in this diagram is gray, but the centre is more likely to be warm-coloured.

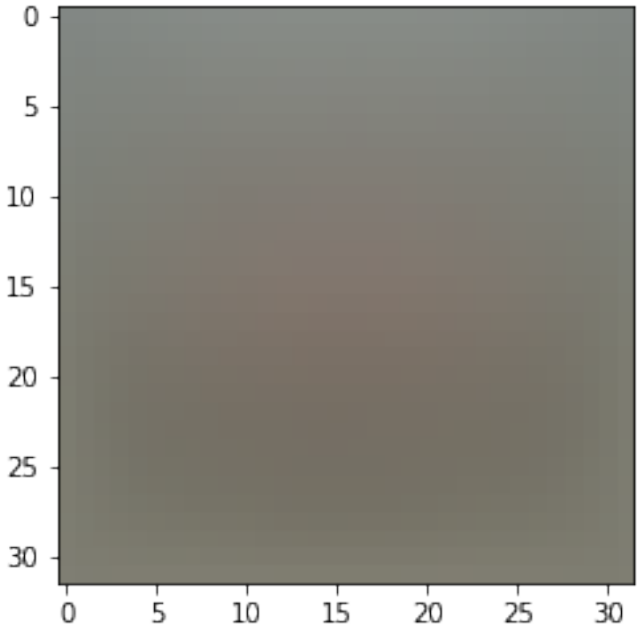

Figure 6: Visualization of the mean value of the training set.

4.2 Model Selection

We tried a lot of models including SVM, CNN, ResNet, and WideResNet. The specific training result can be found in the history folder in our submission folder. The accuracy of each model is shown in the Tabla 1.

Model	Train Accuracy	Validation Accuracy	Test Accuracy
SVM	57.0%	∼	∼
Gaussian Naive Bayes	32.5%	∼	∼
CNN-7	84.87%	82.72%	80.72%
ResNet-18	99.964%	93.2%	92.341%
WideResNet-20	99.98%	95.95%	94.90%

Table 1: Train, validation and test accuracy of multiple model attempted.
The ∼ symbol represents item not tested.

4.2.1 SVM and Naive Bayes

Applying kernel SVM directly to the input data is computationally impractical. However, we still attempted to train the model using kernel SVM model. Therefore, SVM with RBF kernel is used on the results of Fourier transform, yielding a training accuracy of 57.0%. Though not very high, still admirable considering the simplicity of the model.

The Gaussian Naive Bayes model is also used after performing the Fourier transform, yielding a training accuracy of 32.5%.

4.2.2 Deep Neural Network

The deep neural network is a machine learning model that accepts the input matrix, calculates the output matrix, and stacks different kinds of computation layers. Different from simple linear classifiers such as logistic regression, the neural network provides a more complex decision boundary by adding non-linear activation function after each fully connected layer.

At the beginning of network design, we found that the first few networks constructed by us have too low training accuracy. This is possible because of the low capacity of the model. After increasing the complexity of the network, the training accuracy increased. As the number of layers of a network increase, the capacity of the network increases, and the network becomes more capable of fitting more complex data. Therefore, after proof of concept, we increased the number of layers of the neural network. However, after a certain stage, merely increasing the size of the network became impractical, because training speed became unacceptably slow. Increasing the batch size and using PRelu have eased the problem but as the number of layers increases, the speed slowed down again, and other resource consumption became a limit of our training process.

4.2.3 Convolutional Neural Network

General DNN is mostly composed of fully connected linear layers, which requires storing the weight matrix connected to all the entries of the previous matrix. Actually, for image classification problems, we can take advantage of the pattern or edges in the image to reduce the fully connected layer to partially connected layer. By stacking the convolution layers, the model can learn the detail of the image from the small edges to general patterns as the network going deeper.

Due to the fact that the training set consists of images, to increase the computation speed and exploit the locality of information stored in images, CNN is used in this project. The very simple neural network from PyTorch's tutorial consists only three convolution layers, one pooling layer and three full-connected layers, but has a validation accuracy of more than 50% [7]. After increasing the convolution and fully-connected layers' size and increasing batch size, the validation accuracy became around 82.72%. (See Tweaked tutorial network in our history files.) Its structure is shown in the diagram 7.

4.2.4 Residual Neural Network [2]

After literature reviews, we found that the slow-down of training speed after the network became deeper is possible due to the problem called vanishing gradients, meaning that the gradient of a network will become so small, when being back-propagated, that improvement of the network by the step-down following the gradient does not affect the top layers effectively.

As shown in the diagram, the derivatives of each activation function on their input are between 0 and 1, and for a network that is unlikely to overfit, each linear transform from one layer to another is unlikely to have large entry, resulting in that the derivatives of the loss function on the parameters of top layers are likely to be "vanished".

RNN resolves the vanishing gradient problem by connecting smaller blocks of NN both parallelly (green lines in the Figure: 8 and sequentially (blue lines). It is possible that the gradient propagates from the direct decedent layer has vanished, but it is unlikely that the gradient propagate from much lower layer do.

By using the model ResNet-18 with test set patch size 6000, we achieve 93.23% validation accuracy, and 92.34% test accuracy on the kaggle contest.

