Handwritten Digit Classification Using Tensorflow

By: Benjamin Alterman, Adam Kardorff, Jorie Noll, Uluc Ozdenvar

Project Goal and Motivation

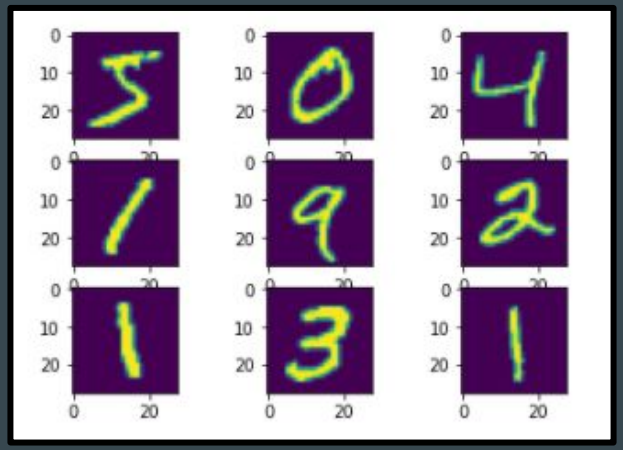
- Our project motivation was to gain a deeper understanding of neural networks and their real world applications.
- After researching different applications of neural networks, we decided to explore how neural networks are utilized to recognize visual patterns from a variety of images using convolutional neural networks (CNN) and Tensorflow.
    - CNN visual recognition can be used for facial recognition software, document analysis, advertising, and more.

Project Goal and Motivation

- Our project scope honed in on training a convolutional neural network that can recognize handwritten digits.
- Handwritten digit analysis can be utilized to create software that converts handwritten documents into digital formats.
- Document conversion software can be used by students and professionals to transform handwritten notes into formatted text documents that archive important information digitally.

About the Data

- We the MNIST database of handwritten digits for our data set.
- The handwritten digit database has a training set of 60,000 examples, and a test set of 10,000 examples.
- Each example is a handwritten digit that has been centered in a fixed-size image.
- This data is converted to a 28x28 matrix with values ranging from 0 - 255 based on density.



