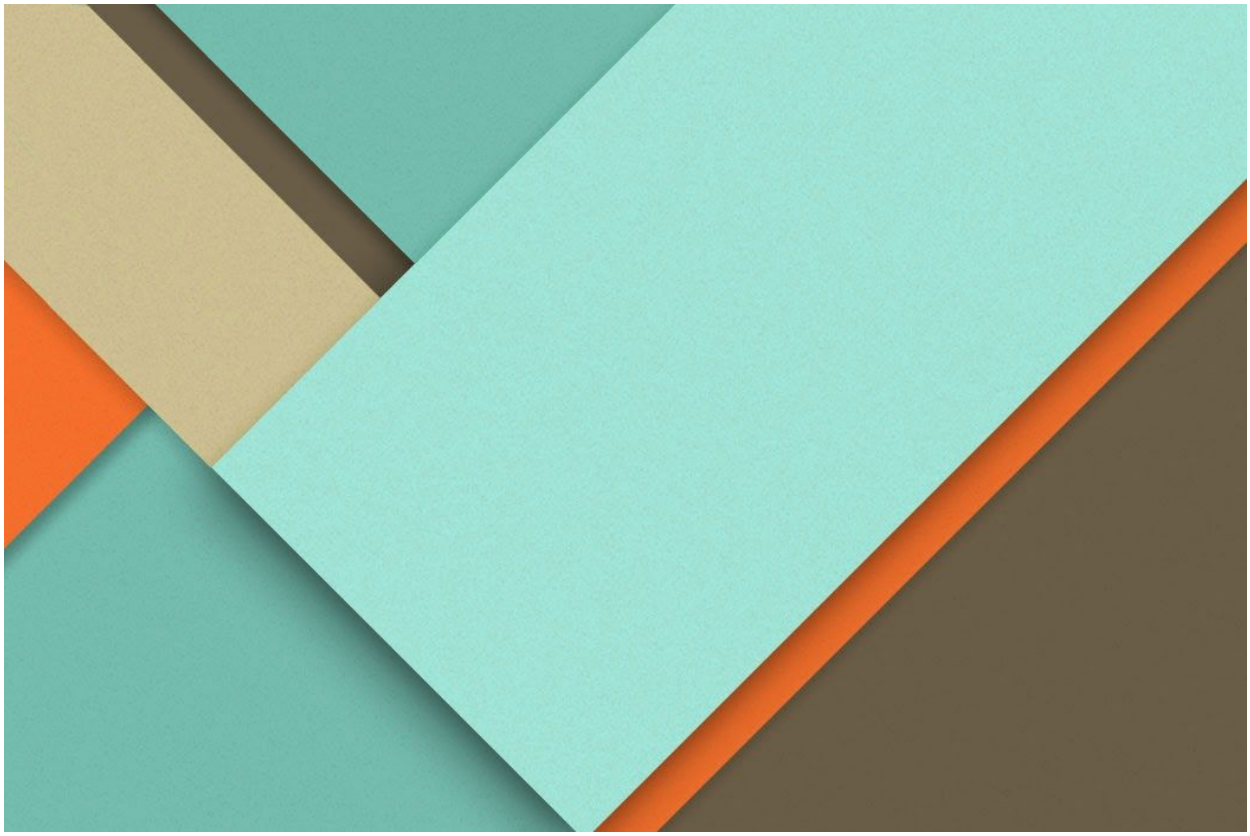


Fashion-Mnist

(using Multilayer Perceptron(MLP) and Convolutional Neural Network(CNN))

Bazil Ahmed
Mtech AI
Sr.No: 16677
Deep Learning

Overview

In this project, I have trained an MLP and a CNN model to classify the Fashion Mnist Dataset in 10 classes.

Goals

1. Choose suitable hyperparameters for MLP and CNN such that it replicates the probability distribution of the training data by an estimated model(minimizing the loss function).
2. The model should not overfit the training data and should be generalizable within a suitable accuracy measure on the test data.