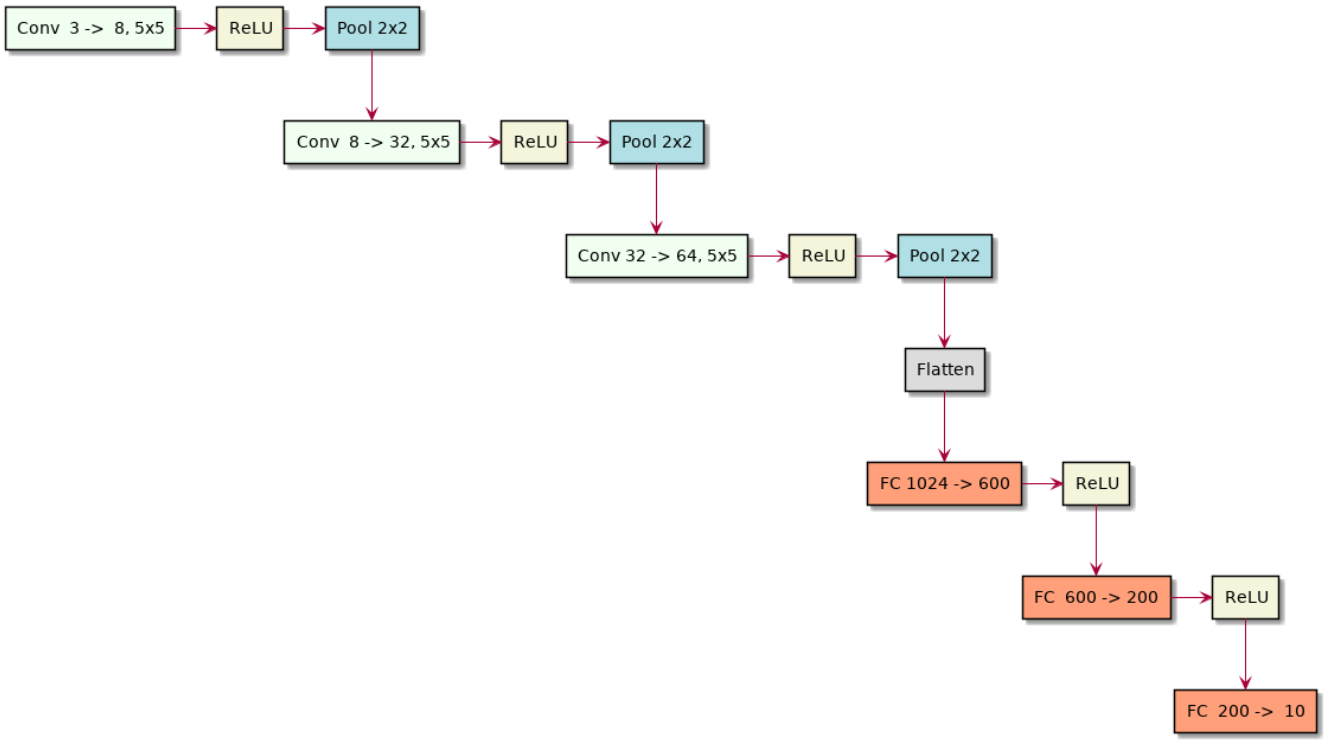

Figure 7: Modified network form PyTorch's tutorial. Validation accuracy: 82.72 %, training accuracy:84.87%

4.2.5 Wide Residual Network [3]

We noticed that for ResNet-18, the validation accuracy is converged at 93.00%. Limited by the computation time and memory, we cannot increase the depth of the neural network too much. Therefore, we decide to use WideResNet. The structure of WideResNet is shown in Figra: 9

The study of normal ResNet focuses mainly on the order of activation inside a ResNet block and the depth of residual networks. WideResNet increases performance by widening the residual blocks. According to [3], the wide network has 50 times fewer layers and being more than 2 times faster. Besides, drop out is used in this model to avoid overfitting.

Finally, we get hour best local validation accuracy 95.95% and best test accuracy on the kaggle contest 94.9%.

4.3 Activation Function

The most traditional activation function in a neural network is a logistic function

$$g(x) = \frac{1}{1 + \exp(x)}$$

. The hyperbolic tangent function is very similar to logistic function and also widely used. However, they are likely to get saturated in both directions, leading to vanishing gradients (see 4.2.4). ReLU is preferred in deep learning and used in our project. The definition of ReLU is :

$$g(x) = \max(0, x)$$

.

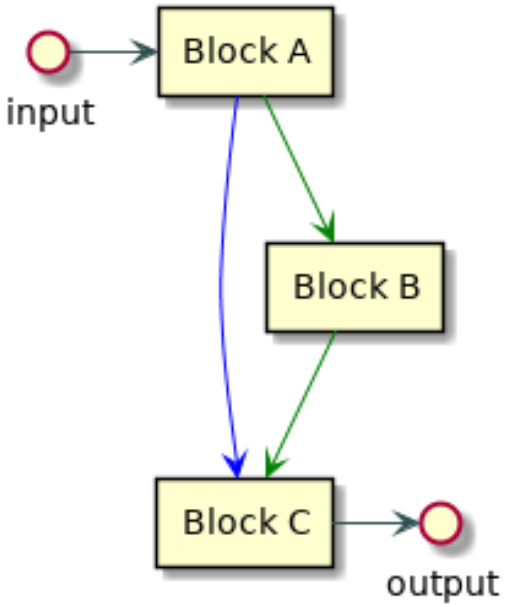

Figure 8: The structure of part of ResNet, C and B can be regarded as connected both parallelly and sequentially.

