CS6005 DEEP LEARNING

MINI PROJECT 1 – 11/04/2021

Convolutional Neural Network for Fashion MNIST Dataset

Name: Reuel Samuel Sam
Registration Number: 2018103053
Batch: P

NOTE: *Resubmission – Originally submitted on 11th April 2021. Changes made – Section 3 (Modules) and Section 6 (Results) – Updated as of 19th April 2021*

Abstract – Image classification is an active research area and has been studied in popular applications such as driverless vehicles and emergency robots. We propose a convolutional neural network (CNN)-based architecture using the Fashion MNIST Dataset, which has a total of 70000 images. These images are divided into training and test sets, each with 60000 and 10000 images, respectively. The CNN model opted for this study does not make use of a pooling layer after every convolution layer. Instead, a max pooling layer is added after two convolution layer.

I. PROBLEM STATEMENT

Image Classification, being an active research topic, has gained a lot of traction in recent years. This field is great to begin exploring deep learning and get used to the libraries and packages involved. The specific problem at hand is the classifications of images present in the Fashion MNIST Dataset. This dataset contains 70000 grayscale images of pixel size 28x28 classified into 10 different classes. The challenge here is to classify these images accurately into their respective classes. To do so, a model built on a Convolutional Neural Network is adopted.

P.T.O

II. DATASET DETAILS

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Zalando intends Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms [1]. It shares the same image size and structure of training and testing splits.

The 10 different target classes in this dataset are as follows:

- ⇨ T-Shirt/Top
- ⇨ Trouser
- ⇨ Pullover
- ⇨ Dress
- ⇨ Coat
- ⇨ Sandal
- ⇨ Shirt
- ⇨ Sneaker
- ⇨ Bag
- ⇨ Ankle Boot

There are 7000 images per category and it provides a new benchmark for Deep Learning Algorithms at a higher complexity when compared to the original Fashion MNIST.

III. MODULES

Python comes with a rich library of modules that make coding rather convenient. This is no different when it comes to Deep Learning. Python has plenty of Deep Learning Libraries that can be made use of. Following are the modules that have been imported and applied in this project

- ⇨ *TensorFlow:* Open Source Machine Learning Platform
- ⇨ *Keras:* Open Source Library that provides python interface for artificial neural networks
- ⇨ *Matplotlib:* Graphical Library (Not necessarily required)
- ⇨ *Numpy:* Mathematical Library

Apart from Python modules, the code can be broken down into modules that run independently:

- *Loading of Data:* The dataset, being quite well known, has been imported from Keras Library's dataset collection. The dataset is retrieved and stored as train images, train labels, test images and test labels. These are then used accordingly.

- *Preprocessing of Data:* Since the dataset contains grayscale images, it has to be reshaped from (60000, 28, 28) to (60000, 28, 28, 1) for training images and (10000, 28, 28) to (10000, 28, 28, 1) for testing images. The fourth value in this shape indicates the number of color channels. This needs to be defined to avoid an error when using Keras CNN library. Apart from this, the dataset is converted to float type normalized by dividing the pixel values by 255

- *Model Creation:* The architecture of the model has been expanded upon in the next section along with a diagram for a visual representation. The main difference in this code in comparison with most codes is the use of one Pooling Layer after two Convolution Layers apart from a Pooling Layer after every Convolution Layer.

- *Hyperparameters:*
  - Epochs: 5
  - Optimizer: SGD
  - Learning Rate: 0.01
  - Momentum: 0.9

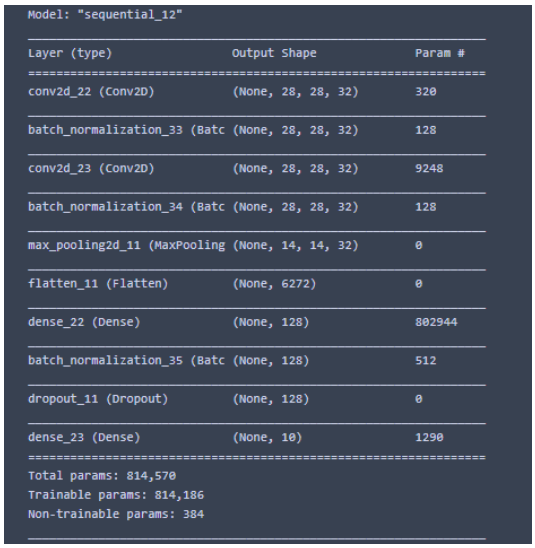
IV. CNN MODEL SUMMARY

The Model Architecture adopted to address the problem at hand (classification of Fashion MNIST Dataset) follows a basic CNN Structure. The different layers used are:

