Handwritten Letter Classification

Justin Farnsworth

September 2, 2020

Summary

In this project, the Keras and TensorFlow packages were used to implement a neural network model. A sequential model was used to identify grayscale images of handwritten letters. These images were converted to vectors of pixel values beforehand. The pixel values were pre-processed via binarization and pixels with low variability were omitted.

After training, the model achieved an accuracy **over 98%**. All letters had a recall rate and precision rate of over 90%. Most of the letters obtained over 95% recall rate and some even achieved upwards 99%. Similarly, the precision rates were also very high as they were mostly over 97%.

A brief analysis of the prevalence of the data showed that some letters appeared more often than others. For example, the letters O and S appeared the most, while I and F appeared the least. Despite the disparity, each letter had over 1,000 images.

The data can be accessed here: https://www.kaggle.com/ashishguptajiit/handwritten-az

Loading The Data

For this project, the following packages were used:

```
# Required packages
if (!require(tidyverse)) { install.packages("tidyverse"); library(tidyverse) }
if (!require(caret)) { install.packages("caret"); library(caret) }
if (!require(matrixStats)) { install.packages("matrixStats"); library(matrixStats) }
if (!require(keras)) { install.packages("keras"); library(keras) }
if (!require(tensorflow)) { install.packages("keras"); library(keras) }
if (!require(reticulate)) { install.packages("reticulate"); library(reticulate) }
```

Before continuing, the following lines of code were run in order to set up the Anaconda environment. This allowed us to use the Keras and TensorFlow packages. Note that `install_keras()` and `install_tensorflow()` may trigger a reset, which would prevent the rest of the code block to run. To address this, the lines were run one at a time. Conveniently, this only needed to be done only once.

```
# Creates the Anaconda environment
conda_create("Keras_TensorFlow")

# Installs Keras in the Anaconda environment
# NOTE: This may trigger a reset, preventing the rest of this code block from running
use_condaenv("Keras_TensorFlow", required = TRUE)
```

```r
install_keras()

# Installs TensorFlow in the Anaconda environment
# NOTE: This may trigger a reset, preventing the rest of this code block from running
use_condaenv("Keras_TensorFlow", required = TRUE)
install_tensorflow()
```

After the Anaconda environment is set up, we connected to it using the following line of code.

```r
# Connect to the Anaconda environment
use_condaenv("Keras_TensorFlow", required = TRUE)
```

Then we loaded the data and converted it into a matrix. The column names were dropped as well.

```r
# Load the data
# Source: https://www.kaggle.com/ashishguptajiit/handwritten-az
data <- as.matrix(
  read_csv(
    unz("A-Z_Handwritten_Data.zip", "A-Z_Handwritten_Data.csv"),
    col_names = FALSE
  )
)
colnames(data) <- NULL
```

There are 372451 rows and 785 columns. The 1st column represents the letters A-Z using the numbers 0-25 respectively, while the remaining columns represent the pixel values ranging from 0 to 255. The higher the value, the darker the pixel.

The following table shows the prevalence of each letter in the dataset. We observed that the most prevalent letters are O and S, while the least prevalent are I and F.

```
## # A tibble: 26 x 3
##    Number Letter Total
##     <dbl> <chr>  <int>
##  1      0 A      13870
##  2      1 B       8668
##  3      2 C      23409
##  4      3 D      10134
##  5      4 E      11440
##  6      5 F       1163
##  7      6 G       5762
##  8      7 H       7218
##  9      8 I       1120
## 10      9 J       8493
## 11     10 K       5603
## 12     11 L      11586
## 13     12 M      12336
## 14     13 N      19010
## 15     14 O      57825
## 16     15 P      19341
## 17     16 Q       5812
## 18     17 R      11566
## 19     18 S      48419
```

```
## 20     19 T      22495
## 21     20 U      29008
## 22     21 V       4182
## 23     22 W      10784
## 24     23 X       6272
## 25     24 Y      10859
## 26     25 Z       6076
```

