IST-718 Homework/Lab-3 (Week9)

Prof. Jonathan M Fox
Student: Sharat Sripada (vssripad)

Introduction

The objective of this lab was to process image data, explore algorithms and build models to predict or accurately label a test image given a large training set of 60,000 - 28x28 images from the MNIST database. The categories of images were broadly organized into:

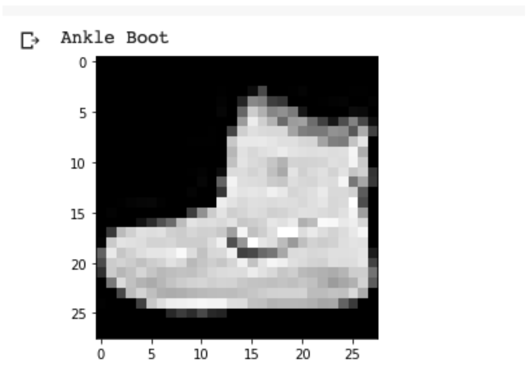
- Fashion (commonly known as the Fashion MNIST)
- Handwritten digits

The lab also exclusively introduced students to the fundamental constructs of neural networks namely, *convolutional neural net (CNN)* and *multi-layer perceptron (MLP).* Measurements of accuracy, trade-offs and performance or run-time of models shall be captured and used to draw conclusions.

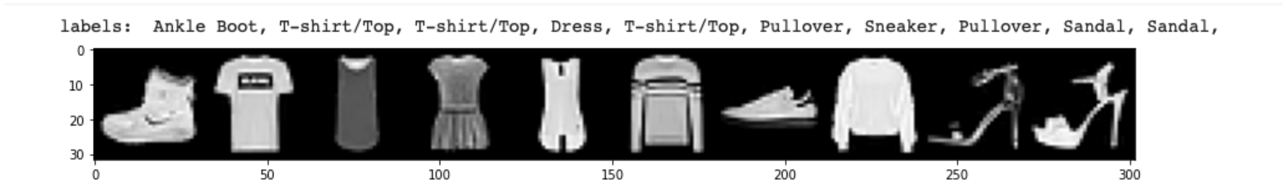
Exploratory Data Analysis

Initial exploratory data analysis was mostly around verifying shape, dimension and image plots from the fashion MNIST and handwritten digits datasets.

Fashion MNIST

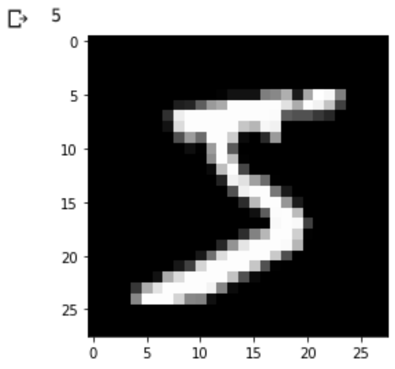


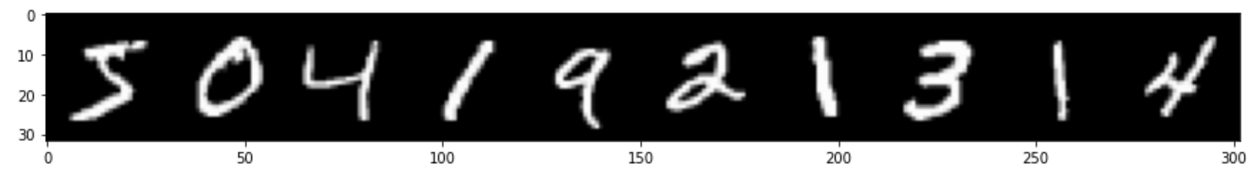
```
Ankle Boot
```



```
labels:  Ankle Boot, T-shirt/Top, T-shirt/Top, Dress, T-shirt/Top, Pullover, Sneaker, Pullover, Sandal, Sandal,
```

Handwritten digits MNIST


```
5
```

```
<matplotlib.image.AxesImage at 0x7f5d46cfc150>
```


Source of data and Python modules

In an attempt to explore different techniques Python modules PyTorch and Tensorflow/Keras was used. Data was correspondingly sometimes obtained from *torchvision.datasets* or *tensorflow.keras.datasets*. In some instances, data was also directly ingested from csv files using the pandas read_csv utility.

Further, for the purpose of utilizing hardware accelerators or GPUs experiments were done on Google Colab.

Exploring Neural Networks

All experiments were run using the following two models for both, the Fashion-MNIST and Handwritten digits – MNIST datasets:

- Convolutional Neural Net/CNN (PyTorch)

  Vs

- Multi-layer Perceptron/MLP (Keras)

MLPs differ from CNNs in that the architecture is less complex (comprises a few dense layers and an output layer), do not comprise computationally intensive convolutional operations and other hidden layers. Therefore, we should expect MLPs to run faster on datasets (given constant EPOCHs and batch-sizes). It would still be interesting however to see if CNNs converge faster and produce better accuracy.

