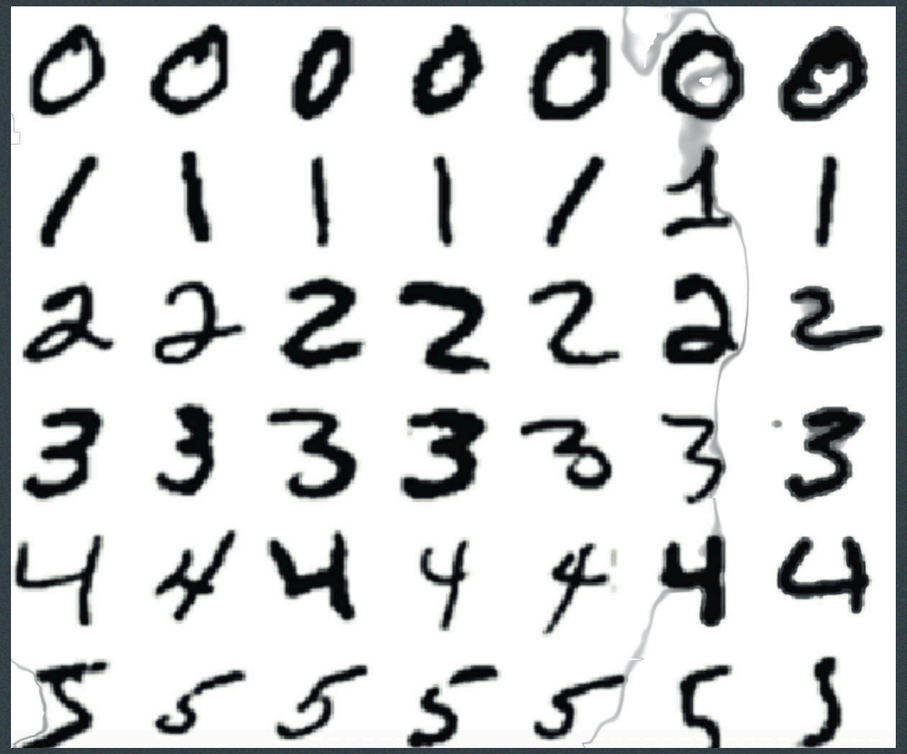
# Digit Classifier AI Algorithm

## Explained Step by Step



The first 80 images of each class in the MNIST handwritten digits dataset

# Project Objective

To create a simple convolutional neural network that takes the input of a handwritten digit and returns the classification of the digit from 0 to 9

# Resources Used

The model is written using Python in the IDE VSCode which are connected to a GitHub repo

The dataset being used is the Modified National Institute of Standards and Technology (MNIST) database which contains 60K training images and 10K test images of handwritten digits

# Libraries Used

numpy — to perform mathematical operations

tensorflow — a python library for dataflow progamming

keras — a deep learning API in tensorflow that helps training neural networks

matplotlib — a python library for data visualization

seaborn — a python library for visualizing statistical data

# Step 1: Getting the Data

```
# load dataset
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()
```

x is an array holding information about the pixel values of the images

y is a label that represents what number (0-9) the data is

# The Shape of the Data

```
Training data shape: (60000, 28, 28)
Training labels shape: (60000,)
Test data shape: (10000, 28, 28)
Test labels shape: (10000,)
```