Specifications of the final model (MLP)

Number of hidden layers: 2

Loss function (objective function): cross-entropy

Optimization technique: Adam

Learning rate: 0.001

Batch size for optimization step: 100

Activation function: RELU

Input layer width: 784

Input feature: pixel value

Output layer width: 10

Epochs: 50

Accuracy on test_data: 88.32%

Specifications of the final model (CNN)

Number of CNN layers: 2

Number of FFN layers: 3

Avg pooling layers: 2

Stride length: 2

Kernel filter size:5

Loss function (objective function): cross-entropy

Optimization technique: Adam

Learning rate: 0.001

Batch size for optimization step: 1024

Activation function: RELU

Input layer width: 784

Input feature: pixel value

Output layer width: 10

Epochs: 200

Accuracy on test_data: 89.66%

Precautions used:

1. Training data were randomly permuted to prevent it from any biasing because of batch-wise optimization.
2. While training the model, test data was completely kept aside and only focus was the minimization of the loss on training data.
3. The training set was split into trainable and validation sets(8:2 ratio).
4. With the same hyperparameters, different accuracy on the validation set was observed because of the different shuffling of the input data. Hence random seeding was given for the same shuffling.
5. After every epoch of training, tested on val_set and then used early stopping on the accuracy of val_set by saving the model whenever there was an increase in the val_set accuracy.

6. **Important observation**: early stopping on" val_set loss" was not aligning with early stopping on "Val_set accuracy" as after some epochs the loss on val_set kept increasing but the acc was still improving. Hence finally used accuracy as the stopping criteria.
7. Used RELU as an activation fn because it doesn't have any saturation problem which could have caused a vanishing gradient problem. Although it does have a problem of the gradient at 0, but that gets handled by the subgradient method.
8. The learning rate was kept low because I observed some cycle out in loss value which might be the case of missing local minima.
9. Adam optimization was used because it contains some flavor of the conjugate gradient optimization.

Libraries used in main.py:

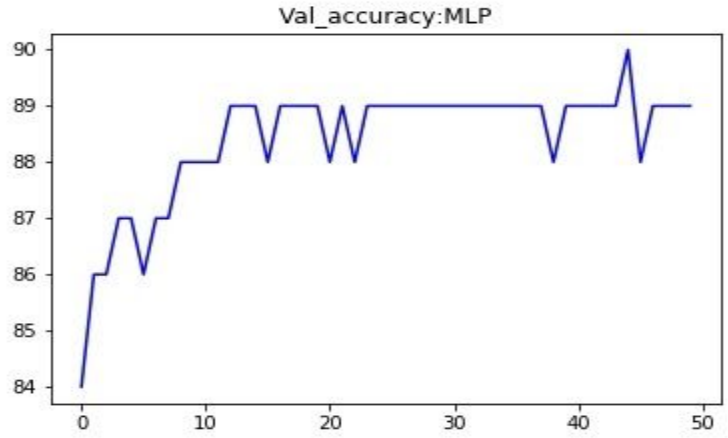
```
import numpy as np

import pdb

import os

from tqdm import tqdm

from matplotlib import pyplot as plt

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

from torch.autograd import Variable

from torch.utils.data import Dataset, DataLoader

from skimage.util import random_noise

import torchvision

from torchvision.datasets import FashionMNIST

from torchvision import transforms

from sklearn.model_selection import train_test_split

from utils import AverageMeter
```