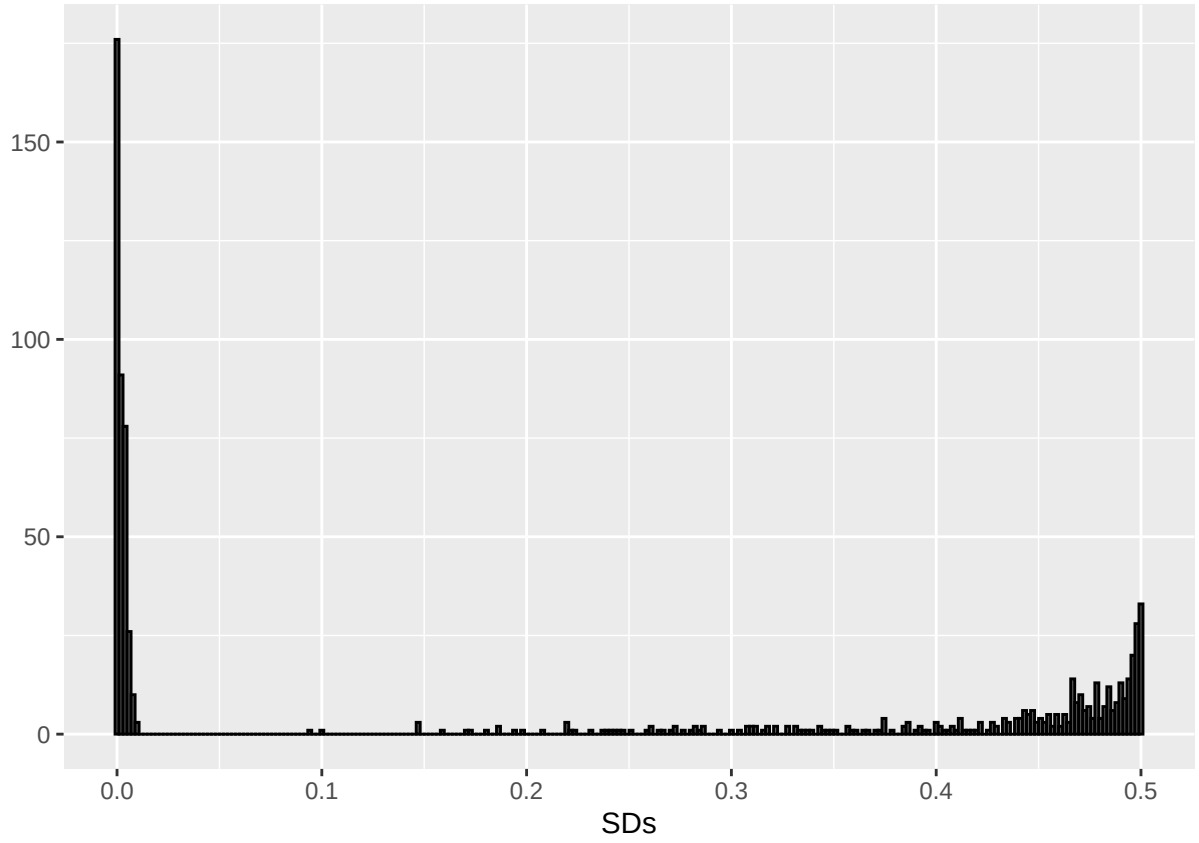
Pre-Processing The Data

The pixel values ranged from 0 to 255, where 0 represented a white pixel and 255 represented a black pixel. The values in between were generally gray pixels. To eliminate grey smudges, the values were binarized. All values closer to 0 were converted to 0 and vice versa. We also extracted the labels, which was denoted `y`, from the pixel values, which was denoted `x`.

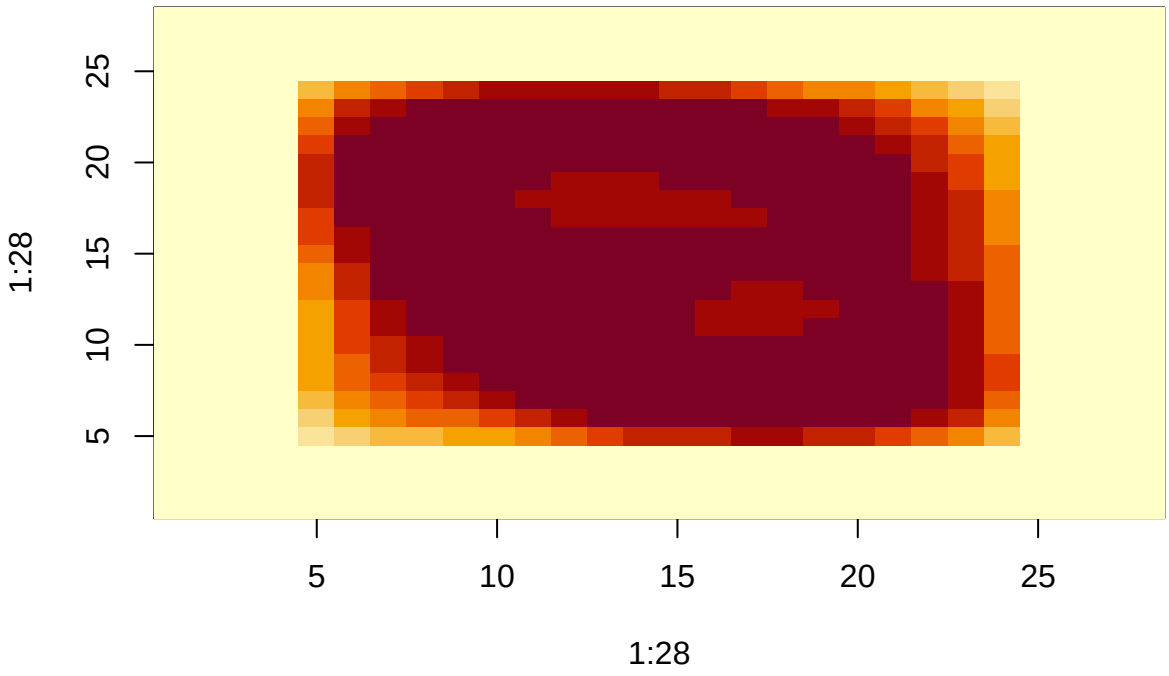
```
# Separate the features (x) from the labels (y)
# Binarize x by converting small numbers to 0 and large numbers to 1
x <- (data[,2:785] >= 255/2) * 1
y <- data[,1]
```

Then we calculated the standard deviation for each column in `x` and plotted their variabilities, allowing us to see spaces where the letters occupied the most as well as where the threshold should be.

```
# Save the column SDs for faster computation
SDs <- colSds(x)

# Plot the frequency of SDs
qplot(SDs, bins = 256, color = I("black"))
```




```
# Plot the pixels and their variabilities
image(1:28, 1:28, matrix(SDs, 28, 28))
```



