CS 4200 Class Project Report and Test Runs

CS 4200.01-1 Artificial Intelligence

By Nicholas Magtangob

Due : 5 / 9 / 25

Introduction
For the class project (CS 4200), we are tasked to create our own implementation of AI in any sort of project, preferably one that has us create and train a model.

For my project, I decided to go with a TensorFlow Keras Convolutional Neural Network, which would be used to read in any 28x28 pixel black and white (1 color channel) image, and then give a prediction as to what uppercase letter of the English Alphabet that it may be.

Project
The project can effectively be divided into 4 sections.

1. Data Collection
2. Data Processing
3. Build and Compile the Model
4. Allow for User Input

For the first data collection phase, I initially tried to individually draw each image for uppercase letters. This got nowhere very quickly! In reality, I needed at least a few hundred for each uppercase letter which would mean at least 2600 drawings at bare minimum, where I would obviously more realistically need around 1000 per letter. I obviously couldn't practically draw 26000 jpg images, so I looked towards Kaggle to get a dataset.

After a bit of searching, I was able to find a dataset which had about 370000 images of handwritten uppercase letters with a size of 28x28 pixels in black and white:
https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format

Now all that was left was to read in the data.
Because it had so many data points, the original CNN had severe overfitting issues. On top of this, the dataset was extremely imbalanced, as it represented some characters much more than others. To counteract both of these issues, undersampling was performed on the dataset to only represent each character an equal amount of times. This not only cut the dataset's size down, but also ensured that the data was balanced. In the end, the balanced dataset had around 29000 entries.

Next was to create the model. Here, I ensured that there were both convolutional layers and max pooling layers, so as to make sure the model was a convolutional model and would be able to read and classify images as well as possible. With a bit of fiddling and parameterizing, as well a lot of testing, the final model looked as follows:

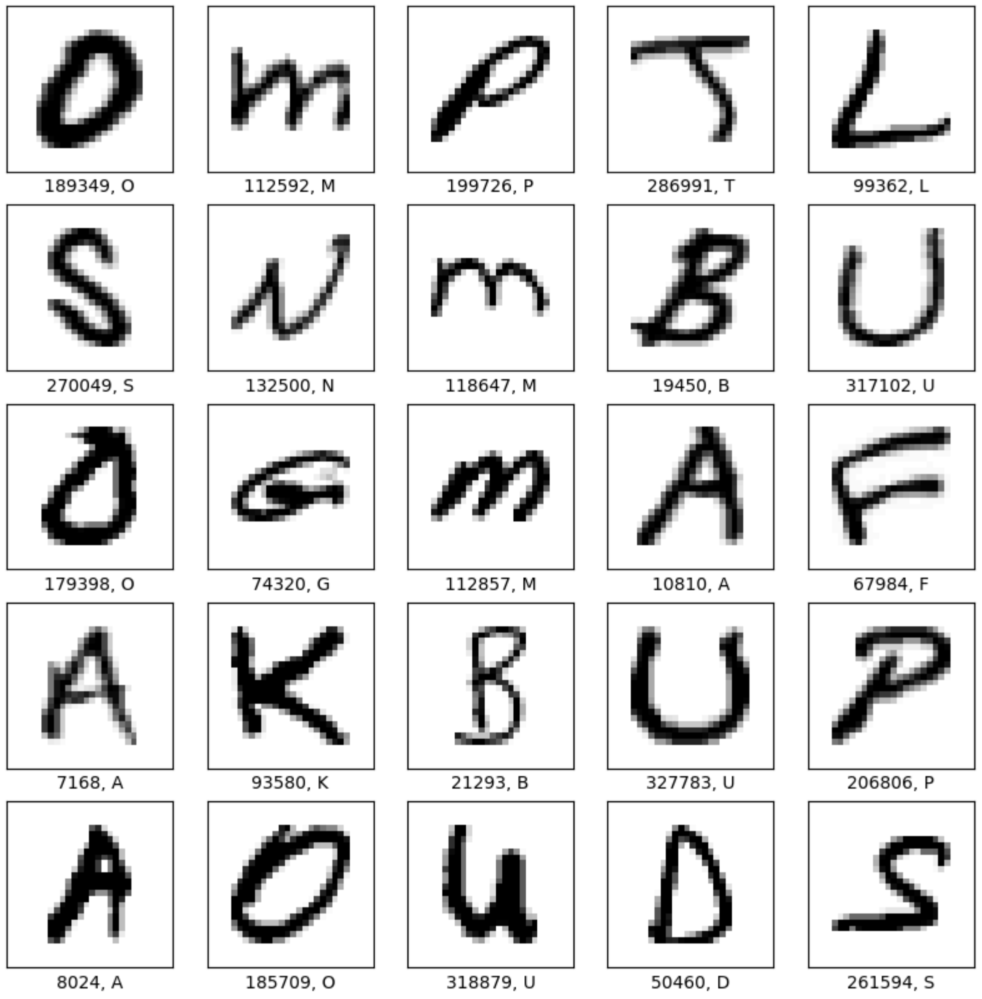
```
model = tf.keras.Sequential([
   tf.keras.layers.Input(shape=(28, 28, 1)),
   tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
   tf.keras.layers.MaxPooling2D((2, 2)),
   tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
   tf.keras.layers.MaxPooling2D((2, 2)),
   tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(64, activation='relu'),
   tf.keras.layers.Dense(26, activation='softmax')
])


model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
             metrics=['accuracy'])
```

