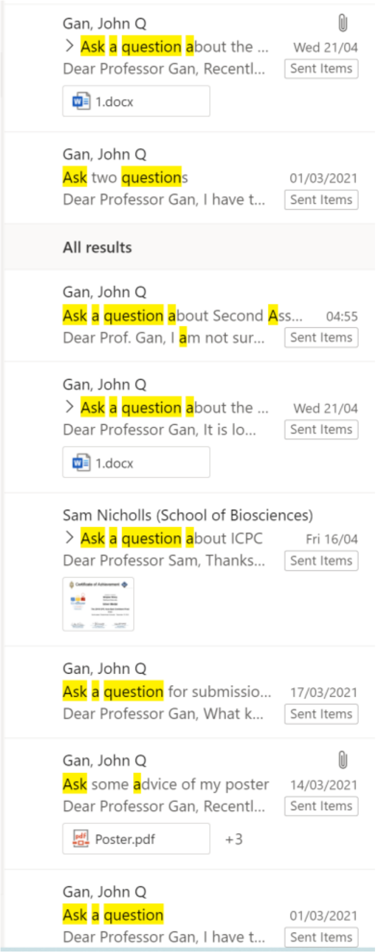
CE301 PDO

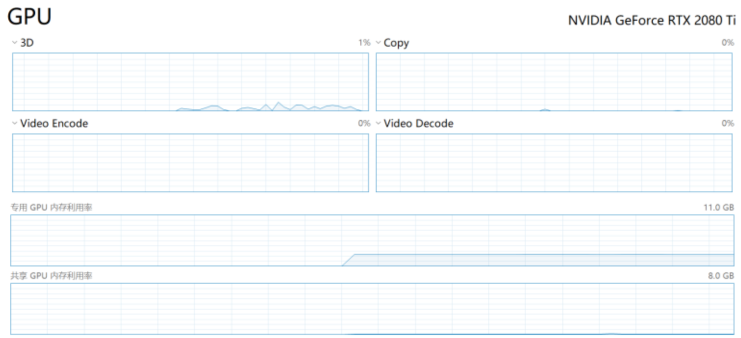
Image classification based on machine learning

Student Name: Wang, Binglun(bw17116@essex.ac.uk)
Register Number: 1708333
Degree Course: Electronic System Engineering(joint degree program with NWU)
Project Supervisor: Prof. John Gan
Second Assessor: Dr. Nick Zakhleniuk

Ackowledgements

- My deepest gratitude is first to Prof. Gan, my dearest supervisor, for his constant guidance in the past few months.
- I also want to thank Prof. Feng and Northwest University for supporting my study and project.
- Lastly, thanks for some my friends.



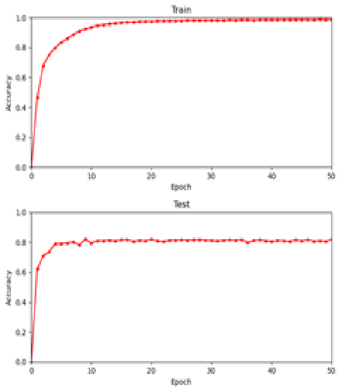


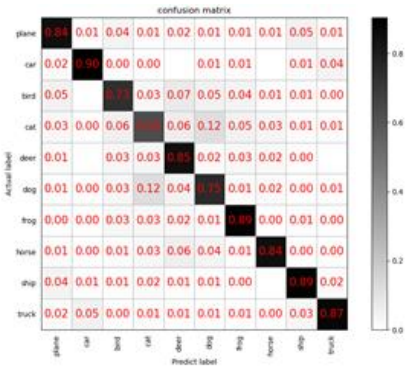
Introduction: Motivation

1. I love math and algorithms. I am good at mathematical theory, dynamic programming and graph theory. Some individuals recommend me to learn machine learning.
2. I have a lot of respect for one contestant of ICPC named Zhan Mingyuan, who is an Algorithm Researcher of computer vision in Sensetime. So, I try to follow his path.
3. AI is a relatively new field, I love exploring and challenges.

Project Objectives

- This project mainly summaries and investigates several significant aspects of modern convolutional neural networks (CNNs). Four typical neural networks are discussed in this project.
- Besides, the effectiveness of optimizers, data augmentation, dropout layer, and batch normalization are thoroughly analyzed with extensive experiments.
- Finally, a GUI system is created to visually compare results from different CNNs and classify photos in real time with trained models.

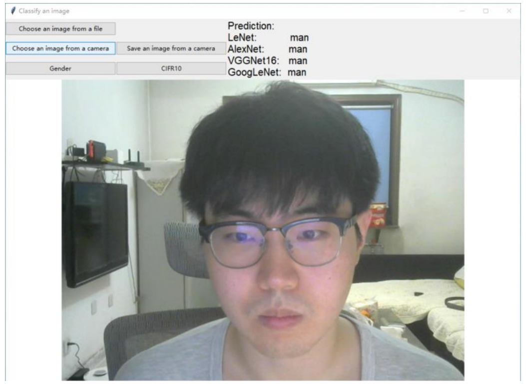


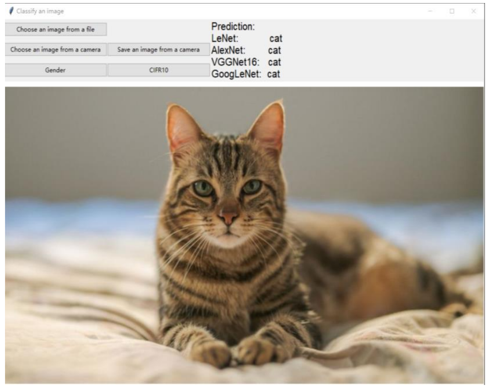


Results and Conclusions

1. With rational and cautious designs, deeper neural networks usually achieves better accuracy on both training set and test set.
2. Both dropout layer and data augmentation are beneficial to bridge the gap between training set and test set.
3. Batch normalization speed up the convergence of CNNs.
4. Optimizers have remarkable influence on the final performance.

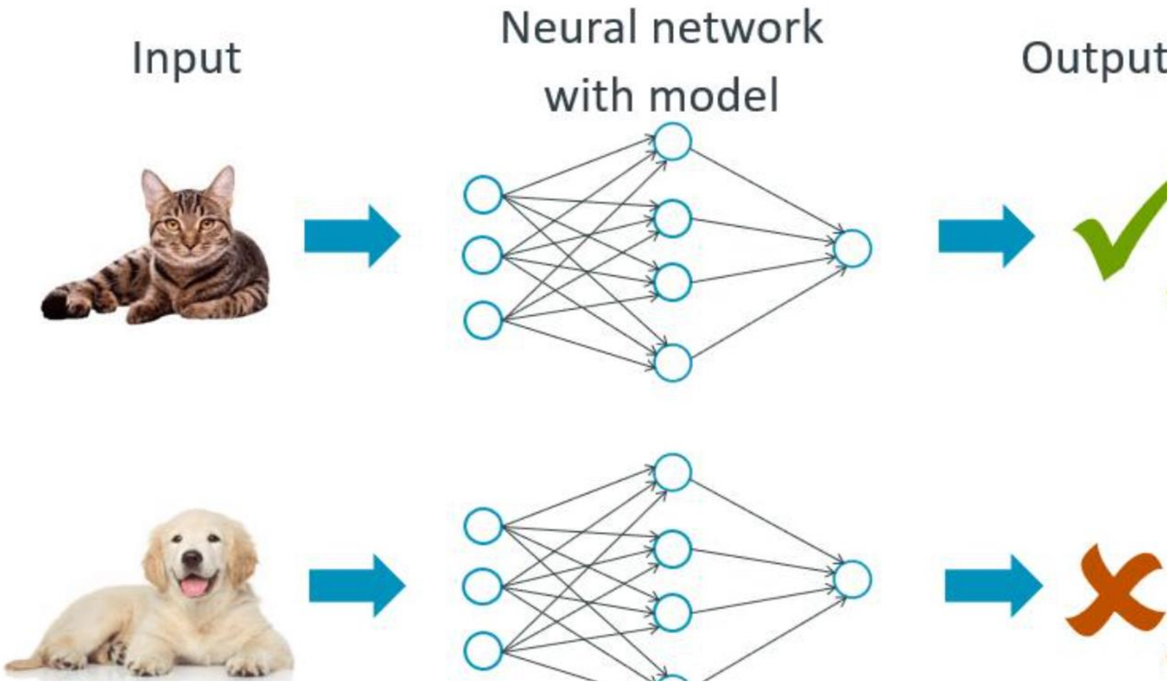
		Adam	Adamw	SGD
LeNet	Train Accuracy	98.60%	97.75%	100.00%
	Test Accuracy	69.38%	69.08%	69.33%
AlexNet	Train Accuracy	99.17%	98.83%	99.51%
	Test Accuracy	81.99%	81.95%	77.66%
GoogLeNet	Train Accuracy	98.08%	98.68%	97.83%
	Test Accuracy	87.06%	88.75%	86.75%





