COSC 4368: Fundamentals of Artificial Intelligence Spring 2024
Problem Set 2 (Individual Tasks[1] Centering on Neural Networks)
First Draft

TASK 3: Build a Classical Neural Network Model for the Fashion MNIST Dataset *Raunak*

Name: Hyundoo Jeong
PSID: 2212332

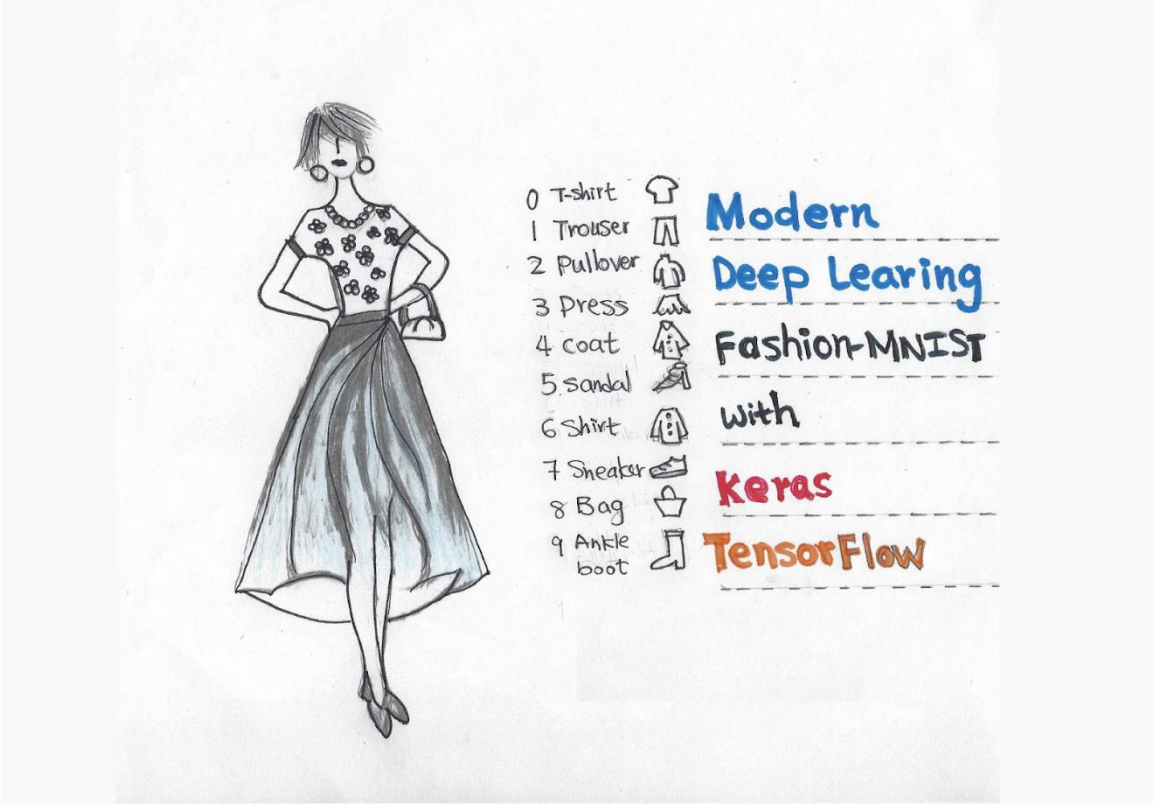


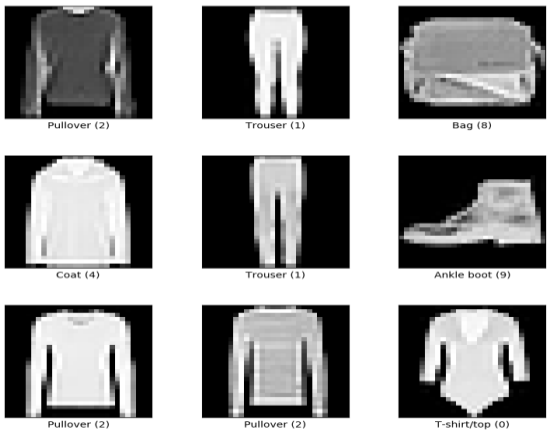
Fig. 1: Training a model to classify images is the archetypal neural network task

Submission Deadlines Task3: Tue., Mar 26, 11:59pm.
Last Updated: March 5, 10a.
Weight Task3: 25 points

In this task, you will explore the world of deep learning by building a neural network to classify clothing items in the Fashion MNIST dataset. You will learn how to build, train, and evaluate a model, gaining valuable insights into neural network capabilities.