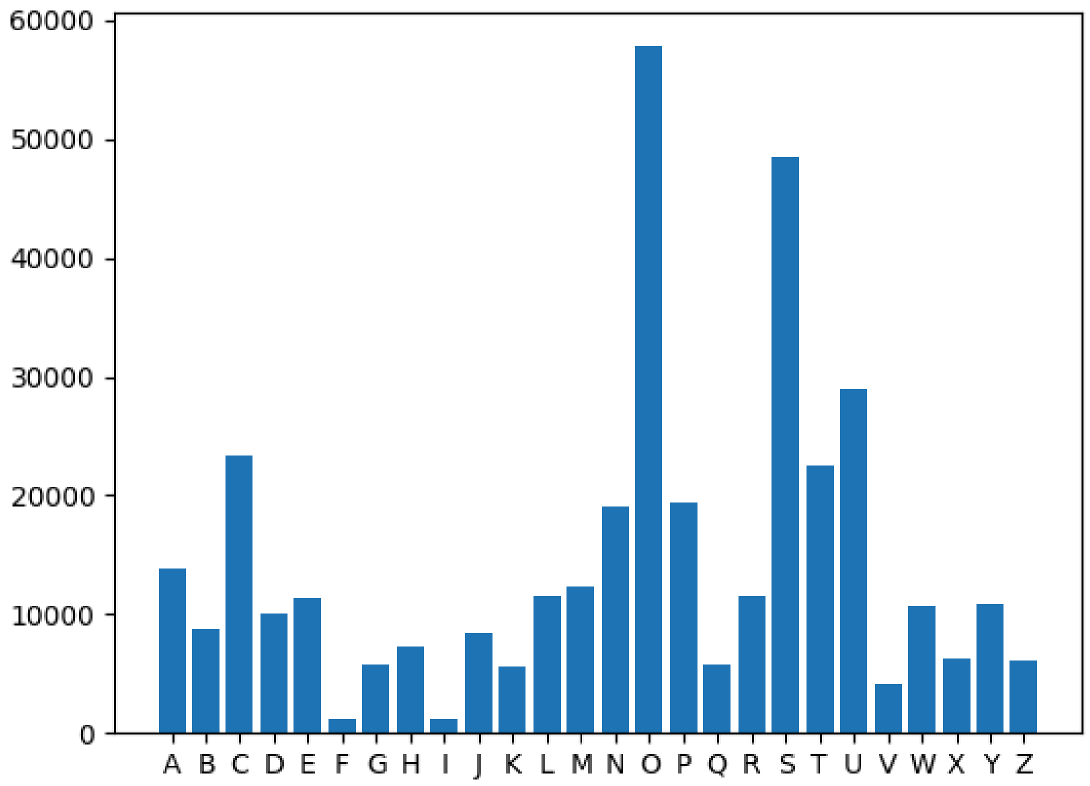
Then, a prediction model was made.
Finally, a simple drawing program for 28x28 images was made on PyGame. The inputs could then be saved and fed to the prediction model, and its outputs printed to the console.

Sample Outputs

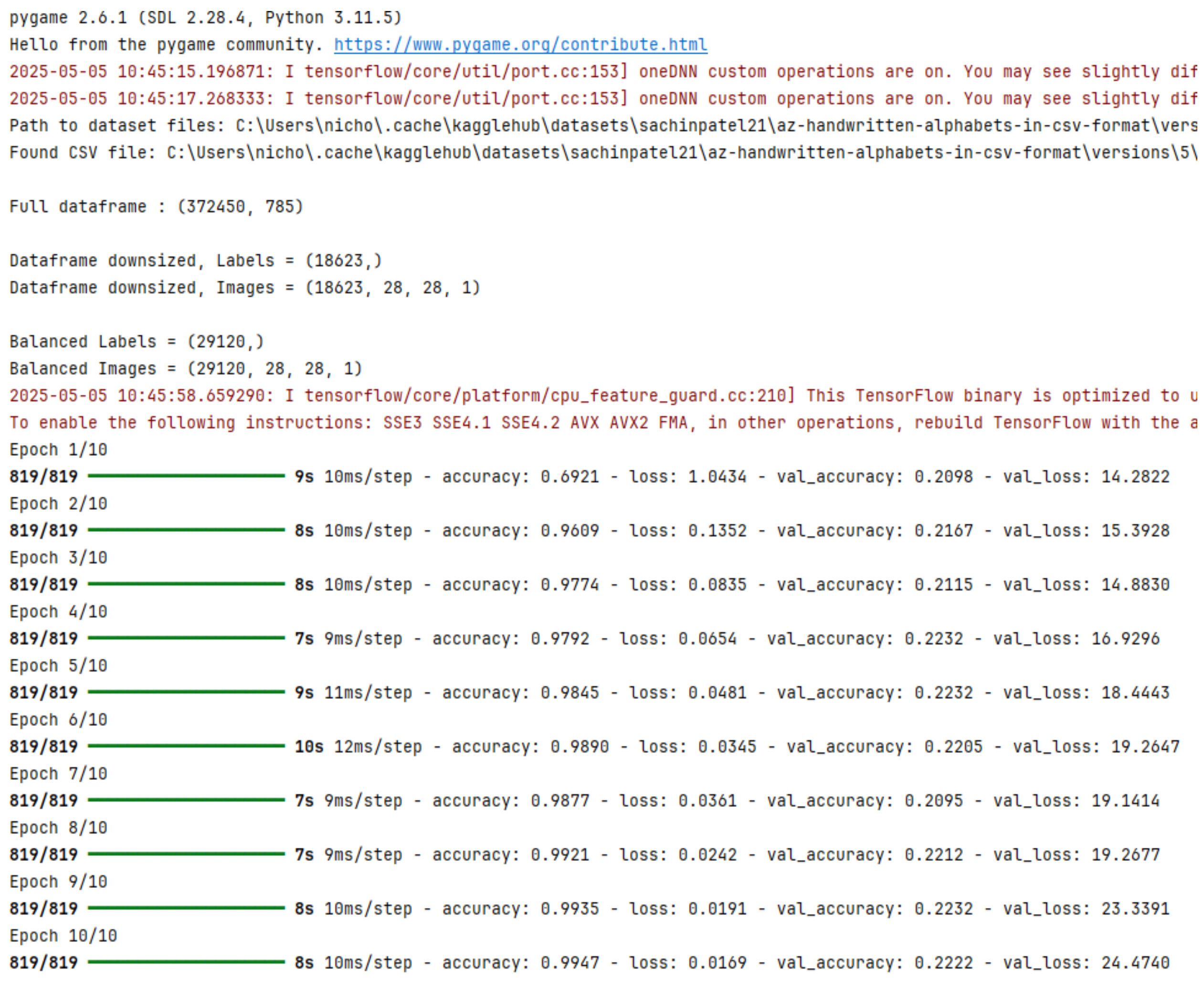
Before the model finishes and compiles, it prints out some information about the dataset, as well as some plots to help visualize some parts of it.