At the beginning of the project, leaky ReLU and PReLU are also used. The definition of them are both

$$g(x) = \max(\alpha x, x)$$

, with the difference that the α for PReLU is a learnable parameter, while that for leaky ReLU is a pre-fixed hyper-parameter (usually 0.1 or 0.001). These functions were used in the earlier constructed (simpler) networks by us and provided some improvement to both training and testing errors. Meanwhile, in the last larger network, we did not use these activation function for simplicity in the final result. Our current test shows that more advanced activation functions do not provide significant performance improvement, comparing to ReLU.

4.4 Batch Normalization

Batch normalisation is a technique to increase training speed and to reduce the likelihood of over-fitting [8]. We embrace the technique in our solutions. (See ResNet 76 BN) To yield a reasonable result, the batch size of the neural network is increased to 32, which became a burden on the GPUs we have.

The test set batch size also has a very significant effect on the testing errors. At the beginning, we set the batch-size of local validation-set to be 16, and the size of Kaggle's testing-set to be 2. We found a unreasonable gap between local validation error and Kaggle's testing error. Two days are spent before we discovered the problem. After re-setting the batch-size, the gap became normal and expected.

4.5 Pre-processing

4.5.1 Fourier transform

We have attempted to perform 2-D Fourier transform on the data set, as a dimensionality-reduction technique. We first perform a 2-D fast Fourier transform on the dataset and extract most low-frequency components, using them as the reduced data.

group name	output size	block type = $B(3,3)$
conv1	32×32	[3×3, 16]
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Figure 9: The structure of the wide resnet[3].

The incentive is that most information of an image perceived by human are the low-frequency components. (Which is also the strategy of JPG's compression). Therefore, by performing FFT, then extracting the low-frequency components, we can reduce the dimension of the input effectively and transform them to a more easy-to-process format.

When applied to SVM, the training speed became practical. (using the original data for training SVM is impractical due to the time complexity of SVM) When applied to NN, it is unreasonable to use convolution layers. By using full-connected layers thought the network, the speed-up is not apparent.

Images may contain some high-frequency noises, we can remove the noises by eliminating the high-frenquency components after performing FFT, then reverse FFT. Though some information may be lost during the de-noise process, the overall shape in the original images are kept.

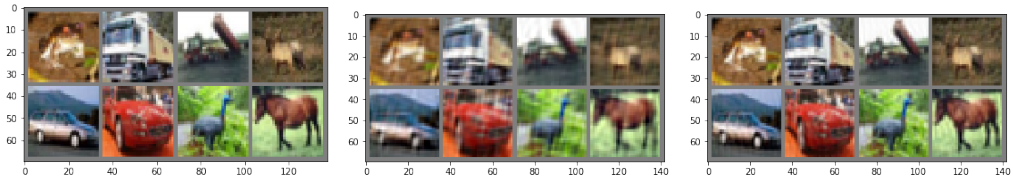

Figure 10: Examples of removing high-frequency components from images. Left: no component removed. Middle: 5×5 out of 32×32 entries of FFT result kept. Right: 10×10 out of 32×32 entries of FFT result kept.

4.5.2 Whitening

Whitening is a form of pre-processing that first center the data and subtract the mean, then compute the SVD of the covariance matrix that tells us about the correlation structure in the data, and finally project the original data matrix to the column space of $U\Sigma$ as shown in (2)

$$\begin{aligned} XX^T &= U\Sigma U^T \\ X_{whitening} &= (\Sigma^{-1}U^T)X \end{aligned} \tag{2}$$

By using whitening, the correlation between each dimension can be reduced. However, in the image classification problem, it is not easy to calculate the SVD of the covariance matrix because of the large data set and a large number of features. Besides, if we are using CNN, we need to keep information

locality of the images, which will be destroyed by applying PCA on images as a while as shown in Figure: 11.

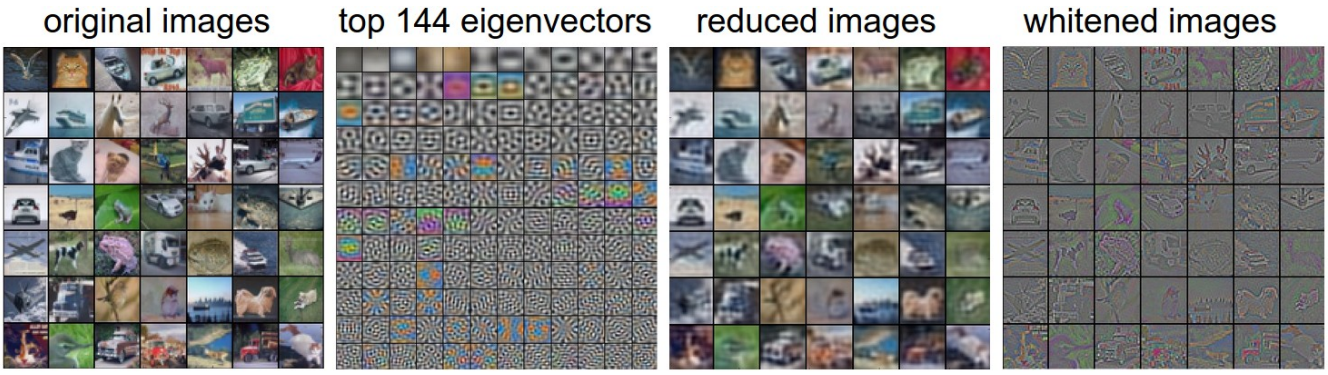

Figure 11: Example of visualizing the whitened images [6] .