NOTE: It could take several iterations to get the layers, optimizers, activation functions, drop-out rates, learning rates etc. in place.

Convolution Neural Net model

Architecture

Below is the structure of the network or layers of the CNN that would be used to model the datasets:

(1) 2x *Sequential* layers. Each comprising:
   a. **Convolution stage**
      - with kernel size of 3 * 3, padding = 1 and 0 respectively
      - stride 1
   b. **Normalization stage**
   c. **Detector stage**
      - Activation function: ReLU (Rectified linear activation function)
   d. **Pooling stage (Max pooling)**
      - with kernel size of 2 * 2
      - stride 2

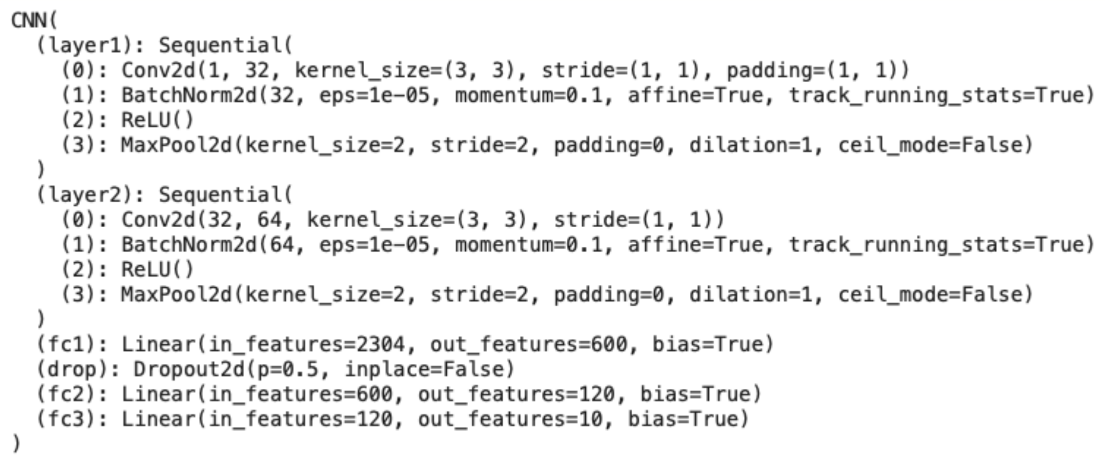
(2) Fully connected or dense layers
   a. 3x Fully connected layers with different in/out features

(3) Dropout layer that has class probability p = 0.25

All the functionality is given in forward method that defines the forward pass of CNN

NOTE: We do not use a softmax/sigmoid activation on the output layer and instead allow Pytorch to apply the built-in Cross-entropy function

Sample of the model output:



```
CNN(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=2304, out_features=600, bias=True)
  (drop): Dropout2d(p=0.5, inplace=False)
  (fc2): Linear(in_features=600, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

Results

Fashion-MNIST

- Iterations/EPOC = 5
- Batch-size = 100
- Validation accuracy of ~89% was obtained.

Further, the accuracy across classes is as seen below:

```
Accuracy of T-shirt/Top: 88.40%
Accuracy of Trouser: 98.40%
Accuracy of Pullover: 87.00%
Accuracy of Dress: 90.20%
Accuracy of Coat: 86.30%
Accuracy of Sandal: 98.60%
Accuracy of Shirt: 65.70%
Accuracy of Sneaker: 90.80%
Accuracy of Bag: 97.20%
Accuracy of Ankle Boot: 98.10%
```

It seems the model was particularly having trouble classifying or labeling *Shirts*. In an attempt to train the model experiments were also run with 20x iterations or EPOCH, which yielded an accuracy of ~90%

Handwritten digits MNIST

- Iterations/EPOCH = 5
- Batch-size = 100
- Validation accuracy of ~98% was obtained.

Performance across all classes showed low variance:

