ECS 189G-001
Deep Learning

Winter 2023

Course Project: Stage 3 Report

Team Information

Enter Your Team Name Here (delete the extra rows if your team has less than 4 students)		
Name : Chung Ying Hsu	ID: 920918764	Email: cyhsu@ucdavis.edu

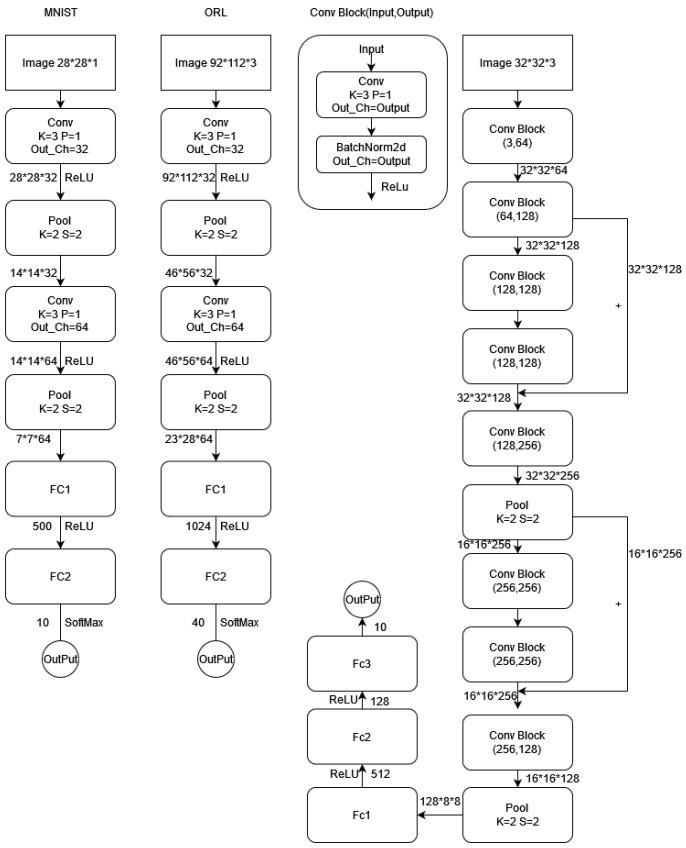
Section 1: Task Description

In this project, we need to train three different image datasets. These three datasets are ORL, CIFAR, and MNIST, each with different feature and label quantities. We need to use CNN models to train them and try to improve the test accuracy.

Section 2: Model Description

Link

Section 3: Experiment Settings



3.1 Dataset Description

MNIST dataset: MNIST is a dataset of handwritten digits used for training and testing image processing and machine learning models. It consists of a training set of 60,000 images and a test set of 10,000 images. Each image is a grayscale 28 x 28 pixel image of a handwritten digit.

ORL dataset: ORL is a face dataset that contains grayscale images of 40 subjects. Each subject has 10 images with different facial expressions, lighting conditions, and facial details. The images are 92 x 112 pixels in size. The test set size is 40 and the training set size is 360.

CIFAR dataset: CIFAR is a collection of datasets that contain color images of different objects. The CIFAR-10 dataset contains 60,000 images of 10 classes, with 6,000 images per class. The test set size is 10000 and the training set size is 50000.

3.2 Detailed Experimental Setups

MNIST:

Fold:10
Depth: 2Conv, 2Pool, 2Fc
Layer dimensions: 28*28*1 -> 28*28*32(Conv1) -> 14*14*32(Pool1)->14*14*64(Conv2) -> 7*7*64(Pool2) -> 500(Fc1) -> 10(Fc2)
Learning rate: 5e-3
Epoch: 10 each fold
Batch: Mini-Batch(40)
Loss function: CrossEntropy
Because these three data sets are multi-classification problems, I chose to use CrossEntropy for all three.
Opimizer: ASGD

ORL:

Depth: 2Conv, 2Pool, 2Fc
Layer dimensions: 112*92*3 -> 112*92*32(Conv1) -> 56*46*32(Pool1)->56*46*64(Conv2) -> 28*23*64(Pool2) -> 1024(Fc1) -> 40(Fc2)
Learning rate: 1e-3
Epoch: 60
Batch: Full-Batch(Because the entire test set is not very large, although the picture is about 16 times larger than MNIST, the entire test set has only 360 pictures. So I choose to use Full-Batch for training.)
Loss function: CrossEntropy
Opimizer: ASGD

CIFAR:

Depth: 4Conv, 2ResBlock ,2Pool, 3Fc
Layer dimensions: 32*32*3 -> 32*32*64(Conv1) -> 32*32*128(Conv2)->32*32*128(ResBlock) -> 32*32*256(Conv3) -> 16*16*256(Pool1) -> 16*16*256(ResBlock) -> 16*16*128(Conv4) -> 8*8*128(Pool2) -> 512(Fc1) -> 128(Fc2) -> 10(Fc3)
Learning rate: 1e-3
Epoch: 10
Batch: Mini-Batch**(400)**
Loss function: CrossEntropy
Opimizer: Adam