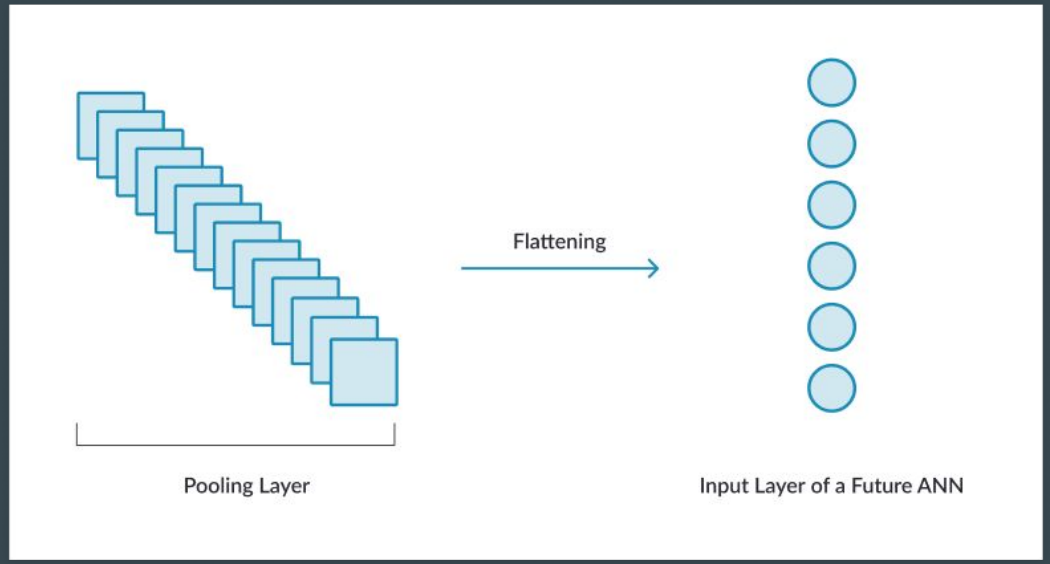
A sample set of the data

Data Analysis Algorithm

- Our project goal was to train a convolutional neural network that can identify handwritten digits from a variety of images.
- We aimed to do this by deploying a sequential model.
- By definition, a **convolutional neural network** is a class of deep neural networks used most commonly to analyze visual imagery.
    - Convolutional neural networks utilize multilayer perceptrons, which are generally fully connected networks in which each neuron in one layer is connected to all neurons in the next layer.
    - A major advantage of convolutional neural networks in comparison to other image classification algorithms is the little amount of preprocessing required
- A **sequential model** is a deep learning approach which allows one to build a model layer by layer.
    - In a sequential model, each layer has weights that correspond to the layer that follows it.
    - These layers interact with each other at every level, so the current output is dependent on the previous input layer.
    - A sequential model is appropriate for a plain stack of layers, where each layer has exactly one input tensor and one output tensor.

Flatten Layer

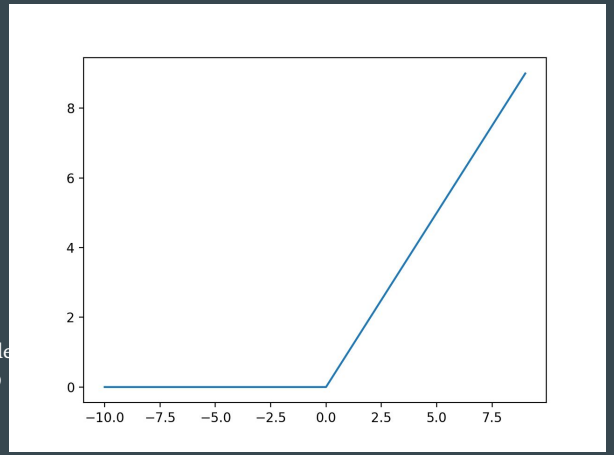
- Start by flattening input 2-dimensional shapes into a 1-dimensional array.



https://www.google.com/url?sa=i&url=https%3A%2F%2Fmissinglink.ai%2Fguides%2Fkeras%2Fusing-keras-flatten-operation-cnn-models-code-examples%2F&psig=AOvVaw2MyQxVa7wYs7EvoLFkM7l3&ust=1606871961965000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCPC_5NbOq-0CFQAAAAAdAAAAABAD

Dense Layer - ReLu Activation

- The dense layer ensures that each neuron in previous layer is connected to a neuron in the next layer.
- The rectified linear activation function, or ReLU for short, is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.
- The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.
- 128 neurons were selected for the hidden layer.



https://www.google.com/url?sa=i&url=https%3A%2F%2Fmachinelearningmastery.com%2Frectified-linear-activation-function-for-de
mca5RRZIG6fE&ust=1606872105420000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCNjPvJrPq-0CFQAAAAAdAAAAABAD

Output Layer - Softmax Function

- The softmax function is a multiclass logistic regression function that turns a vector of K real values into a vector of K real values that sum to 1.
- The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities.
- 10 neurons in the output layer for 10 classes from 0-9.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

https://www.google.com/url?sa=i&url=https%3A%2F%2Fdeepai.org%2Fmachine-learning-glossary-and-terms%2Fsoftmax-layer&psig=AOvVaw1K5v1KIQgfCR3NsZpv00F0&ust=1606872231249000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCMiW-NfPq-0CFQAAAAAdAAAAABAD

Tensorflow Keras model

```
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28,28)),
                                    tf.keras.layers.Dense(128, activation='relu'),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

Fitting the model

- Our next step was to fit the model to the training data.
- In doing this, we had a 99% accuracy with the training data by using 7 epochs.

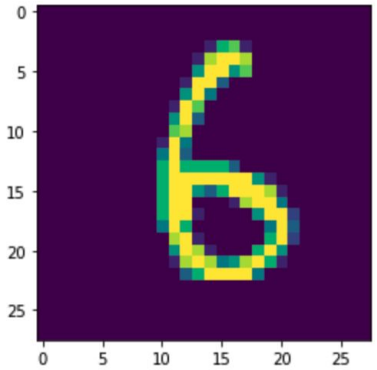
```
# fit the model to the training data
"""An epoch refers to one iteration through the training data. Accuracy reaches 99% in the 7th epoch."""
model.fit(training_images, training_labels, epochs=7)