```
Accuracy of 0: 99.49%
Accuracy of 1: 99.74%
Accuracy of 2: 99.13%
Accuracy of 3: 99.11%
Accuracy of 4: 98.37%
Accuracy of 5: 99.10%
Accuracy of 6: 98.64%
Accuracy of 7: 98.54%
Accuracy of 8: 99.18%
Accuracy of 9: 98.71%
```

The run-time for both datasets was comparable at ~66sec for 5x EPOCHs.

Multi-layer perceptron architecture (layers)

The structure or layers of the MLP are far more simpler comprising four layers only, namely:

(1) Fully connected or Dense layers
    a. 3x 512 dense layers using ReLU activation function
(2) Output layer using softmax

NOTE: Categorical cross-entroy will be used as loss function and adding drop-out layers to reduce overfitting or generalization would be considered.

Results

Fashion-MNIST

- Iterations/EPOC = 5
- Batch-size = 100
- Validation accuracy of ~89% was obtained.

Handwritten digits MNIST

- Iterations/EPOCH = 5
- Batch-size = 100
- Validation accuracy of ~98% was obtained.

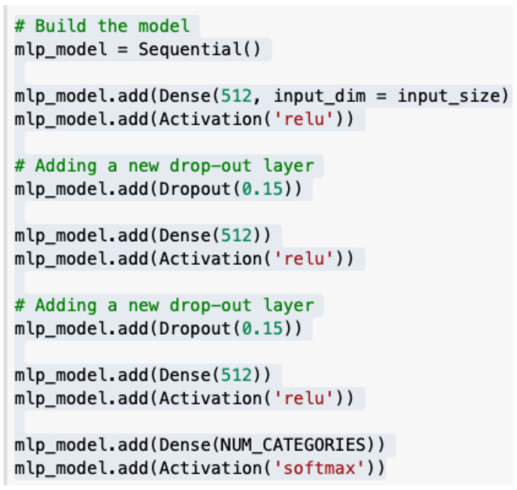
While the accuracy was comparable between MLP and CNN models the run-time for MLP was ~ 1/10th the time of CNNs.

Problem of Overfitting

A neural network has the property to memorize the characteristics of training data. This is called overfitting. To avoid this tendency, the model uses a regularizing layer or function. A commonly used regularizing layer is referred to as a **Dropout layer**.

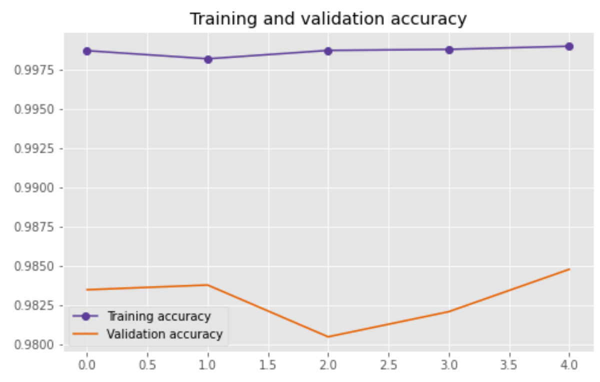
The Dropout layer randomly removes the fraction of units from participating in the next layer and hence makes neural networks robust to unforeseen input data. This is because the network is trained to predict correctly, even if some units are missing.

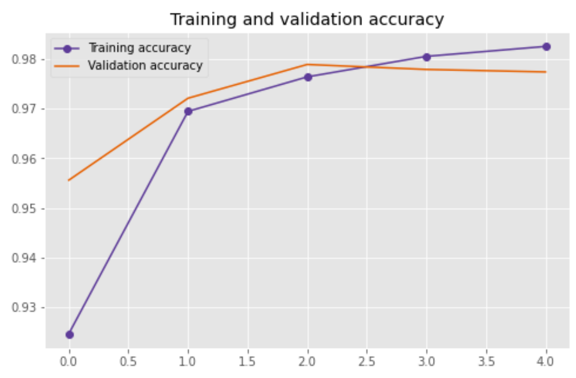
In the Handwritten digits MNIST dataset, two additional drop-out layers were introduced:



```
# Build the model
mlp_model = Sequential()

mlp_model.add(Dense(512, input_dim = input_size)
mlp_model.add(Activation('relu'))

# Adding a new drop-out layer
mlp_model.add(Dropout(0.15))

mlp_model.add(Dense(512))
mlp_model.add(Activation('relu'))

# Adding a new drop-out layer
mlp_model.add(Dropout(0.15))

mlp_model.add(Dense(512))
mlp_model.add(Activation('relu'))

mlp_model.add(Dense(NUM_CATEGORIES))
mlp_model.add(Activation('softmax'))
```

The corresponding training and validation accuracy were plotted but no significant improvements were observed:





Conclusions

The experiments and corresponding results are summarized here (all with GPU on Google Colab):

Method	MNIST	Epoch	Sample	Drop-out	Learning-Rate	Accuracy	Run-Time (sec)
CNN	Fashion	5	100	0.25	0.001	89	65
CNN	Fashion	5	100	0.5	0.001	89	66
CNN	Fashion	20	100	0.25	0.001	90	257
MLP	Fashion	5	100	x	x	89	10
CNN	Digits	5	100	0.25	0.001	98.9	65
MLP	Digits	5	100	x	x	97	9
MLP	Digits	5	100	0.3	x	97	9

CNNs are popular and widely preferred by data-scientists for its ability to solve complex problems in the field of machine-learning, deep-learning and computer-vision. However, in this context (datasets pertaining to Lab-3) MLPs seem to offer at par accuracy at considerably lower run-times and hence offer better performance.

Future Work

Epoch, Sample, Drop-out, Learning-Rate (some of the variables above) along-with subtleties around organizing CNN layers – combinations of convolution, pooling layers etc. and choosing between optimizers or activation functions could be some experiments in store. The trade-off however will possibly always remain that CNNs are more compute hungry.