The number of images in the X training and test array respectively

The size of the images, 28 by 28 pixels

# Normalizing the Data

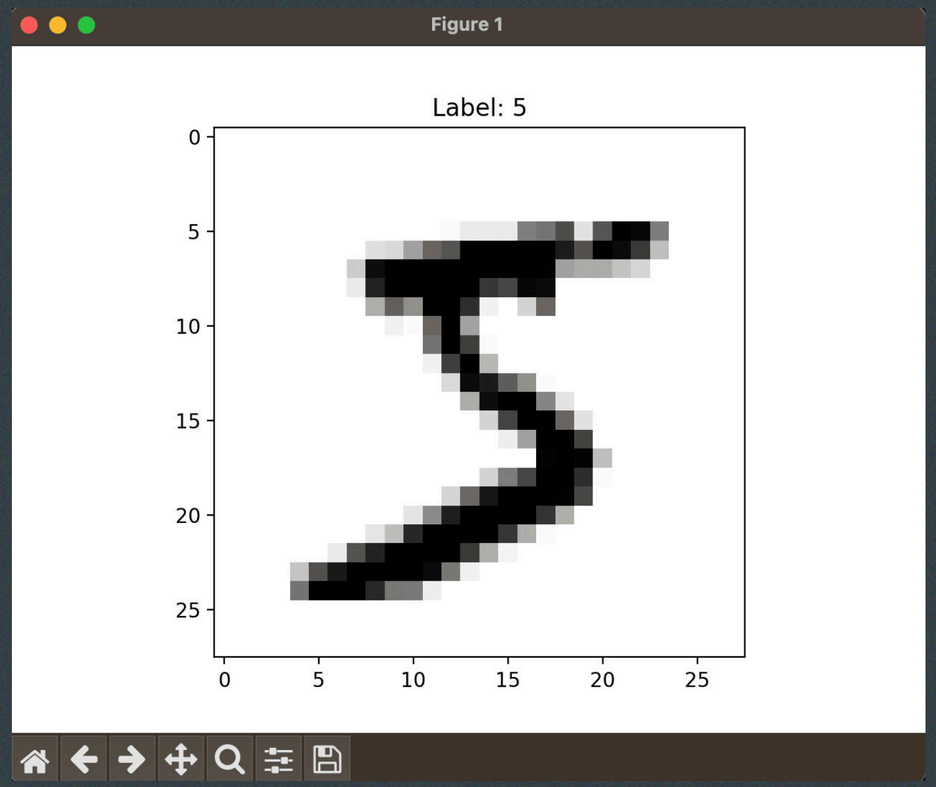
Right now each image is given as an array of 28x28 numbers ranging from 0 (white) to 255 (black)

To make the ML model more efficient you can scale the data so that the value are between 0 and 1

```
X_train = X_train / 255
X_test = X_test / 255
```

# Checking the Data

```
# use matplotlib to see data
index = 0
plt.imshow(X_train[index], cmap=plt.cm.binary)
plt.title(f"Label: {y_train[index]}")
plt.show()
print(y_train[index])
```




# Formating for Later

```
# flatten the data for the dense layers
X_train_flat = X_train.reshape(len(X_train), (28 * 28))
X_test_flat = X_test.reshape(len(X_test), (28 * 28))
```

# Step 2: Set up Fully Connected Neural Network using Tensorflows Keras API

```
# nueral network model
model = keras.Sequential([
    # layer 1
    keras.layers.Dense(128, input_shape=(784,), activation = 'relu'),
    # layer 2
    keras.layers.Dense(64, activation = 'sigmoid'),
    # layer 3
    keras.layers.Dense(12, activation = 'sigmoid'),
    # layer 4
    keras.layers.Dense(10, activation = 'softmax')
])
```

# Explaining the model: layer 1

```
keras.layers.Dense(128, input_shape=(784,), activation = 'relu'
```

use the 'dense' class in keras to create a fully connected layer

the activation function applied to every neuron in the layer

number of neurons in the layer

the orginal dimensions of the images from the dataset were 28x28x1 but were turned into a 1 Dimensional vector of length 784

$$R(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Source:https://artemoppermann.com/activation-functions-in-deep-learning-sigmoid-tanh-relu/

# Explaining the model: layer 2

```
keras.layers.Dense(64, activation = 'sigmoid'
```

use the 'dense' class in keras to create a fully connected layer

number of neurons in the layer

the activation function applied to every neuron in the layer

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

source:https://builtin.com/machine-learning/sigmoid-activation-function

# Explaining the model: layer 3

```
keras.layers.Dense(12, activation = 'sigmoid'
```

use the 'dense' class in keras to create a fully connected layer

the activation function applied to every neuron in the layer

number of neurons in the layer

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

source:https://builtin.com/machine-learning/sigmoid-activation-function

# Explaining the model: layer 4

```
keras.layers.Dense(10, activation = 'softmax'
```

use the 'dense' class in keras to create a fully connected layer

the activation function applied to every neuron in the layer

Softmax converts the output of each neuron into a probability distribution across all classes (from 0-9)

the last layer is 10 neurons because there are 10 possible digits

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

source:https://developer.apple.com/documentation/accelerate/bnnsactivationfunction/2915301-softmax

