Giving Signatures a Purpose

New Mexico

SuperComputing Challenge

Final Report

April 8 2020

Team 15

Desert Academy

Team Members:

Jonatan Kaare-Rasmussen

Ethan Garcia

Teachers:

Steff Eiter

Project Mentor:

Chris Zappe

Abstract:

In this project our goal was to make a signature forgery classification system. To do this we decided to use a Neural Network to classify signatures. To use a Neural Network you need to train it with a label dataset which, in our case, is a set of hundreds of labeled signatures. We collected these by having people sign a signature 90 times. We then scanned the pages in and separated them into separate image files using photoshop. We trained the network on the scanned signatures and the network achieved 96 percent accuracy when we tested it on data it has never seen before.

Problem:

Signatures have been a way of validation for centuries, but in today's world they are not checked for validation even though we have the tools to do it accurately. When you buy something with your credit card you need to write a signature. But that data does not go anywhere, it is just a formality. So, our goal is to fix this issue and write a piece of software that checks if the given signature is the user's signature or if it is forged.

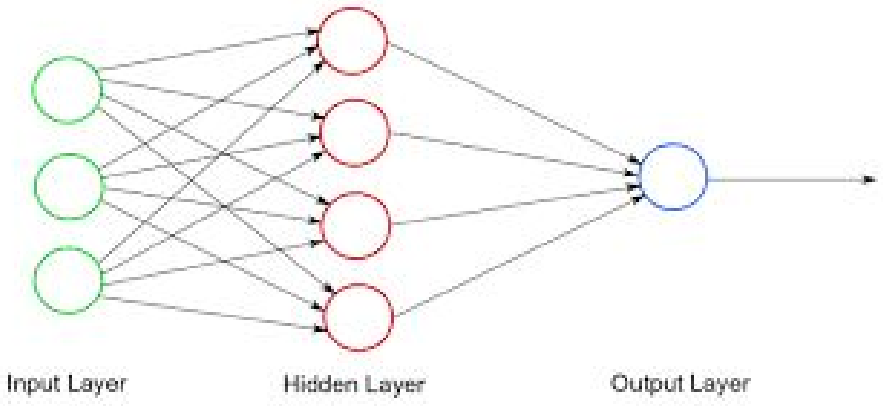
Method:

To solve this problem we decided to use Artificial Intelligence. We created a Neural Network that when given an image of a signature it can verify its validity . To create our network we used a free and open-source google python library called TensorFlow. Using TensorFlow as well as a few other libraries to assist in formatting the data and matrix calculations we were able to make a network that can classify a test data set of 25 images with 96 percent accuracy.

What is a Neural Network:

A Neural Network is really just a big function. You input numbers and it outputs numbers. In order to make the network output the correct numbers you need to adjust the values of the functions parameters. These parameters are called weights. The structure of a network is often described by nodes that are arranged in layers and each node in the first layer is connected to every node in the next layer by a weight. All a weight is is a number that alongside the node it is connected to on the first layer helps determine the value of a node in the next layer Every node in the previous layer has a role in determining the value of any given node in the next layer. In figure one you can a visual representation of an extremely simple

Neural Network. In this example the black arrows represent weights. This figure is a great visual representation of how a Neural Network is structured



(Fig. 1) A visual representation of a Neural Network.

TensorFlow and Keras:

TensorFlow is a Google library that was written in python to simplify the process of making a Neural Network. TensorFlow was made to be flexible and fast and is used by both people new to AI, and professionals. TensorFlow allowed us to use an open source library called Keras. TensorFlow has a version of Keras built in which simplifies the whole process of building a Neural Network. Keras is a low-level Neural Network library that focuses on experimentation and fast prototyping.

Our Process:

We started out this project by making a proof of concept to check if we could make working Neural Networks. We used a MNIST fashion dataset from the internet to train our network and achieved about 90 percent accuracy. The dataset consisted of images of clothes that the network had to classify. This is different from signatures, but it worked well as a proof-of-concept.

