Image Classification using Convolutional Neural Networks

Alok Patel Student ID : 260954024
Aishwarya Ramamurthy Student ID : 260963956
Katyayani Prakash Student ID : 260964511

Abstract

In this project, we investigated the performance of two deep convolutional networks: LeNet and ResNet on a 10-class image classification problem. We trained the models and evaluated their performances for varying epochs, and employing data augmentation, and dropout regularization. We observed that ResNet outperformed LeNet. Validating the findings that depth of the neural network is a key factor for high accuracy, ResNet-34 exhibited better performance than ResNet-18. Finally, bagging method of ensemble learning was used on the best 9 models to obtain final predicted accuracy of 97.533% in Kaggle competition.

1 Introduction

In this project, we have attempted 10-class image classification by training and comparing the performance of various convolutional neural networks in a supervised classification task on a given set of images. The training data comprised of 60,000 images of a modified version of the MNIST + Fashion-MNIST datasets. Each image contained articles and digits, and the true output was the class of the digit presented in the image. The true label, i.e the digit, of all images was provided in a separate file, *Train_labels.csv*. Our goal was to correctly classify the images and predict their labels.

1.1 Background and Related Work

Convolutional Neural Networks have long been a benchmark technique for image classification. LeCun et al. (1989) were the first ones to use back-propagation to learn the convolution coefficients directly from images of hand-written numbers. LeCun et. al (1998) also proposed the LeNet architecture that remains a state-of-art deep neural network in image classification, with many models, like AlexNet developed on its basis. Other CNNs, that have a demonstrated performance in this area, are VGG-16 developed by Simonyan & Zisserman (2015), and ResNet by He et al. (2015). While studying the effects of various parameters on model performance, we came across Koziarski & Cyganek (2018) who experimentally evaluated the impact of low resolution images on deep neural networks and concluded that ResNet Model fetched the highest accuracies. Increasing image resolution, introducing data augmentation, and tweaking hyper-parameters like depth, learning rate and optimizer function, have been proven to be efficient methods of increasing accuracy [2].

1.2 Project Overview

For image classification, we trained 2 CNN models: LeNet, and ResNet. Experiments were carried out to validate the effect of number of iterations (epochs), data augmentation, and dropout regularization, on the accuracy and generalization of a model. We split the training dataset into training and validation set (in 80/20 ratio) to test the generalization of a model. The best models were submitted in Kaggle to check accuracy on (30% of) the test set. Finally, the ones which demonstrated highest generalization were all bagged together to emulate ensemble learning. Using majority voting to obtain our final set of predictions, we achieved our highest test accuracy of 97.533% in Kaggle.

Submitted as part of coursework for ECSE 551, Winter 2021. Do not distribute.

2 Dataset

Figure 1 visually validates that the training dataset is majorly balanced with Class 1 having maximum examples of 6742 images and Class 5 having lowest examples with 5421 images.

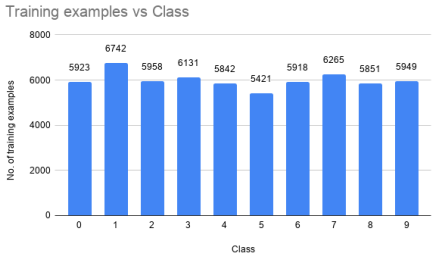


Figure 1: Class Distribution of Training Data

To increase the accuracy of our models, we performed 2 pre-processing steps on our dataset:

1. **Image Resizing** : Building upon the findings of Koziarski & Cyganek (2018), we developed a code for resizing the original low resolution 28x28 images to higher resolution of 224x224.
2. **Data Augmentation** : Data Augmentation refers to techniques used to increase the amount of data by adding slightly modified copies of already existing data. It has proved to be a key factor in improving the performance of a CNN by reducing overfitting [2]. Thus, we increased the total size of the dataset, by adding 10,000 randomly rotated images to the original set, and compared the performance of models on both these datasets.

3 Proposed Approach

As mentioned in Section 1.2, the task flow of this project can be categorized into 2 major parts. First, we trained 2 deep neural networks to study their performance. After rigorous training and experimentation on them, we selected 9 best models and performed bagging method on them for fetching the most accurate possible prediction to submit to the Kaggle Competition.

3.1 Training the Convolutional Neural Networks

Based on our literature review and theoretical knowledge, we decided on 2 benchmark CNNs. We have attempted to fine-tune each model by experimenting with varying optimizers, introducing dropout, applying data augmentation techniques etc. Since we didn't have class labels for test data, we split the training data into training and validation sets in 80-20 ratio. Please see below for a brief summary of each model implementation.

3.1.1 LeNet-5

LeNet is a convolutional neural network structure proposed by Yann LeCun et al. in 1998. It consists of 5 convolutional layers, hence it's called LeNet-5. It is a fundamental bechmark method of image classification. Therefore, we began our task with implementing it and testing its performance. Figure 2 shows the LeNet architecture utilized in this project for an input image of size 28x28.

We experimented with varying epochs to arrive at the most optimal model. However, LeNet failed to give us high test accuracy, therefore, we moved to a deeper network, the ResNet.

3.1.2 ResNet-18 and ResNet-34

ResNet models were proposed in "Deep Residual Learning for Image Recognition" by He et al. in 2015. Fig 3 shows the detailed model architecture of 5 versions of ResNet models in PyTorch, containing 18, 34, 50, 101, 152 layers respectively [5]. Intuitively, we believed ResNet will perform better than LeNet due to having more layers. Koziarski and Cyganek (2018) also demonstrated that

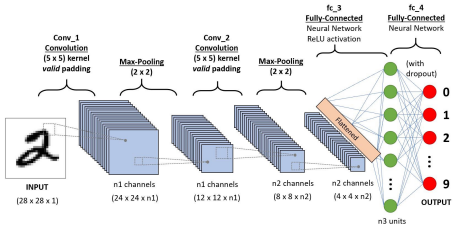
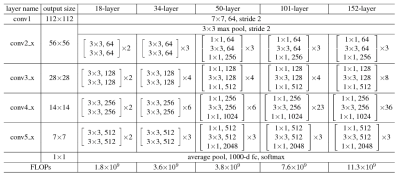


Figure 2: LeNet Architecture [4]

ResNet fetched highest accuracies on low resolution images. Therefore, in this project, we have trained ResNet-18 and 34 to classify our low resolution 28x28 images.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3: ResNet Architecture [5]

We experimented with data augmentation and dropout feature to test accuracy of the models. Results from these experiments are listed in Section 4.

3.2 Selection of best model by Bagging Method

After performing a series of rigorous experiments on each model and testing validation accuracies on training data, and test accuracy (on 30% of test data, for select promising models) in Kaggle, we employed the ensemble method of bagging to arrive at the overall best model. The 9 best models were all bagged together, and majority voting was used on their outputs to make the final prediction.

4 Results

First, LeNet model was trained with varying epochs. Although training accuracy increased with increasing epochs, validation accuracy started reducing after 3250 epochs (presumably due to overfitting). We submitted the best performing model out of this to Kaggle to test (on 30% of test data), and it gave us only 91% accuracy. Table 1 summarizes the results of these experiments.