# Step 3: Train on dataset

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

model.fit(X_train_flat, y_train, epochs = 5)
```

# Explaining the training process

optimizer = 'adam'

the optimizer is an algorithim that controls how the model updates its weights based on the gradients computed during backprop

Adaptive Moment Estimation (adam) combines two popular algorithms (AdaGrad and RMSprop) to continuously adjust the learning rate during training

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

source:https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be

# Explaining the training process

```
loss = 'sparse_categorical_crossentropy'
```

loss functions measure the difference between the models predictions (y-hat) and the true label (y)

sparse categorical crossentropy is a loss function used for multi-class classification problems were the labels are integers

$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

source:https://fmorenovr.medium.com/sparse-categorical-cross-entropy-vs-categorical-cross-entropy-ea01d0392d28

# Explaining the training process

```
metrics = ['accuracy']
```

the metric being used to judge the success of the model here is accuracy

when compiled the model calculates the percentage of correct predictions

# Performance on Training data

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.7282 - loss: 1.1515
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9583 - loss: 0.1857
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9747 - loss: 0.1037
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9800 - loss: 0.0760
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9850 - loss: 0.0576
```

The loss calculated by the sparse categorical crossentropy equations

98.50% accuracy

# Performance on Test Data

97.28% accuracy

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 421us/step - accuracy: 0.9728 - loss: 0.1041
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 431us/step
```

The loss calculated by the sparse categorical crossentropy equations

The 0.0122 difference in accuracy of the models performance from training to test data is negligible

# Step 5: Create Confusion Matrix

```
confusion_matrix = tf.math.confusion_matrix(labels=y_test, predictions=y_pred_labels)
sn.heatmap(confusion_matrix, annot=True, fmt='d')
print(confusion_matrix)
```

To see if there are any numbers that are more likely to be inaccurately labeled than others

# Interpreting the Confusion Matrix

| | 0 | . | . | . | . | . | . | . | . | . | . | . | . | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | [[ 964 | 0 | 0 | 2 | 0 | 1 | 6 | 1 | 1 | 5] | | | | |
| . | [ 0 | 1121 | 3 | 1 | 0 | 0 | 1 | 1 | 8 | 0] | | | | |
| . | [ 3 | 1 | 1011 | 1 | 7 | 0 | 1 | 5 | 3 | 0] | | | | |
| . | [ 0 | 0 | 6 | 975 | 0 | 7 | 3 | 8 | 4 | 7] | | | | |
| . | [ 0 | 0 | 3 | 0 | 970 | 0 | 3 | 2 | 0 | 4] | | | | |
| . | [ 4 | 1 | 1 | 10 | 1 | 860 | 7 | 0 | 5 | 3] | | | | |
| . | [ 4 | 2 | 2 | 1 | 7 | 4 | 934 | 0 | 4 | 0] | | | | |
| . | [ 0 | 1 | 12 | 0 | 1 | 0 | 1 | 1010 | 1 | 2] | | | | |
| . | [ 7 | 0 | 3 | 0 | 7 | 5 | 4 | 5 | 938 | 5] | | | | |
| 9 | [ 1 | 3 | 0 | 2 | 19 | 1 | 1 | 11 | 3 | 968]], | | | | |

the diagonal number represent the amount of times the model correctly identified a digit as itself

the number 6 here represents the number of images of digit '0' that were incorrectly classified as '6'

similarly the number 4 here represents the number of images of the digit '6' that were incorrectly classfied as '5'

# Step 6: Create interface using Flask

```python
# import libraries
from flask import Flask, render_template, request, jsonify
from tensorflow import keras
import numpy as np
from PIL import Image, ImageOps
```

```python
# initialize Flask application
app = Flask(__name__)

