# Handwriting Recognition

## Using Neural Networks

Matthew Smith

# 1 The Goal

- Build a machine learning model that can read an image of a handwritten alphanumeric character and correctly label it
- Can be used to read scanned, handwritten forms

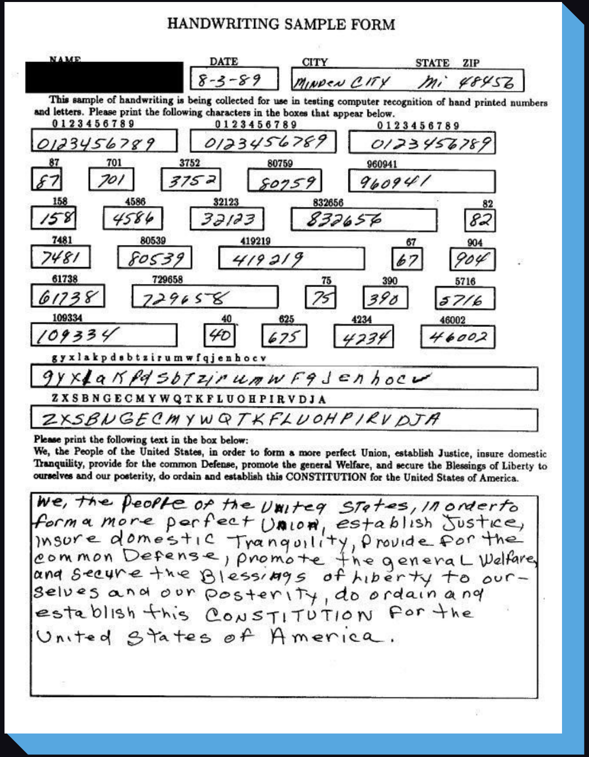
 = A

 = B

 = C

# 2 The Data

- NIST Handprinted Forms and Characters Database
- Contains 814,255 images from over 3,600 different writers
- Includes numbers, uppercase letters, and lowercase letters
- All images are 128 x 128 pixels



HANDWRITING SAMPLE FORM

NAME [redacted] DATE 8-3-89 CITY MINDEN CITY STATE MI ZIP 48456

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0123456789 0123456789 0123456789

87 701 3752 80759 960941

158 4586 32123 832656 82

7481 80539 419219 67 904

61738 729658 75 390 5716

109334 40 625 4234 46002

gyxlakpdsbtzirumwfqjenhocv

ZXSBNGECMYWQTKFLUOHPIRVDJA

Please print the following text in the box below:

We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

# 3 Encoding

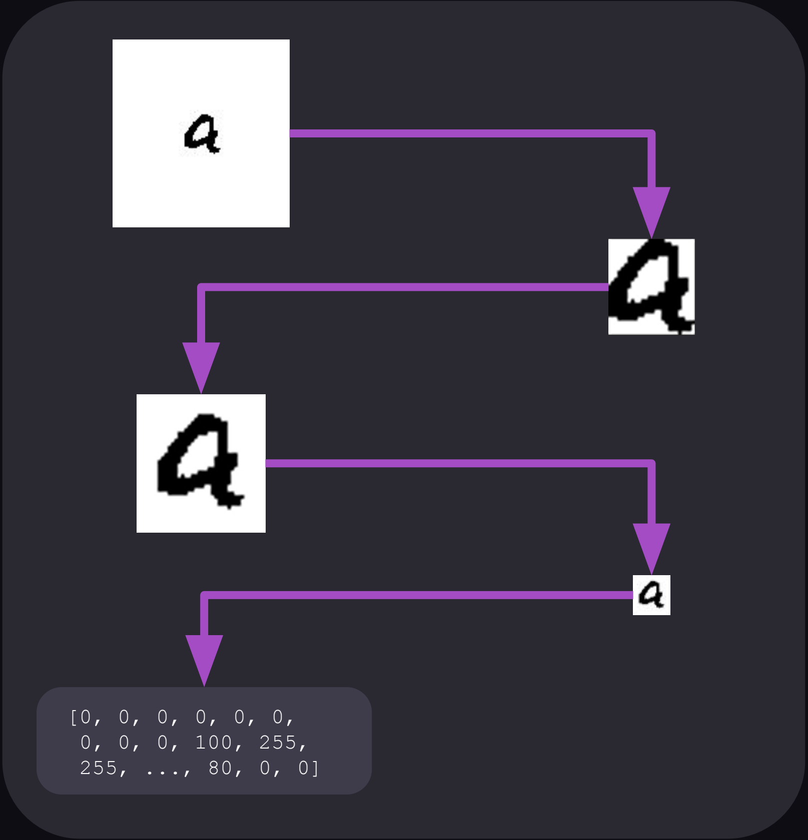
- Images must be converted to a format readable by a program
- General idea:
  - Crop image
  - Lower image resolution
  - Interpret image as 2D array of values
  - Flatten into 1D array

```
[[0, 0, 0, ..., 0],
 [0, 0, 0, ..., 0],
 [0, 0, 0, ..., 0],
 [0, 0, 0, ..., 0],
 [0, 0, 0, ..., 0],
 ...
 [0, 0, 0, ..., 0]]
```

```
[0, 0, 0, 0, 0, 0,
 0, 0, 0, 100, 255,
 255, ..., 80, 0, 0]
```