[1] Collaboration with other students is not allowed!

Dataset:
Fashion MNIST is a popular image classification dataset containing 70,000 grayscale images of 10 different clothing items (T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot). We will use this dataset to train our neural network to identify these clothing items. Below some samples are given from Fashion MNIST dataset:



Tasks:

1. Preprocessing (You can use method used in notebook given for Task 4)

i. Download the Fashion MNIST dataset using TensorFlow or another suitable library.

ii. Preprocess the data: normalize pixel values, split the data into training and testing sets.

2. Building the Neural Network (5 pts)

i. Choose a neural network architecture suitable for image classification. Consider using a Convolutional Neural Network (CNN) due to its effectiveness in this domain.

Answer: I will create CNN model captures important image features using convolutional and pooling layers, then classifies the images with dense layers and 'softmax' function, all optimized for accuracy with Adam optimizer, and loss funcion with using categorical cross-entropy

ii. Define the network architecture in your chosen library (e.g., TensorFlow, PyTorch). Use layers like convolutional, pooling, and dense layers.

```
In [25]: import tensorflow as tf
         from tensorflow.keras import layers, models

         # Define the model with convolutional layers
         model = models.Sequential([
           # Convolutional layer with 32 filters, kernel size of 3, activation 'relu', and input shape as the first layer
           layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
           # MaxPooling layer with pool size of 2
           layers.MaxPooling2D((2, 2)),
           # Second convolutional layer with 64 filters
           layers.Conv2D(64, (3, 3), activation='relu'),
           # Second MaxPooling layer with the same pool size
           layers.MaxPooling2D((2, 2)),
           # Third convolutional layer with 64 filters
           layers.Conv2D(64, (3, 3), activation='relu'),
           # Flatten the output of the last convolutional layer to feed it into dense layers
           layers.Flatten(),
           # Dense layer with 64 units and 'relu' activation
           layers.Dense(64, activation='relu'),
           # Output dense layer with 10 units for classification and 'softmax' activation
           layers.Dense(10, activation='softmax')
         ])
```

Answer: I used TensorFlow and using conv2D layer-> as considering Convolutional layer, setting up Carpooling to reduce computation, and using layer. Dense as considering dense layer.

iii. Compile the model, specifying the optimizer (e.g., Adam) and two separate loss function (e.g., categorical cross-entropy).

```
In [26]: #compile model
         model.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.KLDivergence(),
              metrics=['accuracy'])
```

Answer: I set up two different loss function: one is categorical cross-entropy, another is KLDivergence(), but after trying to fit in another model of loss's function of KLDivergence, it happened keep error in terms of model fitting process because of trace, so I couldn't create this model fully.

3. Training and Evaluation (10 pts)

i. Train the model on the training set with epochs numbering between 10/15/20.
Answer: I chose epochs number 20

```
history = model.fit(
  X_train, y_train,          # Training data and labels
  epochs=20,                 # Train for 20 epochs, or try 10 or 15 as per recommendation
  validation_data=(X_test, y_test),  # Validation data to monitor the performance
)
```

ii. Monitor the training process by logging metrics like accuracy and loss after each epoch.

Answer: accuracy is gradually increasing, loss function is decreasing because model is learning and improving prediction on this training set, and model is making fewer mistakes as training progress.

iii. Evaluate the model's performance on the test set using metrics like accuracy, precision, and recall for each class.

```
Train Accuracy Score: 0.70138
Test Accuracy Score: 0.7018
              precision    recall  f1-score   support

           0       0.65      0.75      0.70       942
           1       0.91      0.93      0.92      1027
           2       0.60      0.56      0.58      1016
           3       0.74      0.81      0.77      1019
           4       0.55      0.64      0.59       974
           5       0.50      0.72      0.59       989
           6       0.51      0.09      0.15      1021
           7       0.74      0.83      0.78      1022
           8       0.91      0.80      0.86       990
           9       0.83      0.91      0.87      1000

    accuracy                           0.70     10000
   macro avg       0.70      0.70      0.68     10000
weighted avg       0.70      0.70      0.68     10000
```

Answer: My accuracy is 70% and this model has moderate level of performance and is generalizing well, given that the training set and test set are close each other.

iv. Experiment with different hyperparameters to improve the model's performance.
   - learning rate (0.1/0.25/0.30),
   - number of layers (2/3/5),
   - neurons per layer (10/20/30)