# load trained model
model = keras.models.load_model('/Users/keerthanajayamoorthy/Desktop/Digit Classifier Project/model.h5')
```

# Make sure the input files are formatted in a way the algorithm will understand

```python
# formatting for input image
def prepare_image(image):
    # convert to grayscale
    image = ImageOps.grayscale(image)
    # Resize to 28x28
    image = image.resize((28,28))
    # convert to numpy array
    image = np.array(image)
    # normalize pixel values to [0, 1]
    image = image / 255
    # flatten the image
    image = image.reshape(1, 28*28)
    return image
```

# Create a page for users to input files and run the formatted image through the classifier

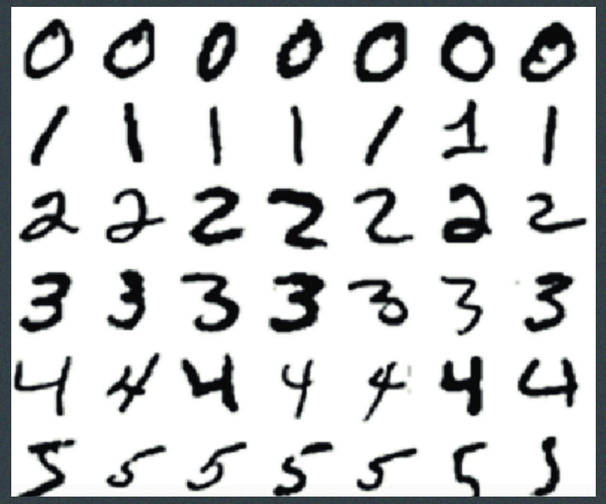
```python
# create home page
@app.route('/')
def home():
    return render_template('index.html')
```

```python
# create page to upload image
@app.route('/predict', methods=['POST'])
def predict():
    # checks to see if user uploaded a file
    if 'file' not in request.files:
        return jsonify({'error': 'no file'}), 400
    # retrieves the uplloaded file from the request
    file = request.files['file']
    # check if the file has a valid filename
    if file.filename == '':
        return jsonify({'error': 'no filename'}), 400

    # opens up the image for processing
    image = Image.open(file.stream)
    # prepares the image using the function above
    prepared_image = prepare_image(image)