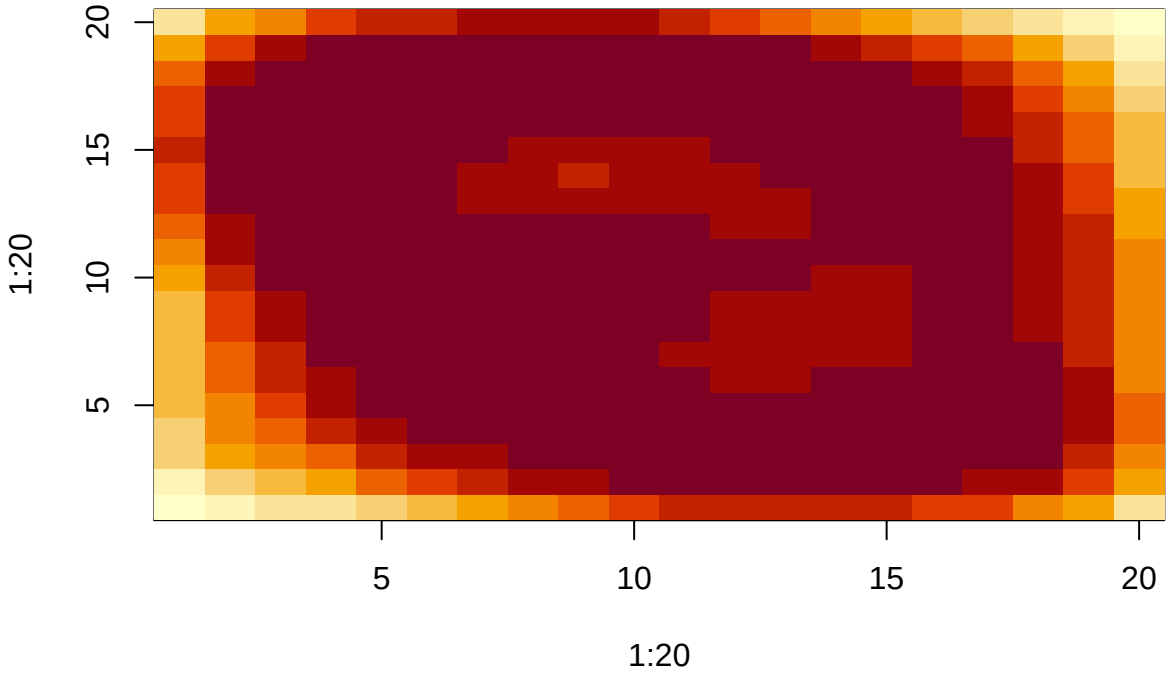
As expected, the most variability was found to be in the center. Conveniently, the area is comprised of a 20x20 region directly in the center of the image.

To drop the columns with little to no variability (the area outside the 20x20 region in the center), we established the threshold to be 0.05.

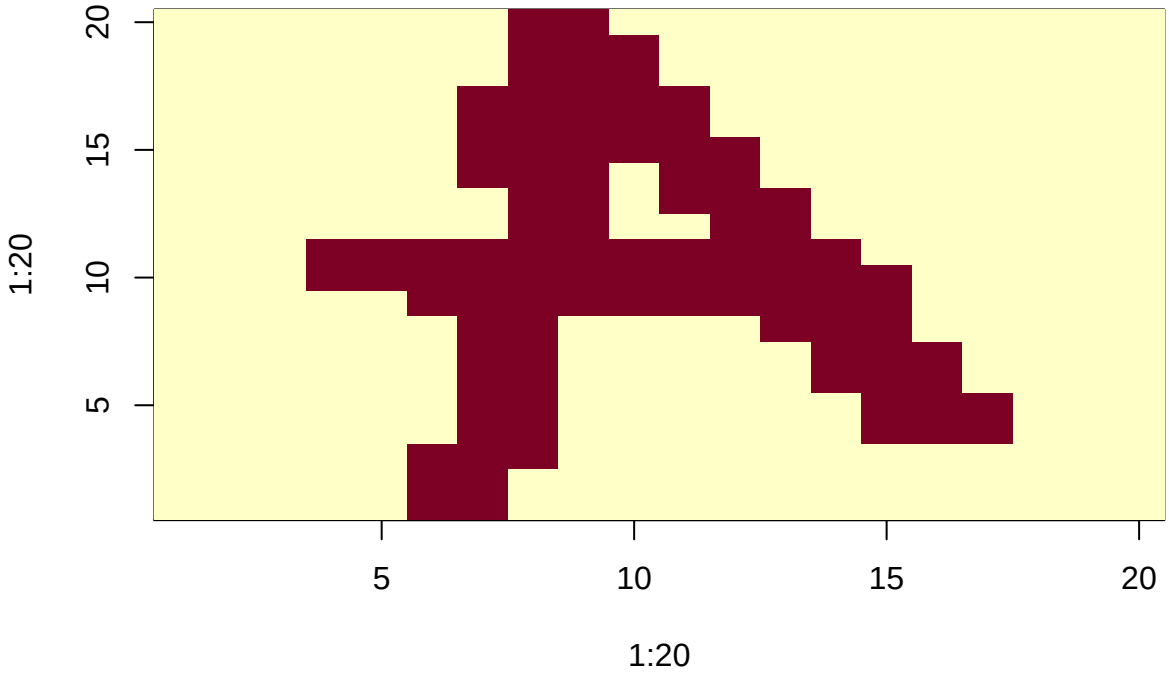
```
# Keep columns with higher variability
x <- x[,SDs >= 0.05]
```