One of the first things it does is print visual images of what the data looks like.
This can help us visualize what the model is learning from, and why it may struggle with some characters as well as excel with others.





Additionally, the program prints a plot of the original distribution of the dataset. As you can see, some characters are severely underrepresented whilst other characters are greatly overrepresented. This led me to conduct a data analysis phase on the original dataset, because the first iteration of the model had extreme overfitting problems.

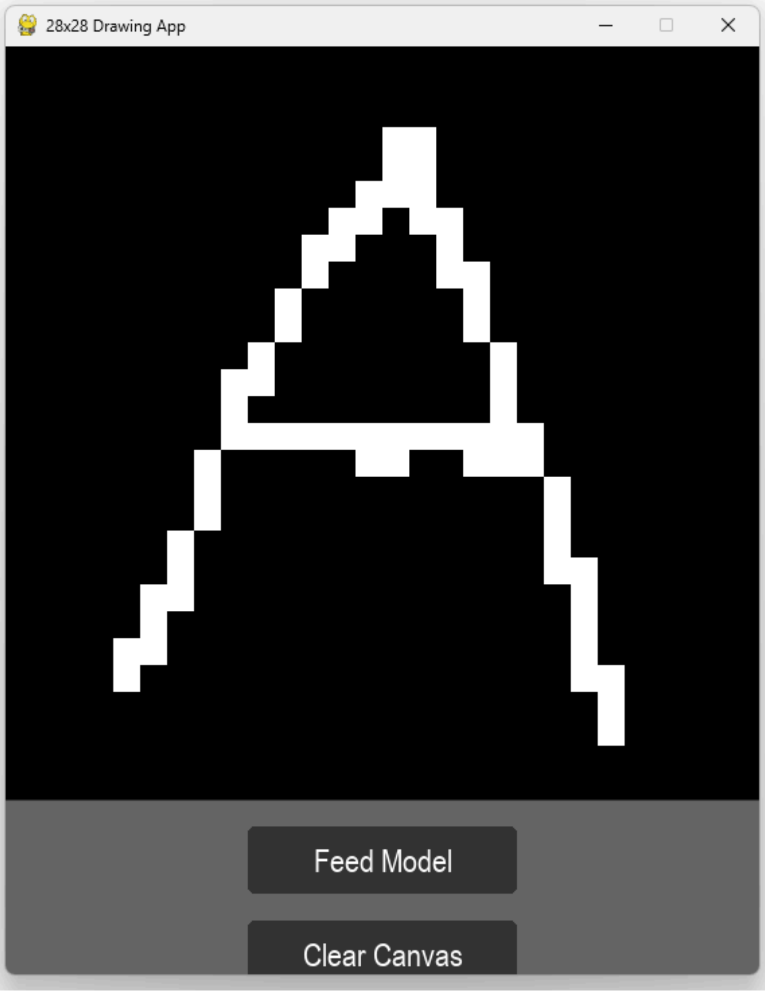


```
pygame 2.6.1 (SDL 2.28.4, Python 3.11.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
2025-05-05 10:45:15.196871: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly dif
2025-05-05 10:45:17.268333: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly dif
Path to dataset files: C:\Users\nicho\.cache\kagglehub\datasets\sachinpatel21\az-handwritten-alphabets-in-csv-format\vers
Found CSV file: C:\Users\nicho\.cache\kagglehub\datasets\sachinpatel21\az-handwritten-alphabets-in-csv-format\versions\5\

Full dataframe : (372450, 785)

Dataframe downsized, Labels = (18623,)
Dataframe downsized, Images = (18623, 28, 28, 1)

Balanced Labels = (29120,)
Balanced Images = (29120, 28, 28, 1)
2025-05-05 10:45:58.659290: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to u
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the a
Epoch 1/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.6921 - loss: 1.0434 - val_accuracy: 0.2098 - val_loss: 14.2822
Epoch 2/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.9609 - loss: 0.1352 - val_accuracy: 0.2167 - val_loss: 15.3928
Epoch 3/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.9774 - loss: 0.0835 - val_accuracy: 0.2115 - val_loss: 14.8830
Epoch 4/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.9792 - loss: 0.0654 - val_accuracy: 0.2232 - val_loss: 16.9296
Epoch 5/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.9845 - loss: 0.0481 - val_accuracy: 0.2232 - val_loss: 18.4443
Epoch 6/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.9890 - loss: 0.0345 - val_accuracy: 0.2205 - val_loss: 19.2647
Epoch 7/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.9877 - loss: 0.0361 - val_accuracy: 0.2095 - val_loss: 19.1414
Epoch 8/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.9921 - loss: 0.0242 - val_accuracy: 0.2212 - val_loss: 19.2677
Epoch 9/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.9935 - loss: 0.0191 - val_accuracy: 0.2232 - val_loss: 23.3391
Epoch 10/10
819/819 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.9947 - loss: 0.0169 - val_accuracy: 0.2222 - val_loss: 24.4740
```