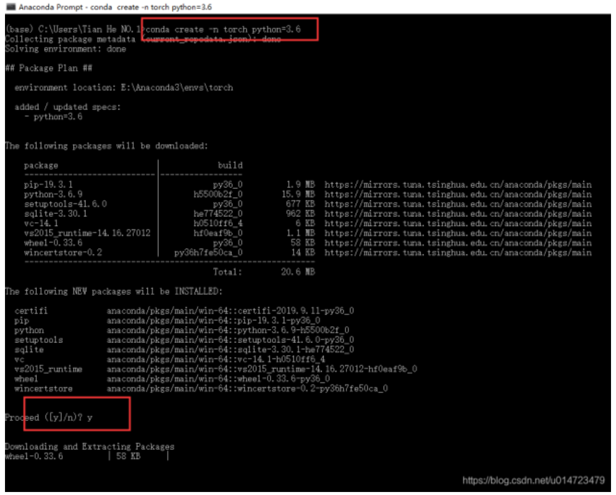
Background

- Image Classification
- Machine Learning



Platform

- PyTorch is an open source Python machine learning library, based on Torch
- Anaconda is very easy to configure your environment and the libraries you need.
- Tkinter is a library in Python. It is convenient for developing small-scale GUIs.

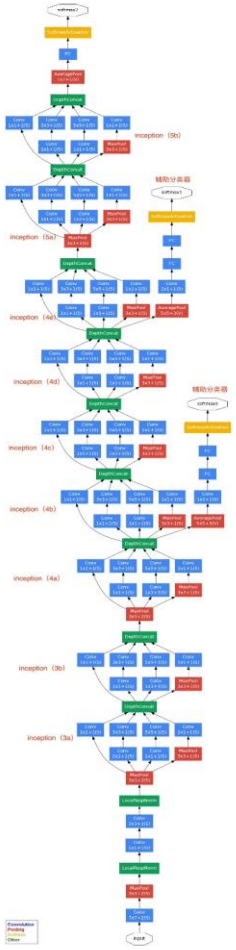


Implementation

Convolutional Neural Networks (4 models)

LeNet	AlexNet	VGGNet_11	VGGNet_13	VGGNet_16	VGGNet_19
input(32 * 32 RGB image)	input(224 * 224 RGB image)				
conv5-16	conv11-48	conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool(2, 2)	maxpool(3, 2)	maxpool(2, 2)			
conv5-32	conv5-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool(2, 2)	maxpool(3, 2)	maxpool(2, 2)			
	conv5-192	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
	conv5-192	maxpool(2, 2)			
	conv5-128	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
		maxpool(2, 2)			
	maxpool(3, 2)	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
		maxpool(2, 2)			
FC-120	FC-2048	FC-4096			
FC-84	FC-2048	FC-4096			
FC-10					
soft-max					

GoogLeNet:



Convolutional Neural Networks (4 models)

LeNet (AlexNet is almost same code)

```python
import torch.nn as nn
import torch.nn.functional as F
import torch

# Tensor [batch, channel, height, width]

class LeNet(nn.Module):
    def __init__(self, num_classes = 10, init_weights = False):
        super(LeNet, self).__init__()

        self.features = nn.Sequential(
            # (3, 32, 32)
            nn.Conv2d(3, 16, kernel_size = 5),  # (6, 28, 28)
            nn.ReLU(),  # inplace = True can Sacrificing time for memory
            nn.MaxPool2d(kernel_size = 2, stride = 2),  # (6, 14, 14)
            nn.Conv2d(16, 32, kernel_size = 5),  # (16, 10, 10)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2),  # (16, 5, 5)
        )

        self.classifier = nn.Sequential(
            nn.Dropout(p = 0.7),
            nn.Linear(32 * 5 * 5, 120),
            nn.ReLU(),
            nn.Dropout(p = 0.7),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, num_classes),
        )

        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = self.features(x)
        # (C, H, W) -> (C * H * W)
        x = torch.flatten(x, start_dim = 1)
        x = self.classifier(x)
        return x

    def _initialize_weights(self):
        for m in self.modules(): # iterator over all modules in the network
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode = 'fan_out', nonlinearity = 'relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)
```

VGGNet

```python
def make_features(cfg: list):
    layers = []
    in_channels = 3
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size = 2, stride = 2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size = 3, padding = 1)
            layers += [conv2d, nn.ReLU(True)]
            in_channels = v
    return nn.Sequential(*layers)


cfgs = {
    'vgg11': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'vgg13': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'vgg16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M'],
    'vgg19': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M'],
}

def vgg(model_name = "vgg16", **kwargs):
    try:
        cfg = cfgs[model_name]
    except:
        print("Warning: model number {} not in cfgs dict!".format(model_name))
        exit(-1)
    model = VGG(make_features(cfg), **kwargs)
    return model
```

GoogLeNet

```python
class Inception(nn.Module):
    def __init__(self, in_channels, ch1x1, ch3x3red, ch3x3, ch5x5red, ch5x5, pool_proj):
        super(Inception, self).__init__()

        self.branch1 = BasicConv2d(in_channels, ch1x1, kernel_size=1)

        self.branch2 = nn.Sequential(
            BasicConv2d(in_channels, ch3x3red, kernel_size=1),
            BasicConv2d(ch3x3red, ch3x3, kernel_size=3, padding=1)   # 保证输出大小等于输入大小
        )

        self.branch3 = nn.Sequential(
            BasicConv2d(in_channels, ch5x5red, kernel_size=1),
            BasicConv2d(ch5x5red, ch5x5, kernel_size=5, padding=2)   # 保证输出大小等于输入大小
        )

        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1, padding=1),
            BasicConv2d(in_channels, pool_proj, kernel_size=1)
        )

    def forward(self, x):
        branch1 = self.branch1(x)
        branch2 = self.branch2(x)
        branch3 = self.branch3(x)
        branch4 = self.branch4(x)

        outputs = [branch1, branch2, branch3, branch4]
        return torch.cat(outputs, 1)
```

```python
class InceptionAux(nn.Module):
    def __init__(self, in_channels, num_classes):
        super(InceptionAux, self).__init__()
        self.averagePool = nn.AvgPool2d(kernel_size=5, stride=3)
        self.conv = BasicConv2d(in_channels, 128, kernel_size=1)  # output[batch, 128, 4, 4]

        self.fc1 = nn.Linear(2048, 1024)
        self.fc2 = nn.Linear(1024, num_classes)

    def forward(self, x):
        # aux1: N x 512 x 14 x 14, aux2: N x 528 x 14 x 14
        x = self.averagePool(x)
        # aux1: N x 512 x 4 x 4, aux2: N x 528 x 4 x 4
        x = self.conv(x)
        # N x 128 x 4 x 4
        x = torch.flatten(x, 1)
        x = F.dropout(x, 0.5, training=self.training)
        # N x 2048
        x = F.relu(self.fc1(x), inplace=True)
        x = F.dropout(x, 0.5, training=self.training)
        # N x 1024
        x = self.fc2(x)
        # N x num_classes
        return x


class BasicConv2d(nn.Module):
    def __init__(self, in_channels, out_channels, **kwargs):
        super(BasicConv2d, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, **kwargs)
        self.relu = nn.ReLU()
        # self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        return x
```

