Homework 3 submission

ECET 612 — Applied Machine Learning

Murat Isik

May 24, 2022

1 Submitted files

For this assignment, besides this report, the following archives were created:

1.1 SRC Folder

* *" Convolution2D"*: This **function** creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.
* *" MaxPooling2D"*: This **function** downsampling by dividing the input into rectangular pooling regions, then computing the maximum of each region.
* *" BatchNormalization"*: This **function** applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.
* *" Activation"*: This **function** ensures a slope larger than one for positive inputs.
* *" Flatten"*: This **function** is to flatten multi-dimensional tensors/arrays. A flattened one-dimensional array includes the total number of elements included in the original tensor but without the batch dimension.
* *" Dense"*: This **function** is on the input and returns the output.
* *" Dropout"*: This **function** randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting.
* *"*+ ***"project3.ipyb"***: This **script** initiates all the necessary function calls, and also generates the graphs for the fashion dataset. A total of 103 epochs have been set aside for training. To get the greatest outcomes from a model, it is also vital to keep it from overtraining and from training too deeply. First, it calculates with the information provided by the training and validation. Then it finds the plot of the graphs by using the *different python* functions. Afterward, it creates graph models. It is critical to keep the model from overtraining, since this might widen the gap between training and validation loss. Keras's EarlyStopping callback has incorporated this.

2 Discussion

We create a CNN model to categorize a fashion MNIST dataset in this project. This dataset looks to be a noisy variant of the MNIST fashion collection. The code is a Python implementation of a fashion MNIST dataset classifier. A multi-layered convolutional neural network underpins the classifier (CNN). The Keras API is utilized to implement all of the models. Google colab was used to implement the code. Below is a plot of some of the images.

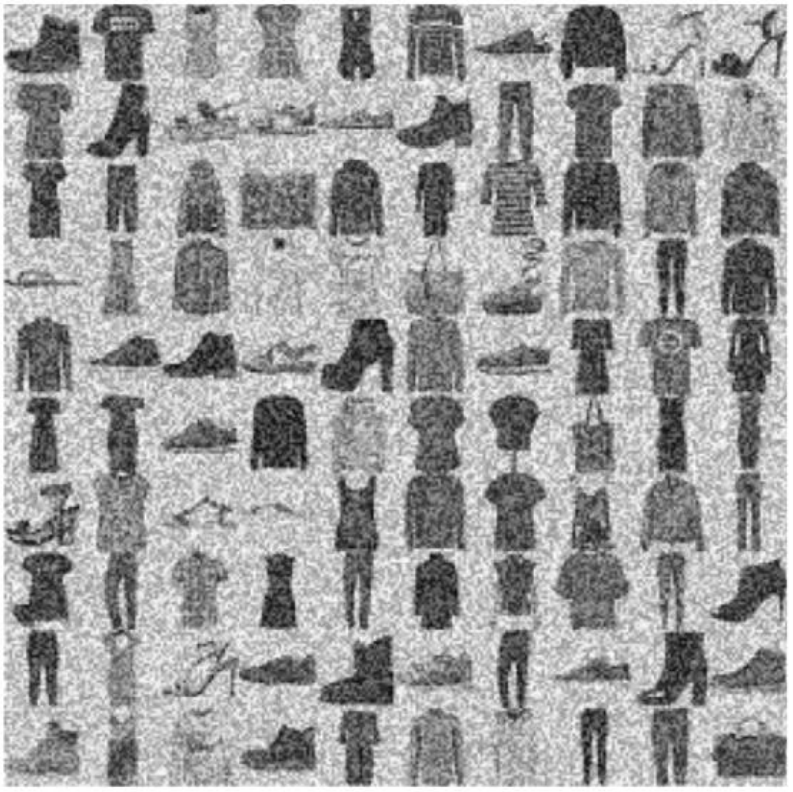

Figure 1: Project dataset.

We'd also want to highlight that we had a lot of difficulties putting up the data at first since it wasn't given in a size that worked well with the architecture described in the book. The problem was resolved by rearranging the data somewhat and clearly specifying that there is only one channel in each image. The offered training dataset is analogous to the MNIST dataset, except each instance has some noise introduced to it. As a result, each instance must be projected into a smaller dimension in order to disregard extraneous attributes.

3. Results

3.1 Dataset

Data (X_train,y_train) will split into 2 sets;

- **Train set:** The training set receives 80% of the whole data, whereas the validation set receives 20% of the total data. The actual training data contained 55,000 photos in all. As a result, 44,000 data points are utilized for training and 11,000 data points are used for validation.

– Training

– Validation

- **Test set**

```
#Split data into sets
   new_shape = list(data_features.shape)
   new_shape.insert(3, 1)
   new_shape = tuple(new_shape)
   reshape_data = data_features.reshape(new_shape)

   data_train, data_val, label_train, label_val =
train_test_split(reshape_data, data_labels, test_size=0.2)
   data_train_2, data_test, label_train_2, label_test =
train_test_split(reshape_data, data_labels, test_size=0.1)
   #Score and validate
   print("CNN:\n====================")
   print("Test Accuracy Score:
"+str(model_res.evaluate(data_train, label_train)))
   print("Validation Accuracy Score:
"+str(model_res.evaluate(data_val, label_val)))
```

Figure 2 Split dataset code

3.2 Grid Search and Hyperparameters Tuning

Grid search is simply an optimization algorithm that allows us to choose the optimum parameters for our optimization issue from a list of parameter alternatives provided by you, thereby automating the 'trial-and-error' process. Although it may be used to a wide range of optimization issues, it is well recognized for its usage in machine learning to determine the parameters at which the model performs best. The model employed includes Convolution, Pooling, and Dense Layer layers (fully connected).