4.5.3 Normalization

The aim of normalization is re-scaling the data to make it easier for a model to fit the data. According to the standard normal distribution, which is a special case of the normal distribution, the mean and standard deviation of the normal random variable are 0 and 1 respectively. Suppose x is a normally distributed random variable, it can be transformed to a standard normal random variable using the following formula:

$$z = \frac{x - \mu}{\sigma} \tag{3}$$

At first, we affinely mapped each of the RGB channels from $[0, 1]$ to $[-0.5, 0.5]$, which achieves a test accuracy 77.55% for the seven-layer neural network on Kaggle. After the data analysis, we decided to use means and standard deviations $\mu = (0.4914, 0.4822, 0.4465)$, $\sigma = (0.2023, 0.1994, 0.2010)$ to normalise the input, using the formula 3, which achieves a test accuracy 78.54% for the seven-layer neural network on kaggle.

4.5.4 Physical Operation

At first, we did not use the physical operations for the seven-layer-CNN, and the test accuracy on Kaggle is 77%, after adding the physical operations: random crop and random horizontal flip, the test accuracy is increased to 78% on Kaggle. When we changed the network to WideNet, we also added cut out operations to randomly cut holes in the image, which also increases the performance.

4.6 Hyper-parameters and Optimizer

Tuning the hyper-parameters and choosing optimizers are the hardest and most crucial parts of training a model. Cross-validation is a powerful method to solve the parameter tuning problem. The idea is to randomly partition the training set into K validation folds, for each fold and each choice of hyper-parameters, choose one fold as the validation set and learn using the weight parameters on the remaining folds. After learning, calculates the average validation accuracy for each hyper-parameters, and choose the one has the best average accuracy. However, the computational difficulty is higher for deep learning than other machine learning methods. Therefore, it is challenging to apply cross-validation to deep learning problems. Besides, some hyper-parameters such as learning rate and batch size should be determined

during the process of training and testing. So we only perform cross-validation on the hyper-parameter weight decay.

4.6.1 Optimizer

At first, we used Adam to optimize the loss function. The benefit of Adam is that the speed of training will be slow down automatically. However, it requires us to carefully tune the hyper-parameters α and β. For a deep neural network, it will take much time to tune hyperparameters. Therefore, we choose SGD after finding out that the speed of Adam is difficult for us to control.

4.6.2 Learning Rate

We tried to use the same learning rate (0.1) at first, the loss function is decreased properly at first. However, WideResNet converged when its accuracy was around $80 \sim 90\%$. Therefore, we try to use the learning rate as shown in the Tabla 2.

Epoches	Learning Rates
0∼49	0.1
50∼99	0.02
100∼139	0.004
140∼200	0.0008

Table 2: Learning rate selection.

4.6.3 Weight Decay

Weight decay represents the level of penalty applied to the weight matrix. It can be used to control overfitting and obtain numerically more stable solutions.

However, the training process of deep neural networks takes much time, which makes it time-consuming to do cross-reference. We came up with a hypothetical approach to tune the weight decay. When we were training the ResNet-18, I change the weight decay from 0.0005 to 0.005, and the accuracy is increased rapidly as shown in the Figure: 12 and Figure: 13.

```
Accuracy of the network on the train images: 98.940000 %
Training starts
[1,   100] double loss: 0.0002107
[1,   200] double loss: 0.0002712
[1,   300] double loss: 0.0003515
[2,   100] double loss: 0.0003236
[2,   200] double loss: 0.0003623
[2,   300] double loss: 0.0003768
[3,   100] double loss: 0.0003597
[3,   200] double loss: 0.0002927
[3,   300] double loss: 0.0003285
[4,   100] double loss: 0.0003377
[4,   200] double loss: 0.0003403
[4,   300] double loss: 0.0003661
[5,   100] double loss: 0.0003579
[5,   200] double loss: 0.0003749
[5,   300] double loss: 0.0003645
Accuracy of the network on the 10000 test images: 90.610000 %
Accuracy of the network on the train images: 98.674000 %
```

Figure 12: Before tuning weight decay during training.

```
In [75]: for i in range(2):
             optimizer = optim.SGD(net.parameters(), lr=1e-3, momentum=0.9, weight_decay=5e-3)
             trainNN(net, trainloader, optimizer, criterion,  5)
             print('Accuracy of the network on the 10000 test images: %f %%' % (
                 100 * cal_accuracy(net, testloader)))
             print('Accuracy of the network on the train images: %f %%' % (
                 100 * cal_accuracy(net, trainloader)))

Training starts
[1,   100] double loss: 0.0002103
[1,   200] double loss: 0.0001714
[1,   300] double loss: 0.0001436
[2,   100] double loss: 0.0000891
[2,   200] double loss: 0.0000968
[2,   300] double loss: 0.0000898
[3,   100] double loss: 0.0000735
[3,   200] double loss: 0.0000786
[3,   300] double loss: 0.0000756
[4,   100] double loss: 0.0000674
[4,   200] double loss: 0.0000728
[4,   300] double loss: 0.0000696
[5,   100] double loss: 0.0000692
[5,   200] double loss: 0.0000634
[5,   300] double loss: 0.0000656
Accuracy of the network on the 10000 test images: 92.920000 %
Accuracy of the network on the train images: 99.906000 %
```

Figure 13: After tuning weight decay during training.