Our dataset has 5 different types of signatures in the training and testing datasets. There are 5 test images and 82 training images for each type of signature. Collecting this data was a big roadblock for this project. Our initial approach was to do everything digitally so we made a simple processing drawing program that could be used to speed up the data collection process. The problem we had with this approach was that most people do not know how to use a

drawing tablet. For this reason we decided to switch tracks. Our new idea was to have people use pen and paper and scan their signatures in. To do this we created a chart that people filled out and we scanned in. After getting the scans we separated the image files using photoshop and prepared them for our network. This worked a lot better but our first dataset had too many types of signatures and not enough of each type for the network to effectively train on it. Training the network on this dataset resulted in a network that could classify signatures with 22 percent accuracy. This was not the result we were looking for so we decided to remake the dataset. We decided to decrease how many types of signatures there were to 5. We increased the amount of signatures per person to 90 signatures. We scanned and processed the data and we ended up with a training dataset of 422 images and a test dataset of 25 images. Also to avoid asking for real signatures. Instead we asked for words, like man123 or spoon. We trained the data on this dataset and the network got a success rate of 80 percent, a huge improvement but not quite good enough. So to improve the network further we experimented with how many hidden layers the network should have and how big they should be. The best combination was a network with 7 layers in total. The input, the output and 5 hidden layers. Each of the hidden layers decreases with size which in a sense funneled the data down from the large amount of input nodes to the very few amount of output nodes.

The Validity:

After seeing that our network classified the signature with a relatively high accuracy we needed to see how it would work in real life conditions. We tried feeding the network forged signatures to see how it would handle them and it classified them correctly with a low accuracy. This is not ideal as a forged signature would ideally be incorrectly classified instead of classified correctly. One important thing to note when considering the networks used in real life is that the network was trained to classify with confidence. This means that if the network were to be fed a signature that it was not trained on the output would say that the signature is in the dataset and it is confident that it is a certain signature. This is important to note because in a real life situation like using a credit card we know what the signature should be so if a signature is written and the network classifies it as someone else's we know that the signature was forged. There is a chance that the network will classify the signature correctly and therefore not flagging it but this chance would be lowered the more people's signatures are used to train the network.. We created a small dataset of forged signatures which the network classified them incorrectly

84 percent of the time. This is very close to 4/5 which shows the correlation between the amount of signatures in the data set as the amount people who wrote their signatures was five and the amount of possible signatures that the network could classify as forged signatures is 4. This means that if we have a network trained on 10,000 peoples signatures the amount of forged signatures that would be classified correctly would be around 0.0001 percent.

Results:

The network we ended up creating worked well considering the dataset that we had to train it on. With thousands of peoples signatures accounted for in the network, the network would work even better than it did now. If the network were to be implemented into a real life situation it would definitely lower the amount of signature forgery when in place. Even in its current state in which it was trained on a far too small dataset according to our validity test it would stop over 80 percent of forged signatures. So in short our model is successful and may actually have a place in real life situations.

Conclusion:

Our goal was to create a program that could flag forged signatures. Our network does this very well and has potential to do it even better with a more diverse dataset. There are a lot of things that we can do to improve our model. One big thing that could be worthwhile to look into is network structure. There are many types of neural network structures that are better at image classification but they get significantly more complicated and require more data. But even without a different structure we made a network that has a real use and could be used to stop credit card fraud.