Answer:

```
Training model with lr=0.1, n_layers=2, n_neurons=10
313/313 - 1s - 2ms/step - accuracy: 0.3268 - loss: 1.4707
Test Accuracy: 0.32679998874664307

Training model with lr=0.1, n_layers=2, n_neurons=20
313/313 - 1s - 3ms/step - accuracy: 0.2869 - loss: 1.5962
Test Accuracy: 0.28690001368522644

Training model with lr=0.1, n_layers=5, n_neurons=30
313/313 - 1s - 2ms/step - accuracy: 0.1144 - loss: 2.2848
Test Accuracy: 0.1143999993801117

Other training models with different parameters except for 3 results
has exactly similar accuracy but loss function rate is slightly
different.
```

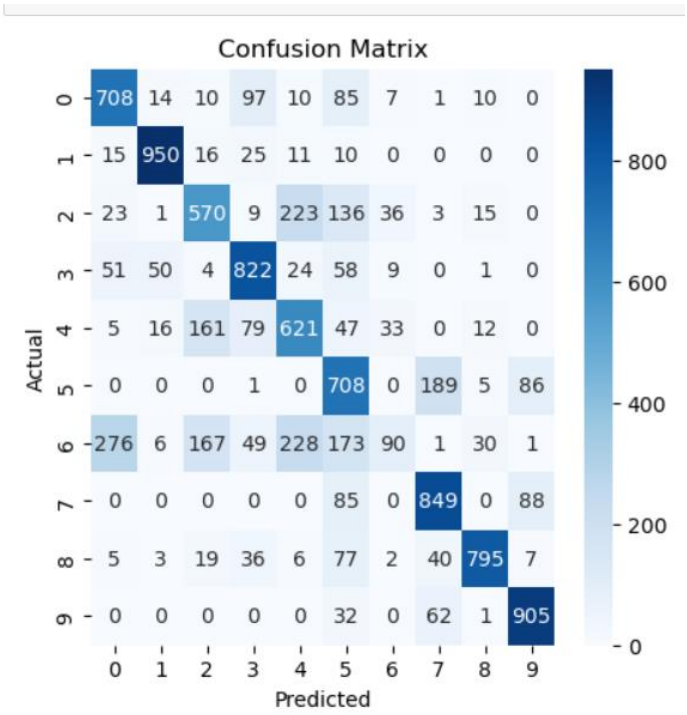
Discuss Result: I noticed 3 different hyperparameters affected accuracy and loss function rate. The problem is here, my model has low accuracy because of maximum 33% which is significantly bad to detect items. The way of improvement is reducing learning rate such as lr = 0.001 and number of layers, and neuron fix value. It would be taking more time, but it will be little bit improving accuracy(Performance).

4. Analysis (10 pts)

i. Analyse the training and evaluation results. Are there any classes with particularly low accuracy?

Answer: Above my evaluated image, class 6 which is Shirt stands out with particularly low accuracy because it has precision 0.51 and recall of 0.09 which is significantly lower than other classes. Also, f1-score of 0.15 confirms that it is poorly represented by this model.

ii. Display the model's confusion matrix. Can you identify any interesting patterns or mistakes? Visualize the lowest performing class.



Answer: The diagonal part of the matrix, which represents the number of correct predictions for each class, shows that classes 1,3,7,9 are well predicted with high value.

Also, when you look at class 6, value is 90 and the lowest value among other classes.

iii. Determine which class is the most difficult to predict by seeing which has the lowest F2. Try to explain why this class is challenging to predict.

Answer: F2 score is calculated by F2 = (1 + 2^2) * (precision * recall) / (4 * precision + recall)

I calculated each of classes.
Class 0 = 0.73, Class 1 = 0.93, Class 2= 0.57, Class 3 = 0.79, Class 4 = 0.62
Class 5 = 0.66, Class 6 = 0.11, Class 7 = 0.81, Class 8 = 0.82, Class 9 = 0.89