Therefore, we think it is possible to use cross-validation for a few epochs after the loss function, which uses the initial weight decay is converged. The mathematical assumption behind the method is that the global minimum is fixed for the same weight decay value, and the loss function after changing (the weight decay) will not converge at the current local minimum. Due to the deadline of the report, we did not try this cross-validation method in practice, but we think it is worth to try later.

4.6.4 Batch Size

For batched gradient descent, using more data items for each step can increase the stability of training and reduce the number of steps to get the optimal answer. Without considering the resources and time needed, increasing batch sizes will generally generate better result in a more stable manner. However, as the number of items in one batch increase, the speed-up effect will become smaller, and the amount of graphics RAM space needed increases. Therefore, it is important to balance the two factors. Generally, we select a batch size that can fit into the graphics memory of our display cards from the following values: 64, 128 or 256. For the batch size used in testing, we will choose from the following values: 1,500, 3,000, 4,000, 6,000, and 12,000 to fill the display card's memory.

5 Process of Reproducing the Best Result

For the process of training, we refer to the file structure and implementation of [4], [7], and [9].

5.1 Data Loading

We use the training set of CIFAR-10 to train the model and do validation, and use the test set to test the final performance of the model. There are $50,000$ images in the training set; we split it into $40,000$ training set and $10,000$ validation set. (Figure: 14)

```
In [6]: # split the training set into training and validation set
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform_train)
[trainset, validationset]= torch.utils.data.random_split(trainset, [40000, 10000])

trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,
                                          shuffle=True, num_workers=2)

validationloader = torch.utils.data.DataLoader(validationset, batch_size=100,
                                          shuffle=True, num_workers=2)


testdata = np.load("./test/y_test.npy").astype(np.uint8)

test_images = [ transform_test(img) for img in testdata]

testloader = torch.utils.data.DataLoader(test_images, batch_size=BATCH_SIZE,
                                          shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

Files already downloaded and verified

In [7]: trainX, trainY = from_cifar_to_np(trainloader)
if not use_forieum:
    trainX = np.reshape(trainX, (40000, 3072))
print("Shape of training data: ")
print(trainX.shape)
print(trainY.shape)

Shape of training data:
(40000, 3072)
(40000,)

In [8]: valX, valY = from_cifar_to_np(validationloader)
if not use_forieum:
    valX = np.reshape(valX, (10000, 3072))
print("Shape of validation data: ")
print(valX.shape)
print(valY.shape)

Shape of validation data:
(10000, 3072)
(10000,)
```

Figure 14: The procedure to split the training set and validation set.

5.2 Pre-processing

The training data is randomly cropped, horizontally flipped and normalised using the parameters $\mu = (0.4914, 0.4822, 0.4465)$, $\sigma = (0.2023, 0.1994, 0.2010)$. Besides, we performed some physical operations such as padding, cropping, flipping, and cut out (notice that the use_cutout is set to true).(Figrua: 15)

5.3 Model

5.3.1 Network Structure

The final network consist of several blocks and some house-keeping layers at the top and bottom. Each block consist of some sub-blocks. Each sub-block (layers in the diagram) has two parts, one main-circuit and one short-circuit. The main-circuit has two convolution layers, an activation layer (ReLU), a drop-out layer and another convolution layer. Meanwhile, if the input and output have the same dimension, the short-circuit part is simply an identity function (performing no transform), but when the dimensions are different, the short-circuit part will be a simple 1×1 convolution layer. (Figure: 16)

5.3.2 Activation Function

For simplicity, all activation functions in this network is ReLU.

```
In [5]: print('==> Preparing data..')


transform_train = transforms.Compose([
        transforms.ToTensor(),
        transforms.Lambda(lambda x: F.pad(x.unsqueeze(0), (4,4,4,4), mode="reflect").squeeze()),
        transforms.ToPILImage(),
        transforms.RandomCrop(32),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
    ])

transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
    ])

if use_cutout:
    transform_train.transforms.append(Cutout(n_holes=N_HOLES, length=LENGTH))

if use_forieum:
    transform_train.transforms.append(remove_high_freq)

if use_pca:
     transform_train.transforms.append(whitening)


==> Preparing data..
```

Figure 15: The pre-processing step.

5.3.3 Batch Normalization

Batch normalisation is performed in each network block to increase the speed of training.

5.4 Training

5.4.1 Loss function

For classification problems, there are many loss functions can be used, including hinge loss and cross-entropy loss. As shown in Figure: 19 both of them are the estimation of the zero-one loss for classification. Compared to zero-one loss and hinge loss, cross-entropy is more smooth, making it a better tool for gradient descent. Besides, ReLU is used for activation, adding softmax (which can be effectively handled by cross-entropy) at the final layer will map the output into the range between 0 and 1, which leads to better probability estimation at the cost of accuracy. Although hinge loss can lead to better accuracy and some sparsity at the cost of misclassifications, we have only ten classes in this problem, which means that the misclassification cases is not as much as classification task for more categories such as CIFAR-100.