- *Convolution Layer*
  - Parts of the image are taken and compared. These parts are called features/kernels
  - Each pixel is multiplied with its corresponding feature pixel. Find mean for that specific feature and replace the original pixel with this new value

- ⇨ *Max Pooling Layer*
  - Used to shrink the image
  - Specific window size is chosen and the window is moved along the ReLU activated image
  - Maximum pixel value within the window is chosen to represent the group
- ⇨ *Fully Connected Dense Layer and Flatten Layer*
  - Conversion of the resultant image into a flattened vector
- ⇨ *Dropout Layer*
  - Random inputs are set to 0 during training while the inputs are scaled up. This is to avoid the problem of over-fitting
- ⇨ *Batch Normalization*
  - Standardizes the input at each layer for a mini batch
  - Rescaling of data to have a mean of 0 and standard deviation of 1

The Keras Model Summary is shown below:

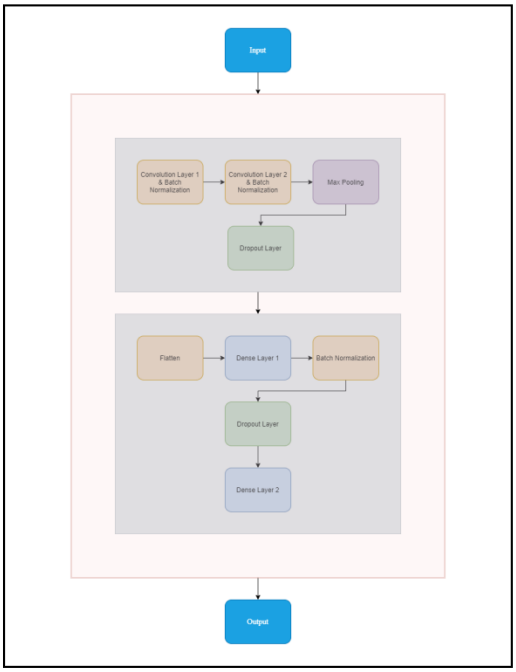


```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
batch_normalization_33 (Batc (None, 28, 28, 32)        128
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 28, 28, 32)        9248
_________________________________________________________________
batch_normalization_34 (Batc (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
flatten_11 (Flatten)         (None, 6272)              0
_________________________________________________________________
dense_22 (Dense)             (None, 128)               802944
_________________________________________________________________
batch_normalization_35 (Batc (None, 128)               512
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_23 (Dense)             (None, 10)                1290
=================================================================
Total params: 814,570
Trainable params: 814,186
Non-trainable params: 384
_________________________________________________________________
```

Keras Model Summary

P.T.O

A System Architecture diagram is also provided to get a better understanding of the model:



System Architecture Diagram

V. CODING SNAPSHOTS

Habitual practice has prompted the code to the Class oriented to allow for importing and integration in GUIs or other programs.

As a result, the code has been divided into methods. These methods have been shown below:

1) *Imports*

```
import tensorflow as tf
from tensorflow.keras import datasets,layers, models
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPool2D,Dense,Flatten,Dropout,Input, AveragePooling2D, Activation,Conv2D, MaxPooling2D, BatchNormalization,Concatenate
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard
from tensorflow.keras import regularizers, optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
```

2) *Initialization of the class*

```
class FashionMNIST_CNN():
    def __init__(self, epochs):
        self.class_names = ["T-shirt/Top", "Trouser", "Pullover", "Dress", "Coat", \
                            "Sandal", "Shirt", "Sneaker", "Bag", "Ankle Boot"]
        self.epochs = epochs
```

3) *Loading of data set*

```
def fetch_data(self):
    dataset = tf.keras.datasets.fashion_mnist.load_data()
    (self.train_images, self.train_labels), (self.test_images, self.test_labels) = dataset
    self.print_shape()
    self.classes = np.unique(train_labels)
    self.nClasses = len(classes)
    print("Total number of clases: ", self.nClasses)
    print("Target Classes: ", self.classes)

def print_shape(self):
    print("Training Data Shape: ", train_images.shape, train_labels.shape)
    print("Testing Data Shape: ", test_images.shape, test_labels.shape)
```

4) *Displaying few samples from the dataset*

```
def example_data(self, n):
    plt.figure(figsize=(10,10))
    for i in range(n):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(self.train_images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[self.train_labels[i]]).set_color('white')
    plt.show()
```