Training Code

1. Open Dataset & Data Augmentation

```
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                             download=False, transform=data_transform["train"])

# train_dataset = datasets.ImageFolder(root = './data',
#                                      transform = data_transform["train"])
train_num = len(train_dataset)


batch_size = 32
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,
                                           shuffle=True, num_workers=0)
# Windows: num_workers must be 0

validate_dataset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                                download=False, transform=data_transform["val"])

# validate_dataset = datasets.ImageFolder(root = image_path + "/val",
#                                         transform = data_transform["val"])
val_num = len(validate_dataset)
validate_loader = torch.utils.data.DataLoader(validate_dataset, batch_size=batch_size,
                                              shuffle=False, num_workers=0)
```

```
data_transform = {
    "train": transforms.Compose([
        transforms.Resize((224, 224)),
        # transforms.RandomResizedCrop(224),  # random crop
        transforms.RandomHorizontalFlip(),  # random reverse
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ]),

    "val": transforms.Compose([
        transforms.Resize((224, 224)),  # cannot 224, must (224, 224)
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
}
```

2. Train & Test

```
net = AlexNet(num_classes=10, init_weights=True)

net.to(device)
loss_function = nn.CrossEntropyLoss()

optimizer = optim.Adam(net.parameters(), lr=0.0002)

save_path = './AlexNet.pth'
best_acc = 0.0
best_acc1 = 0.0
best_acc5 = 0.0
Tbest_acc = 0.0
Tbest_acc1 = 0.0
Tbest_acc5 = 0.0
T1 = 0.0
T5 = 0.0
T10 = 0.0
EPOCH = 90
```

```
for epoch in range(EPOCH):
    # tarin exite dropout
    running_loss = 0.0
    t1 = time.perf_counter()
    acc2 = 0
    for step, data in enumerate(train_loader, start=0):
        images, labels = data
        optimizer.zero_grad()
        outputs = net(images.to(device))  # to GPU or CPU
        predict_y = torch.max(outputs, dim=1)[1]
        acc2 += (predict_y == labels.to(device)).sum().item()

        loss = loss_function(outputs, labels.to(device))
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

        rate = (step + 1) / len(train_loader)
        a = "*" * int(rate * 50)
        b = "." * int((1 - rate) * 50)
        print("\rtrain loss: {:^3.0f}%[{}->{}]{:.3f}".format(int(rate * 100), a, b, loss), end="")
    print()

    train_accurate = acc2 / train_num
    if train_accurate > Tbest_acc:
        Tbest_acc = train_accurate
        torch.save(net.state_dict(), save_path)
    if epoch == 9:
        Tbest_acc1 = Tbest_acc
    elif epoch >= 10 and epoch <= 19:
        Tbest_acc5 = max(Tbest_acc5, Tbest_acc)

    if epoch <= 9:
        T1 += time.perf_counter() - t1
    elif epoch >= 10 and epoch <= 19:
        T5 += time.perf_counter() - t1
    else:
        T10 += time.perf_counter() - t1
```

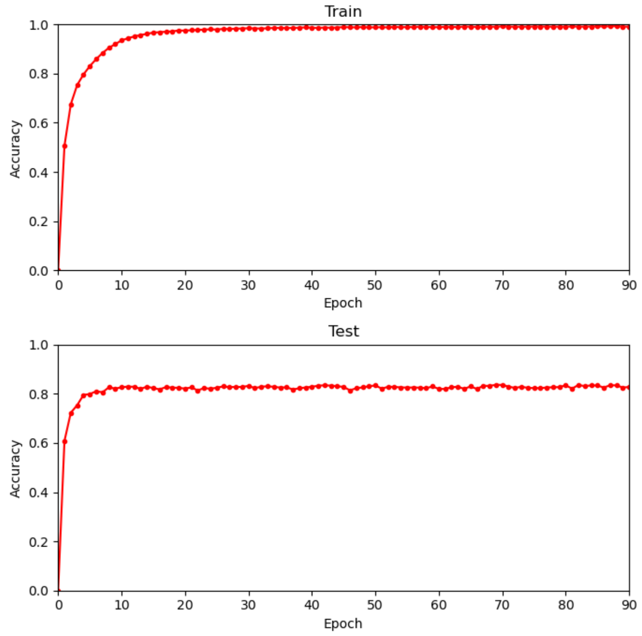
3. Confusion Matrix & Accuracy Curve

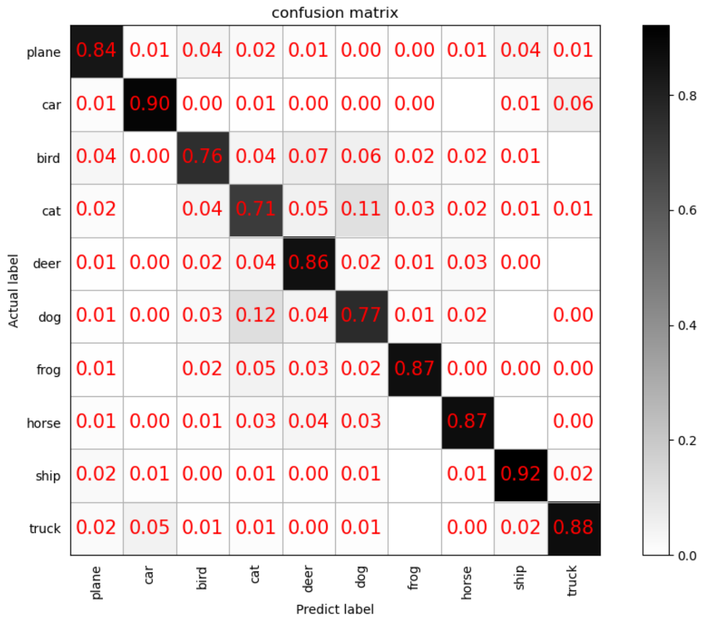
```
def plot_confusion_matrix(cm, savename, title='Confusion Matrix'):
    plt.figure(figsize=(12, 8), dpi=100)
    np.set_printoptions(precision=2)

    # 在混淆矩阵中每格的概率值
    ind_array = np.arange(len(classes))
    x, y = np.meshgrid(ind_array, ind_array)
    for x_val, y_val in zip(x.flatten(), y.flatten()):
        c = cm[y_val][x_val]
        if c > 0.001:
            plt.text(x_val, y_val, "%0.2f" % (c,), color='red', fontsize=15, va='center', ha='center')

    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.binary)
    plt.title(title)
    plt.colorbar()
    xlocations = np.array(range(len(classes)))
    plt.xticks(xlocations, classes, rotation=90)
    plt.yticks(xlocations, classes)
    plt.ylabel('Actual label')
    plt.xlabel('Predict label')

    # offset the tick
    tick_marks = np.array(range(len(classes))) + 0.5
    plt.gca().set_xticks(tick_marks, minor=True)
    plt.gca().set_yticks(tick_marks, minor=True)
    plt.gca().xaxis.set_ticks_position('none')
    plt.gca().yaxis.set_ticks_position('none')
    plt.grid(True, which='minor', linestyle='-')
    plt.gcf().subplots_adjust(bottom=0.15)

    # show confusion matrix
    plt.savefig(savename, format='png')
    plt.show()
```

```
fig, axes = plt.subplots(2, 1, figsize=(7, 7))
fig.subplots_adjust(wspace=0.5, hspace=0.3,
                    left=0.125, right=0.9,
                    top=0.9,    bottom=0.1)
axes[0].set(xlim=[0, EPOCH], ylim=[0, 1.0], title='Train', ylabel='Accuracy', xlabel='Epoch')
axes[1].set(xlim=[0, EPOCH], ylim=[0, 1.0], title='Test', ylabel='Accuracy', xlabel='Epoch')
x = np.arange(EPOCH + 1)
train_y = []
test_y = []
train_y.append(0)
test_y.append(0)
```

```
        test_y.append(val_accurate)
        train_y.append(train_accurate)

axes[0].plot(x, train_y, color='red', marker='.')
axes[1].plot(x, test_y, color='red', marker='.')
fig.tight_layout() #自动调整布局，使标题之间不重叠
plt.show()
```