Therefore, we choose the cross entropy loss to have a better estimation for probability. The cross entropy loss is defined as:

$$\frac{1}{m}\sum_{i=1}^{m} log\ p(Y = y^{(i)}|\mathbf{X} = \mathbf{x}^{(i)}) = \frac{\exp(\mathbf{W}_y\mathbf{x} + b_y)}{\sum_{y' \in \mathcal{Y}} exp(\mathbf{W}_{y'}\mathbf{x} + b_{y'})} \tag{4}$$

which defines multiple linear decision boundaries as following:

$$\begin{aligned} ln\ Pr(Y_i = 1) &= \beta_1 \mathbf{X}_i - ln\ Z \\ &... \\ ln\ Pr(Y_i = K) &= \beta_K \mathbf{X}_i - ln\ Z \end{aligned} \tag{5}$$

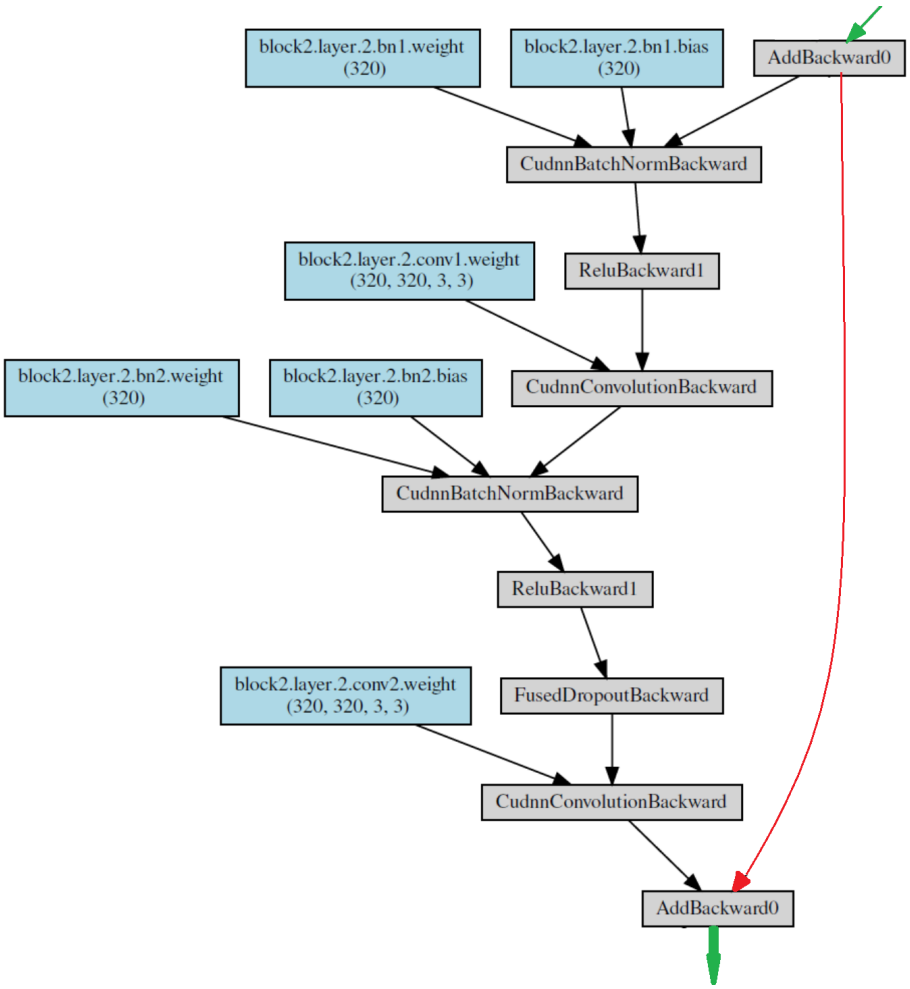

Figure 16: A typical sub-block of the final network, the short-circuit part is drawn in red colour and the input and output are drawn in green colour.

5.4.2 Optimizer and Optimisation

Please refer to the part of optimization and hyper-parameter tuning 4.6.

5.5 Performance Analysis

Figure 19 shows the changes of the loss function during the training process. There are three major stair-steps in this graph, corresponding to different learning rates respectively. Figure 20 showing the change of training and test accuracy also shows this trend. Figure 21 shows the accuracy of different classes, all of which are high.

6 Possible Improvements

Further increasing the size of the network may enhance the performance of the network. However, this needs more computation power and GPU RAM size. Changing the activation function may provide more chances of improvement to the network, but due to the limited time, we have not systematically attempted this yet.

7 Conclusion

During this project, we have attempted multiple machine learning methods to solve the famous CIFAR-10 image-recognition problem. During this project, we have found that neural networks have better performance on this problem, comparing to other traditional learning methods. Deep and thin ResNet with convolution layers and shallow and wide ResNet with convolution layers are proved to be good solutions to this problem by exploiting the locality of information stored in images, making the model more flexible and keeping a balance between time and size.

This project has helped us to understand the necessary procedure of training a deep neural network. We have a deeper understanding of why there are continuous needs for labours in the machine learning fields. This project has encouraged us to do more study in other fields of machine learning and other fields of computer science.

8 Appendices

8.1 Other worth mentioning networks trained in this project

Inspired by [3], the network in the file ReallyBig_block_backed-more_f_in_report is trained. This network takes only 1.5 minutes to train one epoch on GTX 970, and achieve 86 $\sim$ 90% testing accuracy. This network consist of several blocks (as in figure 22) and some house-keeping layers at the top and bottom.