Table 1: LeNet accuracy with varying the number of epochs

Accuracy of different sets (%)			
Epochs	Training Set	Validation Set	Test Set (Kaggle)
1000	86.124	82.173	-
2000	90.181	87.542	-
3250	**94.153**	**89.533**	**91.000**
5000	94.832	88.460	90.833

Due to the under-performance of LeNet, we implemented the much deeper ResNet model. We tried both ResNet-18 and ResNet-34 models. Table 2 summarizes the results from these models, from which we can infer that ResNet models show a sharp increase in accuracy with steep decline in runtime (due to considerably fewer number of epochs required).

We then performed additional experiments to further improve accuracy. Resizing all images of training set took a lot of bandwidth on runtime and RAM, which nullified any improvement in accuracy. We hypothesise this method to be more meaningful for smaller datasets (we tested on a

Table 2: Performance of ResNet-18 & ResNet-34 on training data

	ResNet-18 Accuracy (%)		ResNet-34 Accuracy (%)	
Epochs	Training Set	Validation Set	Training Set	Validation Set
10	99.231	95.647	98.884	94.312
15	99.573	96.412	99.394	95.804

partial dataset of 1000 images and results were encouraging), however, it is not feasible for a dataset as large as ours. Therefore, we moved to the next set of experiments – data augmentation. As stated in Section 2, we increased the size of the training data by introducing 10K images rotated by 30°. (Note: Test Accuracy denotes the accuracy on 30% of test set, as obtained on submitting in Kaggle.)

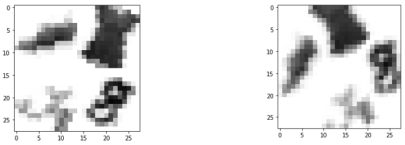


Figure 4: Original Image (Left) and Image Rotated by 45°(Right)

We also tested dropout regularization in ResNet-18, however, it did not improve our test accuracy in Kaggle. Please see Tables 3 and 4 for a summary of our results. The columns Original Dataset and Augmented Dataset present the accuracy of both these datasets, consisting of 60K and 10K observations respectively. Also, we used 45°of rotation in ResNet-18 with dropout, and 30°of rotation everywhere else.

Table 3: ResNet-18 with data augmentation and dropout regularization

Epochs	Dropout	Original Dataset	Augmented Dataset	Avg Accuracy	Loss	Test Accuracy
15	No	99.642	99.221	99.431	0.022	96.266
20	No	99.433	98.890	99.162	0.030	-
20	0.5	99.607	99.453	99.528	0.134	95.966

Table 4: ResNet-34 with data augmentation of 30°at 20 epochs

Epochs	Original Dataset	Augmented Dataset	Avg Accuracy	Loss	Test Accuracy
20	99.520	99.519	99.519	0.012	96.300

Figure 5 shows the trend of accuracy in ResNet-34 for results of Table 4 on training and validation sets (split in 80-20 ratio) with every consequent epoch.

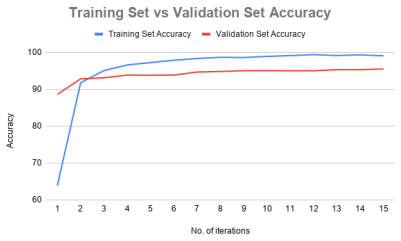


Figure 5: Training Set vs Validation Set Accuracy in ResNet-34

Figure 6 displays the trade-off between accuracy vs runtime in ResNet models. ResNet-18 without augmentation proves to be the most efficient, since it provides highest accuracy with lowest runtime. Data Augmentation can be seen to increase runtimes, as does having more layers in the network.

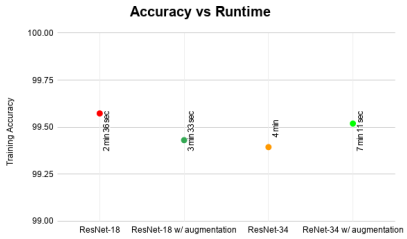


Figure 6: Accuracy vs Runtime of ResNet models

Finally, we selected the 9 best models, and using the ensemble method of bagging, combined them all together. We used majority voting to obtain our final set of predictions, which were then stored in a .csv file and submitted as a final submission to Kaggle. As expected, we achieved our highest test accuracy with this method, **97.533%**. Fig 7 shows the architecture of our bagging ensemble.

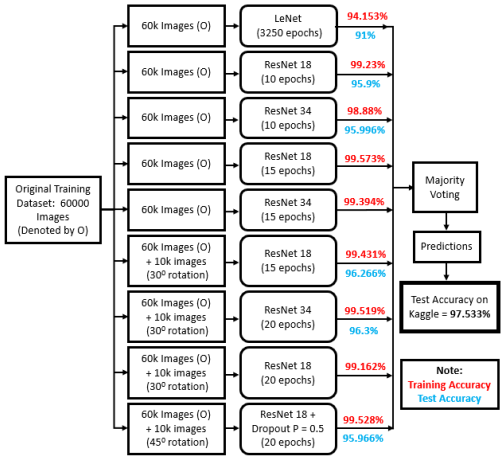


Figure 7: Bagging Method for achieving highly accurate prediction

5 Discussion & Conclusion

On implementing both LeNet and ResNet models, we found that ResNet outperforms the LeNet architecture. We also found that ResNet-34 performs slightly better than ResNet-18. Thus, we can infer that depth of a network plays a huge role in its performance. We also explored various methods of performance improvement, like data augmentation and dropout regularization, and found that while augmentation definitely improved performance, dropout regularization was more subjective. Finally, we applied the ensemble method of bagging, and obtained a prediction that was better than all individual models. This reinstated our intuitive belief in using ensemble method to improve the overall performance by reducing variance.

For future investigation, more models, such as VGG-16, AlexNet or deeper architectures of ResNet (50, 102 etc.) can be explored. Further study of regularization parameters and image resolution on performance should also be conducted.

6 Statement of Contribution

Alok Patel and Katyayani Prakash majorly worked on designing the workflow, developing, training and testing both LeNet and ResNet models inclusive of data augmentation and dropout regularization techniques. Aishwarya Ramamurthy attempted train/test data split and ResNet validation accuracy evaluation. All the team members have made fair contribution to the report.

References

[1] Koziarski, Michał & Cyganek, Boguslaw. (2018). Impact of Low Resolution on Image Recognition with Deep Neural Networks: An Experimental Study. *International Journal of Applied Mathematics and Computer Science*. 28. 735-744. 10.2478/amcs-2018-0056.