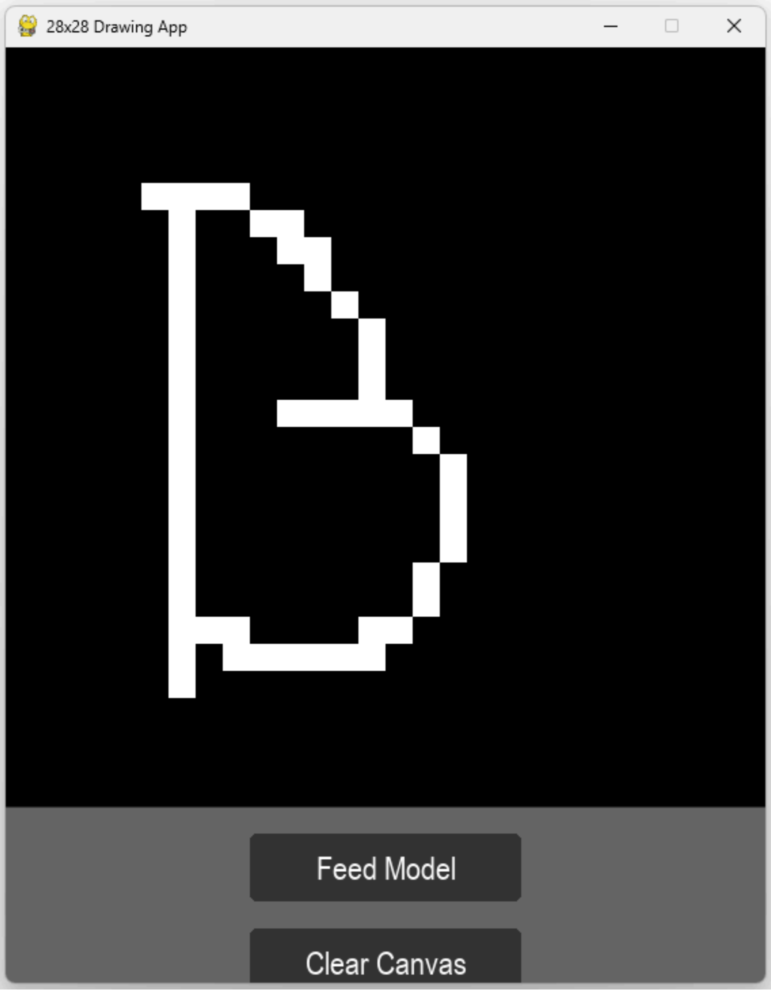
This is all the console output that our program initially prints.
From here, it only prints output when the user feeds the model data and asks for a prediction.

The following images are taken after using the PyGame drawing program, and clicking “Feed Model”
Upon feeding the button to feed the model, a prediction is then printed to the console.

