Fashion MNIST TensorFlow Model

Presented to

Prof. Farzad Amirjavid, PhD in CS
Artificial Intelligence in Canadian Perspective
Lambton College

Prepared by

Karthikeyan Mohan

24th March 2021

Table of Contents

A. Project Overview

We have been provided with three different Artificial intelligence projects such as FB Prophet, Resnext / Resnet model, and the Fashion MNIST dataset to build a TensorFlow. FB Prophet model has already been done as part of the first assignment, and I had already explored the Resnext model. I wanted to learn and do the hands-on project on TensorFlow from scratch. In the class, we worked on one of the MNIST datasets to build a TF model, and it was interesting and exciting to work on. Hence, I chose this one to learn and explore new ideas as part of this assignment.

B. Problem Statement

As part of this project, we have to use the Fashion MNIST dataset to build a TensorFlow model from scratch. This image classification problem is where we have to train the model with the Fashion dataset and identify the image that comes under which Fashion category.

C. Design Approach

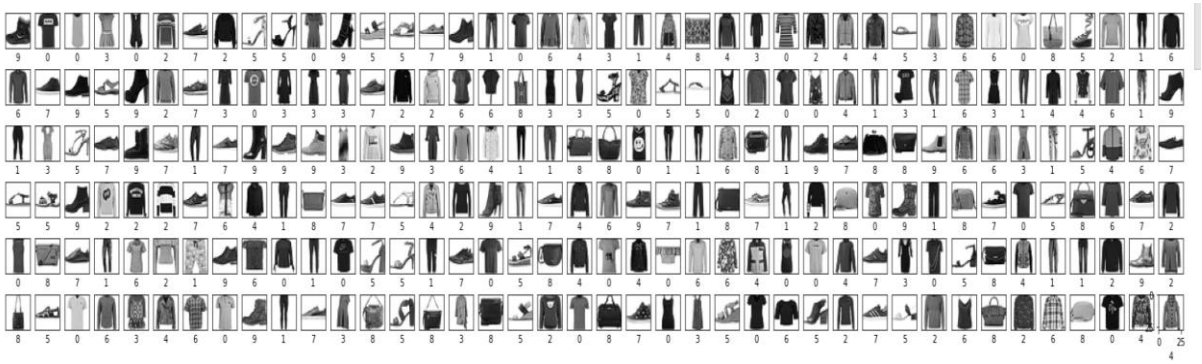
Before start doing the project, I had come up with the below methods to build the model:

- Import the dataset and analyze the input dataset.
- Split the test and train data for training and evaluating the model.
- Identify the method to do Normalization of the dataset.
- Preprocess the data and convert the dataset as required by the model.
- Build the model using the CNN architecture.
- Compile the model and train the model with the required epoch iteration.
- Fit the model created into the test dataset and evaluate the model.
- Finally, provide the image as an input to our model and predicts the output.

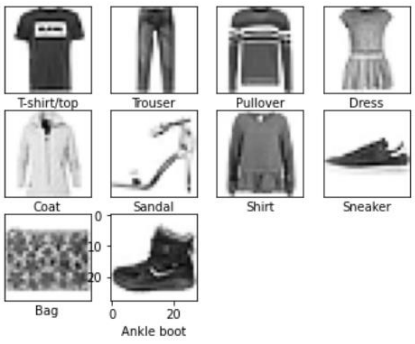
D. Exploratory Data Analysis

- We have imported the NumPy and Matplotlib package required to do the EDA.
- Loaded the Fashion dataset into the training and test data from the TensorFlow dataset.

Written a code to Visualize the sample top 240 data.



- Training data count and pixel size - (60000, 28, 28)
- Test data count and pixel size - (10000, 28, 28)
- There are 10 categories in the output variable, and each class is equally split with 6000 records. Below is the sample in each one of them.