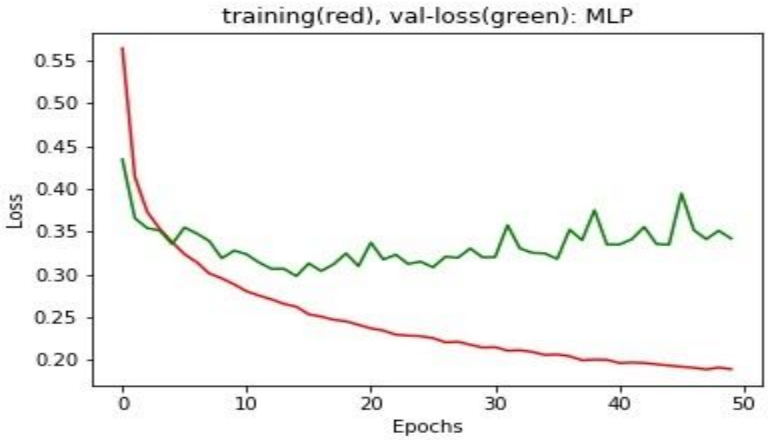
```
from sklearn.metrics import confusion_matrix

from sklearn.metrics import ConfusionMatrixDisplay as cmd
```

Experiments:

Specifics Regarding MLP:

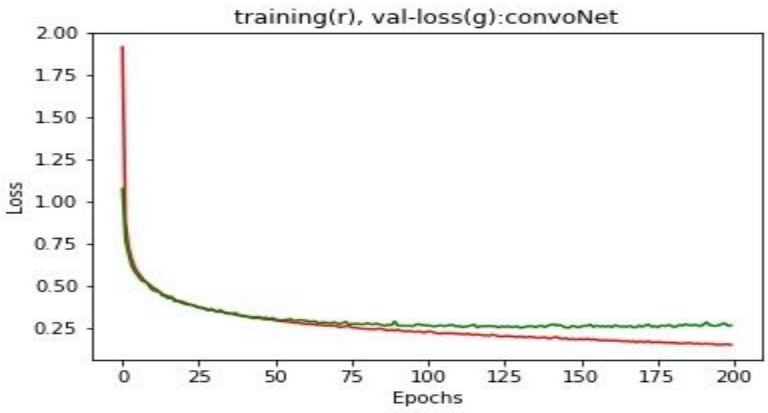
I started with the starter code with 3 hidden layers but then reduced the size without affecting the accuracy on val_ set. Hyperparameters tuning done on val_set then finally was tested on the test data. Notice below that acc spiked up when the loss was degrading. Even tried dropouts with wider layers but still, no improvement was observed although training loss and the val_ loss better aligned with that.