Training Output

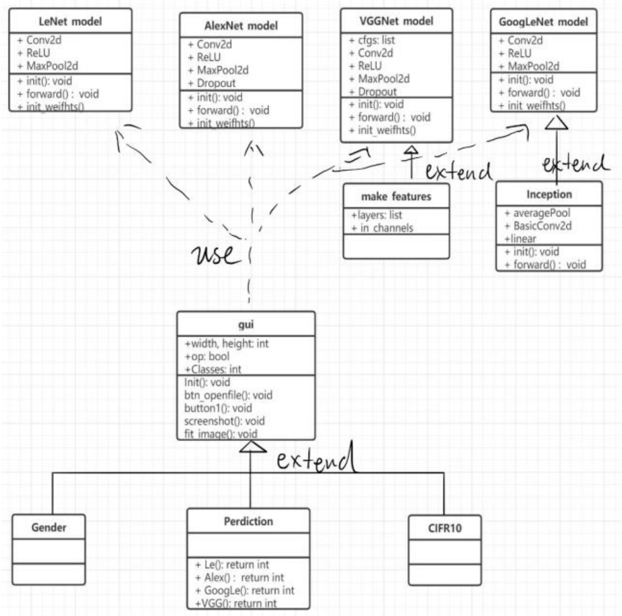




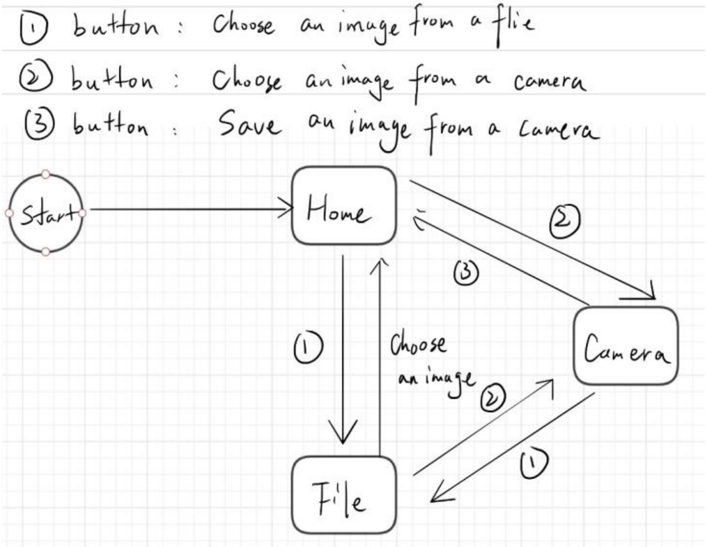

```
T20 : 1397.4328
T30 : 6209.3132
Test
A10 : 0.8270
A20 : 0.8295
A30 : 0.8370
Training
A10 : 0.9352
A20 : 0.9755
A30 : 0.9932
[[841, 10, 39, 24,
[[0.841, 0.01, 0.03
```

GUI

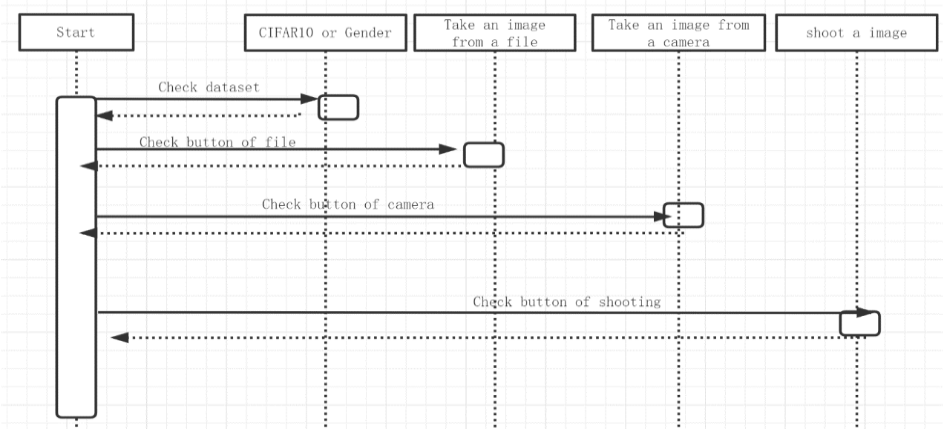
- I use tkinter library to build GUI system as mentioned. There are UML graphical representation of the system.



Class Diagram



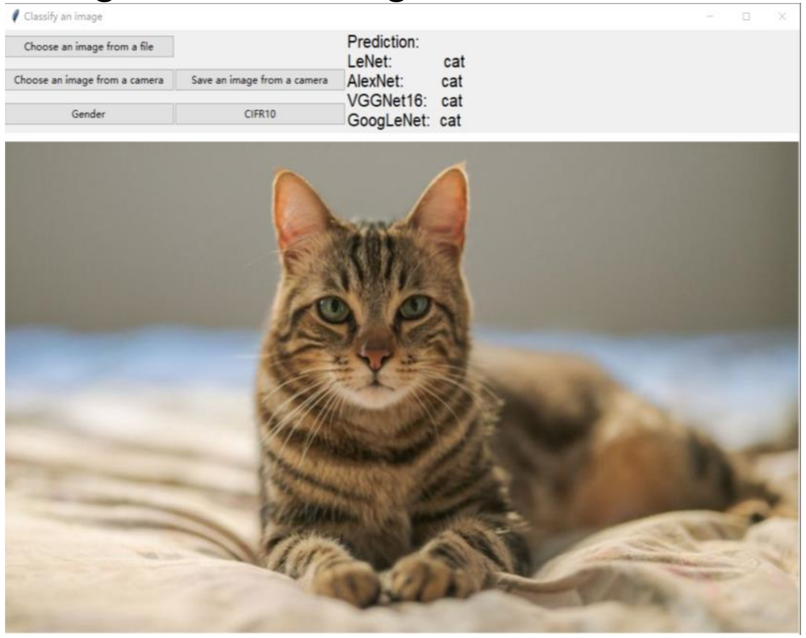
State machine diagram



Sequence diagram

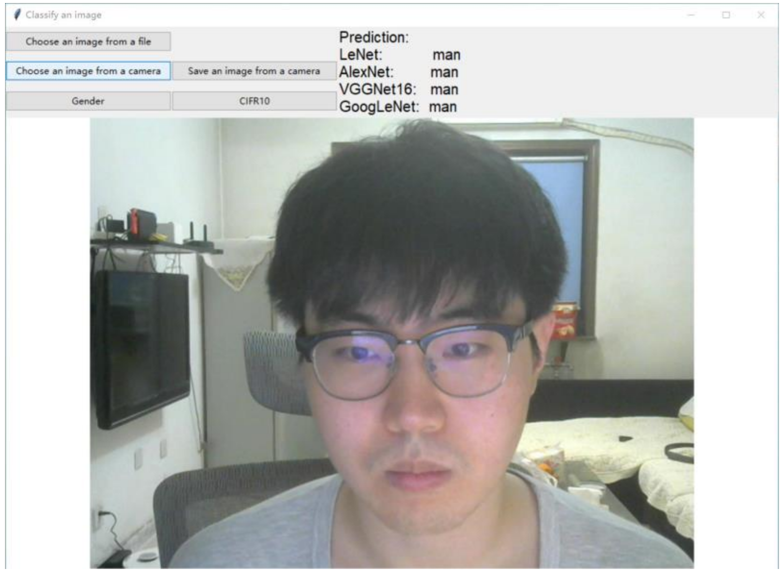
GUI

1) Choose an image from a local file and categorize it through trained CNNs.



2) Capture images from the real-time camera and categorize them.

3) Customized training set and test set.