[2] The Quest of Higher Accuracy for CNN Models. (2021). Retrieved from https://towardsdatascience.com/the-quest-of-higher-accuracy-for-cnn-models-42df5d731faf

[3] Hacking your Image Recognition Model. (2021). Retrieved from https://towardsdatascience.com/hacking-your-image-recognition-model-909ad4176247

[4] A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. (2021). Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[5] PyTorch. (2021). Retrieved from https://pytorch.org/hub/pytorch_vision_resnet/

[6] He, K., Zhang, X., Ren, S., Sun, J. (2015). Deep Residual Learning for Image Recognition. Retrieved from https://arxiv.org/abs/1512.03385

[7] Simonyan, K., Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. Retrieved from https://arxiv.org/abs/1409.1556

[8] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Handwritten Digit Recognition with a Back-Propagation Network. NIPS.

[9] Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings Of The IEEE, 86(11), 2278-2324. doi: 10.1109/5.726791

[10] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*. 32 (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[11] bentrevett/pytorch-image-classification. (2021). Retrieved from https://github.com/bentrevett/pytorch-image-classification

[12] PyTorch 1.0.1 on MNIST (Acc > 99.8%). (2021). Retrieved from https://www.kaggle.com/tonysun94/pytorch-1-0-1-on-mnist-acc-99-8

[13] Pinsky, G., & Peifer, T. (2021). How do I rotate a PyTorch image tensor around it's center in a way that supports autograd?. Retrieved from https://stackoverflow.com/questions/64197754/how-do-i-rotate-a-pytorch-image-tensor-around-its-center-in-a-way-that-supports

[14] Lecture Slides of ECSE-551: Machine Learning For Engineers by Prof. Narges Armanfard

Appendix-A: ReadME for Code

ReadMe:

Note: Look at each cell for more instruction

1) First, load the *Train.pkl* , *Train_labels.csv* , and *Test.pkl* files which will be used for the training and testing pupose of various algorithms using the drive.

2) At the start, code of the pre-processing of the data and the data is loaded into the dataloader for the training and the testing of the data.

3) Then the following cells are the cells which performs the data augmentation, in the form of the Image resize and the Image Rotation.

4) LeNet, VGG, and ResNet CNN are defined and created.

5) Later cells contribute for the training, validation and testing of all the models.

6) Lastly, the test results are stored in the csv file to make it ready for the kaggle, and then the ENSEMBLE method is done using the majority of votin

Appendix-B: Python Code

Preprocessing of the data and its Analysis

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
#Folder of the  drive is accessed
%cd '/content/gdrive/MyDrive/MP3_data'
```

Calling all IMP Libraries

```
## --- Calling all the Neccessary Libraries --- ##
import torch
import pickle
import matplotlib.pyplot as plt
import numpy as np
from torchvision import transforms
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
```

```
from PIL import Image
from tqdm.notebook import tqdm_notebook
import random

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import cv2

#print(torch.__version__)
```

Extra Data Loading for the data observation

```
## ---- Not needed --- ##
# Data directly loaded using the drive as a mount
#If we don't want to use we will not use.
import numpy as np

#train_data_1 = np.load('/content/gdrive/MyDrive/ECSE_551/Assignment-
3/train/Train.pkl', allow_pickle=True).reshape(-1, 28, 28)

train_data_1 = np.load('/content/gdrive/MyDrive/ECSE-
551/imageunderstanding/Train.pkl', allow_pickle=True).reshape(-1, 28, 28)
```

```
#Checking the size of the data and checking the array of one image
import random
#print(train_data_1.shape)      #To print the shape of the dataset

#This data is already been given normalized in the training data file.
a = random.randint(0, 59999)
print("The image index id is: "+str(a))
train_data_1[a]
print(train_data_1[a].shape)
#255*train_data[a]         #To check the not normalized pixel data array.

#To display the random image from the set
import matplotlib.pyplot as plt
plt.imshow(train_data_1[a], cmap="binary")
```

```
# Loading the data here using the pickle library
# Read a pickle file and disply its samples
# Note that image data are stored as unit8 so each element is an integer v
alue between 0 and 255
a = random.randint(0, 59999)
```

```
data = pickle.load( open( './Train.pkl', 'rb' ), encoding='bytes').reshape
(-
1,28,28)    #Reshape from (1,28,28) to (28,28), if RGB then 3 instead of 1
 in image shape
targets = np.genfromtxt('./Train_labels.csv', delimiter=',', skip_header=1
)[:,1:]       #skip_header = 1, will remove the 1st unwanted row of the la
bel file and [:,1:] will remove the 1st column
plt.imshow(data[1236],cmap='binary', vmin=0, vmax=1)
              #As normalized images are given we have taken the vmax=1, ot
herwise we have 255
plt.title("Ground Truth: {}".format(targets[a]))
print("The image index id is: "+str(a))

print("Shape of training data = ",data.shape)
#print(targets)
       #To see the matrix of the targets stored
```

```
import random

#This data is already been given normalized in the training data file.
a = random.randint(0, 59999)
print("The image index id is: "+str(a))
train_data_1[a]
#255*train_data[a]        #To check the not normalized pixel data array.

#To display the random image from the set
import matplotlib.pyplot as plt
plt.imshow(train_data_1[a], cmap="binary")
plt.title("Ground Truth: {}".format(targets[a]))
```

Dataset and Dataloader class for the training of the CNN

Transformation of the Data

```
# Code for the image resize and rotation
size = (224,224)
img_resize = transforms.Resize(size)
img_rotation = transforms.RandomAffine(degrees=30)
#img_transform = transforms.Compose([img_resize, transforms.ToTensor()])
img_transform = transforms.Compose([ img_rotation, transforms.ToTensor()])
```

MyDataset Class for creating the dataset for the CNN

```
# Dataset class and Dataloader class
from skimage.util import img_as_ubyte
```

```
class MyDataset(Dataset):
  def __init__(self, img_file, label_file, transform=None, idx = None):
    self.data = pickle.load( open( img_file, 'rb' ), encoding='bytes')

    self.targets = np.genfromtxt(label_file, delimiter=',', skip_header=1)
[:,1:]           #When the labels file will be loaded the 1st row will be
removed.
    self.transform = transform

    if idx is not None:
      self.targets = self.targets[idx]
      self.data = self.data[idx]

  def __len__(self):
    return len(self.targets)

  def __getitem__(self, index):
    img, target = self.data[index], int(self.targets[index])

    if self.transform is not None:
      img = img.reshape(28,28)
      #print(img.shape)
      img = Image.fromarray(img.astype('float'), mode='1')
      #float is for 0 to 1 image type and mode = 1 is for 0 to 1 same
      #img_as_ubyte(img)
      #plt.imshow(imgs[3],cmap='binary', vmin=0, vmax=1)
      img = self.transform(img)
      #img = img.reshape(1,224,244)

    return img, target
```

Main Cell for creating the Dataloader for the training data

```
## ---- Main Data loader for whole training set ---- ##
# Read image data and their label into a Dataset class #

## Note:- Taking the test data of around 1000 from this 60000 set as well.


dataset = MyDataset('./Train.pkl', './Train_labels.csv', idx=None)

#Loading data using DataLoader
batch_size = 1000
      #We decide it according to the size for the training or test data
```

```
train_data = DataLoader(dataset, batch_size=batch_size, shuffle=True)
      #Shuffle=False if we want to fix the arrangement of data
```

Loading the Test data set for Kaggle

