STAT 419 Final Project: Neural Networks

Daniel DeFoe, Markelle Kelly, Sadie Rhen

ABSTRACT.

Currently the world of machine learning gives statisticians and data scientists a plethora of tools and models to choose from when it comes to analyzing data and making predictions. With all of these choices at the disposal of professionals, neural networks have emerged as a powerful tool for data classification and outcome prediction. Modeling with neural networks has significantly advanced our power to make sense of large datasets and draw meaningful, increasingly accurate conclusions. Neural network architecture is behind the advancement of modern technological breakthroughs such as image and audio recognition.

The following report aims to describe and portray how a neural network can classify image data. This will be done using Python's Keras package with tensorflow. The general structure of the network will be discussed, including its layers, in terms of our data set. The produced model will then be broken down through evaluation of our findings and reflection upon our results.

---

1. INTRODUCTION

The data used to investigate how to build and develop a neural network are images of articles of clothing from the popular "Fashion-MNIST" dataset, obtained from Zalando Research on GitHub. Each observation is a 28x28 pixel black and white image classified as one of 10 possible types of clothing articles. Each image is classified as a t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, or ankle boot. To describe each image, the neural network considers its pixels, counted horizontally row by row. This means that pixel 29 is in the 2nd row, in the 0th column. There are 784 total columns representing the pixels by their darkness, measured on a scale of 1 to 255. There are a total of 70,000 observations (images) in the dataset.

Why Neural Networks?

Convolutional neural networks are one of the most effective strategies for image and pattern recognition and classification. Their convolutional layers, which will be discussed in detail in the next section, allow them to reliably identify patterns and features in images. Additionally, the way neural networks consider different variables and their relationships make them an ideal way to deal with sets of variables that are related in specific ways, like the sets of pixels comprising images.

2. NEURAL NETWORK CREATION

Neural Network Layers

The Keras package for Python used to create the network employs a sequential model, so it allows the model to be defined and built layer by layer. The input layer to the network consists of the data's features as described in the introduction. Collectively the data set's variables serve as predictors, with each pixel assuming the role of a single input node, for a total of 784 nodes.

The hidden layers of the network include different types of layers with various functions that essentially comprise the entire model. It is important to note that the creation of the series of hidden layers is largely synonymous with the creation of the model. The number and nature of these layers plays a significant role in the model's ability to correctly classify data. The

model used for the Fashion-MNIST data was a two dimensional convolutional neural network, so the first layer was a convolutional layer. This layer uses a filter referred to as a kernel to go through the input image, one section at a time, grouping the pixels into receptive fields. For each position on the image, weights assigned to the previously determined groups of pixels are multiplied by each pixel to essentially determine importance by area. Convolutional layer filters act as feature identifiers - the element wise multiplications will increase dramatically if there is a pattern in parts of the image that match the filter. These element wise multiplications end up being represented in a single array of numbers called a feature map. The more kernels there are, the better the network can capture differences between groups of pixels. After this first layer, we added another convolutional layer, then two dense layers. The dense layers conduct operations on the preceding layer, using the calculated weights to connect each node to each neuron in the next layer, further down the network, with updated values based on the weights.

It should be noted that in the defining of the hidden layers, a max-pooling aggregation was applied as well as two dropout functions. Max-pooling helps prevent overfitting by aggregating the maximums of each feature group after the convolutional layers. The dropout functions also help prevent overfitting by setting a portion (for our model, 25%, then 50%) of the activations of a given layer to zero during each step of model training. For further detail on the exact model we used, see our Keras code in the appendix.

Finally, the output layer, containing 10 nodes, contains the neural network's prediction of what the first column of the data, the image classification, should be. From the computing done in the series of hidden layers, the model makes a prediction for each observation of what kind of clothing article the image is. To represent this, one of the 10 nodes, each of which correspond to one of our 10 original categories, will have a value of 1, and the rest will have a value of 0 (the node with 1 representing the predicted category). We saved these predictions for further analysis of which categories our network was most frequently getting wrong (discussed in the next section).

Model Training

We trained our neural network on 60,000 of the 70,000 total images, leaving 10,000 for testing the model.

In order for the network to classify the data properly, the weights and biases of each layer need to be optimized. In order to do this, neural networks need a cost function and optimization algorithm. When a single piece of training data is run through the unoptimized model, the value of each output node is compared to what was expected, generally one for the category to which the piece of training data belongs and zero for all other categories. Popular cost functions include mean square error, cross entropy, and absolute error; we used a categorical cross entropy loss function, which determines the difference between the true and observed probability distributions; minimizing cross entropy maximizes the log likelihood. Since this cost function will be small when the output nodes are close to their expected value, minimizing this cost function by adjusting the weights and bias of each layer of the network will reduce the overall error in the network.

To perform this optimization, we used Adadelta, a gradient descent algorithm, which applies the negative gradient of the cost function to all of the weights and biases in a layer. This reveals

how much each piece should be adjusted in order to decrease the cost function by the greatest amount. Then, this process is repeated for each layer, optimizing the entire network.

