Homework 5: Neural Networks for Fashion Classification

Audrey Shingleton

March 13, 2020

Abstract

In this project, I trained neural networks to classify articles of fashion from 10 different categories. I analyzed the effects of changing various model parameters of the neural network to achieve good classification results. More specifically, I evaluated a fully-connected neural network and a convolutional nueral network.

I. Introduction and Overview

People can often identify various articles of fashion. For this project, I wanted to see if I could train a neural network on the computer to do the same. Using a previously constructed data set with 10 categories, Fashion-MNIST, I trained neural networks to classify the images in the data set and changed various model parameters to achieve the best classification accuracy. In part one, I trained a fully-connected neural network. In part two, I trained a convolutional neural network.

II. Theoretical Background

Neural networks are used to do classification and regression. In order to do this, we need data, a model, and a loss function. The log loss is common for classification. We use a neural network to try to connect the x variable to the y variable in our data.

A neural network is composed of layers of nodes: an input layer which is the training data set, hidden layers which map the previous layer to the next, and an output layer which has the same number of nodes as the amount of classification categories.

A fully connected, or dense, neural network is where every neuron is connected to every neuron in the previous layer. The number of neurons in each layer is the width of the layer. The number of layers is the depth of the network.

A convolutional neural network (CNN) is a method applied to the hidden layers in which a small window sweeps across a layer and transfers the data into a new node through the given activation function. Pooling layers are layers generally used with a convolutional layer. The popular methods are average and maximum pooling, which take the average and maximum value of the nodes in the convolutional window respectively. LeNet-5 is a classic CNN architecture. In general, it alternates convolutional and pooling layers and ends with fully connected layers.

Activation functions are functions that will be differentiated for gradient descent to optimize the neural network training. A common function is the rectified linear unit (ReLU) where $\sigma(x) = max(0, x)$ such that all negative entries in a layer are zeroed out. Additionally, classification problems commonly uses a softmax function for the output layer.

One issue with neural networks is overfitting. This tends to occur if there are more parameters, in this case each weight and bias, than data points. A sign of overfitting is that classification is much better on the training data than the test data. To account for this, data should be split into training, test, and validation data. Validation data plays the role of test data while choosing the right model.

III. Algorithm Implementation and Development

The first step was loading in and pre-processing the data. I split up the full training data set into training and validation data such that I had 55,000 training examples and 5,000 validation examples. Then, I divided the x data by 255.0 so that the values would be floating point between 0 and 1.

The next step is developing the neural network model. I did this using the Python Keras package. For the fully connected network, I first flatten the image and then add in dense layers. For the convolutional neural network, I alternate convolutional and pooling layers, then flattening the data before adding any dense layers. I end both models with a dense layer of 10 neurons and a softmax activation function. I then adjust various model parameters, such as the layer width, network depth, number of filters, etc. until I reach a desired accuracy on the validation data. Both networks are then trained and used to predict the testing data. Confusion matrices of the results are computed to analyze the classification accuracy and errors.

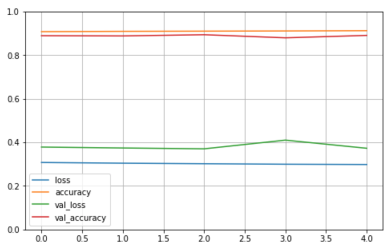
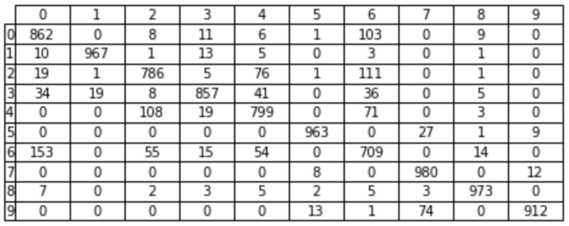


Figure 1: Training and validation losses and accuracy for my fully-connected neural network model