```
# Test data using dataloader
# Used for ResNet
img_file = './Test.pkl'
batch_size = 1000
data = pickle.load( open( img_file, 'rb' ), encoding='bytes')
test_data = DataLoader(data, batch_size = batch_size)
```

Dataloader for the change in image size (Not working)

```
# Image size change
dataset = MyDataset('./Train.pkl', './Train_labels.csv', transform = img_t
ransform, idx=None)

#Loading data using DataLoader
batch_size = 1000           #We decide it according to the size for the tr
aining or test data
train_data = DataLoader(dataset, batch_size=batch_size, shuffle=True)   #S
huffle=False if we want to fix the arrangement of data

#print(train_data)
imgs, labels = (next(iter(train_data)))
```

```
imgs = np.squeeze(imgs)
print(imgs[3][150])
plt.imshow(imgs[3].cpu().numpy(),cmap='gray', vmin=0, vmax=1) #.transpose(
)
#plt.imshow(imgs[12].reshape(224,224),cmap='binary')
```

```
#Seeing rotating image
plt.imshow(imgs[60].reshape(28, 28).cpu().numpy(),cmap='binary', vmin=0, v
max=1)
```

```
# Image Rotation added

dataset = MyDataset('./Train.pkl', './Train_labels.csv', img_transform, id
x=None)

#Loading data using DataLoader
```

```
batch_size = 1000           #We decide it according to the size for the tr
aining or test data
train_data = DataLoader(dataset, batch_size=batch_size, shuffle=True)   #S
huffle=False if we want to fix the arrangement of data

imgs, labels = (next(iter(train_data)))
```

Dataset Making for VGG by 1x28x28 to 1x224x224

```
## ---- Loading the training data for the VGG ----- ##
import pandas as pd

data_vgg = pickle.load( open('./Train.pkl', 'rb' ), encoding='bytes').resh
ape(-1,28,28)
data_train_vgg = torch.tensor(data_vgg, dtype=torch.float32)
df = pd.read_csv(r'./Train_labels.csv')
cls = df['class']
label_vgg = np.array(cls)
```

```
## ------ Function fot the batches of the dataset for VGG ------ ##

def vgg_data(data_vgg, ran):
  res_img = []
  resize_images = np.zeros([10000,224,224])
  for a in range(len(ran)):
    original_img = data_vgg[ran[a]]
    rsz = cv2.resize(original_img,(224,224))
    res_img.append(rsz)
    resize_images[a] = np.array(res_img[a])

    return resize_images
```

```
## ------ Create batches of the dataset for vgg ------ ##

lst  = list(range(0,60000,1))
batch = 10000           #Each split of 10000 (Batch size)
spl = [lst[i:i + batch] for i in range(0, len(lst), batch)]

b1_vgg = vgg_data(data_train_vgg, spl[0])

vgg_b1_labels = label_vgg[:10000]
b1_vgg = b1_vgg.reshape(10000,1,224,224)
b1_vgg = b1_vgg.astype('float32')
```

```
## ---- Updating the dataloader for vgg (10k) ---- ##
batch_size = 50
vgg_data = DataLoader(b1_vgg, batch_size = batch_size, shuffle = True)
```

```
# ---- For the manual images resize ---- ##

resize_images = np.zeros([1000,224,224])
#res_img = np.reshape(res_img,(2,3))

res_img = []

for a in range(1000):
  #print(a)
  #plt.imshow(data[a],cmap='binary', vmin=0, vmax=1)
  #plt.title("Ground Truth: {}".format(targets[a]))
  #plt.show()
  #print("Size of image =",data[a].shape)
  original_img = data[a]
  rsz = cv2.resize(original_img,(224,224))
  #plt.imshow(rsz,cmap='binary', vmin=0, vmax=1)
  #plt.show()
  #print("Size of new image =",rsz.shape)
  res_img.append(rsz)
  resize_images[a] = np.array(res_img[a])
  #print(res_img[a])


#resize_imgages = np.array(res_img[0])
print("Size of resize_imgages",resize_images.shape)
plt.imshow(resize_images[0],cmap='binary')
plt.title('First picture in resized array')
plt.show()
```

```
# Loading the training data for the VGG
data_train_vgg = pickle.load( open('./Train.pkl', 'rb' ), encoding='bytes'
).reshape(-1,28,28)
```

```
# Function for the resizing of the 60 ths data [Extra][Not Needed]

def resize(data, st, en):
  resize_img = np.zeros([en-st,224,224])
  res_img = []
  for a in range(st, en):
```

```
    original_img = data[a]
    rsz = cv2.resize(original_img,(224,224))
    res_img.append(rsz)
    resize_img[a] = np.array(res_img[a])

  return resize_img
```

```
# Converting the image size from 28x28 to 224x224

lst  = list(range(0,60000,1))
batch = 600           #Each split of 600 (Batch size)
split = [lst[i:i + batch] for i in range(0, len(lst), batch)]

d1 = 224
d3 = 1
lst2 = [[ ['#' for col in range(d1)] for col in range(d1)] for row in rang
e(d3)]
final_array = np.array(lst2)        #1st element just sample later removed

#final_array = torch.from_numpy(final_array)
for i in range(len(split)):
#for i in range(1):
  resize_op = resize(data_train_vgg, split[i][0], (split[i][-
1])+1)     #+1 as in range last is not count
  #final_array.append(resize_op)
  #tens = torch.from_numpy(resize_op)
  #torch.cat((final_array, tens), 0)
  final_array = np.concatenate((final_array, resize_op), axis = 0)

#Final_resize = np.array(final_array)
print(final_array, final_array.shape)
print("Size of final resize_images",Final_resize.shape)
```

```
#Converting the resized array to a tensor
VGG_Train = torch.tensor(resize_images, dtype=torch.float32, device='cuda'
)
print("VGG Train image tensor size = ",VGG_Train.shape)
```

Code Snippet for Rotating images [IMP for the Data Augmentation]

```
## --- Image Rotation Functions for the randome 10000 images --- ##

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

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv(r'./Train_labels.csv')
cls = df['class']
label_rotation = np.array(cls)

#Loading the 10000 images(first whole data and then pick selected)
data_rotation = np.load('./Train.pkl', allow_pickle=True)
#label_rotation = np.genfromtxt('./Train_labels.csv', delimiter=',', skip_
header=1)[:,1:]

image_index = np.random.randint(0,60000,size=10000)             #Chossing
random 10 ths images

#Taking the 10000 images and its index randomly from 60 ths set.
rotation = data_rotation[image_index]
image_rot_label = label_rotation[image_index]
img_to_rotate = torch.tensor(rotation)                          #Tensor, a
s required by rotation fun
#image_rot_label = torch.tensor(image_rot_label)
img_to_rotate = img_to_rotate.cuda()
#image_rot_label = image_rot_label.cuda()
#img_to_rotate.device