# Encoding Algorithm

1. Crop the image as tightly as possible
2. Make the image square by padding out the sides
3. Downsize image
4. Encode as array of darkness values



# File Size Reduction

Number of cells:

**834,611,375**

Final file size:

**216 MB**

- Data size needed to be kept low to work within memory/storage constraints
- Features were kept between 0 and 255 and stored as unsigned 8-bit integers to save space
- Data was stored as a .parquet file

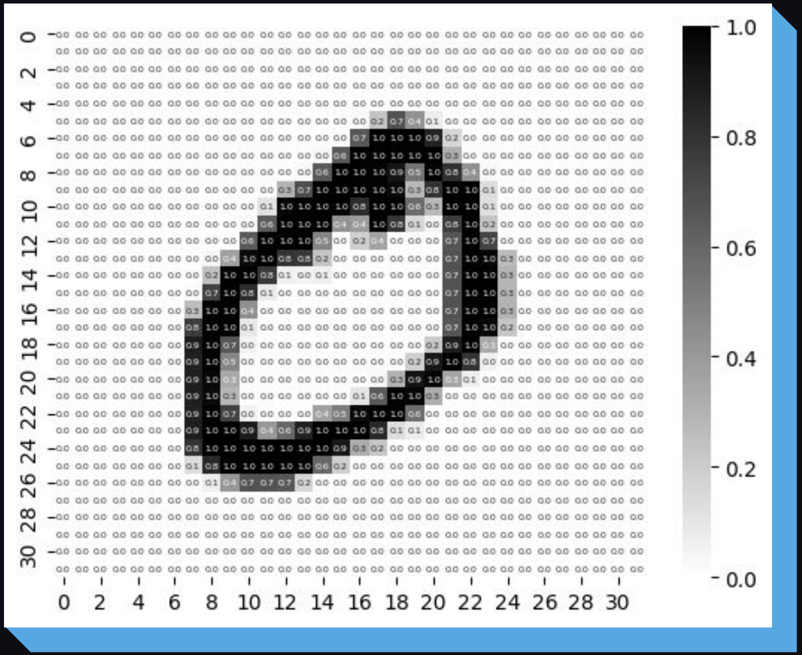
# 4 Data Exploration

- Data could now be loaded into pandas
- Features were normalized to floating-point values between 0 and 1