Since the entire training data set of 60,000 points (one epoch) cannot be run through the network all at once, a cost function and corresponding gradient was computed for batches of 128 points at a time, and the network's weights and biases were optimized based on that. Batches were run until all 60,000 points had been used, and the first epoch was completed. Since only running through the training data once tends to underfit the data, multiple epochs need to be run in order to increase the accuracy of the network. However, running too many epochs can overfit the network to the specific training data, making it less accurate when classifying new data that is not from the training set. By 12 epochs, the increases in accuracy were smaller than .005 per step, and the network had an overall accuracy of .9406 for the training data, so training was considered to be complete. The specific performance of each epoch is included in the appendix.

Model Evaluation

We then evaluated our model based on our test set of 10,000 additional images. Within the training, we brought our loss down from 0.5684 to 0.1660, and increased our accuracy from .7978 after the first epoch to .9406. When we evaluated our neural network with the test data, it achieved a loss of .214 and an accuracy of .9267.

3. FINDINGS

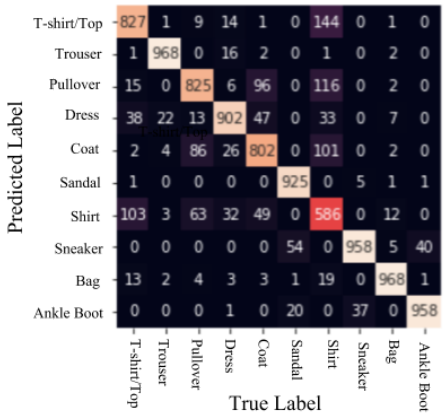

Figure 1: Confusion matrix measuring the accuracy of the network using one epoch

Figure 1 shows the general accuracy of the predictions provided by the network when run using a single epoch. This makes it easy to visualize which articles are incorrectly classified, and why that may be. The confusion matrix shows that the articles most commonly classified incorrectly are shirts, with only 58.6% of them correctly identified. However, this category makes sense as a problematic one; these shirts were most frequently mistaken for t-shirts/tops, pullovers, and coats. These are very similar articles of clothing, with very similar features (comparable shape, visual nature of having sleeves), and this makes it understandable that the network would have trouble consistently classifying them - even humans could have trouble distinguishing shirts from tops. (In this data set, shirts are nicer blouses, while tops are more casual). The articles with the highest proportion correctly classified were trousers and bags, each at 96.8% correct, which makes sense because each of these items has its own unique features. Trousers are the only pant item being classified and bags have a very distinct shape compared to the other items, which are all physically worn on the body. These distinctive visual features allow

trousers and bags to be effectively classified by this somewhat limited neural network.

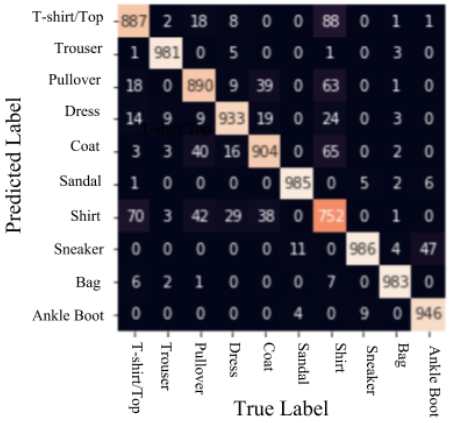

Figure 2: Confusion matrix measuring the accuracy of the network using 12 epochs

Figure 2 is the same confusion matrix as Figure 1, but recreated after running all 12 epochs. The increased accuracy for every category shows the benefit of running multiple epochs. Going through the same training data multiple times significantly improved our accuracy on the test data. Shirts are still the most frequently misclassified, but now over 75% of them are correctly identified, a significant improvement from 58.6% after a single epoch. Our neural network is most accurate for trousers, sandals, sneakers, and bags.

Just by looking at the clear diagonal lines in both confusion matrices, one can conclude that our neural network is closely matching reality, and is a fairly reliable indicator of an image's true clothing classification.

While near 93% accuracy is impressive for a relatively simple model, we could always work to improve our neural network by expanding the hidden layer. According to the dataset's GitHub page, other convolutional neural networks have achieved up to 95% accuracy. To improve our model, we would consider adding another convolutional or dense layer, or pre-processing the data. This page also lists benchmarks for other types of classifiers, including SVC, random forests, and K-neighbors, which all have accuracies in the mid 80 percents, so neural networks appear to be an optimal way to classify this dataset.

It is worth noting that in the modern age, there is often a heightened importance placed on the accuracy of neural network models when it comes to certain systems, especially in the field of image recognition. Artificial intelligence utilizing neural networks is becoming increasingly responsible for the maintenance of systems pertaining to safety and security. To cite a few examples, self driving cars interpreting the highly variable surrounding world require very high accuracy to be able to safely navigate the road, and facial recognition to unlock modern smartphones is often the only barrier between a potential predator and someone's private information. It is in systems such as this that networks must be built with much more extensive and powerful hidden layers to increase overall accuracy. It is imperative for the prediction of these networks to be reliable in order to protect users and coexisting entities.