References

[1] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images , CIFAR-10 data set*, 2009

[2] He, K & Zhang, X & Ren, S & Sun, J. *Identity Mappings in Deep Residual Networks.*, 2016

[3] Sergey Zagoruyko & Nikos Komodakis. *Wide Residual Networks.*, 2017

[4] Alex Lekov. *CIFAR-10 96% PyTorch wResNet 28x10 {SF} v3.2*, 2019

[5] Wikipedia. *Deep Learning*

[6] Fei-Fei Li, Justin Johnson & Serena Yeung. *CS231n: Convolutional Neural Networks for Visual Recognition*, 2019

[7] Sasank Chilamkurthy, et. al. *Deep Learning with PyTorch: A 60 Minute Blitz*. PyTorch, 2017.

[8] Sergey Ioffe & Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training byReducing Internal Covariate Shift*, 2015

[9] Kuang Liu. *Train CIFAR10 with PyTorch*, 2019

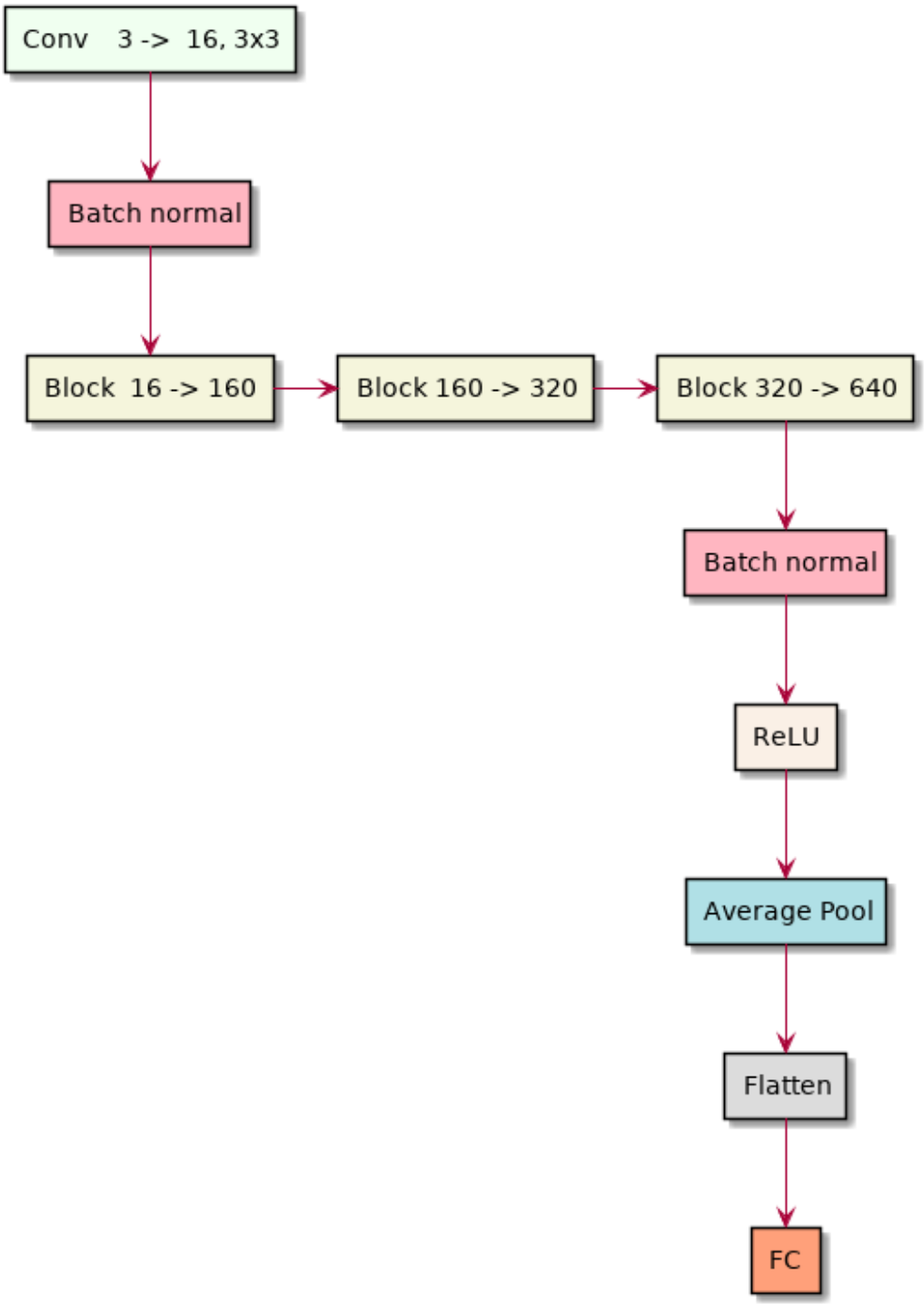

Figure 17: The overall structure of the network.