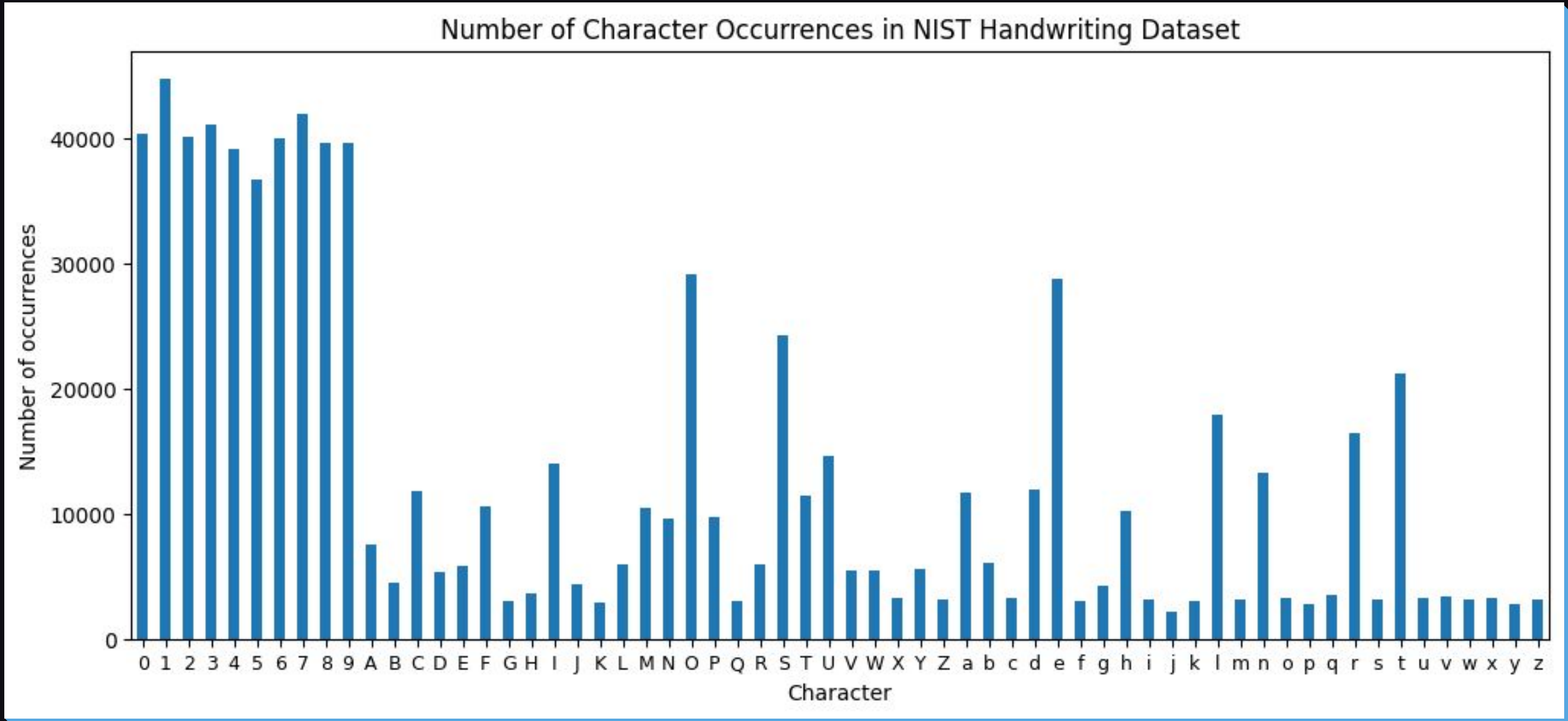
| | px0-0 | px0-1 | px0-2 | px0-3 | px0-4 | px0-5 | ... | px31-27 | px31-28 | px31-29 | px31-30 | px31-31 | label |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 |
| 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.2 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 |
| 2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.3 | 0.4 | ... | 0.2 | 0.1 | 0.0 | 0.0 | 0.0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 814252 | 0.0 | 0.0 | 0.0 | 0.1 | 0.2 | 0.4 | ... | 0.3 | 0.2 | 0.1 | 0.0 | 0.0 | z |
| 814253 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.3 | ... | 0.3 | 0.1 | 0.0 | 0.0 | 0.0 | z |
| 814254 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | z |

# Interpreting Rows as Images

- Each row in the data set can be interpreted as an image
- 0 is white, 1 is black, in between is gray



# Distribution of Data



# 5 Training and Testing Sets

- Data set was shuffled
- 20% of the data was assigned to the testing set
- Labels were mapped to integers between 0 and 61 for use in the machine learning model