The Code:

This is the section of code that imports the third party libraries that we used such as TensorFlow and Numpy.

```
#TensorFlow import Google AI library
import tensorflow as tf
from tensorflow import keras

#Numpy import for matrix calculation
import numpy as np

#matplot import for displaying result
import matplotlib.pyplot as plt
import matplotlib.image as mping

#Other imports for loading datasets
from PIL import Image
import os
```

This the section of code that imports the data set we created. It uses PIL(Python Imaging Library) and the os library to import the image files on the harddrive to an array.

```
#preperation for dataset importation
train_list=os.listdir("C:/Users/jtkaa.LAPTOP-MV6CH47B/Desktop/SC/Post_Prosess/train")
test_list=os.listdir("C:/Users/jtkaa.LAPTOP-MV6CH47B/Desktop/SC/Post_Prosess/test")
#importation of the dataset
train_sig=np.array([np.array(Image.open("C:/Users/jtkaa.LAPTOP-MV6CH47B/Desktop/SC/Post_Prosess/train/"+fname)) for fname in train_list])
test_sig=np.array([np.array(Image.open("C:/Users/jtkaa.LAPTOP-MV6CH47B/Desktop/SC/Post_Prosess/test/"+fname)) for fname in test_list])
```

Here is where we create the labels for our dataset. Also this changes the range of pixel values for in between 0 and 255 to in between 0 and 1 so the network can understand the data.

```
#preperation for the creation of our label matrices
train_labels=np.zeros(410)
test_labels=np.zeros(25)

#Label matrices creation
for i in range(0,5):
    for x in range(1,83):
        train_labels[((i*82)+x)-1]=i
for i in range(0,5):
    for x in range(1,6):
        test_labels[((i*5)+x)-1]=i

#data formating
train_sig=train_sig/255.0
test_sig=test_sig/255.0
```

This is the code that actually creates the model. The Network has 5 deep layers and The Input and output layers. The network is made so that all of the outputs add up to one so that the output says not only what type of signature it is but also would certain it is that it is correct

```
#the creation of the acutal model
model = keras.Sequential([
    #Flatten 2d pixel array in 1d for input
    keras.layers.Flatten(input_shape=(60, 250)),

    #RELU activation in deep layers becuase sigmoid produces dead nodes
    keras.layers.Dense(1200, activation='relu'),
    keras.layers.Dense(800, activation='relu'),
    keras.layers.Dense(400, activation='relu'),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(20, activation='relu'),

    #softmax for percetage output (All outputs add up to 1)
    keras.layers.Dense(5, activation='softmax')
])
```

This is the code that feeds the network the training data. The networks train 5 times on each image in the dataset. We tried to do more than that but the network became overtrained and got great scores on the training set but terrible ones on the test set. This is because the network essentially memorized the training set.

```
#prepare for training (train on accuracy)
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

#Train on the train dataset (x5)
model.fit(train_sig, train_labels, epochs=5)
```

This is the code that tells us how well the network works by testing it on the test dataset.

```
#test for loss and accuracy on the test datasets
test_loss, test_acc = model.evaluate(test_sig,  test_labels, verbose=2)

#create an array for the models output on the test dataset
predictions = model.predict(test_sig)

#display the test dataset along with the networks guess and the test labels
sigs = ['man123', 'Diels-Alder', 'Spoon', 'milk', 'no!']
for i in range(0,25):
    plt.grid(False)
    plt.imshow(test_sig[i], cmap=plt.cm.binary)
    plt.title("prediction: "+ sigs[np.argmax(predictions[i])])
    plt.xlabel("real: "+ sigs[int(test_labels[i])])
    plt.show()
```