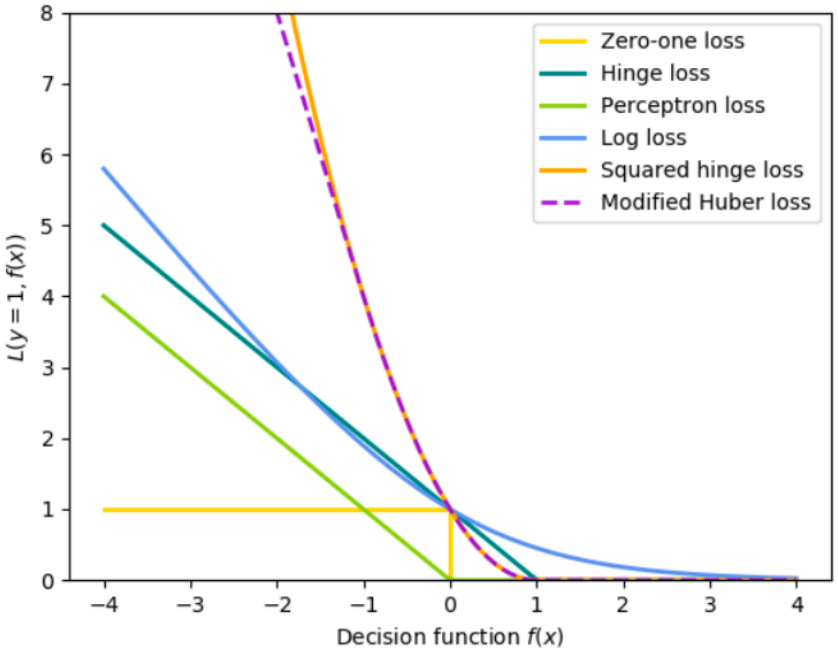

Figure 18: Loss functions for classification [from lecture note].

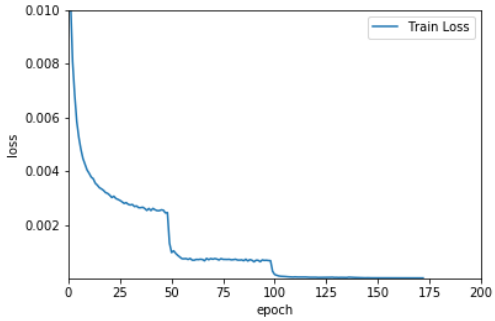

Figure 19: The change of loss function during the training process of Wide ResNet

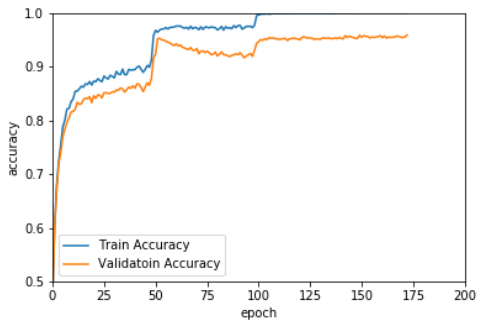

Figure 20: The change of training and test accuracy during the training process of Wide ResNet

Figure 21: The classification accuracy of different categories.

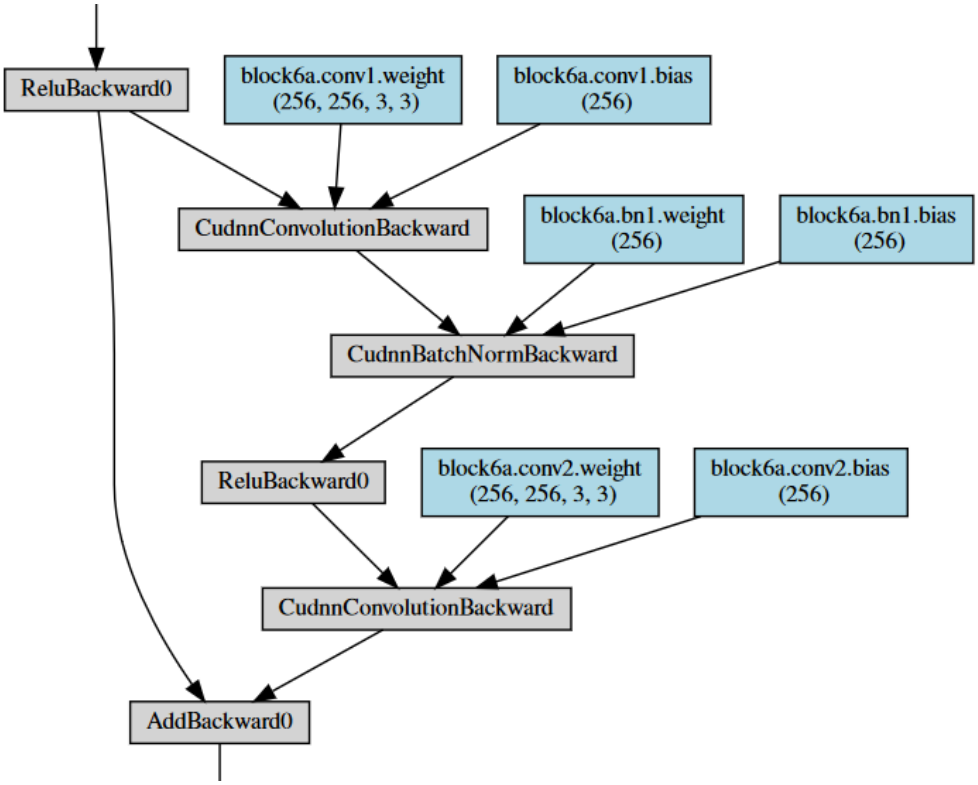

Figure 22: One typical block of the network.