# Funtion that do the rotation of the images
def get_rot_mat(theta):
    theta = torch.tensor(theta)
    return torch.tensor([[torch.cos(theta), -torch.sin(theta), 0],
                         [torch.sin(theta), torch.cos(theta), 0]])

def rot_img(x, theta, dtype):
    rot_mat = get_rot_mat(theta)[None, ...].type(dtype).repeat(x.shape[0],
1,1)
    grid = F.affine_grid(rot_mat, x.size()).type(dtype)
    x = F.grid_sample(x, grid)
    return x

#dtype =  torch.cuda.FloatTensor if torch.cuda.is_available() else torch.F
loatTensor
dtype = torch.cuda.FloatTensor

#im should be a 4D tensor of shape B x C x H x W with type dtype, range [0
,255]:
```

```
see_img = np.squeeze(img_to_rotate)
plt.imshow(see_img[1].reshape(28, 28).cpu().numpy(),cmap='binary', vmin=0,
 vmax=1)
#plt.imshow(imgs[0],cmap='binary',vmin=0,vmax=1) #To plot it im should be
1 x C x H x W
plt.show()

#Rotation by np.pi/6 with autograd support:
rotated_im_data = rot_img(img_to_rotate, np.pi/4, dtype)     # Rotate imag
e by 30 degrees.

#Plot rotated images
see_img1 = np.squeeze(rotated_im_data)
plt.imshow(see_img1[1].reshape(28, 28).cpu().numpy(),cmap='binary', vmin=0
, vmax=1)
plt.show()
print("Rotated imgs size=",rotated_im_data.shape)
```

```
## ----- EXTRA ----- ##
# Read image data and their label into a Dataset class
# Test and train data is uploaded from this cell.

## Note:- Taking the test data of around 6000 from this 60000 set as well.


dataset = MyDataset('./Train.pkl', './Train_labels.csv', idx=None)

#Loading data using DataLoader
batch_size = 1000
      #We decide it according to the size for the training or test data
train_data = DataLoader(dataset, batch_size=batch_size, shuffle=True)
      #Shuffle=False if we want to fix the arrangement of data

#Read a batch of data and their labels and display them
imgs, labels = (next(iter(train_data)))
      #Calling methods (same like for loop)
see_img = np.squeeze(imgs)
plt.imshow(see_img[1].reshape(28, 28).cpu().numpy(),cmap='binary', vmin=0,
 vmax=1)
```

```
# Loading the test.pkl using the array and loading in the tensor
# Used for LeNet

data_test = np.load('./Test.pkl', allow_pickle=True)
```

```
test_data = torch.tensor(data_test, dtype=torch.float32, device='cuda')
#test_data = DataLoader(data_test, batch_size=batch_size, shuffle=False)
```

```
# Shape when dataloader is used.
print(imgs.shape)
print(labels.shape)
```

```
## --- NEW when data is converted to the tensor [Extra] --- ##

#train_img = train_img.astype(np.float32)
#label_img = label_img.astype(np.float32)
train_img = torch.tensor(dataset.data, dtype=torch.float32, device='cuda')
label_img = torch.tensor(dataset.targets, dtype=torch.long, device ='cuda'
).reshape(-1,)

#print(dataset.data)
print("Train image dataset size = ",train_img.shape)
print("Train label dataset size = ",label_img.shape)
```

```
print(img.shape)
```

LeNet – CNN

```
## Class of LeNet CNN model for MNIST+Fashion-MNIST Dataset ##

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class LeNet(nn.Module):
  def __init__(self):
    super(LeNet, self).__init__()

    #First Convolutional Layer which converts (28x28)x1 to (24x24)x6
    self.conv1 = nn.Conv2d(1, 6, kernel_size=5)

    #Max pooling of 2x2, which makes next layer dim as (12x12)x6
    #It is also used after 2nd Conv layer which makes next layer dim as (4
x4)x16
    self.pool = nn.MaxPool2d(2,2)

    #Second Convolutional Layer which converts (12x12)x6 to (8x8)x16
    self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
```

```
    #NN at the FC region
    self.fc1 = nn.Linear(16*4*4,120)       #1st FC layer
    self.fc2 = nn.Linear(120,84)           #2nd FC layer
    self.fc3 = nn.Linear(84,10)            #Output layer

  def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = x.view(-
1,16*4*4)                 #Matrix to vector conversion of Conv to NN(FC) i
/p
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)                        #output layer executed

    return x
    #return F.log_softmax(x, dim=1)

#Below Function is for testing
'''
  def forward(self,x):
    x = F.sigmoid(self.pool(F.tanh(self.conv1(x))))        #For testing pu
rpose
    x = F.sigmoid(self.pool(F.tanh(self.conv1(x))))               #For tes
ting purpose
    x = x.view(-1,16*4*4)
    x = F.tanh(self.fc1(x))
    x = F.tanh(self.fc2(x))
    x = self.fc3(x)
    return F.log_softmax(x)
'''
print()
```

VGG – CNN

```
# Class of VGG CNN model for MNIST+Fasion-MNIST Dataset
import torchvision.models as models

#class VGG():

device = 'cuda'
vgg = models.vgg16(pretrained = False, progress=True)
      #Pretrained is OFF and Progress bar is made ON
vgg.features[0] = nn.Conv2d(1, 64,3,1)
```

```
vgg.classifier[-1] = nn.Linear(4096,1000)
vgg.classifier.add_module('7', nn.ReLU())
#vgg.classifier.add_module('8', nn.Dropout(p=0.5, inplace=False))
vgg.classifier.add_module('8', nn.Linear(1000, 10))
vgg.classifier.add_module('9', nn.LogSoftmax(dim=1))

vgg.to(device)
```

```
#Good to Know the architecture
print(vgg)
```

ResNet – CNN

```
# ---- ResNet 18 ----- #
from torchvision.models.resnet import ResNet, BasicBlock, Bottleneck

class MFResNet(ResNet):
    def __init__(self):
        super(MFResNet, self).__init__(BasicBlock, [2, 2, 2, 2], num_class
es=10) # Based on ResNet18
        self.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=1, padding=3,b
ias=False)
```

```
# ---- ResNet 34 ----- #
from torchvision.models.resnet import ResNet, BasicBlock, Bottleneck

class MFResNet(ResNet):
    def __init__(self):
      super(MFResNet, self).__init__(BasicBlock, [3, 4, 6, 3], num_classes
=10)
      self.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=1, padding=3,bia
s=False)
```

Training and Testing on ResNet

```
# --- ResNet and its optimizer --- #
resnet = MFResNet()

optimizer = optim.Adam(resnet.parameters(), lr = 0.001)
criterion = torch.nn.CrossEntropyLoss()
```

```
## ---- Adding the dropout features for Resnet ---- ##

num_ft = resnet.fc.in_features
resnet.fc = nn.Sequential(nn.Dropout(p=0.5), nn.Linear(num_ft, 10))
resnet = resnet.to('cuda')
```

```
print(resnet)
```

```
## ---- Getting data ready for the rotated images ---- ##
rotated_data = rotated_im_data.cpu().numpy()
batch_size = 1000
train_rot_data = DataLoader(rotated_data, batch_size = batch_size, shuffle
 = False)
```