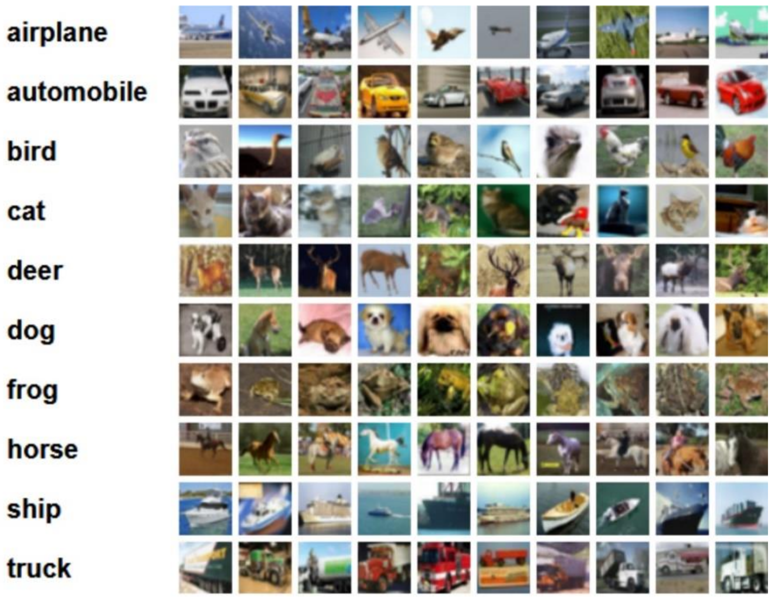
(This figure shows a gender classification from a real-time photo)

Experimental Investigation

Data Set

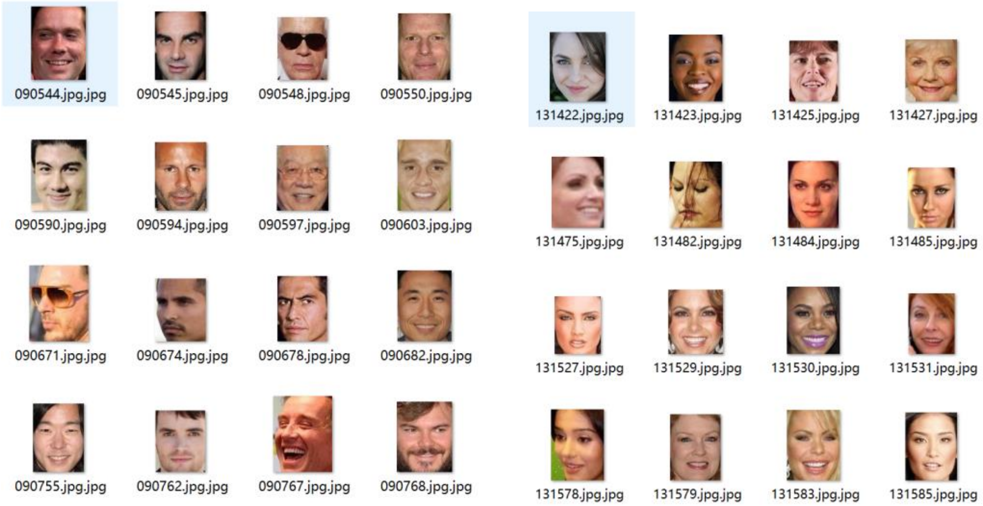
CIFAR10

- 10 Classes
- 50k Training images
- 10k Test images



Gender

- 2 Classes
- 47009 Training images
- 11649 Test images



Data Set

Best performance in CIFAR10 of 4 CNNs

	Optimizer	Train Accuracy	Test Accuracy
LeNet	Adam	96.06%	70.61%
AlexNet	AdamW	99.32%	83.70%
VGGNet	AdamW	99.80%	84.07%
GoogLeNet	AdamW	99.68%	90.13%

Best performance in Gender Dataset of 4 CNNs

	Optimizer	Train Accuracy	Test Accuracy
LeNet	SGD	97.41%	96.11%
AlexNet	AdamW	99.54%	97.21%
VGGNet	AdamW	97.76%	97.09%
GooLeNet	AdamW	99.82%	97.38%

1. Different Optimizers

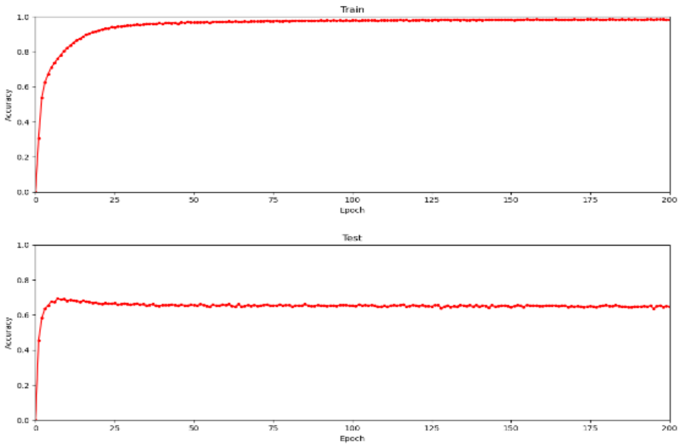
In my experiment I test for the performance of different networks in CIFAR10 dataset with Adam, AdamW and SGD optimizers.

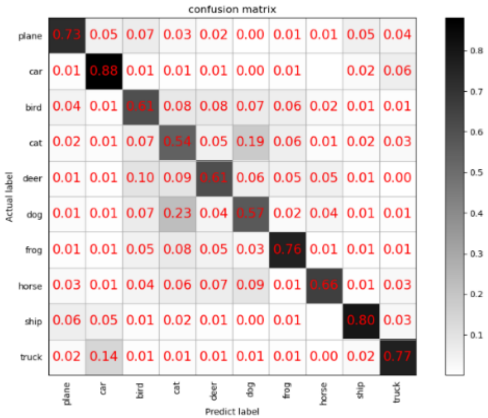
		Adam	Adamw	SGD
LeNet	Train Accuracy	98.60%	97.75%	100.00%
	Test Accuracy	69.38%	69.08%	69.33%
AlexNet	Train Accuracy	99.17%	98.83%	99.51%
	Test Accuracy	81.99%	81.95%	77.66%
GoogLeNet	Train Accuracy	98.08%	98.68%	99.56%
	Test Accuracy	87.06%	88.75%	84.98%
VGGNet	Train Accuracy	99.80%	99.80%	100.00%
	Test Accuracy	83.86%	84.07%	67.61%

1. Different Optimizers

The best choice of LeNet is Adam.

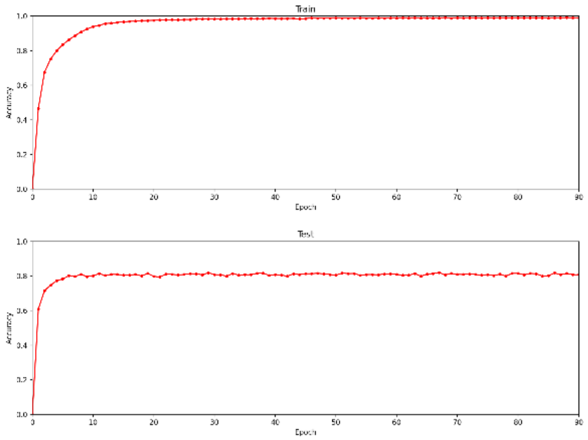
- Training accuracy: 98.63% Test accuracy: 69.38%
- From confusion matrix, LeNet is not good at cat pics and dog pics classification.
- Accuracy of cat is just 54%, and that of dog accuracy is just 57%.
- For test accuracy, there is a slight drop from 5 epochs in 68.80% to 25 epochs in almost 65.00%, after that, it still constant.