Training set:
**651,404** rows

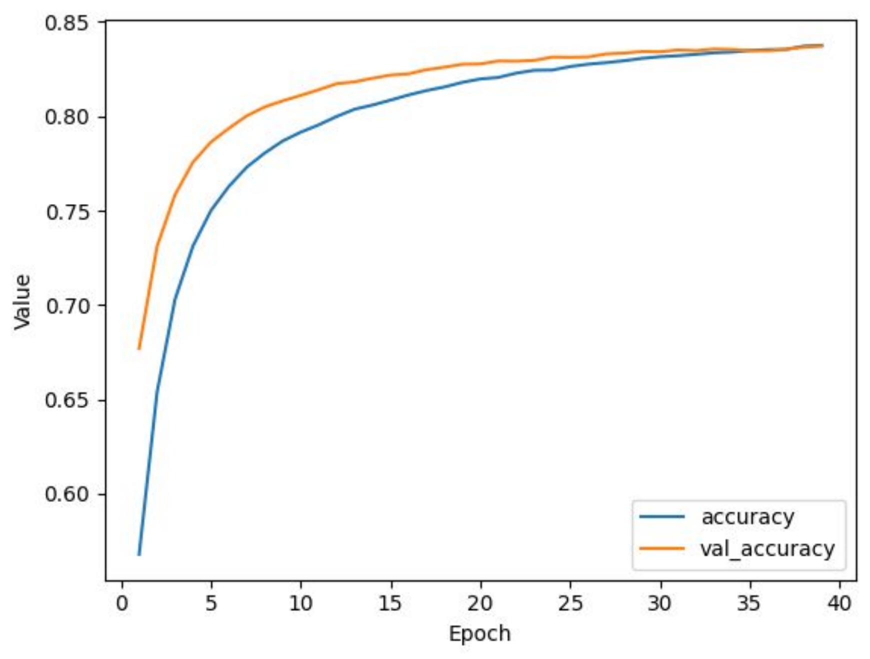
Testing set:
**162,851** rows

# 6 The Neural Network

- Multi-class classification problem
- Features: 1024 pixel values between 0 and 1
- Label: Integer between 0 and 61 corresponding to an alphanumeric character
- Output of a prediction: Softmax probability distribution

# Training the Neural Network

- 20% of training data used as validation set
- Layers:
  - Input layer
  - Hidden layer: 256 neurons
  - Hidden layer: 128 neurons
  - Hidden layer: 64 neurons
  - Dropout regularization layer
  - Output layer: 62 neurons
- 40 epochs
- 0.002 learning rate
- 0.35 dropout rate