E. Data Preparation

1. Data Normalization
   - we normalized the pixel value ranging from black (0) to white (255) into the values from 0 to 1 by converting them to float and dividing it with the maximum value.
   - Our output class is categorical in no order of 0 to 9. If we use this data, our model will give rank the category based on the numerical value. Hence, we need to normalize this value with the one-hot encoding method.
2. Data Sizing
   - The input data image is in with the pixel size 28*28, and in this step, we reshaped the image to all in the single-color channel by reshaping it into (28,28,1).

F. Model Building and Evaluation

We used the sequential API method with the 6 different layers in the below order to build our model. In the first layer, we always need to mention the pixel size for the machine to understand explicitly.

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

In the last layer, we used the number of categories 10 as a parameter to model to understand and complete.

```
model.fit(x=train_x, y=train_y, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 63s 33ms/step - loss: 0.7108 - accuracy: 0.7435
Epoch 2/10
1875/1875 [==============================] - 61s 32ms/step - loss: 0.3201 - accuracy: 0.8838
Epoch 3/10
1875/1875 [==============================] - 61s 32ms/step - loss: 0.2571 - accuracy: 0.9047
Epoch 4/10
1875/1875 [==============================] - 61s 32ms/step - loss: 0.2276 - accuracy: 0.9173
Epoch 5/10
1875/1875 [==============================] - 63s 34ms/step - loss: 0.1931 - accuracy: 0.9308
Epoch 6/10
1875/1875 [==============================] - 61s 33ms/step - loss: 0.1728 - accuracy: 0.9374
Epoch 7/10
1875/1875 [==============================] - 61s 33ms/step - loss: 0.1503 - accuracy: 0.9432
Epoch 8/10
1875/1875 [==============================] - 61s 33ms/step - loss: 0.1359 - accuracy: 0.9502
Epoch 9/10
1875/1875 [==============================] - 62s 33ms/step - loss: 0.1147 - accuracy: 0.9568
Epoch 10/10
1875/1875 [==============================] - 62s 33ms/step - loss: 0.0979 - accuracy: 0.9633
<tensorflow.python.keras.callbacks.History at 0x7feddec37e50>
```

We then fit our trained model into test data by providing 10 epochs to run the model in 10 different patterns to increase its accuracy.

```
opt = SGD(lr=0.01, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

Because it's a categorical value, we used categorical_crossentropy loss in the model compilation and evaluated the model with an accuracy of **0.9116**.

```
model.evaluate(test_x,test_y)
```

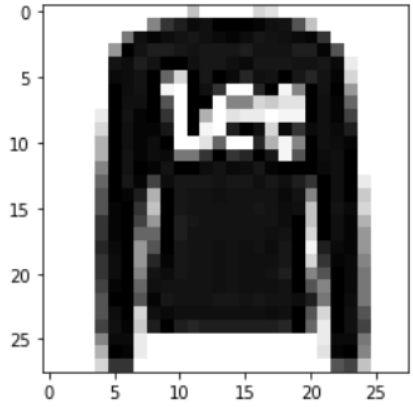
```
313/313 [==============================] - 3s 10ms/step - loss: 0.2781 - accuracy: 0.9128
[0.27810028195381165, 0.9128000140190125]
```

G. Model Prediction

We predicted the result to compare our predicted values with the actual test output data. In the end, we loaded an image from the drive and preprocessed the image. Then we provided the processed image data as an input to the model and predicted the output. We processed all the same steps for our image and predicated the result from our existing model.

```
print("User provided image is below")
plt.imshow(user_input_unprocessed,cmap='Greys')
#print("Predicted value is :",output_class[model_predicion])
print("Predicted value is :",fashion_mnist_labels[model_predicion[0]])
```

```
User provided image is below
Predicted value is : Pullover
```



H. Conclusion

I had completed and developed our model to predict the un-trained data. As a future enhancement, I would like to build it as a simple application to receive input from the user and deploy it on the AWS server. Doing these tasks all by myself from scratch helped me to understand each concept well. I'm motivated to start doing more self-learning projects other than academics.

Code: https://colab.research.google.com/drive/11vugHxzA8yFhOok1niDkUcIApg2Im-J_?usp=sharing