5) *Preprocessing*

```
def preprocess(self):
    self.train_images = self.train_images.reshape((self.train_images.shape[0], 28, 28, 1))
    self.test_images = self.test_images.reshape((self.test_images.shape[0], 28, 28, 1))
    nRows, nCols, nDims = self.train_images.shape[1:]
    self.train_data = self.train_images.reshape(self.train_images.shape[0], nRows, nCols, nDims)
    self.test_data = self.test_images.reshape(self.test_images.shape[0], nRows, nCols, nDims)
    self.input_shape = (nRows, nCols, nDims)

    self.train_data = self.train_data.astype('float32')
    self.test_data = self.test_data.astype('float32')

    self.train_data /= 255
    self.test_data /= 255

    self.train_label_one_hot = to_categorical(self.train_labels)
    self.test_label_one_hot = to_categorical(self.test_labels)
```

6) Model Creation

```
def model_create(self):
    self.model = Sequential()
    self.model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', \
                          padding='same', input_shape=self.input_shape))
    self.model.add(BatchNormalization())
    self.model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', \
                          padding='same'))
    self.model.add(BatchNormalization())
    self.model.add(MaxPool2D((2, 2)))

    self.model.add(Flatten())
    self.model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    self.model.add(BatchNormalization())
    self.model.add(Dropout(0.5))
    self.model.add(Dense(self.nClasses, activation='softmax'))
```

7) Model Summary

```
def model_summary(self):
    self.model.summary()
```

8) Model Compilation

```
def model_compile(self):
    opt = optimizers.SGD(lr=0.01, momentum=0.9)
    self.model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return 0
```

9) Driver Function for the model

```
def model_start(self):
    self.preprocess()
    self.model_create()
    self.model_summary()
    var = self.model_compile()
    print("Model Creation Done")
```

10)Training

```
def train(self):
    self.history = self.model.fit(self.train_data, self.train_label_one_hot, epochs=self.epochs, \
                                   validation_data=(self.test_data, self.test_label_one_hot))
```

11) Evaluation

```
def model_evaluate(self):
    self.test_loss, self.test_acc = self.model.evaluate(self.test_data,  self.test_label_one_hot, verbose=2)

    print("Test Accuracy: ", self.test_acc)
```

12) Driver

```
[89]
model = FashionMNIST_CNN(5)
model.fetch_data()
[90]
model.example_data(15)
[91]
model.model_start()
[92]
model.train()
[93]
model.model_evaluate()
```

VI. RESULTS

The different outputs corresponding to the driver code has been shown below:

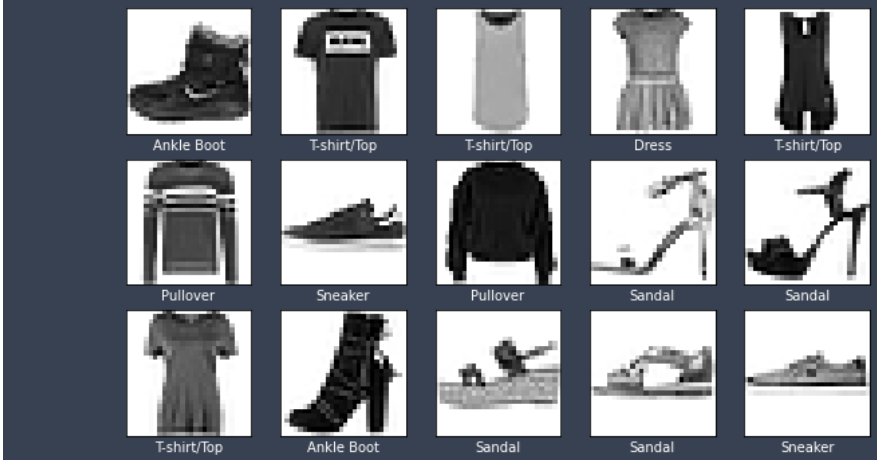
1) *Model Creation and Dataset Fetching*

```
In [89]: model = FashionMNIST_CNN(5)
         model.fetch_data()

Training Data Shape:  (60000, 28, 28) (60000,)
Testing Data Shape:  (10000, 28, 28) (10000,)
Total number of clases:  10
Target Classes:  [0 1 2 3 4 5 6 7 8 9]
```

2) Displaying of example data

```
In [90]: model.example_data(15)
```



3) Preprocessing, creation, compilation and summary of the model

```
In [91]: model.model_start()

Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
batch_normalization_33 (Batc (None, 28, 28, 32)        128
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 28, 28, 32)        9248
_________________________________________________________________
batch_normalization_34 (Batc (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
flatten_11 (Flatten)         (None, 6272)              0
_________________________________________________________________
dense_22 (Dense)             (None, 128)               802944
_________________________________________________________________
batch_normalization_35 (Batc (None, 128)               512
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_23 (Dense)             (None, 10)                1290
=================================================================
Total params: 814,570
Trainable params: 814,186
Non-trainable params: 384
_________________________________________________________________
Model Creation Done
```