    # passes prepared image to the model
    prediction = model.predict(prepared_image)
    # get the prediction
    predicted_label = np.argmax(prediction)
```

# Step 7: Test algorithm on picture I've taken

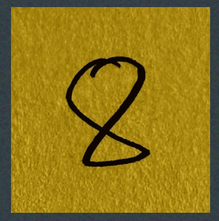
Flask Webpage on local computer

Examples of data the model has been trained on



Real life data I uploaded

Pic 1



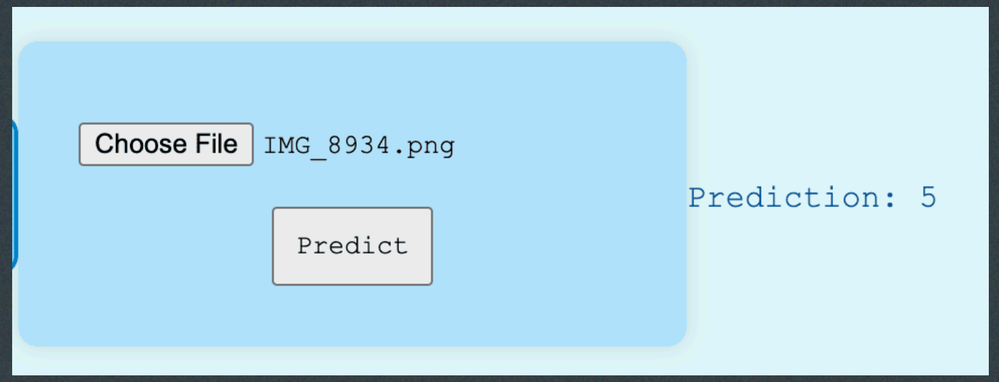
Pic 2

Pic 3

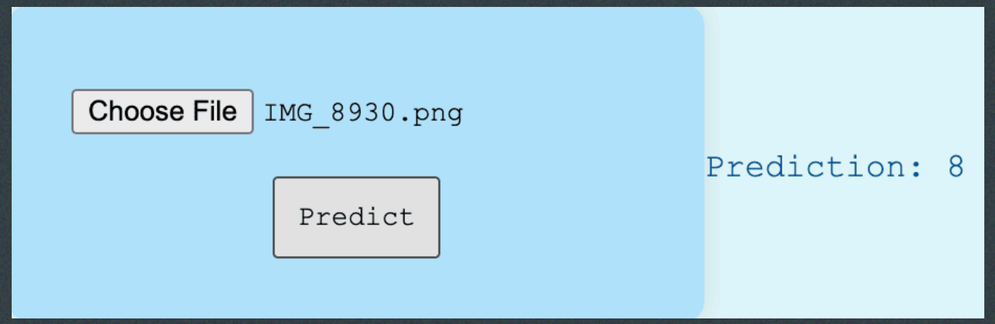
Pic 4

# First round of testing

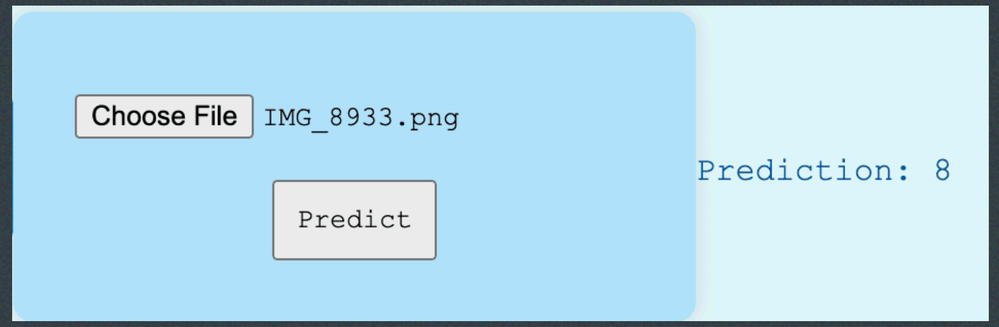
## Result of Testing Pic 1



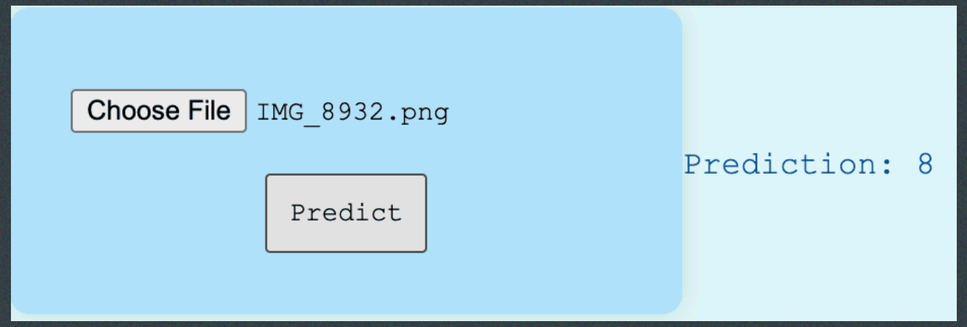

## Result of Testing Pic 3




## Result of Testing Pic 2




## Result of Testing Pic 4



# Analysis of Testing Round 1

Prediction: 5

Prediction: 8

Prediction: 8

Prediction: 8

It seems although the algorithm worked with 98.50% accuracy on the test data and 97.28% accuracy on the train data there seems to be an issue identifying digits when given real life data. It seems that the model is stuck as most of the answers ended up being 8 despite giving the webpage different files each time. And the page was refreshed after submitting every file. I tried quitting and re-running py.app before submitting each picture again and the predictions are on the left.

# Things to change for Round 2 of testing

- Find a way to display the confusion matrix each time the model is run on a real life data file to see what any possible issues could be
- Tweak the model itself, changing the loss function and epoch numbers to see if that helps, maybe even change the layers of the model