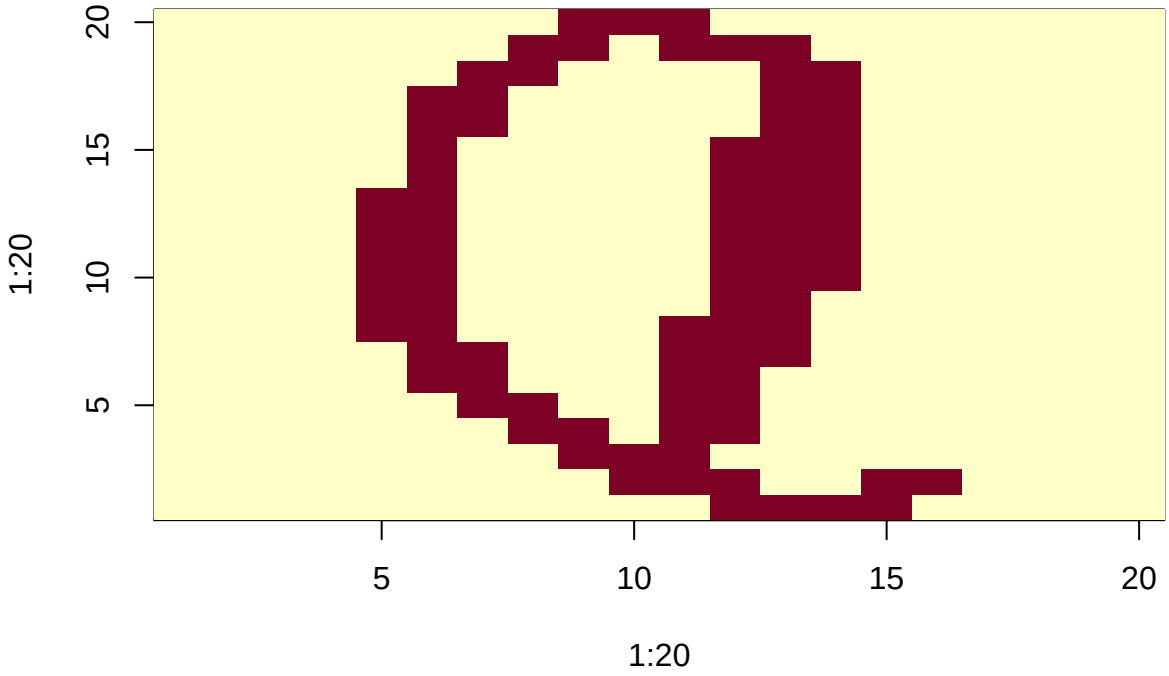
After pre-processing, we observed that there are 400 pixels remaining, which is the correct area of a 20x20 image.

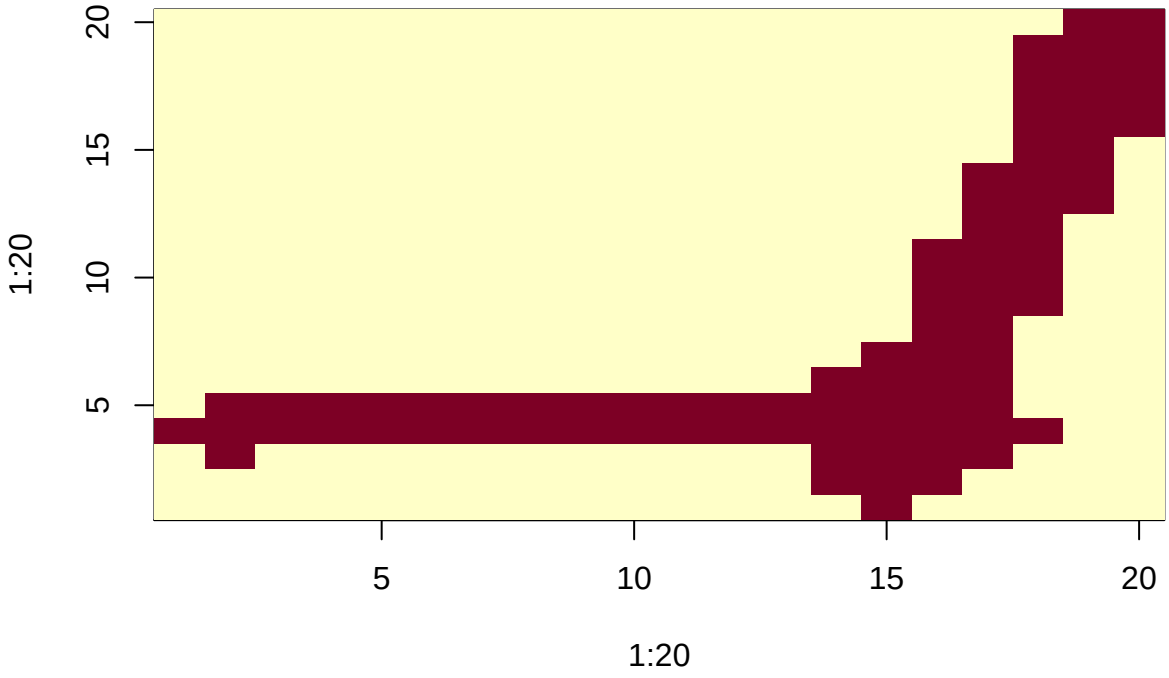
```
# Show the variabilities of the remaining columns
image(1:20, 1:20, matrix(colSds(x), 20, 20))
```