# 7 Evaluating the Model

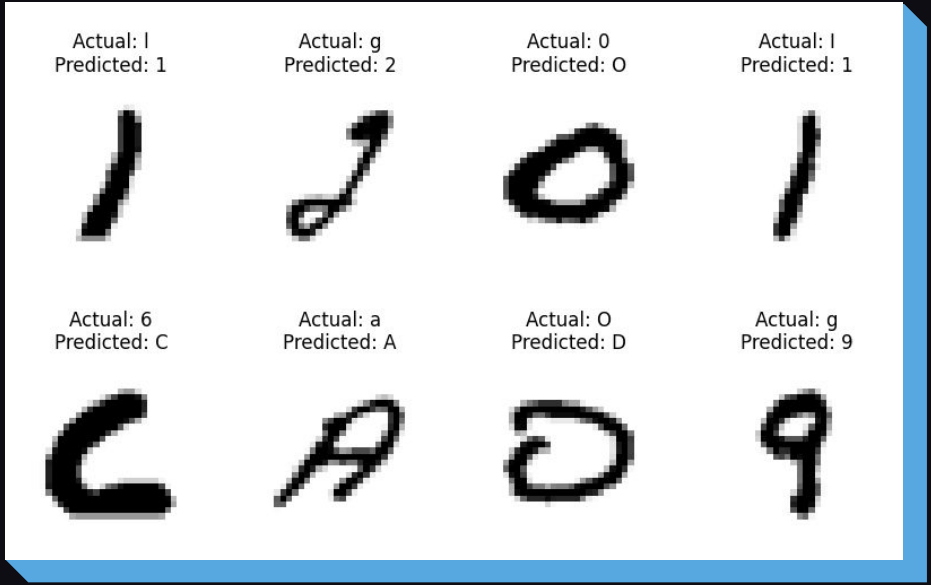
**84%**

accuracy rate

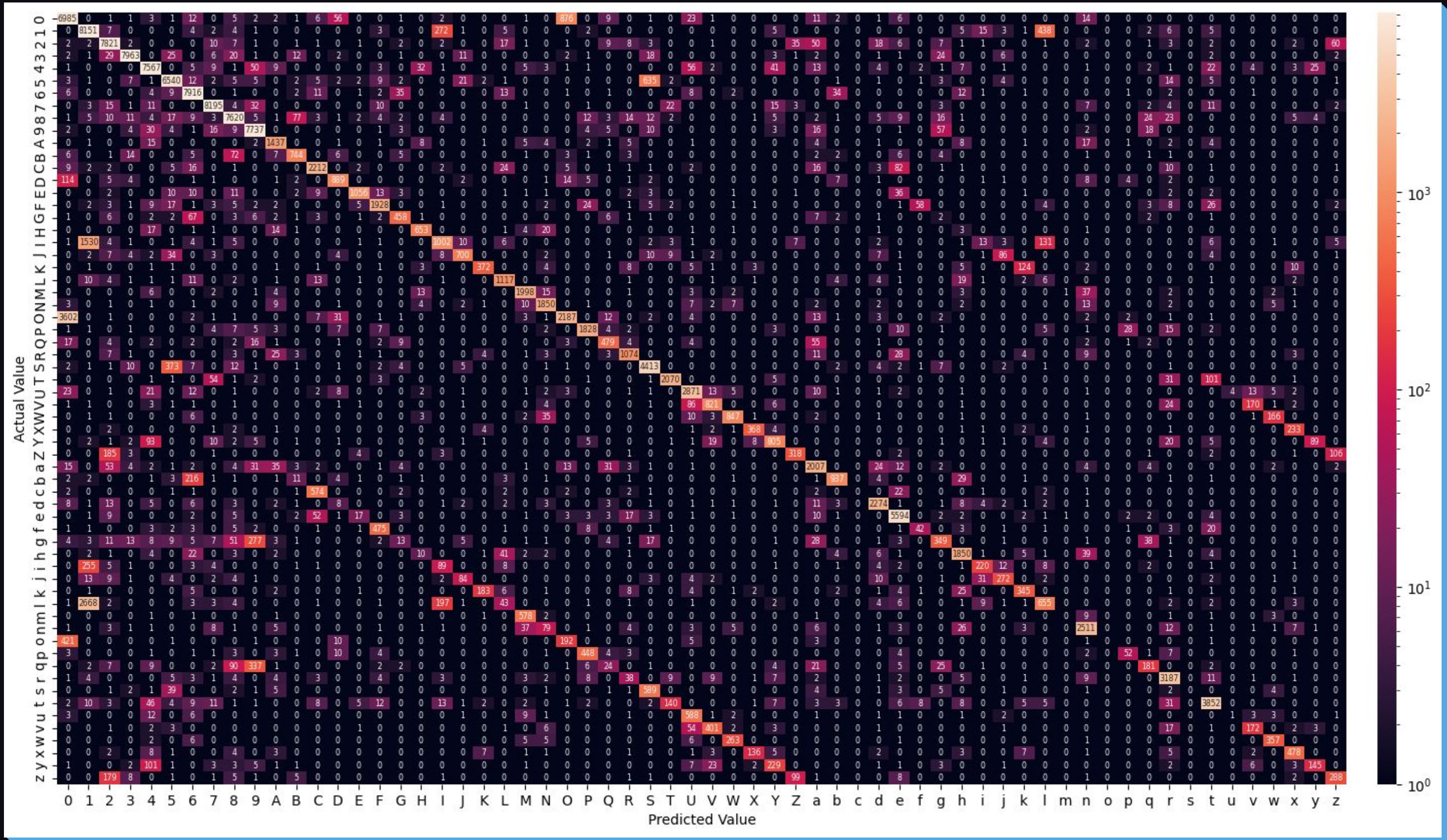
The model made correct predictions on the testing data about 84% of the time.

# Examples of Incorrect Predictions

- Commonly confused characters:
    - `0`, `O`, and `o`
    - `1`, `I`, and `l`
- Sources of errors:
    - Reduced detail from low image resolution
    - Visually indistinguishable characters
    - Poor handwriting