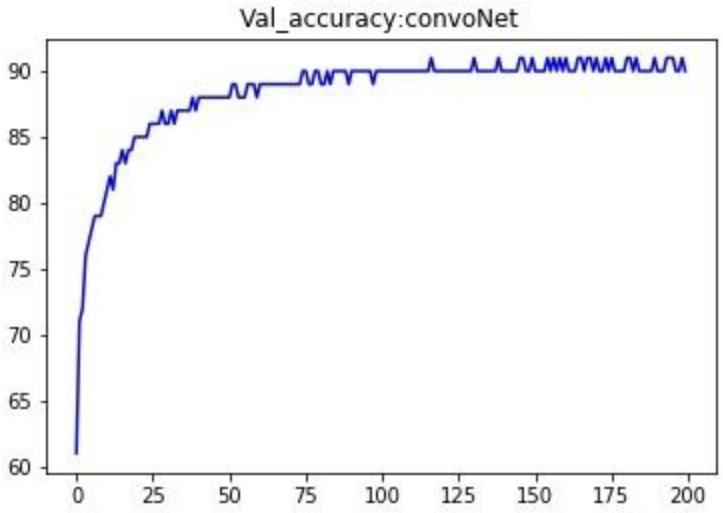


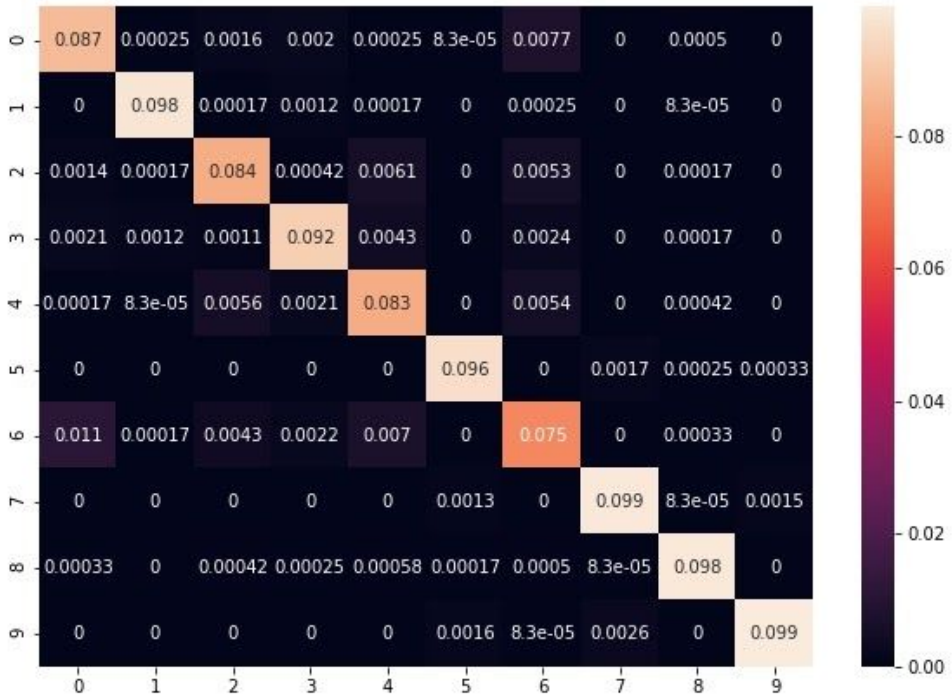


Specifics Regarding CNN:

Again started with the starter code but soon realized there is so much of memory leakage and even the model is quite big. My google collab RAM got exhausted and had to restart again. After trying some variations and blocking memory leakages settled with the final model. Layer details are in the attached .py file.




Val_accuracy:convoNet
90
85
80
75
70
65
60
0
25
50
75
100
125
150
175
200


0 0.087 0.00025 0.0016 0.002 0.00025 8.3e-05 0.0077 0 0.0005 0
1 0 0.098 0.00017 0.0012 0.00017 0 0.00025 0 8.3e-05 0
2 0.0014 0.00017 0.084 0.00042 0.0061 0 0.0053 0 0.00017 0
3 0.0021 0.0012 0.0011 0.092 0.0043 0 0.0024 0 0.00017 0
4 0.00017 8.3e-05 0.0056 0.0021 0.083 0 0.0054 0 0.00042 0
5 0 0 0 0 0 0.096 0 0.0017 0.00025 0.00033
6 0.011 0.00017 0.0043 0.0022 0.007 0 0.075 0 0.00033 0
7 0 0 0 0 0 0.0013 0 0.099 8.3e-05 0.0015
8 0.00033 0 0.00042 0.00025 0.00058 0.00017 0.0005 8.3e-05 0.098 0
9 0 0 0 0 0 0.0016 8.3e-05 0.0026 0 0.099
0 1 2 3 4 5 6 7 8 9
0.08
0.06
0.04
0.02
0.00

Conclusions:

After trying so many variations on MLP the accuracy couldn't surpass CNN model. The reasons could be the spatial locality capturing of CNN which gets lost in MLP. In image classification, spatial locality plays an important role. MLP being fully connected increases the number of learning parameters and hence heavy, causing redundancy and inefficiency compared to CNN which uses shared parameters and weights by filter panning. CNN is also location invariant and looks for patterns in all parts of the image. MLP uses a flattened image and gives no consideration for adjacent image pixels.

What to be expected:

Run the below command in the main.py containing directory.

python3 main.py

It will generate two files namely **convolution-neural-net.txt** and **multi-layer-net.txt**. convNet.pt and MLP.pt named NN model gets picked up from the Model directory automatically by main.py.

cnn_train.py and mlp_train.py are the training files.