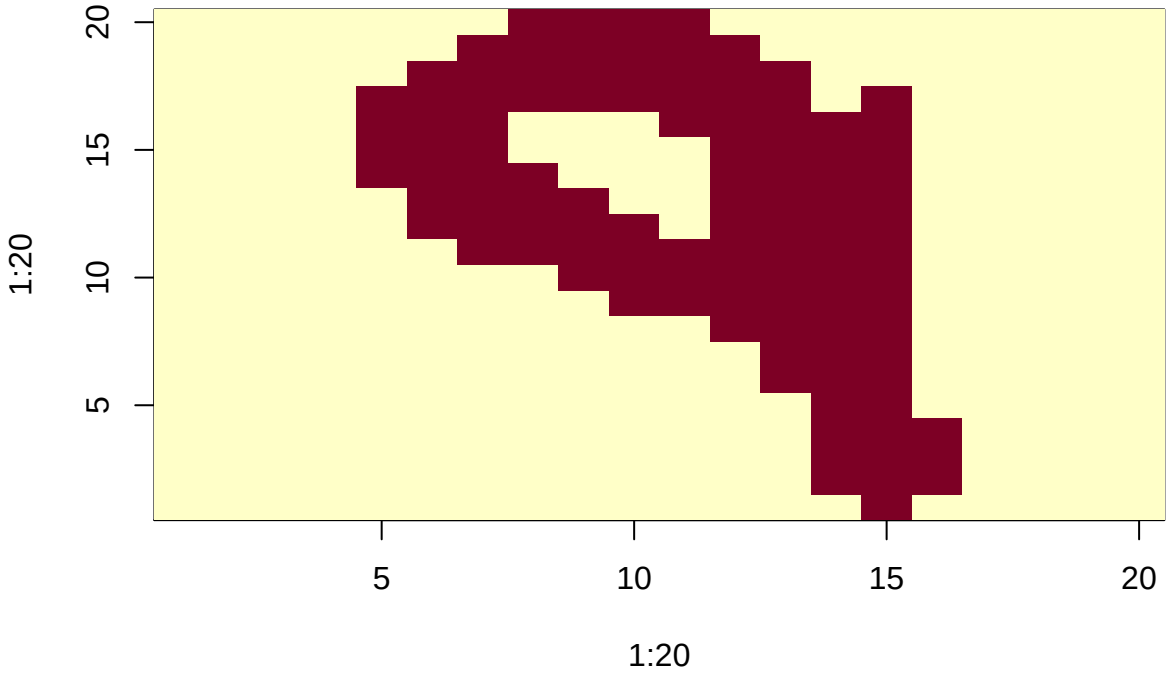


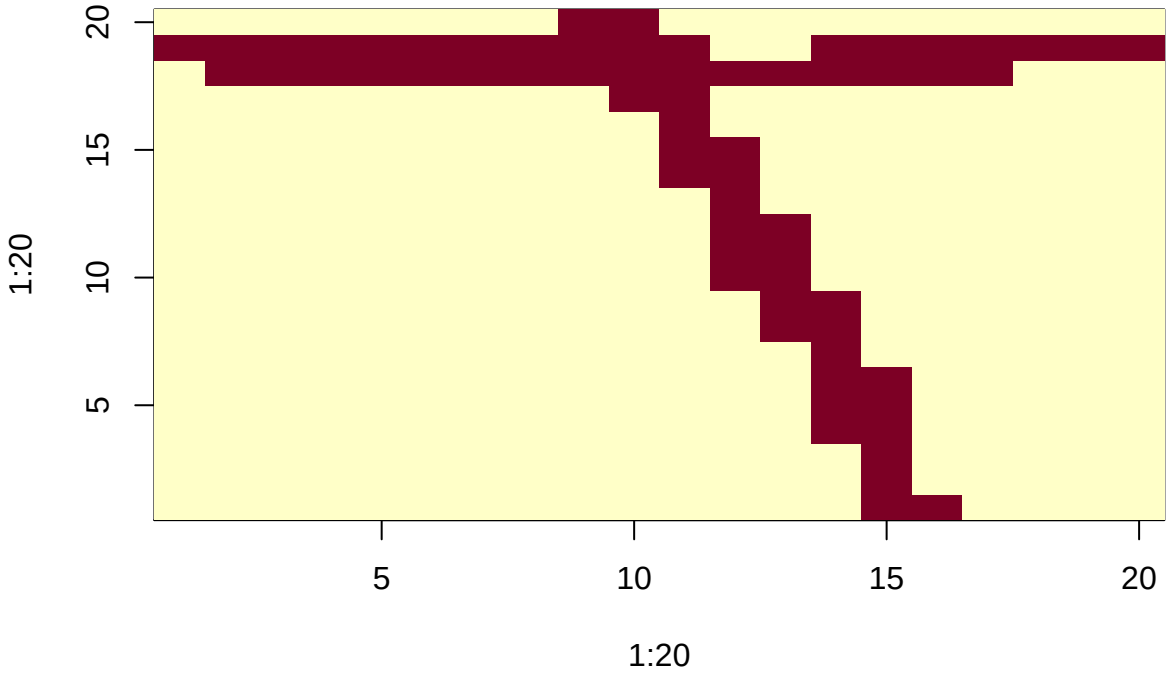
We can observe the images of the handwritten letters after pre-processing.