3.3 Evaluation Metrics

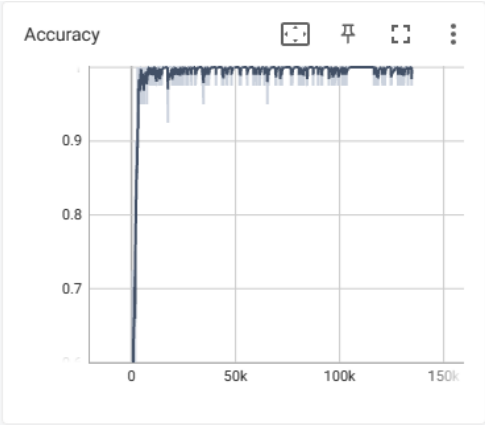
In the experiment, we will use Accuracy, Precision, Recall, F1 score as our evaluation metrics.

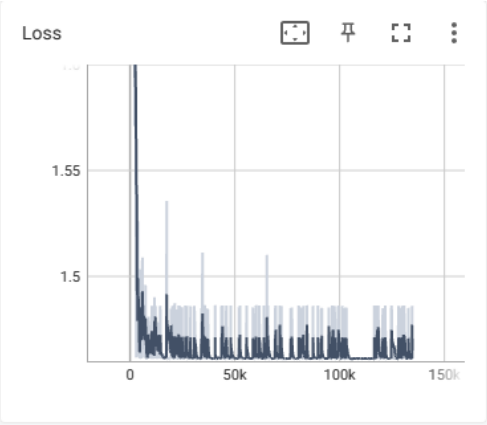
3.4 Source Code

https://drive.google.com/file/d/1KeonFM476KTnA79wV2krMAJ6a_p5Vl6a/view

3.5 Training Convergence Plot

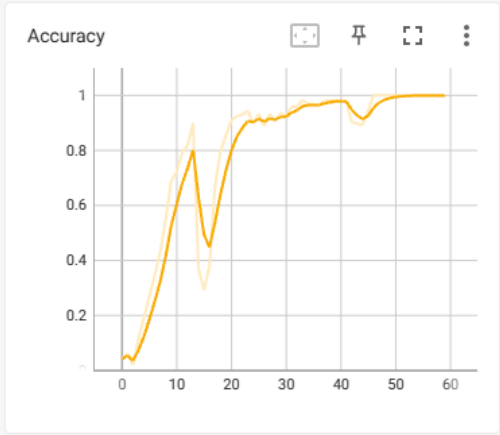
MNIST

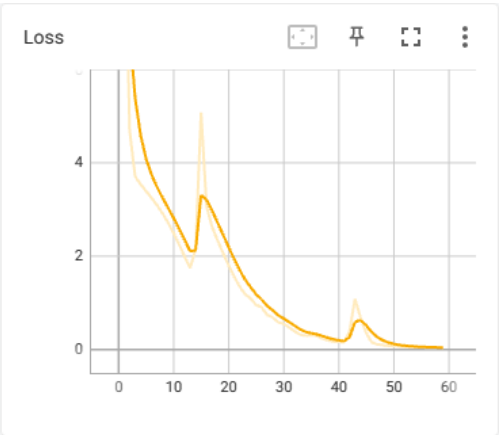




The X axis is the number of batch.

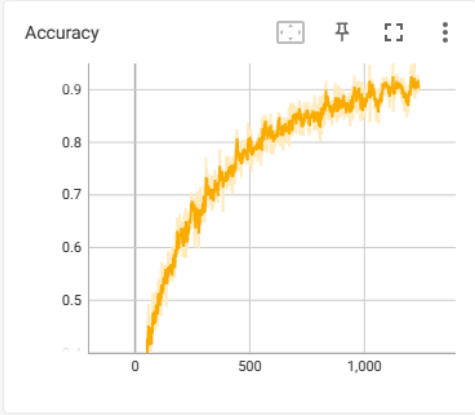
ORL

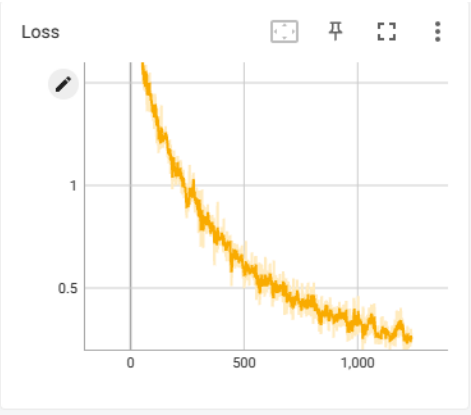




Although the final accuracy in the training set can reach 100%, in the test set, it can only reach 92.5% accuracy.

CIFAR





It can be clearly seen here that the accuracy rate and loss are gradually converging.