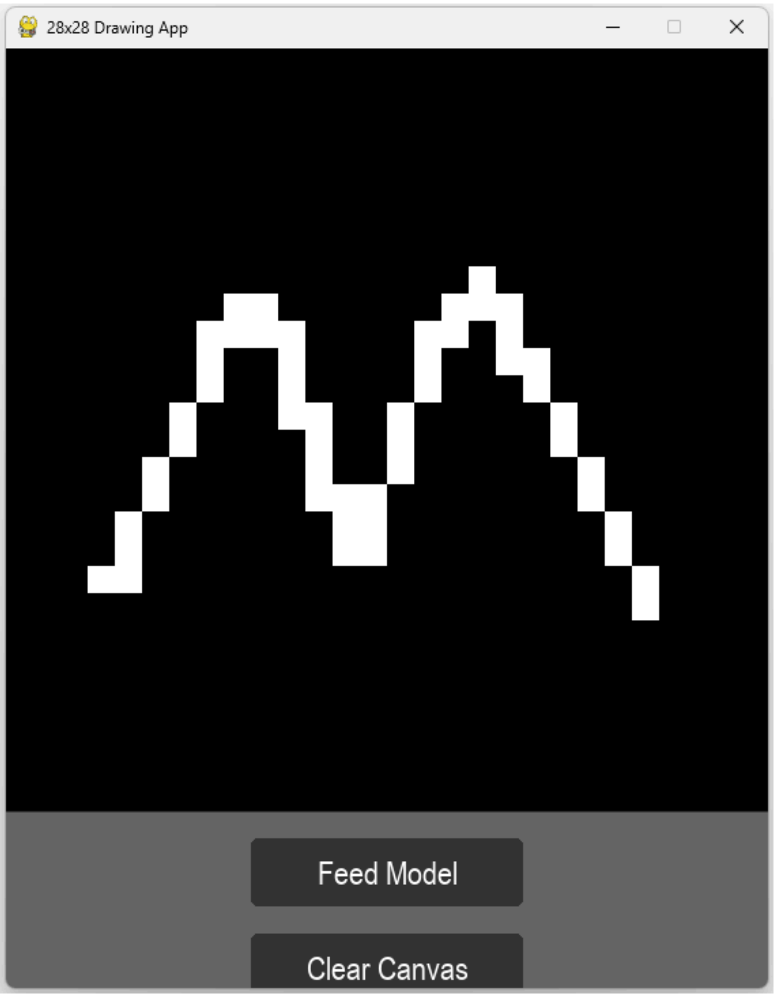
```
Pixel data stored in test_image array:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step

Model predicts image is: "A"
    Confidence: 1.0

A : 100.0%
B : 4.5074916e-10%
C : 6.0741025e-15%
D : 4.3671938e-14%
E : 2.357553e-13%
F : 6.299927e-13%
G : 2.5387846e-11%
H : 4.0624104e-07%
I : 3.0054148e-18%
J : 5.8037147e-15%
K : 1.0352276e-11%
L : 1.4275712e-16%
M : 3.4313576e-09%
N : 2.1635204e-12%
O : 1.1874485e-12%
P : 1.561639e-13%
Q : 4.5634266e-10%
R : 5.8893207e-08%
S : 7.8615256e-14%
T : 7.079299e-12%
U : 1.9761205e-16%
V : 2.9815428e-22%
W : 1.3559833e-14%
X : 5.08403e-17%
Y : 8.3917534e-23%
Z : 1.6342363e-15%
```

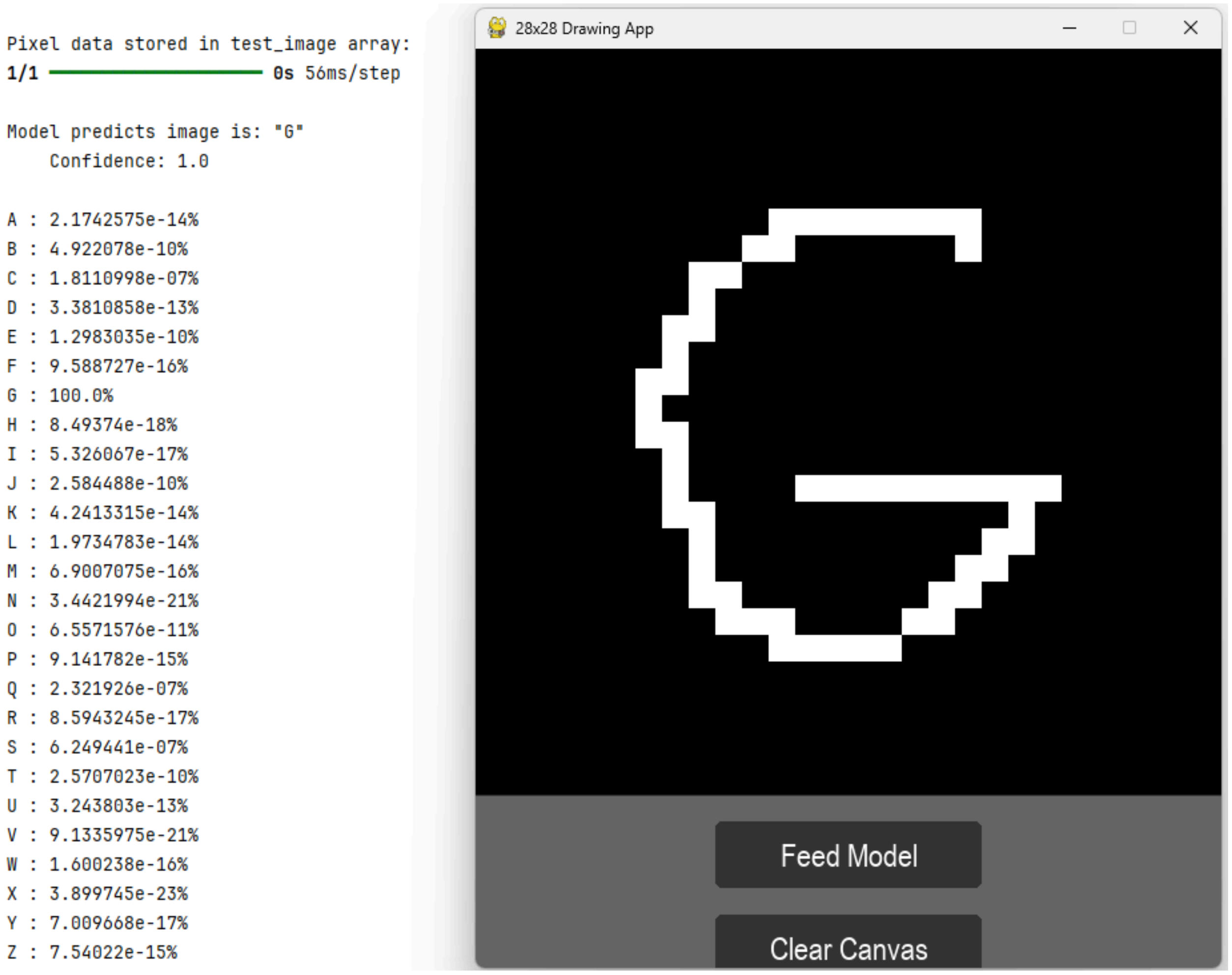


```
Pixel data stored in test_image array:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step

Model predicts image is: "B"
    Confidence: 0.87122065

A : 0.00066119526%
B : 87.12206%
C : 6.4942355e-07%
D : 12.718451%
E : 0.00014316213%
F : 1.0381219e-07%
G : 0.1289328%
H : 2.3300141e-05%
I : 3.230203e-07%
J : 0.0062600253%
K : 3.454425e-05%
L : 1.0866177e-06%
M : 3.460491e-06%
N : 1.5682694e-07%
O : 0.01578954%
P : 4.0973696e-08%
Q : 0.0074551287%
R : 4.5616787e-08%
S : 8.880602e-06%
T : 3.6622896e-07%
U : 0.0001499986%
V : 7.219437e-10%
W : 1.966186e-05%
X : 2.6792324e-09%
Y : 3.7110044e-11%
Z : 4.322483e-10%
```