```
INFO:werkzeug: * Debugger PIN: 105-172-601
INFO:werkzeug:127.0.0.1 - - [02/Sep/2024 16:58:31] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [02/Sep/2024 16:58:31] "GET /static/styles.css HTTP/1.1" 304 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
Result: 5
Probabilities: [2.9835987e-03 7.4200216e-04 9.3748276e-05 2.8650287e-01 4.6159814e-05
 6.3313353e-01 8.9223310e-03 2.7105345e-03 6.2980585e-02 1.8846201e-03]
INFO:werkzeug:127.0.0.1 - - [02/Sep/2024 16:58:39] "POST /predict HTTP/1.1" 200 -
```

Based on what's being logged in the terminal when I upload a file and press submit, it seems like the input file is being passed to the model and the prediction is being sent back to the webpage though I'm not sure why it's stuck on 8 mostly. I've gotten a couple different numbers as prediction as well when I submitted the files again and again; but pretty much 0% of predictions were accurate.

# Round 2 of Testing

[Changes made]

Changed to a convolutional network instead of a feed forward nueral network

## Outcome

Training Accuracy: 0.9850 -> 0.9886
Testing Accuracy: 0.9728 -> 0.9910

```
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 15s 8ms/step - accuracy: 0.9943 - loss: 0.0175 - val_accuracy: 0.9910 - val_loss:
 0.0304
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9886 - loss: 0.0383
Test Accuracy: 0.9910
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

Now the algorithm seems to give more varied outputs but still sticks to either 8, or 4 as the predictions regardless of the different numbers I give. But it does perform better on the testing/training data and when I submit examples from the MNIST database instead of my handwritten real-life data the prediction works a little better and gives more varied predictions.

# Round 2 of Testing: New Nueral Network

had to take out lines formatting the image to be flattened to one dimension in model.py and app.py

```
# nueral network model (changed to cnn from feed forward)
model = keras.Sequential([
    # Conv layer 1
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    keras.layers.MaxPooling2D((2, 2)),
    # Conv layer 2
    keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    keras.layers.MaxPooling2D((2, 2)),
    # Conv layer 3
    keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    # Flatten layer
    keras.layers.Flatten(),
    # Fully connected layer
    keras.layers.Dense(64, activation='relu'),
    # Output layer
    keras.layers.Dense(10, activation='softmax')
])
```

# Things to Change for Round 3 of testing

- Add a threshold to eliminate background noise in the real life data and to help center the digit in the image like the MNIST dataset does
- The MNIST dataset has white digits on a black background so maybe invert the image colors
- Add a dropout layer to reduce overfitting (make sure the model isn't trained too closely on the training data)
- Use a larger variety of real life data, other than the 4 handwritten digits I've been using

# Round 3 of Testing: reduce background noise

```python
# formatting for input image
def prepare_image(image):
    # convert to grayscale
    image = ImageOps.grayscale(image)
    # Resize to 28x28
    image = image.resize((28,28))
    # convert to numpy array and normalize
    image = np.array(image).astype('float32') / 255.0
    # apply threshold to reduce background noise
    image[image < 0.1] = 0.0
    # invert image so it matches the MNIST data the model has been trained on
    image = ImageOps.invert(Image.fromarray(np.uint8(image * 255)))
    image = np.array(image).astype('float32') / 255.0
    # reshape to match the models input shape
    image = image.reshape(1, 28, 28, 1)
    return image
```

Threshold: if a pixel value is below a certain threshold (0.1 in my case) then it just becomes white to make the background more clear