```
# --- Training of the rotated + originial dataset on ResNet --- #
import time
from tqdm.notebook import tqdm_notebook

lst  = list(range(0,10000,1))
batch = 1000
split = [lst[i:i + batch] for i in range(0, len(lst), batch)]

test_counter = [i*len(train_data.dataset) for i in range(3)]
best_acc = 0
avg_acc = 0

iterations = 5        #Setting no. of epochs
progress_bar = tqdm_notebook(iterable=range(iterations), position=0, leave
=True)

start = time.time()
for epoch in progress_bar:
  resnet.train()
  correct1 = 0
  total1 = 0
  correct2 = 0
  total2 = 0
  for batch_idx, (data, target) in enumerate(train_data):
    data, target = data.cuda(), target.cuda()
    optimizer.zero_grad()
    output1 = resnet(data).cuda()
    _, predicted = torch.max(output1, 1)
    correct1 += (predicted == target).sum().item()
```

```
    loss1 = criterion(output1, target)
    total1 += target.size(0)
    acc1 = (correct1 / total1) * 100.0
    progress_bar.set_description(f"Loss : {loss1.item():.3f}, Main Data Ac
curacy : {acc1:.3f}")
    loss1.backward()
    optimizer.step()

  print(f'Epoch : {epoch+1} ; Training Accuracy on main data: {acc1:.3f}')

  var = 0
  for _, dat in enumerate(train_rot_data):          #As 10 data will be so
 10 times loops
    tar = image_rot_label[split[var]]
    tar = torch.tensor(tar)
    dat, tar = dat.cuda(), tar.cuda()
    optimizer.zero_grad()
    output2 = resnet(dat).cuda()
    _, pred = torch.max(output2, 1)
    correct2 += (pred == tar).sum().item()
    loss2 = criterion(output2, tar)
    total2 += tar.size(0)
    acc2 = (correct2 / total2) * 100.0
    progress_bar.set_description(f"Loss : {loss2.item():.3f}, Rotated data
 Accuracy : {acc2:.3f}")
    loss2.backward()
    optimizer.step()
    var = var + 1

  print(f'Epoch : {epoch+1} ; Training Accuracy on rotated data: {acc2:.3f
}')
  avg_acc = (acc1+acc2)/2

  if avg_acc>best_acc:
    best_acc = avg_acc
    torch.save(resnet.state_dict(), '/model.pth')
    torch.save(optimizer.state_dict(), '/optimizer.pth')

  print(f'Epoch : {epoch+1} ; Training Accuracy over whole set(avg): {avg_
acc:.3f}')

#print(f'Accuracy of the network on the 60000 train images: {(correct / to
tal) * 100.0:.3f} %')
```

```
## --- Random data split - Training and Validation Set for ResNet --- ##

dataset = MyDataset('./Train.pkl', './Train_labels.csv', idx=None)

#Defining lengths of training and validation set
train_dsize = int(0.8*len(dataset))
valid_dsize = len(dataset) - train_dsize
train_set, valid_set = torch.utils.data.random_split(dataset, [train_dsize
, valid_dsize])

#Loading data using DataLoader
batch_size = 1000
      #We decide it according to the size for the training or test data
train_dat = DataLoader(train_set, batch_size=batch_size, shuffle=True)
       #Shuffle=False if we want to fix the arrangement of data
valid_dat = DataLoader(valid_set, batch_size=batch_size, shuffle=True)
```

```
## ----- Training Function for the ResNet ----- ##
from tqdm.notebook import tqdm_notebook

def train_resnet(iteration, train_dat):
  resnet.train()
  correct = 0; total = 0;
  for batch_idx, (data, target) in enumerate(train_dat):
    data, target = data.cuda(), target.cuda()
    optimizer.zero_grad()
    output = resnet(data).cuda()
    _, predicted = torch.max(output.data, 1)

    correct += (predicted == target).sum().item()
    loss = criterion(output, target)
    total += target.size(0)
    acc = (correct / total) * 100.0
    progress_bar.set_description(f"Loss : {loss.item():.3f}, Training Accu
racy : {acc:.3f}")

    loss.backward()
    optimizer.step()

  print(f'Epoch : {iteration}; Training Accuracy: {acc:.3f}')

#print(f'ResNet Accuracy on Training Set (48000 images): {(correct / total
) * 100.0:.3f} %')
```

```
## ----- Validation Function for the ResNet ----- ##
from tqdm.notebook import tqdm_notebook

def validate_resnet(val_iter, valid_dat):
  correct = 0; total = 0;
  best_acc = 0
  for batch_idx, (data, target) in enumerate(valid_dat):
    data, target = data.cuda(), target.cuda()
    output = resnet(data).cuda()
    _, predicted = torch.max(output.data, 1)
    correct += (predicted == target).sum().item()
    total += target.size(0)
    acc = (correct / total) * 100.0
    progress_bar.set_description(f"Validation Accuracy : {acc:.3f}")

    if acc>best_acc:
      best_acc = acc

  print(f'Epoch : {val_iter}; Validation Accuracy: {acc:.3f}')
  return best_acc

#print(f'ResNet Accuracy on Validation Set (12000 images): {(correct / tot
al) * 100.0:.3f} %')
```

```
## ----- Running the epochs of ResNet ----- ##
import time

epoch = 20          #Set the number of iterations to run
store_epoch = 0
progress_bar = tqdm_notebook(iterable=range(epoch), position=0, leave=True
)
best_acc = 0
start = time.time()
for i in progress_bar:
  train_resnet(i+1, train_dat)
  val_acc = validate_resnet(i+1, valid_dat)
  if val_acc>best_acc:                                #Best Validation acc
. is stored
    best_acc = val_acc
    store_epoch = i
    torch.save(resnet.state_dict(), '/model.pth')
    torch.save(optimizer.state_dict(), '/optimizer.pth')

end = time.time()
```

```
print(f'At Epoch : {store_epoch} the best Validation Accuracy acheived: {b
est_acc:.3f}')
```

```
# --- Training of the Dataset on ResNet (For Whole training data)--- #
import time
from tqdm.notebook import tqdm_notebook

train_losses = []
train_counter = []
test_losses = []
test_counter = [i*len(train_data.dataset) for i in range(3)]
best_acc = 0

iterations = 10          #Setting no. of epochs
progress_bar = tqdm_notebook(iterable=range(iterations), position=0, leave
=True)

start = time.time()
for epoch in progress_bar:
#for epoch in range(iterations):
  resnet.train()
  correct = 0
  total = 0
  for batch_idx, (data, target) in enumerate(train_data):
    data, target = data.cuda(), target.cuda()
    optimizer.zero_grad()
    output = resnet(data).cuda()
    #predicted = torch.argmax(output, dim=1)
    _, predicted = torch.max(output.data, 1)
    # print(predicted)
    correct += (predicted == target).sum().item()
    loss = criterion(output, target)             #negative log liklhood lo
ss
    total += target.size(0)
    acc = (correct / total) * 100.0
    progress_bar.set_description(f"Loss : {loss.item():.3f}, Accuracy : {a
cc:.3f}")
    # print(loss)
    loss.backward()
    optimizer.step()

    if acc>best_acc:
      best_acc = acc
      torch.save(resnet.state_dict(), '/model.pth')
```