4. REFERENCES

3Blue1Brown. "But What *Is* a Neural Network? | Chapter 1, Deep Learning." *YouTube*, YouTube, 5 Oct. 2017, www.youtube.com/watch?v=aircAruvnKk.

3Blue1Brown. "Gradient Descent, How Neural Networks Learn | Chapter 2, Deep Learning." *YouTube*, YouTube, 16 Oct. 2017, www.youtube.com/watch?v=IHZwWFHWa-w.

Clabaugh, Caroline, et al. "Neural Networks - History." *Neural Networks*, 2000, cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html.

Izenman, Alan Julian. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*.

Springer, 2013.
Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Han Xiao, Kashif Rasul, Roland Vollgraf. arXiv:1708.07747. github.com/zalandoresearch/fashion-mnist
Sharma, Sagar. “Epoch vs Batch Size vs Iterations – Towards Data Science.” *Towards Data Science*, 22 Sept. 2017, towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

5. APPENDIX

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
2018-06-01 15:07:20.912367: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this Tensor
 was not compiled to use: AVX2 FMA
60000/60000 [==============================] - 273s 5ms/step - loss: 0.5684 - acc: 0.7978 - val_loss: 0.3581 - val_acc: 0.8714
Epoch 2/12
60000/60000 [==============================] - 331s 6ms/step - loss: 0.3587 - acc: 0.8721 - val_loss: 0.3213 - val_acc: 0.8806
Epoch 3/12
60000/60000 [==============================] - 300s 5ms/step - loss: 0.3109 - acc: 0.8902 - val_loss: 0.2776 - val_acc: 0.8987
Epoch 4/12
60000/60000 [==============================] - 312s 5ms/step - loss: 0.2770 - acc: 0.9003 - val_loss: 0.2601 - val_acc: 0.9049
Epoch 5/12
60000/60000 [==============================] - 329s 5ms/step - loss: 0.2550 - acc: 0.9072 - val_loss: 0.2483 - val_acc: 0.9095
Epoch 6/12
60000/60000 [==============================] - 316s 5ms/step - loss: 0.2340 - acc: 0.9159 - val_loss: 0.2408 - val_acc: 0.9140
Epoch 7/12
60000/60000 [==============================] - 299s 5ms/step - loss: 0.2182 - acc: 0.9221 - val_loss: 0.2350 - val_acc: 0.9160
Epoch 8/12
60000/60000 [==============================] - 279s 5ms/step - loss: 0.2078 - acc: 0.9247 - val_loss: 0.2210 - val_acc: 0.9196
Epoch 9/12
60000/60000 [==============================] - 301s 5ms/step - loss: 0.1930 - acc: 0.9303 - val_loss: 0.2216 - val_acc: 0.9188
Epoch 10/12
60000/60000 [==============================] - 320s 5ms/step - loss: 0.1833 - acc: 0.9342 - val_loss: 0.2177 - val_acc: 0.9236
Epoch 11/12
60000/60000 [==============================] - 302s 5ms/step - loss: 0.1739 - acc: 0.9371 - val_loss: 0.2210 - val_acc: 0.9246
Epoch 12/12
60000/60000 [==============================] - 325s 5ms/step - loss: 0.1660 - acc: 0.9406 - val_loss: 0.2140 - val_acc: 0.9267
Test loss: 0.21397165847420693
Test accuracy: 0.9267
```

Figure 3. Output for our neural network

```
33  model = Sequential()
34  model.add(Conv2D(32, kernel_size=(3, 3),
35                   activation='relu',
36                   input_shape=input_shape))
37  model.add(Conv2D(64, (3, 3), activation='relu'))
38  model.add(MaxPooling2D(pool_size=(2, 2)))
39  model.add(Dropout(0.25))
40  model.add(Flatten())
41  model.add(Dense(128, activation='relu'))
42  model.add(Dropout(0.5))
43  model.add(Dense(num_classes, activation='softmax'))
44
45  model.compile(loss=keras.losses.categorical_crossentropy,
46                optimizer=keras.optimizers.Adadelta(),
47                metrics=['accuracy'])
48
49  model.fit(x_train, y_train,
50            batch_size=128,
51            epochs=12,
52            verbose=1,
53            validation_data=(x_test, y_test))
54  score = model.evaluate(x_test, y_test, verbose=0)
55  y_new = model.predict_classes(x_test)
56  y_actual = []
57  for array in y_test:
58      mylist = list(array)
59      y_actual.append(mylist.index(1.0))
60  y_actual = np.array(y_actual)
61  mydata = pd.DataFrame({'actual':y_actual,'predicted':y_new})
62  mydata.to_csv("mydata.csv")
63  print('Test loss:', score[0])
64  print('Test accuracy:', score[1])
```

Figure 4. Our model code in Keras