	0	1	2	3	4	5	6	7	8	9
0	862	0	8	11	6	1	103	0	9	0
1	10	967	1	13	5	0	3	0	1	0
2	19	1	786	5	76	1	111	0	1	0
3	34	19	8	857	41	0	36	0	5	0
4	0	0	108	19	799	0	71	0	3	0
5	0	0	0	0	0	963	0	27	1	9
6	153	0	55	15	54	0	709	0	14	0
7	0	0	0	0	0	8	0	980	0	12
8	7	0	2	3	5	2	5	3	973	0
9	0	0	0	0	0	13	1	74	0	912

Figure 2: Confusion matrix for my trained fully-connected neural network model

IV. Computational Results

Part 1: Fully Connected Neural Network

When developing a fully connected neural network model, I found that a learning rate closer to one in the Adam optimizer tended to have lower validation accuracy. There also seemed to be less accuracy when the network had too many hidden layers or layers that were very wide. Additionally, I found that after a certain amount of epochs (iterations over the training data), the validation accuracy no longer continued to increase and instead seemed to hover below and above a certain value.

The best model I found had 89.08% validation data accuracy and 88.08% test data accuracy. Since the test data accuracy was only slightly below the validation data accuracy, the model does not overfit the training data. This is also shown in Figure 1 as the training loss is only slightly below the validation loss.

The most common mistake from my model was predicting class 6 (a shirt) when the image was actually in class 0 (a t-shirt/top) (Figure 2).

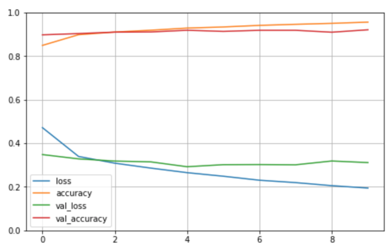
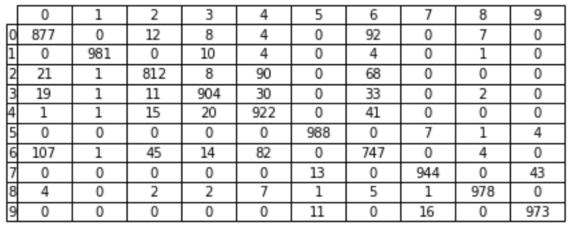


Figure 3: Training and validation losses and accuracy for my CNN



	0	1	2	3	4	5	6	7	8	9
0	877	0	12	8	4	0	92	0	7	0
1	0	981	0	10	4	0	4	0	1	0
2	21	1	812	8	90	0	68	0	0	0
3	19	1	11	904	30	0	33	0	2	0
4	1	1	15	20	922	0	41	0	0	0
5	0	0	0	0	0	988	0	7	1	4
6	107	1	45	14	82	0	747	0	4	0
7	0	0	0	0	0	13	0	944	0	43
8	4	0	2	2	7	1	5	1	978	0
9	0	0	0	0	0	11	0	16	0	973

Figure 4: Confusion matrix for my trained convolutional neural network model

Part 2: Convolutional Neural Network

For the convolutional neural network, I found that only a few convolutional layers were necessary to achieve 92.12% validation data accuracy. Similar to part one, adding more layers didn't necessarily lead to higher accuracy. I tested out using different filter and kernel sizes. Additionally, I looked at using average pooling and maximum pooling between convolutional layers. Maximum pooling seemed to give my model better accuracy.

My final model achieved 91.26% test data accuracy. This is only slightly lower than the accuracy on the validation data. However, there may be some overfitting as the training loss looks to move away from and below the validation loss (Figure 3).

The most common mistake of this model was predicting class 6 (shirt) when the image was actually in class 0 (t-shirt/top) (Figure 4). This was also the most common mistake from the fully-connected model, but I had 46 fewer of this mistake with convolutional layers.

V. Summary and Conclusions

Overall, I found that neural networks were able to achieve fairly high classification accuracy (about 90%). Convolutional neural networks seemed to perform better than the fully-connected networks. Additionally, I found that just adding more complexity to the model, such as more layers, a larger filter size, etc. did not always lead to better accuracy. Thus, I found it was important to evaluate and adjust the model using various parameters.