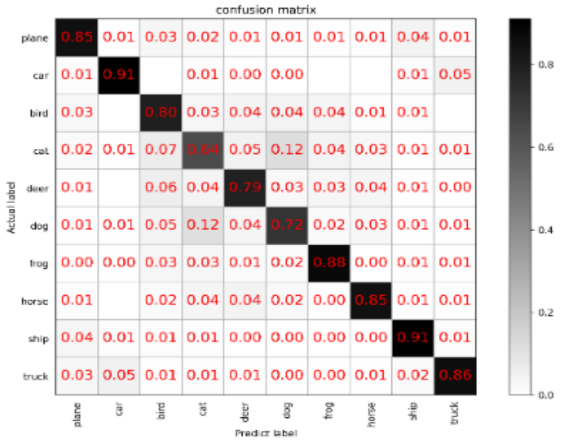




1. Different Optimizers

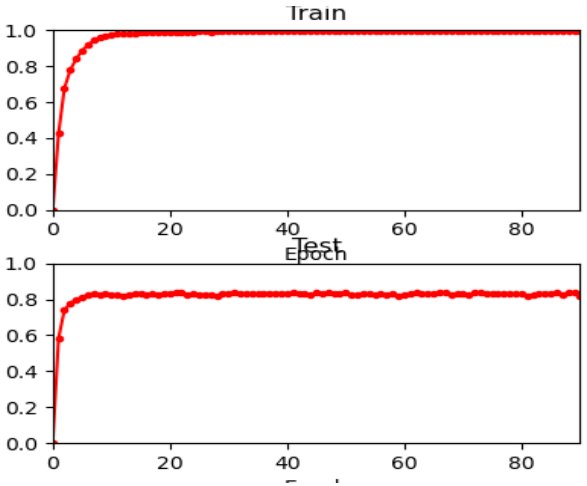
- The best choice of AlexNet also is Adam.
- Train Accuracy: 99.17% Test Accuracy: 81.99%
- From confusion matrix, I got AlexNet are not good at cat pics and dog pics classification. Cat accuracy is just 64%. And dog accuracy is just 72%.
- But, AlexNet are good at car and ship pics classification. They have the same accuracy: 91%

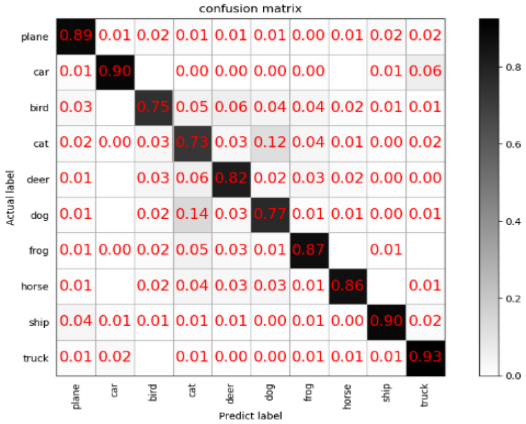




1. Different Optimizers

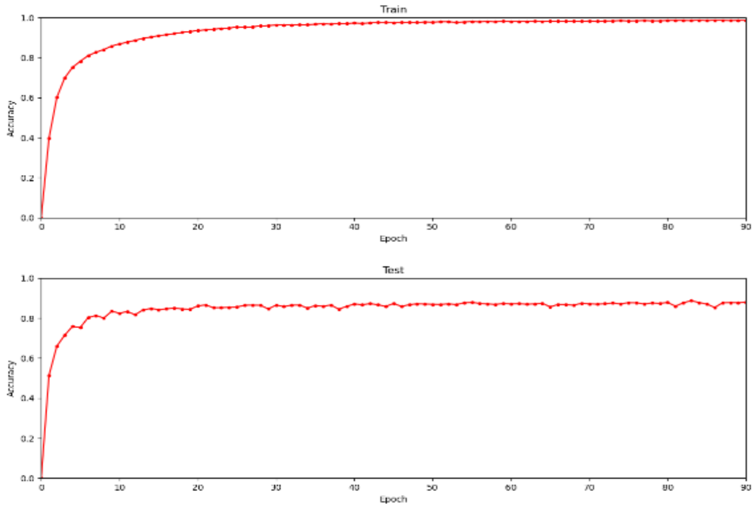
- The best choice of VGGNet16 is AdamW.
- Train Accuracy: 99.80% Test Accuracy: 84.07%
- From confusion matrix, I got VGGNet are not good at cat pics and bird pics classification. Cat accuracy is just 73%. And dog accuracy is just 75%.
- But, AlexNet are good at car, ship and ship pics classification. Their accuracy all higher than 90%.

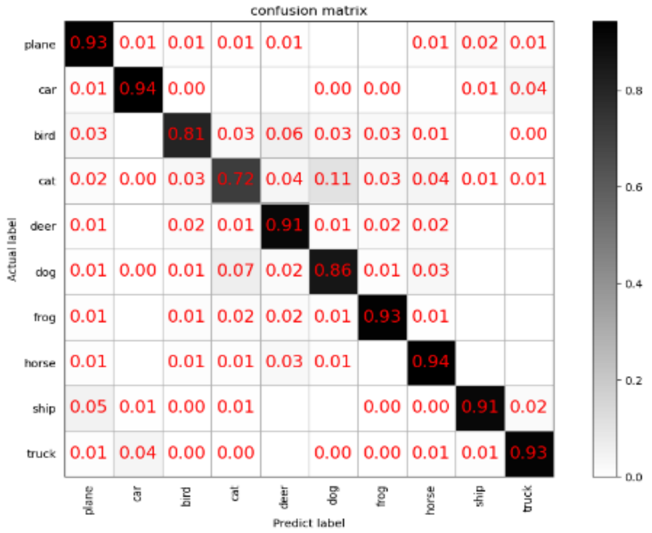




1. Different Optimizers

- The best choice of GoogLeNet is AdamW.
- Train Accuracy: 98.68% Test Accuracy: 88.75%
- From confusion matrix, I got GoogLeNet are not good at cat pics classification. Cat accuracy is just 72%.





- With rational and cautious designs, deeper neural networks usually achieves better accuracy on both training set and test set.

If a network can divide the input space more and more densely, it can fit more complicated functions or probability distributions. Therefore, the number of linear regions in the fixed input space of a neural network reflects the complexity of the function expressed by the network from one side; the maximum number of divisions that can be obtained in the input space also reflects the structure of a certain type of architecture.

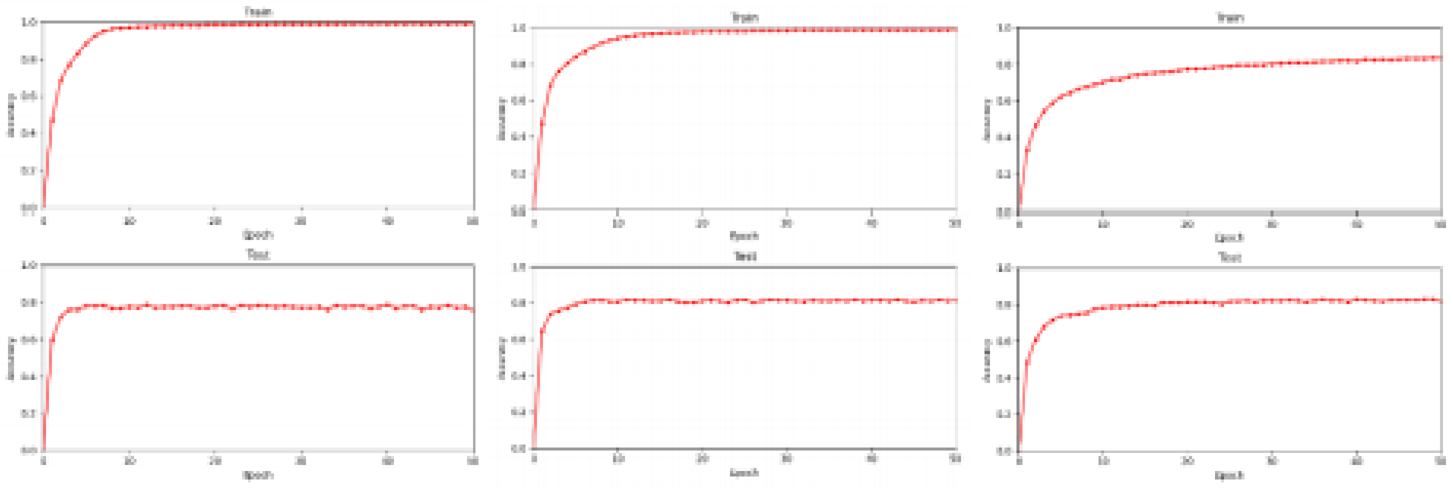
- Optimizers have remarkable influence on the final performance.

The purpose of training a neural network is to find suitable parameters for the network to adapt to the data set. Different optimizers are to make the loss function as small as possible. Therefore, different optimizers perform differently in different networks.