Epoch 1/7
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2559 - accuracy: 0.9272
Epoch 2/7
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1129 - accuracy: 0.9669
Epoch 3/7
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0780 - accuracy: 0.9760
Epoch 4/7
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0584 - accuracy: 0.9824
Epoch 5/7
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0459 - accuracy: 0.9856
Epoch 6/7
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0348 - accuracy: 0.9893
Epoch 7/7
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0285 - accuracy: 0.9913
<tensorflow.python.keras.callbacks.History at 0x7f95ec9f30f0>
```

The code to fit the model to the training data

Sample Test Result

```
[10]  1 # select image from testing data from 0-9999
      2 plt.imshow(test_images[3000])
```

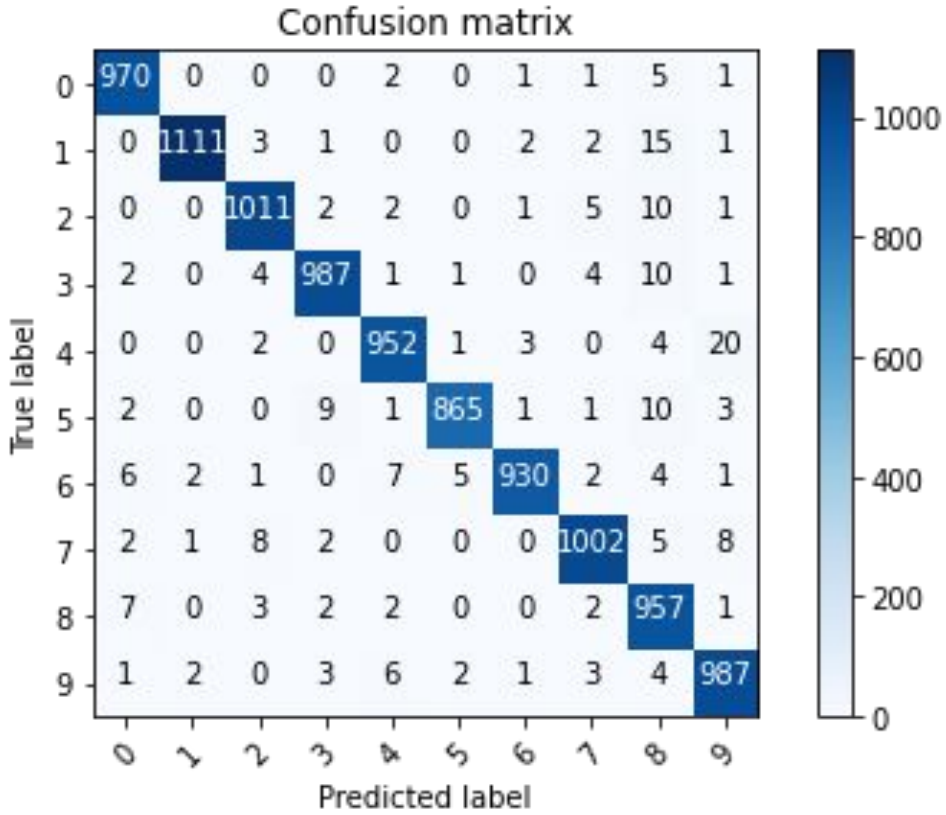
```
<matplotlib.image.AxesImage at 0x7f4298b8d4e0>
```




```
[11]  1 # run model to predict the number of the image
      2 prediction=model.predict(test_images)
      3 """np.argmax finds the neuron with the highest probability."""
      4 print("The number is:", np.argmax(prediction[3000]))
```

```
The number is: 6
```

Confusion Matrix



Experimental Results and Conclusions

- We got a 99% accuracy on the training data, and 97% accuracy on the testing data.
- We gained a greater understanding of how deep learning can be achieved using large data sets.
- We explored how neural networks have real world applications, such as imagery recognition using a convolutional neural network.
- We learned how to use Tensorflow and the Keras API, which have many industry applications.
- https://colab.research.google.com/drive/1hWWWSn5tjGgkVK9fe7fK2lc8i7lV-Wzt#scrollTo=cfPfFjhAKe4K