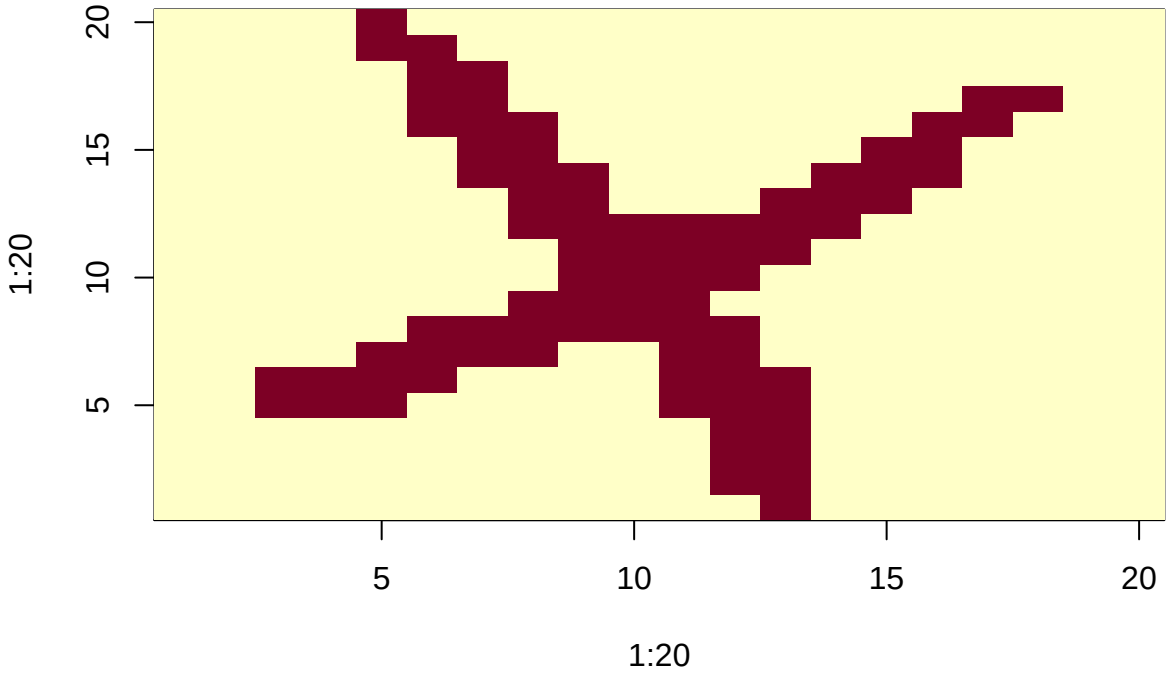

1:20
5
10
15
20
1:20
5
10
15
20


1:20
5
10
15
20
1:20
5
10
15
20


1:20
5
10
15
20
5
10
15
20
1:20


1:20
5
10
15
20
1:20
5
10
15
20


1:20
5
10
15
20
1:20
5
10
15
20



Training & Test Sets

For this project, we split the data into a training set and a test set. The training set consisted of 80% of the data while the test set consisted of the remaining 20%.

```
# Split the data into a training set (80%) and a test set (20%)
set.seed(2)
test_index <- createDataPartition(y, p = 0.2, list = FALSE)

train_x <- x[-test_index,]
train_y <- y[-test_index] %>% to_categorical(26)

test_x <- x[test_index,]
test_y <- y[test_index] %>% to_categorical(26)
```

After splitting the data, we observed that there were 297959 rows in the training set and 74492 rows in the test set.

Sequential Model

The model that was used for classification was a sequential model. The following block of code generates the model.

```
# Generate the model
model <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(ncol(x))) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 26, activation = "softmax")

# View the model
summary(model)
```

```
## Model: "sequential"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense (Dense)                (None, 512)               205312
## _________________________________________________________________________
## dropout (Dropout)            (None, 512)               0
## _________________________________________________________________________
## dense_1 (Dense)              (None, 256)               131328
## _________________________________________________________________________
## dropout_1 (Dropout)          (None, 256)               0
## _________________________________________________________________________
## dense_2 (Dense)              (None, 26)                6682
## =========================================================================
## Total params: 343,322
## Trainable params: 343,322
## Non-trainable params: 0
## _________________________________________________________________________
```

Since there were 26 categorical outcomes, categorical cross-entropy was as our loss function. We also used accuracy to measure correctness.

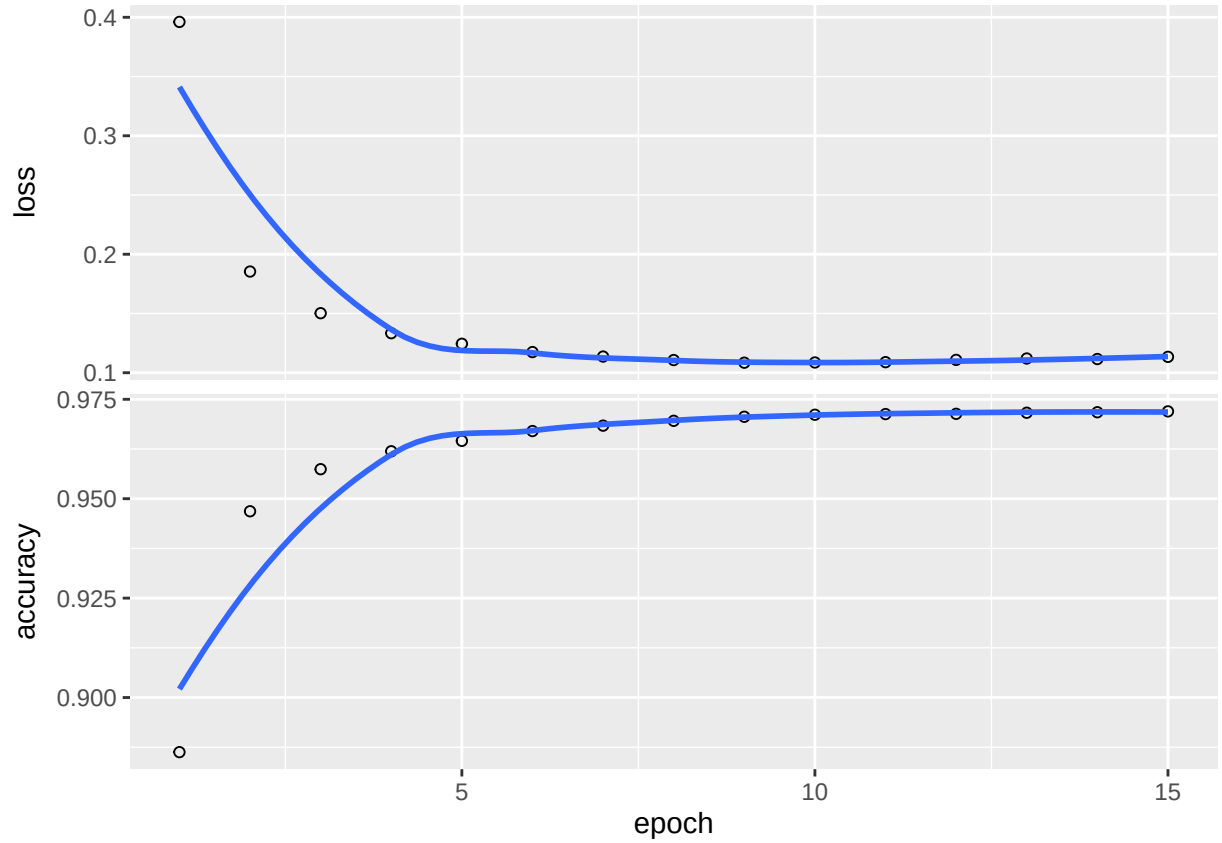
```
# Compile the model
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_rmsprop(),
  metrics = c("accuracy")
)
```