Appendix A. Python functions used and explanations

- keras.model.Sequential: takes in nueral network layers to construct a model
- keras.model.compile: compiles the neural network
- keras.model.fit: trains the compiled model
- keras.model.evaluate: tests the trained model on given data
- keras.model.predict classes: predicts the classes of given data
- confusion matrix: creates a confusion matrix given predicted and labeled data

Appendix B. Python codes

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix

fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()

X_train_full.shape
# pre-process data
X_train_full = X_train_full / 255.0
X_test = X_test / 255.0

# validation
X_valid = X_train_full[:5000]
y_valid = y_train_full[:5000]
```

```
X_valid.shape

# training
X_train = X_train_full[5000:]
y_train = y_train_full[5000:]
X_train.shape

# Part 1: Fully connected neural network
from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="relu",
        kernel_regularizer=tf.keras.regularizers.l2(0.0001))

model = tf.keras.models.Sequential([
    # define layers
    # flatten 28x28 image
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    # how many neurons
    # widen
    # deepen
    my_dense_layer(300),
    my_dense_layer(120),
    my_dense_layer(120),
    # need 10 neurons (classes), 10 probs
    my_dense_layer(10, activation="softmax")
])

# training options
# loss function
# learning_rate - start with guess and update
# how much to change weights at each step
model.compile(loss="sparse_categorical_crossentropy",
             optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
             metrics=["accuracy"])

# epochs - how many steps - use more
history = model.fit(X_train, y_train, epochs=5,
        validation_data=(X_valid, y_valid))

# 89.08
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()

y_pred = model.predict_classes(X_train)
```

```
conf_train = confusion_matrix(y_train, y_pred)
print(conf_train)
# rows pred, cols real, diag correct
# largest off diag
# predict 6 shirt, actually 0 t-shirt

model.evaluate(X_test, y_test)

y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
print(conf_test)

# 88.08 test accuracy
fig, ax = plt.subplots()

fig.patch.set_visible(False)
ax.axis('off')
ax.axis('tight')

df = pd.DataFrame(conf_test)
ax.table(cellText=df.values, rowLabels=np.arange(10),colLabels=np.arange(10),
        loc='center',cellLoc='center')
fig.tight_layout()
plt.savefig('conf_mat.pdf')

# Part 2: convolutional neural network
X_train = X_train[..., np.newaxis]
X_valid = X_valid[..., np.newaxis]
X_test = X_test[..., np.newaxis]

my_dense_layer = partial(tf.keras.layers.Dense, activation="relu",
        kernel_regularizer=tf.keras.regularizers.l2(0.0001))

my_conv_layer = partial(tf.keras.layers.Conv2D,
        activation="relu", padding = "valid")

model = tf.keras.models.Sequential([
    my_conv_layer(32, 4, padding = "same", input_shape=[28,28,1]),
    tf.keras.layers.MaxPooling2D(2),
    my_conv_layer(32,4),
    tf.keras.layers.Flatten(),
    my_dense_layer(300),
    my_dense_layer(100),
    # need 10 neurons (classes), 10 probs
    my_dense_layer(10, activation="softmax")
])
```

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=["accuracy"])

history = model.fit(X_train, y_train, epochs=10,
         validation_data=(X_valid, y_valid))
# 92.12%
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()

y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train, y_pred)
print(conf_train)

model.evaluate(X_test, y_test)
# 91.26
y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
print(conf_test)
# print conf matrix
fig, ax = plt.subplots()

fig.patch.set_visible(False)
ax.axis('off')
ax.axis('tight')

df = pd.DataFrame(conf_test)
ax.table(cellText=df.values, rowLabels=np.arange(10),
                 colLabels=np.arange(10),
         loc='center',cellLoc='center')
fig.tight_layout()
plt.savefig('conf_mat2.pdf')
```