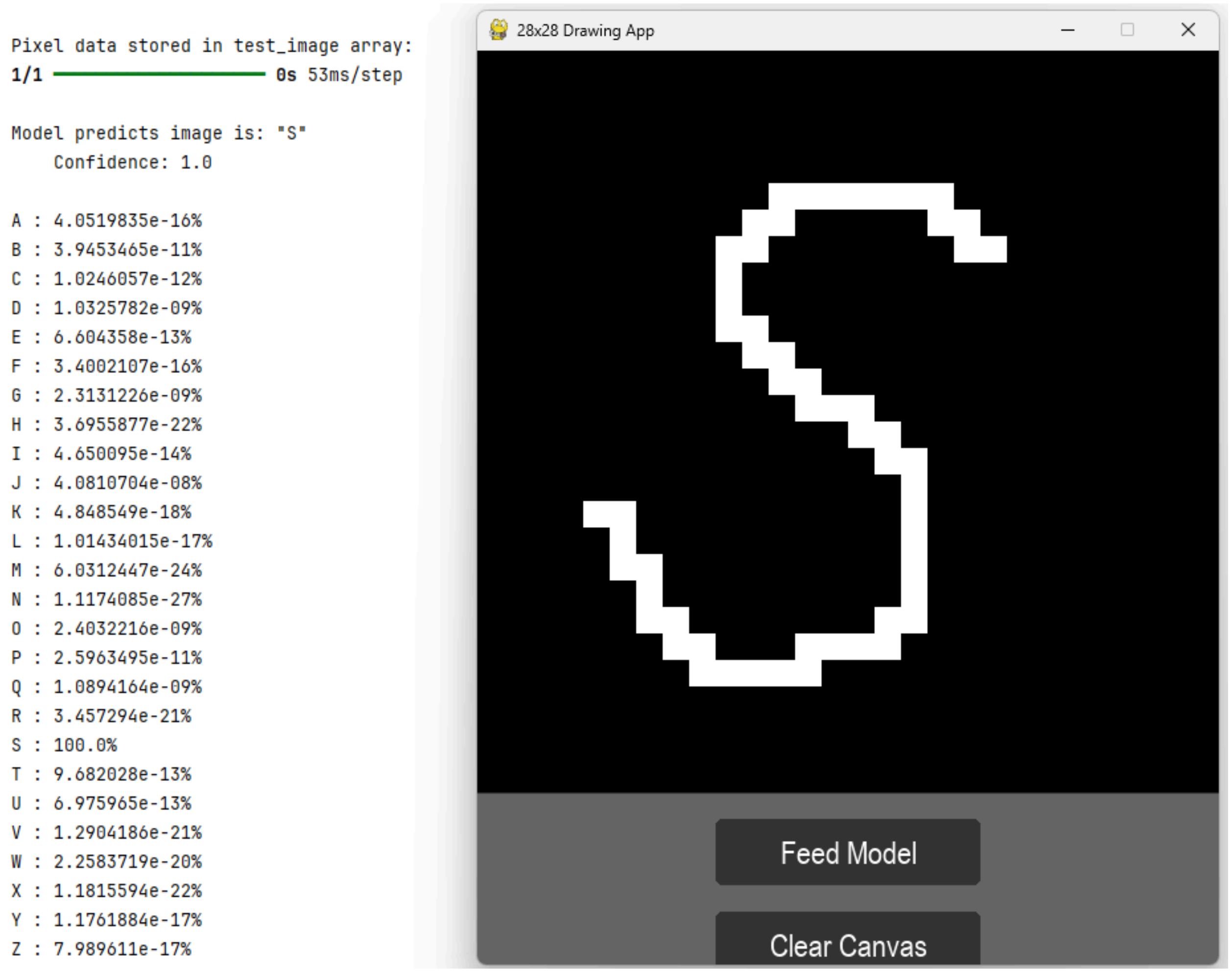


```
Pixel data stored in test_image array:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step

Model predicts image is: "M"
    Confidence: 1.0

A : 5.3750764e-08%
B : 1.2126244e-10%
C : 3.7083569e-13%
D : 1.6023337e-08%
E : 1.1156488e-13%
F : 4.446827e-14%
G : 1.5414566e-13%
H : 2.2296065e-12%
I : 3.2811916e-19%
J : 8.227063e-14%
K : 1.13812355e-10%
L : 7.73932e-14%
M : 100.0%
N : 2.936343e-07%
O : 1.770111e-07%
P : 8.736766e-16%
Q : 2.7827085e-12%
R : 2.2778952e-09%
S : 2.5450265e-17%
T : 1.0993105e-16%
U : 3.6279504e-09%
V : 3.074356e-15%
W : 2.3626907e-08%
X : 7.4907154e-13%
Y : 4.054833e-14%
Z : 3.648327e-15%
```