4) Training

```
In [92]: model.train()

Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 10s 166us/sample - loss: 0.4397 - accuracy: 0.8459 - val_loss: 0.3343 - val_accuracy: 0.878
9
Epoch 2/5
60000/60000 [==============================] - 9s 155us/sample - loss: 0.3010 - accuracy: 0.8922 - val_loss: 0.2625 - val_accuracy: 0.9031
Epoch 3/5
60000/60000 [==============================] - 9s 154us/sample - loss: 0.2591 - accuracy: 0.9074 - val_loss: 0.2584 - val_accuracy: 0.9072
Epoch 4/5
60000/60000 [==============================] - 9s 155us/sample - loss: 0.2330 - accuracy: 0.9181 - val_loss: 0.2622 - val_accuracy: 0.9045
Epoch 5/5
60000/60000 [==============================] - 9s 154us/sample - loss: 0.2135 - accuracy: 0.9240 - val_loss: 0.2379 - val_accuracy: 0.9128
```

5) Accuracy and evaluation

```
In [93]: model.model_evaluate()

10000/10000 - 1s - loss: 0.2379 - accuracy: 0.9128
Test Accuracy:  0.9128
```

For a better analysis, the model was trained once again for 15 epochs. The results did not bear riper fruit as compared to 5 Epochs:

Training

```
In [15]: model.train()

Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 [==============================] - 14s 240us/sample - loss: 0.4464 - accuracy: 0.8426 - val_loss: 0.4721 - val_accuracy: 0.851
7
Epoch 2/15
60000/60000 [==============================] - 10s 159us/sample - loss: 0.3025 - accuracy: 0.8923 - val_loss: 1.2066 - val_accuracy: 0.613
5
Epoch 3/15
60000/60000 [==============================] - 9s 155us/sample - loss: 0.2757 - accuracy: 0.9019 - val_loss: 0.3739 - val_accuracy: 0.8733
Epoch 4/15
60000/60000 [==============================] - 9s 157us/sample - loss: 0.2335 - accuracy: 0.9164 - val_loss: 0.2810 - val_accuracy: 0.9053
Epoch 5/15
60000/60000 [==============================] - 10s 159us/sample - loss: 0.2093 - accuracy: 0.9237 - val_loss: 0.2403 - val_accuracy: 0.914
8
Epoch 6/15
60000/60000 [==============================] - 10s 161us/sample - loss: 0.1959 - accuracy: 0.9294 - val_loss: 0.2752 - val_accuracy: 0.907
2
Epoch 7/15
60000/60000 [==============================] - 10s 161us/sample - loss: 0.1907 - accuracy: 0.9317 - val_loss: 0.2575 - val_accuracy: 0.909
4
Epoch 8/15
60000/60000 [==============================] - 9s 156us/sample - loss: 0.1760 - accuracy: 0.9362 - val_loss: 0.2269 - val_accuracy: 0.9225
Epoch 9/15
60000/60000 [==============================] - 9s 155us/sample - loss: 0.1601 - accuracy: 0.9419 - val_loss: 0.2536 - val_accuracy: 0.9163
Epoch 10/15
60000/60000 [==============================] - 9s 155us/sample - loss: 0.1498 - accuracy: 0.9451 - val_loss: 0.2485 - val_accuracy: 0.9190
Epoch 11/15
60000/60000 [==============================] - 9s 156us/sample - loss: 0.1363 - accuracy: 0.9499 - val_loss: 0.2415 - val_accuracy: 0.9216
Epoch 12/15
60000/60000 [==============================] - 9s 157us/sample - loss: 0.1283 - accuracy: 0.9531 - val_loss: 0.2402 - val_accuracy: 0.9230
Epoch 13/15
60000/60000 [==============================] - 10s 159us/sample - loss: 0.1218 - accuracy: 0.9556 - val_loss: 0.9081 - val_accuracy: 0.894
3
Epoch 14/15
60000/60000 [==============================] - 10s 161us/sample - loss: 0.1318 - accuracy: 0.9521 - val_loss: 0.2448 - val_accuracy: 0.921
7
Epoch 15/15
60000/60000 [==============================] - 10s 161us/sample - loss: 0.1302 - accuracy: 0.9519 - val_loss: 0.2658 - val_accuracy: 0.919
4
```

Evaluation

```
In [16]: model.model_evaluate()

10000/10000 - 1s - loss: 0.2658 - accuracy: 0.9194
Test Accuracy:  0.9194
```

VII. CONCLUSIONS

As seen from the output screenshots, the model performs relatively well with this dataset. It managed to score a Test Accuracy of 91.28%. On training this model with 15 epochs instead of 5, the accuracy was noted to have boosted to 91.94%.

As a result, the basics of CNN and its implementation have been understood via this experimental project.

VIII. REFERENCES

[1] Xiao, H., Rasul, K., and Vollgraf, R., "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms", 2017.