3.6 Model Performance

MNIST

Traning Set

Accuracy-Score: 0.9984074074074074
Precision-Score: 0.9984074798298332
Recall-Score: 0.9984074074074074
F1-Score: 0.9984072918018304

Testing Set

Accuracy-Score: 0.9919
Precision-Score: 0.9919160992460553
Recall-Score: 0.9919
F1-Score: 0.9918981890849163

It can be clearly seen that the accuracy is much better than using MLP training. It is not necessary to use a very complex model architecture to achieve an accuracy of 99.17%.

ORL

Training Set

Accuracy-Score: 1.0
Precision-Score: 1.0
Recall-Score: 1.0
F1-Score: 1.0

Testing Set

Accuracy-Score: 0.95
Precision-Score: 0.925
Recall-Score: 0.95
F1-Score: 0.9333333333333332

CIFAR

Training Set

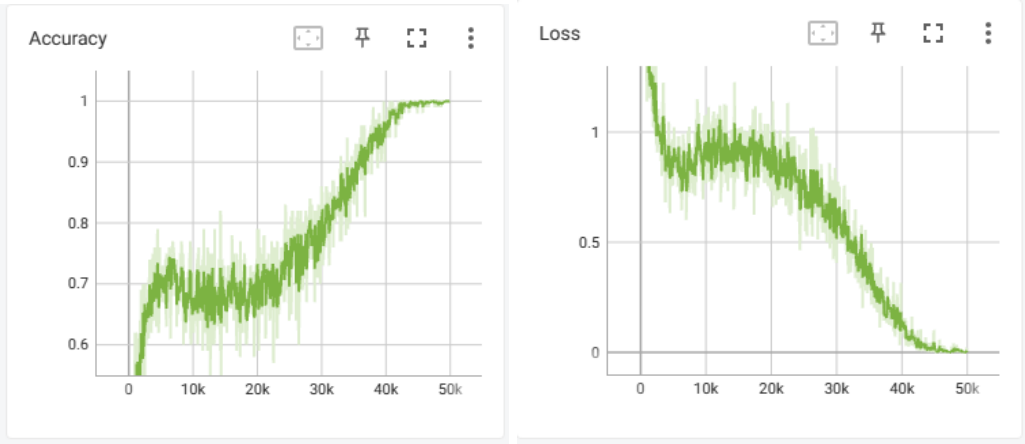
Accuracy-Score: 0.91388
Precision-Score: 0.9163360911330813
Recall-Score: 0.91388
F1-Score: 0.9136757284105399

Testing Set

Accuracy-Score: 0.8186
Precision-Score: 0.822742265196997
Recall-Score: 0.8186
F1-Score: 0.8178063447554501

3.7 Ablation Studies

1. Since MNIST and ORL are relatively simple, I found a relatively good answer without changing too many model structures and parameters. So I spend a lot of time training CIFAR.



After training for 100 epochs, I got this result. Seems fine, but it's overfitting. Even though I used ResNet in this model, the final test set accuracy was only 60%.
I used 6 convolutional layers, 5 pooling layers, and 3 Res in this model test.
Therefore, the conclusion this time tells me that in order to prevent overfitting, I need to add a verification set to the training for verification to observe whether the training accuracy has been overfitted, which can avoid wasting time on training an overfitting model .
2.
This is a good example
Epoch: 19 Test_Accuracy: 0.5914750000000002 Valid_Accuracy: 0.548
Epoch: 23 Test_Accuracy: 0.6309500000000002 Valid_Accuracy: 0.5518999999999998
Epoch: 26 Test_Accuracy: 0.653375 Valid_Accuracy: 0.5554000000000001
Epoch: 27 Test_Accuracy: 0.6648999999999998 Valid_Accuracy: 0.5548
The accuracy of the verification set has been unable to increase, so I stopped this training to modify the model.
3.
I found that although I used VGG16, there was still no way to make Valid_Accuarcy reach 65%. Until I found the problem, tensor.reshape cannot be used in such model training. Instead, transpose the nparray input dimension before converting the nparray to tensor. After improving this problem, the following results were obtained.
Epoch: 46 Test_Accuracy: 0.962540322580645 Valid_Accuracy: 0.79
4.

```
self.conv1 = conv_block(3, 64)
self.conv2 = conv_block(64, 128, pool=True)
self.res1 = nn.Sequential(conv_block(128, 128), conv_block(128, 128)).cuda()

self.conv3 = conv_block(128, 256, pool=True)
self.conv4 = conv_block(256, 512, pool=True)
self.res2 = nn.Sequential(conv_block(512, 512), conv_block(512, 512)).cuda()

self.classifier = nn.Sequential(nn.MaxPool2d(4).cuda(),
                                nn.Flatten().cuda(),
                                nn.Linear(512, 10).cuda())
```

This structure allowed me to get a good result of Accuracy-Score: 0.8304. This is what I made with reference to other people's structures, so I put this structure here.