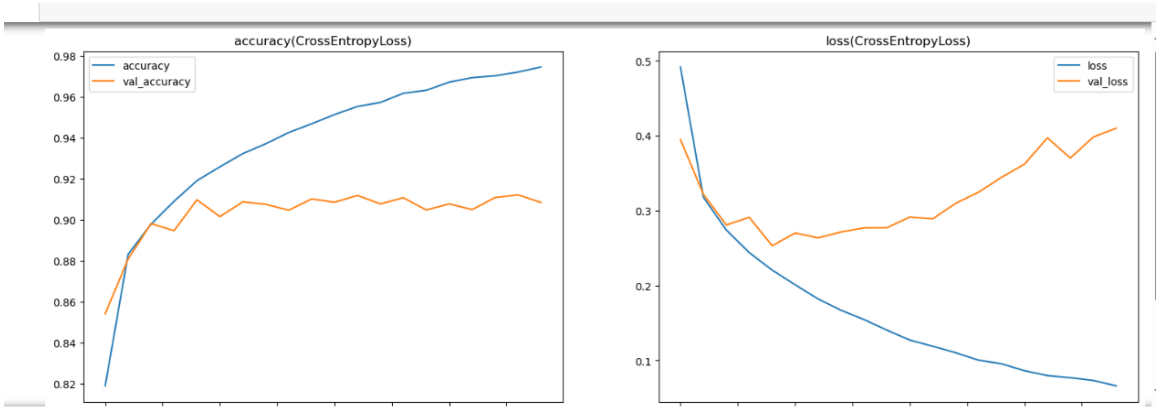
Class 6 has the lowest F2 value, and it is the most difficult to predict with each of classes. The potential reason for class 6(Shirts) being difficult prediction is overlapping feature of other classes. For example, Both Class 0 and Class 3(T-shirt/Dress) are similar features of shirt, so some of data overlapped and it made it hard for the model to distinguish between them.

Submission:

- Submit a report including:
  - Clear explanation of your chosen architecture and its rationale.
  - Visualization of training and evaluation metrics (graphs, confusion matrices).
  - Your answers to the analysis questions.
  - Discuss the limitations of your model and potential improvements for future exploration. -> 3. Training and Evaluation: Discuss Result part.

Bonus: (up to 5 pts)

- Visual explanations concerning what kind of errors the Neural Network models makes and which pairs of classes are hard to distinguish for the Neural network model.

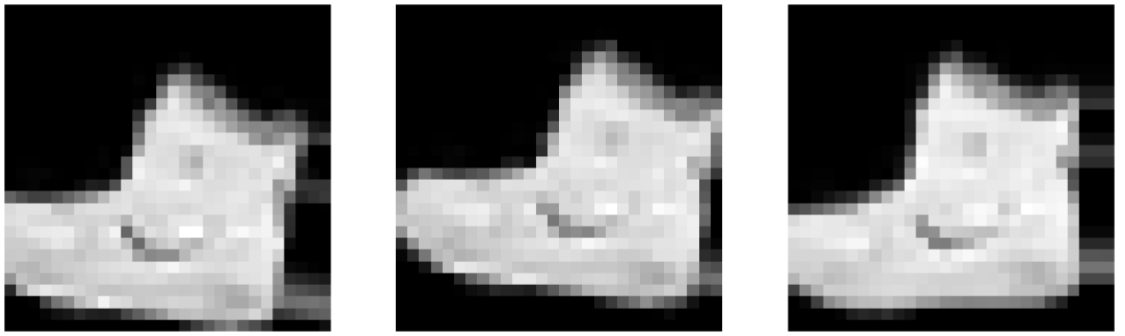


Answer: overfitting could be happening based on my graph because my training set is drastically increasing but my validation data is not significantly increasing such as it is slowly little bit increasing and converge.

- Implement data augmentation techniques to improve model generalizability and accuracy.

```
[91]: # This will do preprocessing and realtime data augmentation
augmented_datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    rotation_range=25,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=False,  # randomly flip images
    vertical_flip=False)  # randomly flip images
```

Answer: my data augmentation is set up to perform geometric transformations to the image (rotation, shift) which can help model to distinguish different object of image, and it will be improving accuracy.



- Apply your model to classify images from a different dataset of your choice.

Resources:

- Fashion MNIST dataset: http://pytorch.org/vision/stable/generated/torchvision.datasets.FashionMNIST.html
- TensorFlow tutorials: https://www.tensorflow.org/tutorials
- PyTorch tutorials: https://pytorch.org/tutorials/

Pre-Processing Example:

```
from keras.utils import to_categorical
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

data_train = pd.read_csv('../input/fashion-mnist_train.csv')
data_test = pd.read_csv('../input/fashion-mnist_test.csv')

img_rows, img_cols = 28, 28
input_shape = (img_rows, img_cols, 1)

X = np.array(data_train.iloc[:, 1:])
y = to_categorical(np.array(data_train.iloc[:, 0]))

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=13)

X_test = np.array(data_test.iloc[:, 1:])
y_test = to_categorical(np.array(data_test.iloc[:, 0]))

X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
X_val = X_val.reshape(X_val.shape[0], img_rows, img_cols, 1)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_val = X_val.astype('float32')
X_train /= 255
X_test /= 255
X_val /= 255
```