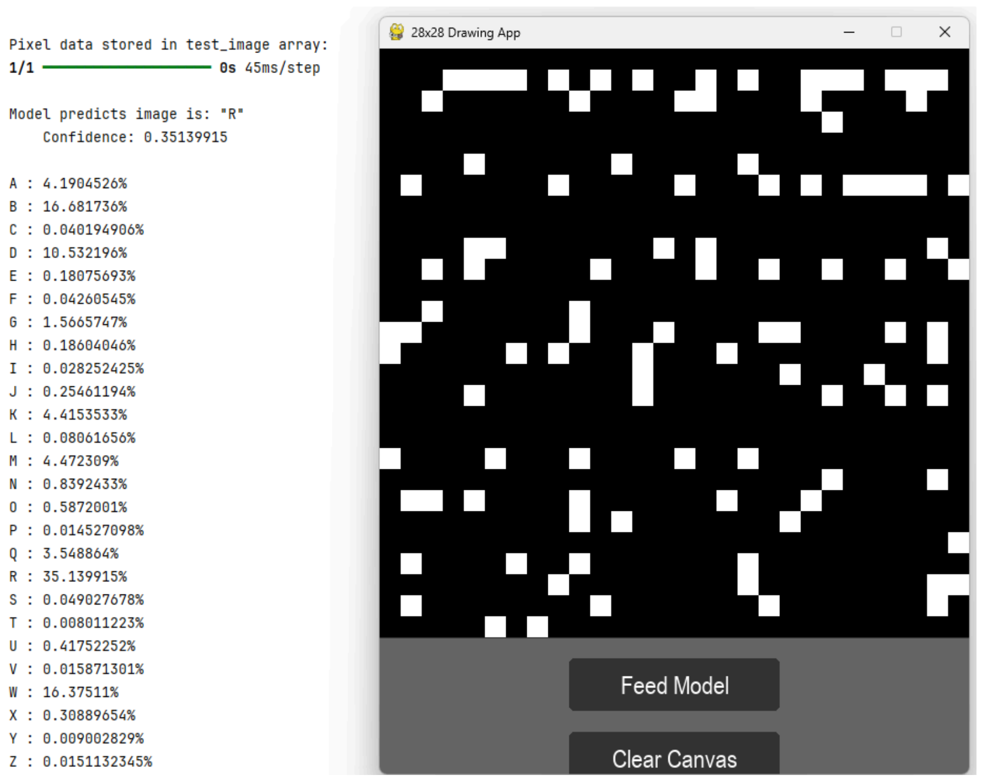


```
Pixel data stored in test_image array:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step

Model predicts image is: "G"
    Confidence: 1.0

A : 2.1742575e-14%
B : 4.922078e-10%
C : 1.8110998e-07%
D : 3.3810858e-13%
E : 1.2983035e-10%
F : 9.588727e-16%
G : 100.0%
H : 8.49374e-18%
I : 5.326067e-17%
J : 2.584488e-10%
K : 4.2413315e-14%
L : 1.9734783e-14%
M : 6.9007075e-16%
N : 3.4421994e-21%
O : 6.5571576e-11%
P : 9.141782e-15%
Q : 2.321926e-07%
R : 8.5943245e-17%
S : 6.249441e-07%
T : 2.5707023e-10%
U : 3.243803e-13%
V : 9.1335975e-21%
W : 1.600238e-16%
X : 3.899745e-23%
Y : 7.009668e-17%
Z : 7.54022e-15%
```