```
      torch.save(optimizer.state_dict(), '/optimizer.pth')

  acc = 100 * correct / total
  # print('Epoch : ',epoch,' Accuracy: %d',acc)

print(f'Accuracy of the network on the 60000 train images: {(correct / tot
al) * 100.0:.3f} %')
```

```
## ------ Testing the test dataset on resnet for Kaggle ----
- # (Test data)
# We are loading the save model from the directory

model_test = MFResNet()                    #Temporary model to save the inbu
ilt model (in cpu)
num_ft = model_test.fc.in_features
model_test .fc = nn.Sequential(nn.Dropout(p=0.5), nn.Linear(num_ft, 10))
model_test.to('cuda')
model_test.load_state_dict(torch.load('/model.pth'))


prediction = np.zeros(10000)
n = 0
for d in test_data:
  d = d.cuda()
  out = model_test(d).cuda()
  _, pred_test = torch.max(out, 1)
  pred_test = pred_test.cpu()
  pred_test = pred_test.tolist()
  for i in range(batch_size):
    prediction[n] = pred_test[i]
    n = n + 1

print(prediction)
```

Training and Testing on VGG

```
# --- VGG and its optimizer --- #

optimizer = optim.Adam(vgg.parameters(), lr = 0.01)
#criterion = torch.nn.NLLLoss()
criterion = torch.nn.CrossEntropyLoss()
```

```
## ---- Training the 10k resize images on vgg ---- ##
import time
```

```
lt  = list(range(0,9984,1))
bat = 64          #Each split of (Batch size)
splt = [lst[i:i + bat] for i in range(0, len(lt), bat)]

test_counter = [i*len(vgg_data) for i in range(3)]
best_acc = 0

iterations = 100         #Setting no. of epochs
progress_bar = tqdm_notebook(iterable=range(iterations), position=0, leave
=True)

start = time.time()
for epoch in progress_bar:
#for epoch in range(iterations):
  vgg.train()
  correct = 0
  total = 0
  var = 0
  for _, data in enumerate(vgg_data):
    target = vgg_b1_labels[splt[var]]
    target = torch.tensor(target)
    data, target = data.cuda(), target.cuda()
    optimizer.zero_grad()
    output = vgg(data).cuda()
    _, predicted = torch.max(output, 1)
    correct += (predicted == target).sum().item()
    loss = criterion(output, target)
    total += target.size(0)
    acc = (correct / total) * 100.0
    progress_bar.set_description(f"Loss : {loss.item():.3f}, Accuracy : {a
cc:.3f}")
    loss.backward()
    optimizer.step()
    torch.cuda.empty_cache()
    var = var + 1

  acc = 100 * correct / total
  print(f'Epoch : {epoch+1} Accuracy: {acc:.3f}')

print(f'Accuracy of the network on the 60000 train images: {(correct / tot
al) * 100.0:.3f} %')
```

```
# --- Training of the Dataset on VGG (Function of train)--- #
```

```
import time

train_losses = []
train_counter = []
test_losses = []
test_counter = [i*len(VGG_Train) for i in range(3)]
best_acc = 0

iterations = 10         #Setting no. of epochs
#progress_bar = tqdm_notebook(iterable=range(iterations), position=0, leav
e=True)

start = time.time()
#for epoch in progress_bar:
for epoch in range(iterations):
  vgg.train()
  correct = 0
  total = 0
  for batch_idx, (data, target) in enumerate(VGG_Train):
    data, target = data.cuda(), target.cuda()
    optimizer.zero_grad()
    output = vgg(data).cuda()
    #predicted = torch.argmax(output, dim=1)
    _, predicted = torch.max(output.data, 1)
    # print(predicted)
    correct += (predicted == target).sum().item()
    loss = criterion(output, target)             #negative log liklhood lo
ss
    total += target.size(0)
    acc = (correct / total) * 100.0
    #progress_bar.set_description(f"Loss : {loss.item():.3f}, Accuracy : {
acc:.3f}")
    # print(loss)
    loss.backward()
    optimizer.step()

    if acc>best_acc:
      best_acc = acc
      torch.save(network.state_dict(), '/model.pth')
      torch.save(optimizer.state_dict(), '/optimizer.pth')


  acc = 100 * correct / total
  # print('Epoch : ',epoch,' Accuracy: %d',acc)
```

```
print(f'Accuracy of the network on the 60000 train images: {(correct / tot
al) * 100.0:.3f} %')
```

Training and Testing the dataset on LeNet

```
## --- Optimizer for the LeNet --- ##
import torch.optim as optim

network = LeNet().to("cuda")
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(network.parameters() ,lr = 0.001)
```

```
## --- Random data split - Training and Validation Set for LeNet --- ##

dataset = MyDataset('./Train.pkl', './Train_labels.csv', idx=None)

#Defining lengths of training and validation set
train_dsize = int(0.8*len(dataset))
valid_dsize = len(dataset) - train_dsize
train_set, valid_set = torch.utils.data.random_split(dataset, [train_dsize
, valid_dsize])

#Loading data using DataLoader
batch_size = 1000
      #We decide it according to the size for the training or test data
train_dat = DataLoader(train_set, batch_size=batch_size, shuffle=True)
       #Shuffle=False if we want to fix the arrangement of data
valid_dat = DataLoader(valid_set, batch_size=batch_size, shuffle=True)
```

```
## ----- Training Function for the LeNet ----- ##
from tqdm.notebook import tqdm_notebook

def train_lenet(iteration, train_dat):
  network.train()
  correct = 0; total = 0;
  for batch_idx, (data, target) in enumerate(train_dat):
    data, target = data.cuda(), target.cuda()
    optimizer.zero_grad()
    output = network(data).cuda()
    _, predicted = torch.max(output.data, 1)

    correct += (predicted == target).sum().item()
    loss = criterion(output, target)
    total += target.size(0)
```

```
    acc = (correct / total) * 100.0
    progress_bar.set_description(f"Loss : {loss.item():.3f}, Training Accu
racy : {acc:.3f}")

    loss.backward()
    optimizer.step()

  print(f'Epoch : {iteration}; Training Accuracy: {acc:.3f}')

#print(f'ResNet Accuracy on Training Set (48000 images): {(correct / total
) * 100.0:.3f} %')
```

```
## ----- Validation Function for the LeNet ----- ##
from tqdm.notebook import tqdm_notebook

def validate_lenet(val_iter, valid_dat):
  correct = 0; total = 0;
  best_acc = 0
  for batch_idx, (data, target) in enumerate(valid_dat):
    data, target = data.cuda(), target.cuda()
    output = network(data).cuda()
    _, predicted = torch.max(output.data, 1)
    correct += (predicted == target).sum().item()
    total += target.size(0)
    acc = (correct / total) * 100.0
    progress_bar.set_description(f"Validation Accuracy : {acc:.3f}")

    if acc>best_acc:
      best_acc = acc

  print(f'Epoch : {val_iter}; Validation Accuracy: {acc:.3f}')
  return best_acc

#print(f'ResNet Accuracy on Validation Set (12000 images): {(correct / tot
al) * 100.0:.3f} %')
```