2. Data Augmentation

I test AlexNet in CIFAR 10 with different data augmentation in 50 epochs.

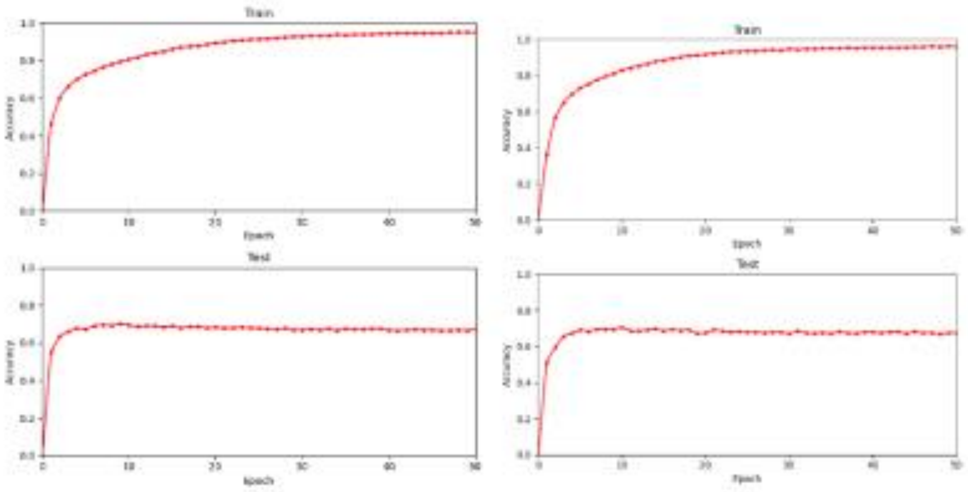
	Without	Random Flip	Random Crop
Train Accuracy	99.23%	98.83%	83.00%
Test Accuracy	78.79%	81.92%	83.84%



3. Dropout layers

Before AlexNet, there is no dropout technique. So, my experiment in this part is adding some dropout layers in LeNet.

	Without	Dropout 0.3	Dropout 0.45	Dropout 0.6	Dropout 0.55
Train Accuracy	95.33%	95.70%	95.65%	96.36%	96.06%
Test Accuracy	70.17%	68.94%	69.61%	69.78%	70.61%



- Both dropout layer and data augmentation are beneficial to bridge the gap between training set and test set.
- Firstly, because the training data cannot represent all the data, the training network cannot rely too much on the data set. Dropout can reduce the co-adaptation relationship between neurons, and discarding some information randomly can make the network more versatile. For example, biological inheritance, it is precisely because of the existence of genetic mutations that organisms continue to survive in the face of environmental changes.

- Both dropout layer and data augmentation are beneficial to bridge the gap between training set and test set.
- Secondly, in machine learning, training data set is required a lot. If we use data augmentation, it can play a role in expanding the data set. For a large number of data sets, it still has many functions. For example, there is a large dataset which has two different classes. In the first class the head of the car is all to the left, and the second is just the opposite. So, for a car with the right-turning head, this network will easily misjudge it as the second class. At this time, data augmentation played a big role.

4. Batch Normalization

In paper of ResNet, there is a new technique called Batch Normalization. I test AlexNet with this technique in CIFAR 10 dataset.

	Without	BN
Train Accuracy	99.17%	99.32%
Test Accuracy	81.99%	83.70%

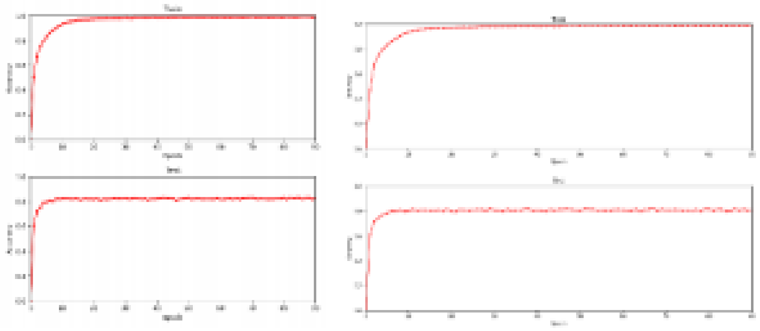


Figure 5.13 Without & BN (from left to right)

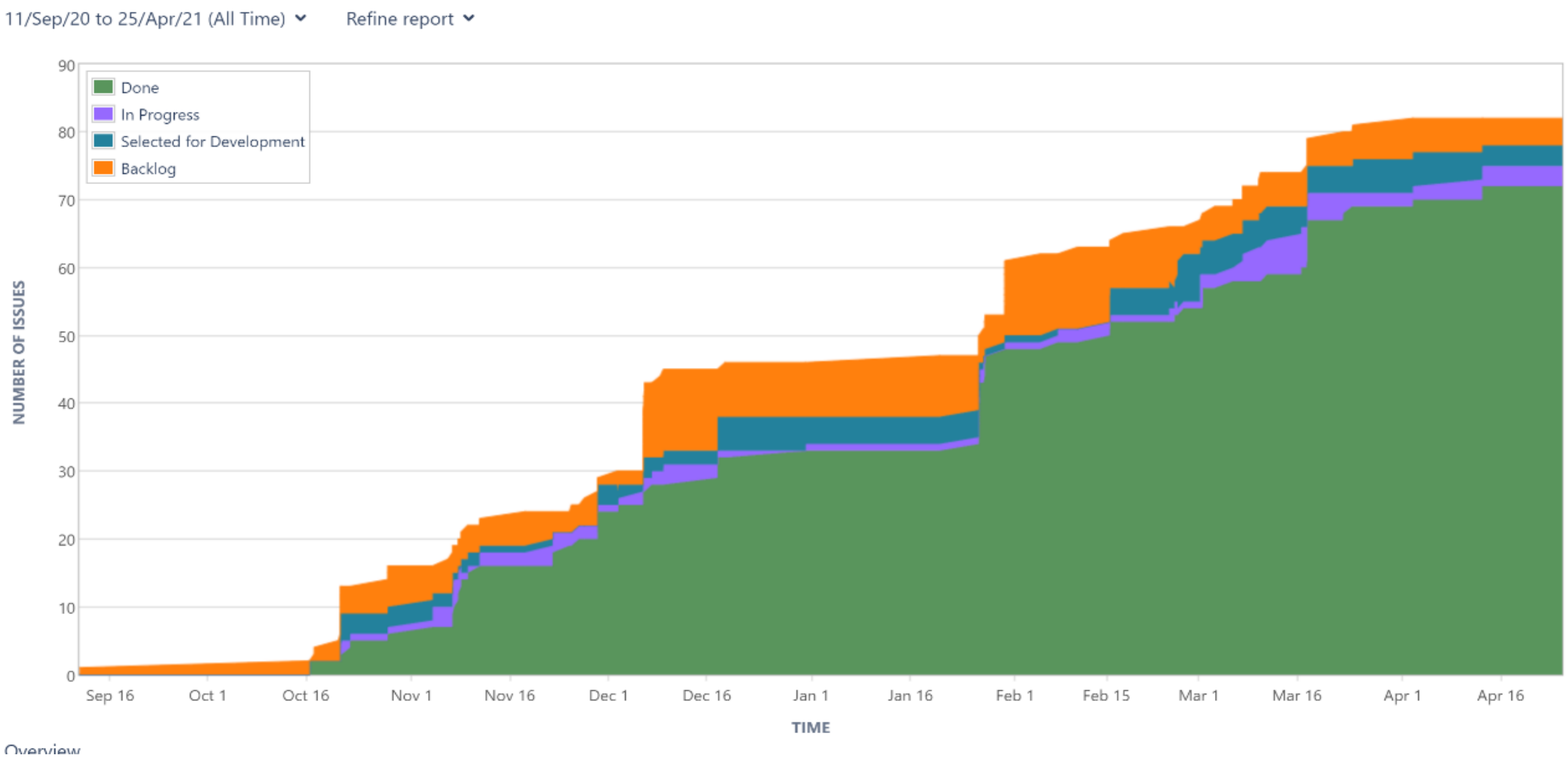
- Batch normalization speed up the convergence of CNNs.