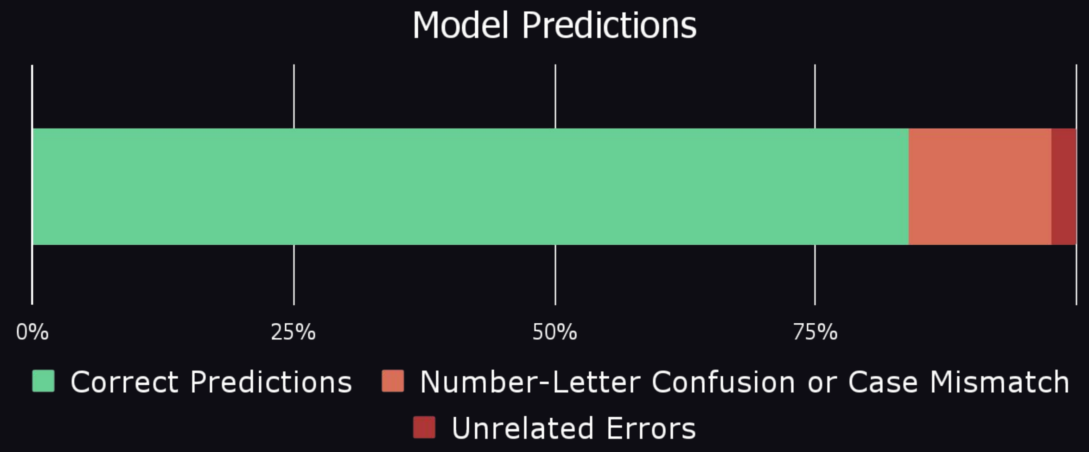


# Confusion Matrix



# Common Errors

- About 60% of incorrect predictions resulted from confusing numbers and letters
- 25% of incorrect predictions resulted from mismatching letter case

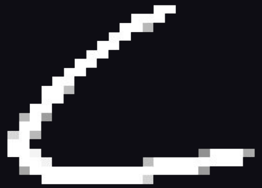


# 8 Further Testing of the Model

- What would happen if we generalized the model to new data?

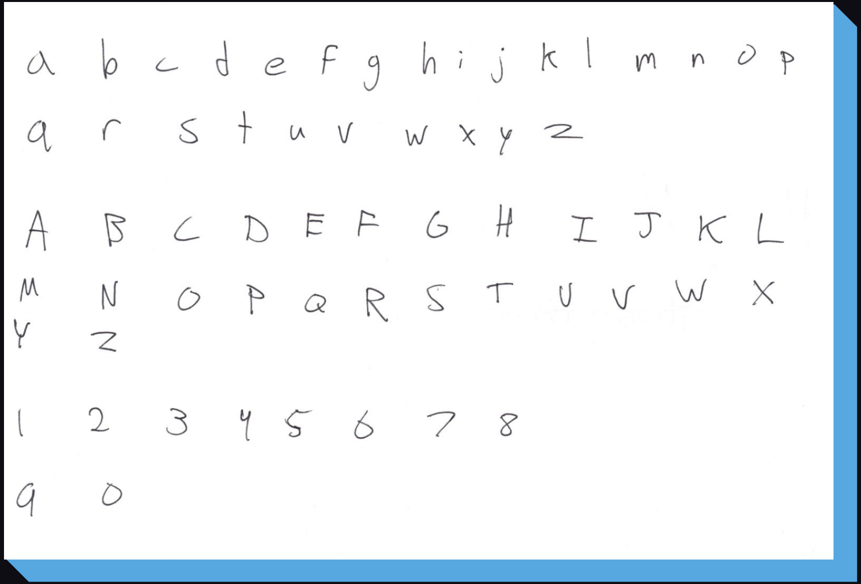
 = ?

 = ?