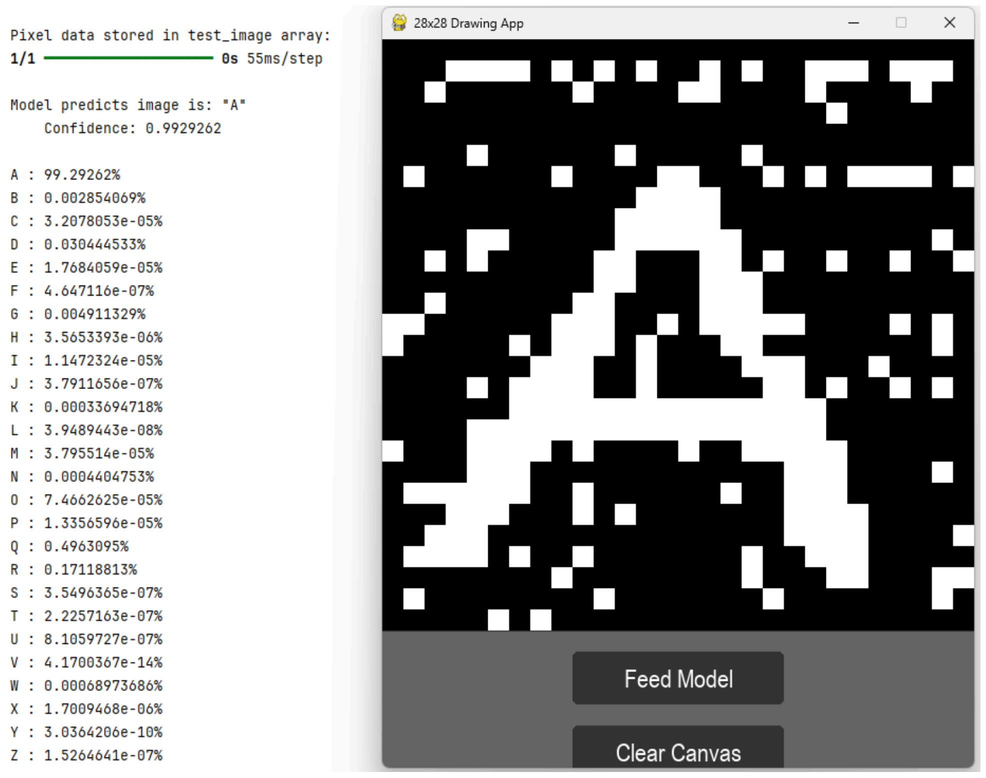


```
Pixel data stored in test_image array:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step

Model predicts image is: "S"
    Confidence: 1.0

A : 4.0519835e-16%
B : 3.9453465e-11%
C : 1.0246057e-12%
D : 1.0325782e-09%
E : 6.604358e-13%
F : 3.4002107e-16%
G : 2.3131226e-09%
H : 3.6955877e-22%
I : 4.650095e-14%
J : 4.0810704e-08%
K : 4.848549e-18%
L : 1.01434015e-17%
M : 6.0312447e-24%
N : 1.1174085e-27%
O : 2.4032216e-09%
P : 2.5963495e-11%
Q : 1.0894164e-09%
R : 3.457294e-21%
S : 100.0%
T : 9.682028e-13%
U : 6.975965e-13%
V : 1.2904186e-21%
W : 2.2583719e-20%
X : 1.1815594e-22%
Y : 1.1761884e-17%
Z : 7.989611e-17%
```

As one can see from all of these test cases, the model is often extremely accurate. It actually lists a 100% confidence value, because all of the other values are so low. Of course, many of these drawn characters are drawn carefully and more neatly. This means the model isn't being challenged so much. Therefore, we explore a challenging case in the following two images:



```
Pixel data stored in test_image array:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step

Model predicts image is: "R"
    Confidence: 0.35139915

A : 4.1904526%
B : 16.681736%
C : 0.040194906%
D : 10.532196%
E : 0.18075693%
F : 0.04260545%
G : 1.5665747%
H : 0.18604046%
I : 0.028252425%
J : 0.25461194%
K : 4.4153533%
L : 0.08061656%
M : 4.472309%
N : 0.8392433%
O : 0.5872001%
P : 0.014527098%
Q : 3.548864%
R : 35.139915%
S : 0.049027678%
T : 0.008011223%
U : 0.41752252%
V : 0.015871301%
W : 16.37511%
X : 0.30889654%
Y : 0.009002829%
Z : 0.0151132345%
```



```
Pixel data stored in test_image array:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step

Model predicts image is: "A"
    Confidence: 0.9929262

A : 99.29262%
B : 0.002854069%
C : 3.2078053e-05%
D : 0.030444533%
E : 1.7684059e-05%
F : 4.647116e-07%
G : 0.004911329%
H : 3.5653393e-06%
I : 1.1472324e-05%
J : 3.7911656e-07%
K : 0.00033694718%
L : 3.9489443e-08%
M : 3.795514e-05%
N : 0.0004404753%
O : 7.4662625e-05%
P : 1.3356596e-05%
Q : 0.4963095%
R : 0.17118813%
S : 3.5496365e-07%
T : 2.2257163e-07%
U : 8.1059727e-07%
V : 4.1700367e-14%
W : 0.00068973686%
X : 1.7009468e-06%
Y : 3.0364206e-10%
Z : 1.5264641e-07%
```

As one can see, the Convolutional Neural Network looks for characters and features! It manages to ignore the noise and make an accurate prediction!