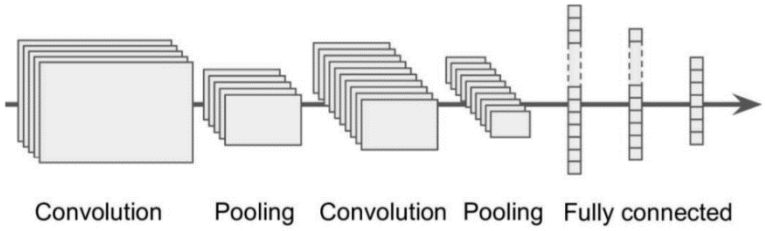

Figure 3 CNN Layers

Because the total parameters must be under 50,000, the filters and unit size will be restricted to 36 at maximum.

Functions are explained under the SRC Folder part.

3.3 CNN Architecture

This project's purpose is to develop a model with as few parameters as feasible while yet giving greater accuracy. I decided that the textbook example of a convolutional neural network (CNN) would be a decent place to start, even though it contains much too many parameters for the needs of my direction. I then contemplated utilizing the architecture because it has demonstrated to have superior accuracy with fewer parameters while still being deeper.

While the grid search reports that the best parameters are:

- activation = 'relu'
- dropout_rate = 0.2
- epochs = 103
- optimizer = 'Adadelta'

The low dropout rate might result in overfitting. As a result, the first dense layer has a smaller dropout value (0.2), whereas the next has a greater value (0.4).

The total number of parameters in the network is 51,054 and among these parameters, the number of trainable parameters is 50,606.

```
=================================================================
Total params: 51,054
Trainable params: 50,606
Non-trainable params: 448
_________________________________________________________________
```

3.4 Performance and Results

The training accuracy is 91.41% and the validation accuracy is 88.34%.

```
model.load_weights('murat.h5')
print("Training Acuracy:", model.evaluate(xtrain, ytrain, verbose=0)[1])
print("Test Accuracy", model.evaluate(xtest, ytest, verbose=0)[1])
```

```
Training Acuracy: 0.9148181676864624
Test Accuracy 0.883363664150238
```

Fig. 4 represents the curves for accuracy and loss –

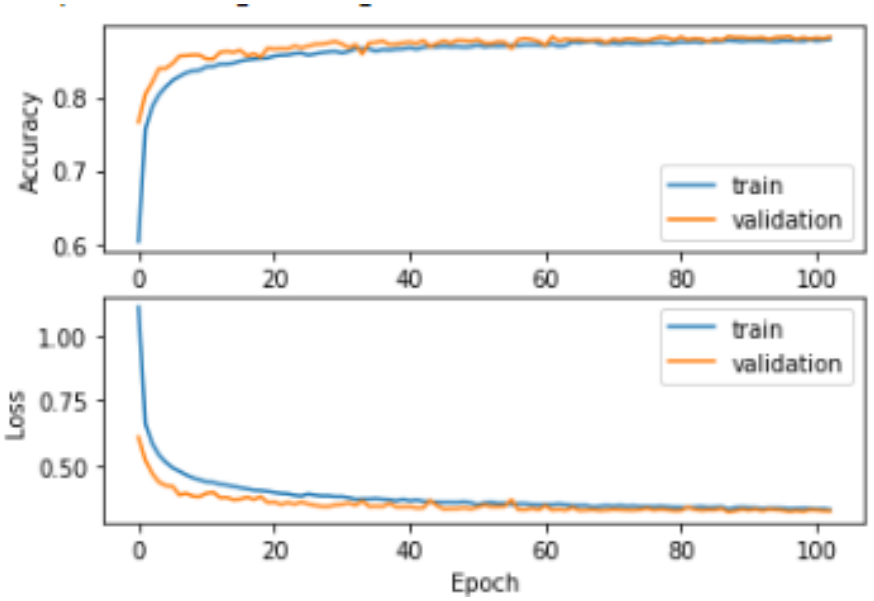

Figure 4 Learning curves

3.5 Confusion Matrix for CNNs

Given a CNN, a confusion matrix shows where the model is getting confused, i.e. which classes the model predicts correctly and which classes the model predicts incorrectly.

```
#"""**Loading and testing the model performance**"""
labels = [0,1,2,3,4,5,6,7,8,9]
model_name = 'murat.h5'
model_path = os.path.join(path, model_name)
model.load_weights(model_path)
predicts = model.predict(xtest)
ypred = np.argmax(predicts, axis = 1)
cm = confusion_matrix(ytest_bak, ypred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, disp
lay_labels=labels)
disp.plot(cmap=plt.cm.Blues)
```

Figure 5 Confusion Matrices Code

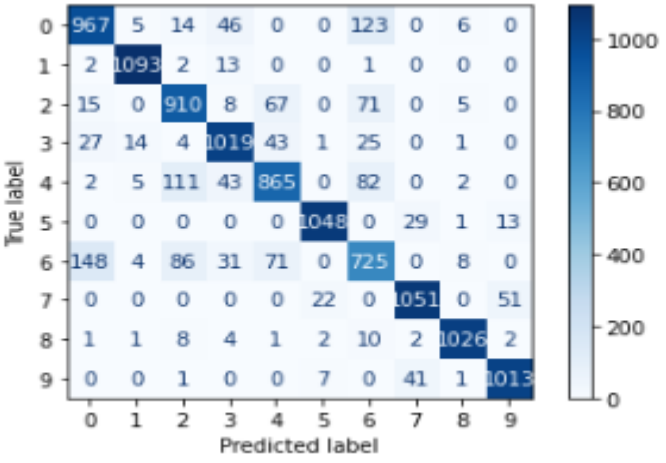

Figure 6 Confusion Matrices Results