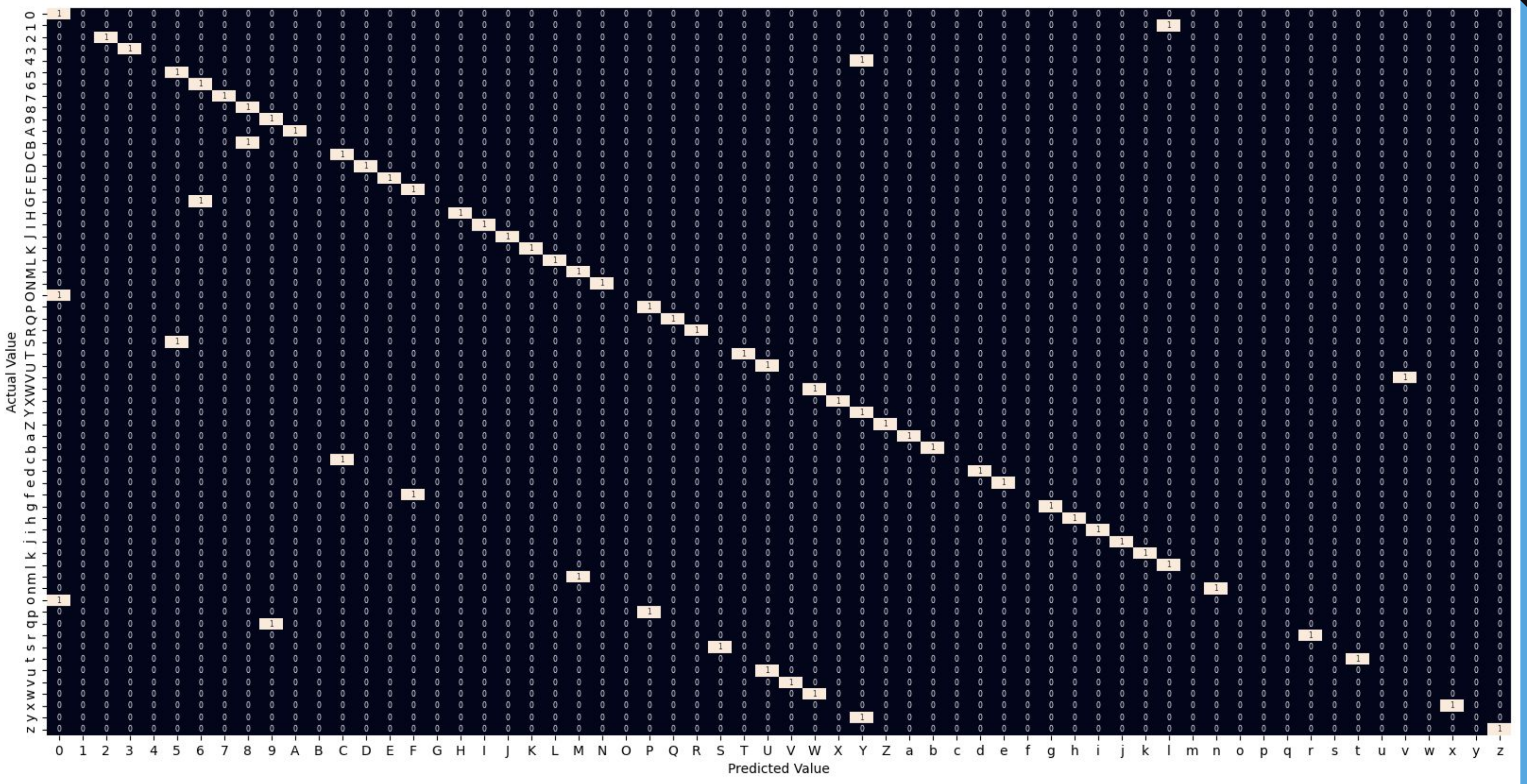
 = ?

# My Handwriting

- I wrote one of each alphanumeric character
- Scanned and encoded them the same way as the original data



# Confusion Matrix



# Accuracy of Model on New Data

**62**

samples

**70%**

accuracy rate

- Model achieved 70% accuracy on my handwriting
- Reasons for reduced accuracy:
  - My handwriting is bad
  - Low sample size
  - Different data distribution
  - Different scanning, encoding, and normalizing process than data on which model was trained

# 9 Conclusion and Possible Improvements

Accuracy lied between **83%-84%** at best - how to raise it?

- More epochs = higher accuracy, but training takes much longer
- Next time, I would try a **convolutional neural network**
  - Demonstrate high success in computer vision problems
  - Could take into account data in neighboring pixels
- Some inaccuracy resulted from normalization/encoding
  - Reduced size meant reduced detail
  - Encoding algorithm removed size disparities that distinguished letters

# Future Applications

How to avoid the most common errors in practice:

## 1. Differently trained models for different fields

- Most fields use only numbers or only letters, not both
- No need to expect letters in a numeric field, or vice versa
- Would remove 60% of incorrect predictions

## 2. Loss of case sensitivity

- Many mistakes resulted from confusing uppercase and lowercase forms of letters
- Case sensitivity is nonessential in most forms
- Would remove 25% of incorrect predictions

# Thank you