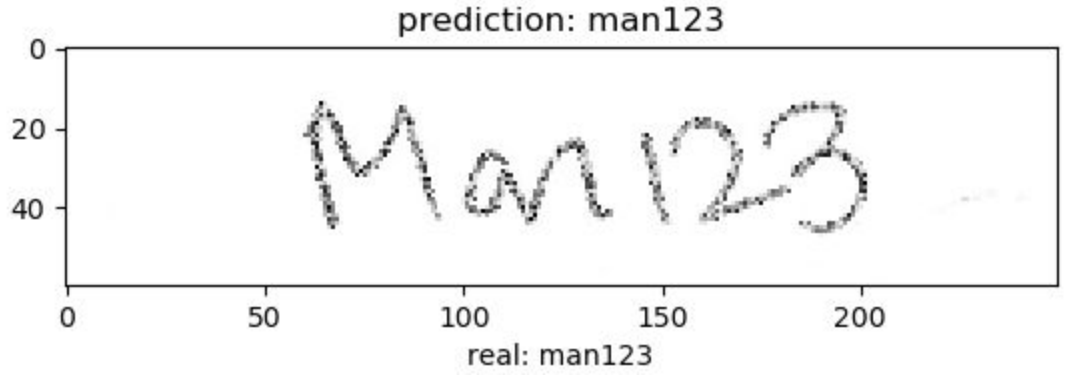
This is how it looks when the network is training. You can see that the accuracy goes up after the first time training on the data as well as the loss going down.

```
Epoch 1/5
410/410 [==============================] - 2s 4ms/sample - loss: 0.9481 - accuracy: 0.5780
Epoch 2/5
 32/410 [=>............................] - ETA: 0s - loss: 0.1920 - accuracy: 0.9375
```

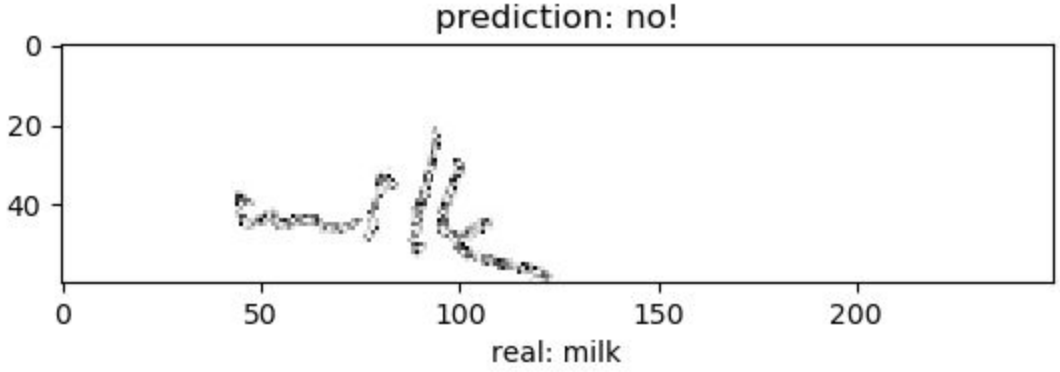
This is how it looks after the network has completed Training. You can see that with each epoch or iteration through the data the accuracy goes up and the loss goes down. You can also see that the network is slightly overtrained because the accuracy goes down in the test data set.

```
Train on 410 samples
Epoch 1/5
410/410 [==============================] - 2s 4ms/sample - loss: 0.9481 - accuracy: 0.5780
Epoch 2/5
410/410 [==============================] - 1s 2ms/sample - loss: 0.2049 - accuracy: 0.9488
Epoch 3/5
410/410 [==============================] - 1s 2ms/sample - loss: 0.0359 - accuracy: 0.9927
Epoch 4/5
410/410 [==============================] - 1s 2ms/sample - loss: 0.0975 - accuracy: 0.9780
Epoch 5/5
410/410 [==============================] - 1s 2ms/sample - loss: 0.0436 - accuracy: 0.9902
25/1 - 0s - loss: 0.1239 - accuracy: 0.9600
```

This is the individual image view of the network's success on the test dataset.



This is an example of the network making the mistake.



Works Cited:

Nielsen, and Michael A. “Neural Networks and Deep Learning.” *Neural Networks and Deep Learning*, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/.

Sanderson, Grant, director. *Neural Networks*. *YouTube*, YouTube, 1 Aug. 2018, www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi.

"TensorFlow." *TensorFlow*, www.tensorflow.org/.

Yiu, Tony. "Understanding Neural Networks." *Medium*, Towards Data Science, 4 Aug. 2019, towardsdatascience.com/understanding-neural-networks-19020b758230.