- After the data is normalized and standardized, the solution speed of gradient descent can be accelerated, which makes it possible to use a larger learning rate for more stable gradient propagation, and even increase the generalization ability of the network.

Demo

Jira

My project is basically divided into two stages: the first stage covered three months from September 16 to December 16; the second stage covered another three months from January 20 to April 20.

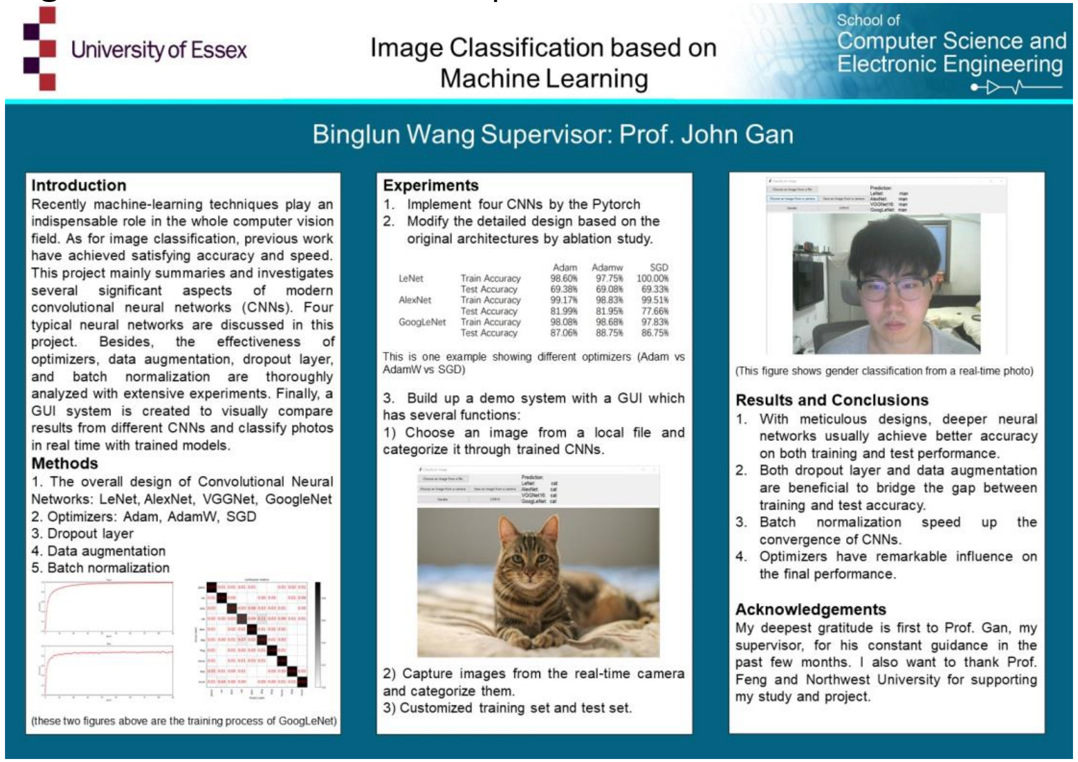


Gitlab

This image below is my Gitlab, where you can see my code of different networks and GUI, with records of my experiments and technical documents.

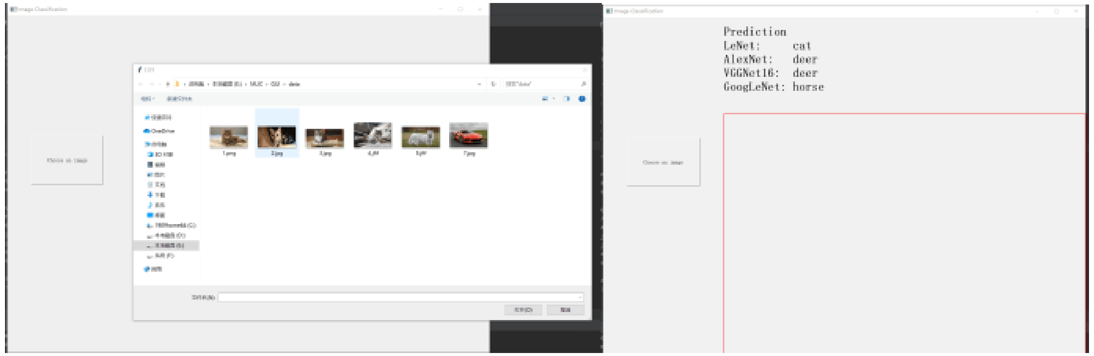
Name	Last commit	Last update
AlexNet	Upload New File	4 months ago
DataSet	Add new directory	4 months ago
GUI	Upload New File	1 month ago
GoogLeNet	Upload New File	4 months ago
LeNet	Upload New File	4 months ago
Learning Resources	Add new directory	4 months ago
Poster and Abstract	Upload New File	1 month ago
Some General Code for Experime...	This code can draw images of the confusion ...	4 months ago
VGG	This code is VGGNet	4 months ago
Week 11	Upload New File	4 months ago
Week2(Challenge Week)	Upload New File	4 months ago
Weekly_Report	Upload New File	1 month ago

To sum up, in this project, I compared four different networks, namely, LeNet, AlexNet, VGGNet, GoogleNet, modified the detailed design based on the original architectures by ablation study, and construct a GUI system to visually compare resultsfrom different CNNs and classify photos in real time with trained models. Through training on CIFER10 and gender dataset, there are four conclusions: deeper neural networks usually achieve better accuracy on both training and test performance, both dropout layer and data augmentation are beneficial to bridge the gap between training and test accuracy, batch normalization speed up the convergence of CNNs, and optimizers have remarkable influence on the final performance.



In doing these experiments I encountered many obstacles, and I would like to list some of the most representative ones and offer some solutions.

1. The first thing is I used QT Library in Python to construct GUI initially, but I found it was not compatible with the environment of Pytorch which led to many bugs, so tkinter is a good choice in alternating QT Library.



2. The second thing is that training networks can cost a lot of time, so selecting a small amount of datasets or epochs in test will save time in case there are some bugs in your code.

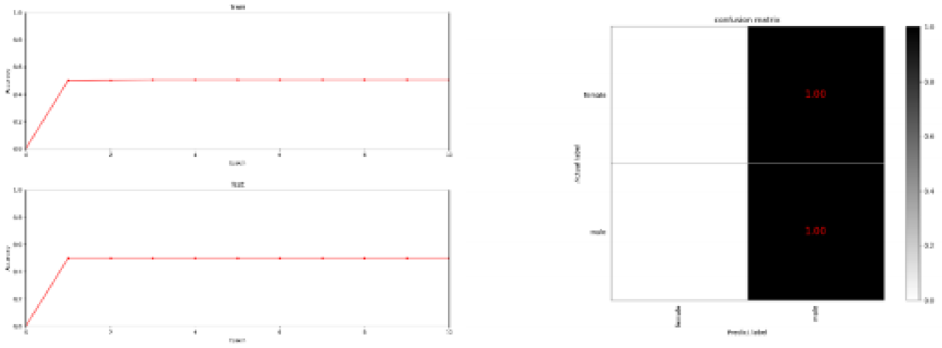


Figure 6.2 The incorrect performance

There are some limitations in my project:

- The networks and datasets that I tested are not enough due to my poor GPU.
- The complement of my project is not an efficient process due to my limited ability, and the lack of more new techniques to do experiments.
- Most importantly, universal image classification has been very matured, thus it would be better if I can focus on more specific datasets such as medical images. Therefore, in the future, I will use better equipment to continue my study.

Reference

[1] Alex Krizhevsky; Ilya Sutskever; Geoffrey E. Hinton;"ImageNet Classification with Deep Convolutional Neural Networks".

[2] Python; anaconda; Wikipedia

[3] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; Jackel, L. D. (December 1989). "Backpropagation Applied to Handwritten Zip Code Recognition". Neural Computation.

[4] Karen Simonyan ∗ & Andrew Zisserman + Visual Geometry Group, Department of Engineering Science, University of Oxford; "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION"

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich; "Going Deeper with Convolutions".

[6] CIFAR10, pytorch official tutorial

Thanks for Listening!