Once the model was set up, we began training our data. Only 15 epochs were used when fitting the model.

```
# Fit the model
fit <- model %>% fit(
  train_x,
  train_y,
  epochs = 15,
  batch_size = 256
)
```

The following plot shows the results from fitting the model.

```
# Plot the results from fitting the model
plot(fit)
```



After evaluating the model, we saw that the model is able to correctly identify the letter over 98% of the time!

```
# Evaluate the accuracy using the test cases
model %>% evaluate(test_x, test_y)
```

```
##       loss   accuracy
## 0.07197014 0.98460239
```

In the following section, we saved the predictions in order to further analyze the results.

```
# Predict the results
predictions <- model %>% predict_classes(test_x)
```

Results

The model was able to achieve an accuracy over 98%, as shown below.

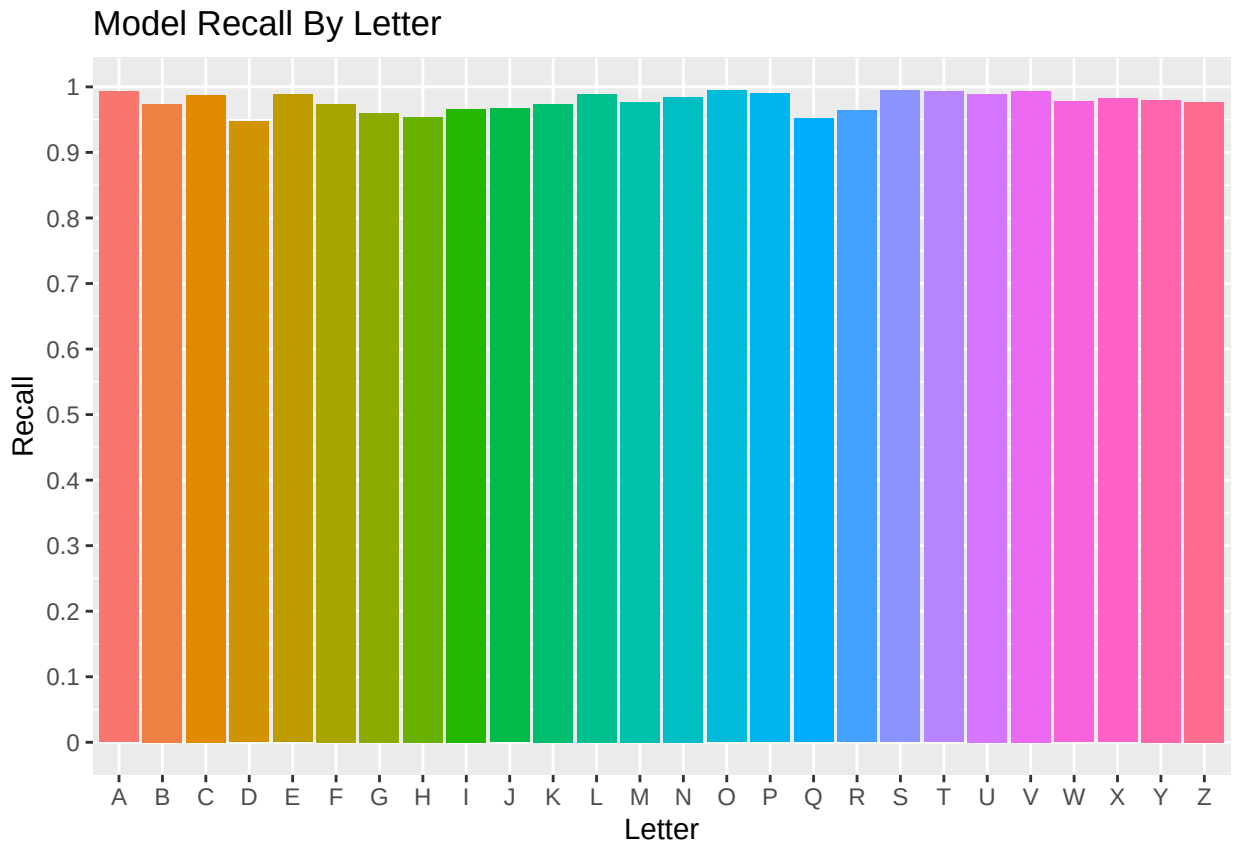
```r
# Combine the observations and the predictions into one table
results <- tibble(Prediction = predictions, Observation = y[test_index])

# Compute the accuracy
mean(results$Prediction == results$Observation)
```

```
## [1] 0.9846024
```

When anaylzing the recalls for each letter, we managed to achieve over 90% for all the letters. Most of the letters were correctly identified 95% of the time, however some letters were correctly predicted over 99% of the time!

```
## # A tibble: 26 x 4
##    Observation Letter Count Recall
##          <dbl> <chr>  <int>  <dbl>
##  1           0 A       2754  0.993
##  2           1 B       1763  0.974
##  3           2 C       4543  0.988
##  4           3 D       2017  0.947
##  5           4 E       2344  0.988
##  6           5 F        229  0.974
##  7           6 G       1227  0.960
##  8           7 H       1469  0.954
##  9           8 I        238  0.966
## 10           9 J       1680  0.967
## 11          10 K       1112  0.974
## 12          11 L       2316  0.989
## 13          12 M       2403  0.977
## 14          13 N       3846  0.985
## 15          14 O      11587  0.995
## 16          15 P       3884  0.990
## 17          16 Q       1145  0.952
## 18          17 R       2324  0.964
## 19          18 S       9675  0.994
## 20          19 T       4400  0.993
## 21          20 U       5770  0.989
## 22          21 V        829  0.994
## 23          22 W       2235  0.978
## 24          23 X       1346  0.982
## 25          24 Y       2166  0.980
## 26          25 Z       1190  0.977
```

The following graph visualizes the table directly above.