```
## ----- Running the epochs of LeNet (Train+Val) ----- ##
import time

epoch = 150          #Set the number of iterations to run
store_epoch = 0
progress_bar = tqdm_notebook(iterable=range(epoch), position=0, leave=True
)
best_acc = 0
```

```
start = time.time()
for i in progress_bar:
  train_lenet(i+1, train_dat)
  val_acc = validate_lenet(i+1, valid_dat)
```

```
# Function of train for the data from dataloader (training over whole data
set) # {Main}

import time

train_losses = []
train_counter = []
test_losses = []
test_counter = [i*len(train_data.dataset) for i in range(3)]

best_acc = 0
iterations = 2500
       #Set number of epoch here
progress_bar = tqdm_notebook(iterable=range(iterations), position=0, leave
=True) #New added, library important statement added at the start

start = time.time()
for epoch in progress_bar:
  network.train()
  correct = 0
  total = 0
  for batch_idx, (data, target) in enumerate(train_data):
    data, target = data.cuda(), target.cuda()
    optimizer.zero_grad()
    output = network(data).cuda()
    #predicted = torch.argmax(output, dim=1)
    _, predicted = torch.max(output.data, 1)
    correct += (predicted == target).sum().item()
    loss = criterion(output, target)             #negative log liklhood lo
ss
    total += target.size(0)
    progress_bar.set_description(f"Loss : {loss.item():.3f}, Accuracy : {(
correct / total) * 100.0:.3f}")
    #print(loss)
    loss.backward()
    optimizer.step()

    if acc>best_acc:
      best_acc = acc
```

```
      torch.save(network.state_dict(), '/model.pth')
      torch.save(optimizer.state_dict(), '/optimizer.pth')

end = time.time()
print(f'Accuracy of the network on the 60000 train images: {(correct / tot
al) * 100.0:.3f} %')

print(f"Time for {epoch} epoch is {end - start} ")
```

```
# --- Testing on the test dataset for kaggle --- #

#model = torch.load('/model.pth')
#out = model(test_data).cuda()
out = network(test_data).cpu()
_, pred_test = torch.max(out, 1)
print(pred_test)
```

For storing the test results

```
## --- For storing the o/p of the test dataset --- ##

import pandas as pd
from google.colab import files

test_id = np.arange(10000)
#df = pd.DataFrame(pred_test.numpy(),test_id, columns = ['class'])        #
(LeNet)Storing the predicted results in the panda dataframe
df = pd.DataFrame(prediction,test_id, columns = ['class'])               #R
esNet
#print(df2)     # columns = ['class', 'id']

df.to_csv('submission.csv')                                             #Sto
ring panda dataframe into .csv file
files.download('submission.csv')                                         #Do
wnloading the csv file
```

APPLYING ENSEMBLE

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
%cd '/content/gdrive/MyDrive/MP3_data/Submissions'
```

```
#Uploading the results of different models
import pandas as pd
import numpy as np

#1 - LeNet
df1 = pd.read_csv(r'./submission1.csv')
cls1 = df1['class']
label1 = np.array(cls1).tolist()

#2 - ResNet18
df2 = pd.read_csv(r'./submission3.csv')
cls2 = df2['class']
label2 = np.array(cls2).tolist()

#3 - ResNet34
df3 = pd.read_csv(r'./submission4.csv')
cls3 = df3['class']
label3 = np.array(cls3).tolist()

#4 - ResNet18
df4 = pd.read_csv(r'./submission5.csv')
cls4 = df4['class']
label4 = np.array(cls4).tolist()

#5 - ResNet34
df5 = pd.read_csv(r'./submission6.csv')
cls5 = df5['class']
label5 = np.array(cls5).tolist()

#6 - ResNet18 (70k)
df6 = pd.read_csv(r'./submission7.csv')
cls6 = df6['class']
label6 = np.array(cls6).tolist()

#7 - ResNet18 (70k); high epoch
df7 = pd.read_csv(r'./submission8.csv')
cls7 = df7['class']
label7 = np.array(cls7).tolist()

#8 - ResNet34 (70k)
df8 = pd.read_csv(r'./submission9.csv')
cls8 = df8['class']
label8 = np.array(cls8).tolist()

#9 - ResNet18 (70k + dropout)
```

```
df9 = pd.read_csv(r'./submission10.csv')
cls9 = df9['class']
label9 = np.array(cls9).tolist()

cls = [cls1, cls2, cls3, cls4, cls5, cls6, cls7, cls8, cls9]
LABEL = pd.concat(cls, axis = 1).to_numpy()
LABEL = LABEL.tolist()
#print(LABEL)
```

```
# Function for the selection of the maximum majority voting #
def most_frequent(List):
  return max(set(List), key = List.count)

max_voting = []
for i in range(10000):
  lst = LABEL[i]
  max_voting.append(most_frequent(L[i]))

print(max_voting)
```

```
#Checking deviation from the submission7 file i.e. last highest acc one on
 kaggle#
correct = 0
for i in range(10000):
  if max_voting[i]==label6[i]:
    correct = correct + 1

acc = (correct / 10000) * 100.0
print(acc)
```

```
# Storing the mojority voted outputs #
from google.colab import files

test_id = np.arange(10000)
df = pd.DataFrame(max_voting,test_id, columns = ['class'])
df.to_csv('submission.csv')
files.download('submission.csv')
```

Extra Below:

```
## ---- New for the data to tensor --- ## (Not needed now)

for epoch in range(2000):
```

```
  optimizer.zero_grad()
  outputs = network(train_img).cuda()
  #predict = torch.argmax(outputs,dim=1)
  #print(outputs)
  #print(predict)
  loss = criterion(outputs, label_img).cuda()
  print(f"{epoch} : loss - {loss.item()}")
  loss.backward()
  optimizer.step()
```

```
#Running the network to see if it's implemented well
net = Net()
print(net)

#Seeing learnable parameters of the network
params = list(net.parameters())
print("\nLength of parameters =",len(params))
print("Size of first layer weight =",params[0].size())  # conv1's .weight
```

```
#Alternate training fucntion, as shown in Pytorch tutorial

for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_data, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
```

```
print('Finished Training')
```

EXTRA

Below section is not needed now, only needed for mounting the data from Kaggle to the Drive

```
!pip install -q kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!ls ~/.kaggle
!chmod 600 /root/.kaggle/kaggle.json
```

```
from google.colab import files
files.upload()
```

```
!kaggle competitions download -c imageunderstanding -p root_path/
```

```
root_path = '/content/gdrive/MyDrive/ECSE_551/Assignment-3'
```

```
!unzip -q '/content/gdrive/MyDrive/ECSE_551/Assignment-
3/test/Test.pkl.zip'
```