# Round 3 of Testing: reduce overfitting

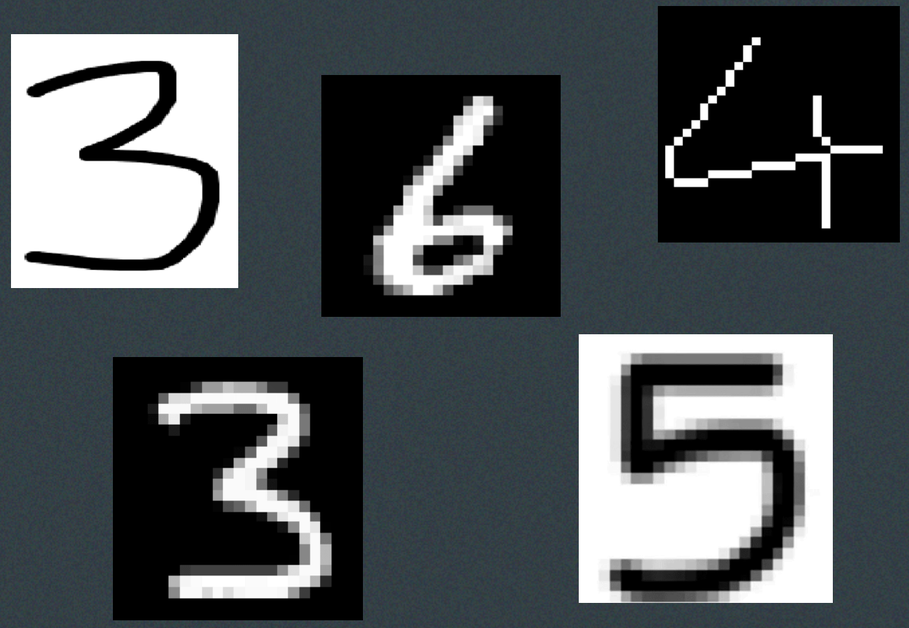
```
model = keras.Sequential([
    # Conv layer 1
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Dropout(0.25),
    # Conv layer 2
    keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Dropout(0.25),
    # Conv layer 3
    keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    # Flatten layer
    keras.layers.Flatten(),
    # Fully connected layer
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dropout(0.5),
    # Output layer
    keras.layers.Dense(10, activation='softmax')
])
```

Add dropout layers in my model so 25% or 50% of the neurons are randomly set to 0 every time the model runs so it doesn't rely too heavily on one neuron

# Round 3 of Testing: varied real life data

Before: the same digits I wrote on a yellow background and MNIST data example I found online

Now I've also added these example from the web:

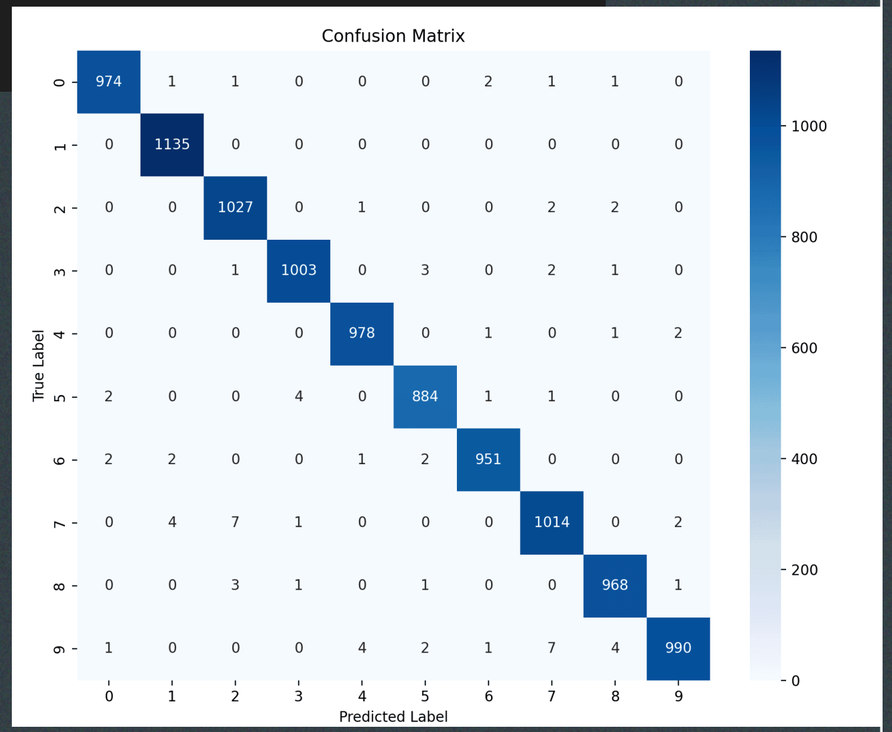


# Results of Round 3 of testing

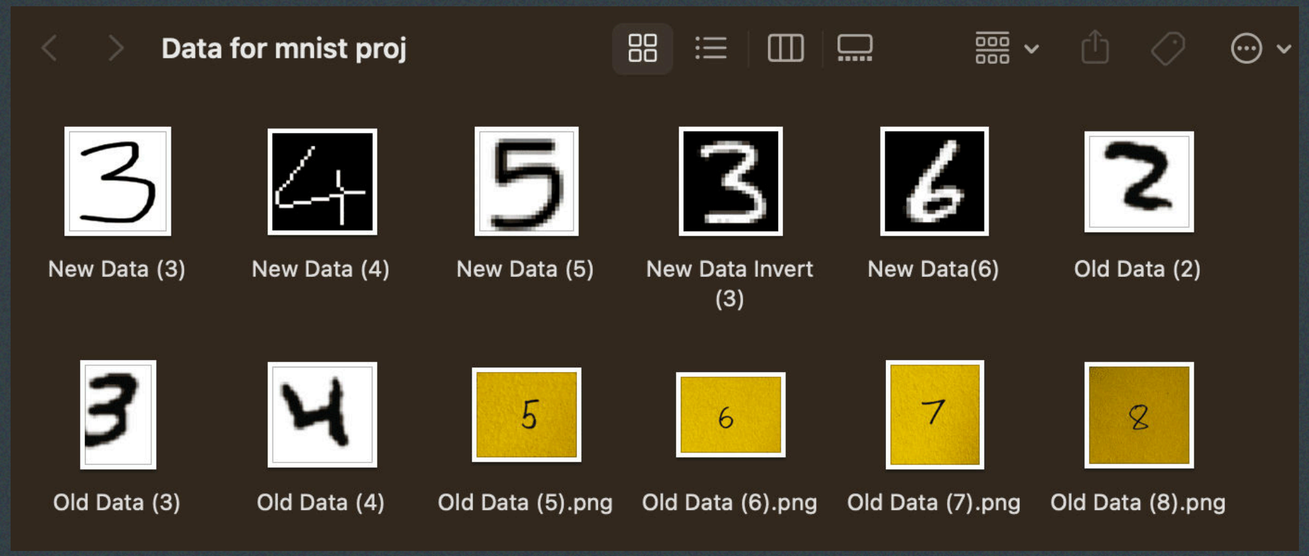
```
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.9830 - loss: 0.0608 - val_accuracy: 0.9924 - val_loss:
 0.0272
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9909 - loss: 0.0317
Test Accuracy: 0.9924
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

Training Accuracy: 0.9850 -> 0.9886 -> 0.9830
Testing Accuracy: 0.9728 -> 0.9910 -> 0.9924

Although important note, even though the training accuracy went down the val_accuracy went up which means that it tested well on the new epoch of data so it should work better than before on unseen data because the val_accuracy from the last round was 0.9910



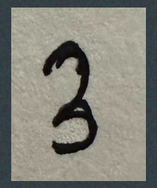
# Results of Round 3 on real life data




The model worked accurately 87.5% of the time on the images that were black and white but consistently got the predictions wrong on the yellow images

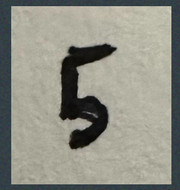
# Results of Round 3 on real life data

Prediction: 1



Prediction: 3

Prediction: 4



Prediction: 5

Since it didn't seem to work on the yellow I tried testing more pictures I took that were black and white this time after quitting and restarting the application and those were the predictions 75% accuracy

# Conclusion and Future Plans

It seems that the model works on only black and white images. If I continued working on this model to improve it I would train the model on a dataset that consisted of a mix of the MNIST dataset and a colorful images that have been formatted to be black and white. This way the model would perform better on real-life data which is not usually monochrome black and white.

Try out the Website for Yourself!

# Thanks!

Você tem alguma pergunta?

seuemail@freepik.com
+91 620 421 838
seusite.com