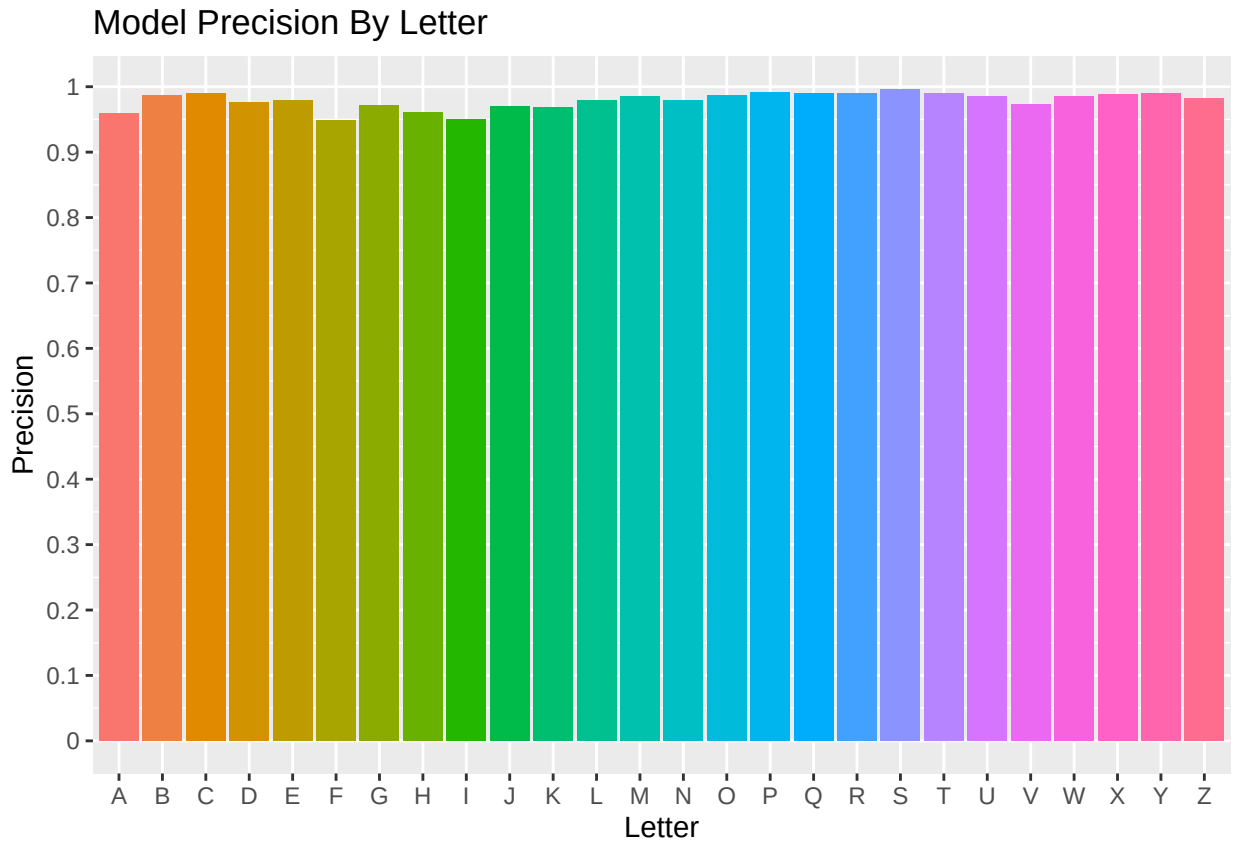


When analyzing the precision of each letter, we observed that all of them were also well over 90%. Most of them were over 97%!

```
## # A tibble: 26 x 4
##    Prediction Letter Count Precision
##         <dbl> <chr>  <int>     <dbl>
##  1          0 A       2848     0.960
##  2          1 B       1739     0.987
##  3          2 C       4537     0.989
##  4          3 D       1957     0.976
##  5          4 E       2365     0.980
##  6          5 F        235     0.949
##  7          6 G       1213     0.971
##  8          7 H       1459     0.961
##  9          8 I        242     0.950
## 10          9 J       1674     0.971
## 11         10 K       1118     0.969
## 12         11 L       2339     0.979
## 13         12 M       2383     0.985
## 14         13 N       3869     0.979
## 15         14 O      11685     0.986
## 16         15 P       3879     0.991
## 17         16 Q       1102     0.989
## 18         17 R       2264     0.989
## 19         18 S       9651     0.997
```

```
## 20          19 T        4416      0.989
## 21          20 U        5787      0.986
## 22          21 V         847      0.973
## 23          22 W        2216      0.986
## 24          23 X        1338      0.988
## 25          24 Y        2145      0.990
## 26          25 Z        1184      0.982
```

The following graph visualizes the table directly above.



Conclusion

Using the sequential model and binarization, we were able to correctly identify over 98% of the letters. The accuracies by letter were high, yet fairly balanced. None of the letters had a recall rate or precision rate under 90%.

A limitation that occurred was the varying prevalences of the letters. Some letters occurred significantly more than others, giving the model more data for that letter than others. Meanwhile, other letters didn't occur as often, meaning the model had a limited amount of data for that letter. Despite this, the model performed fairly well.