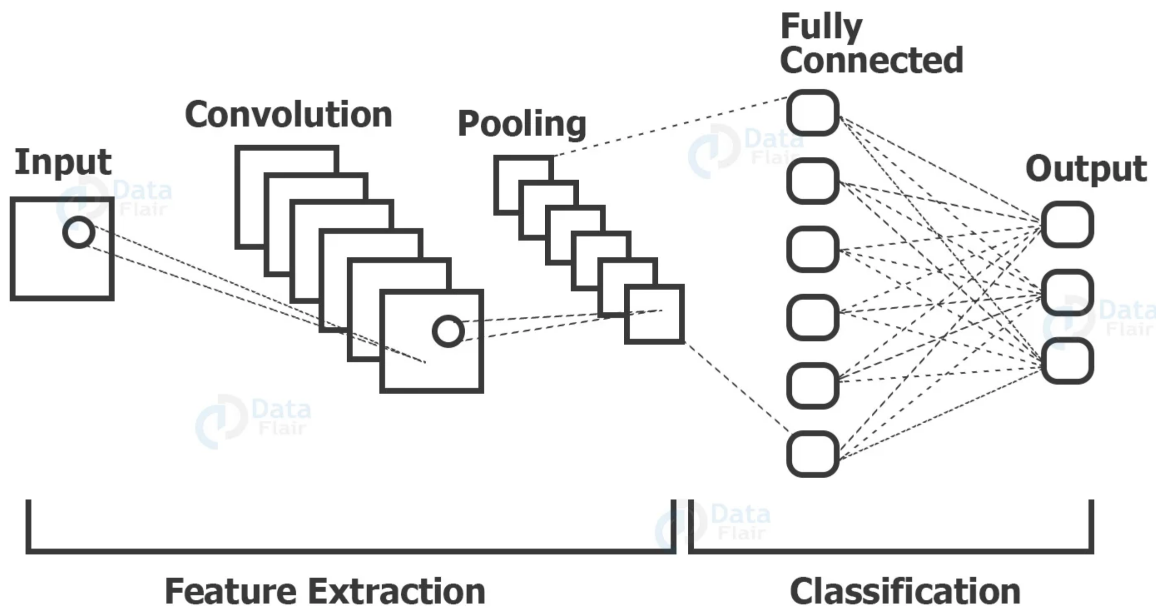


Handwritten Alphabet Recognition

19.04.2023

—

Likhith Raj DV
cse200001039@iiti.ac.in

Abhishek Jaisawal
cse200001001@iiti.ac.in

Overview

- Libraries
- Data Preprocessing
- Model1
- Model2
- Model3
- Image Prediction

Libraries

numpy and pandas

Pandas is used for data preprocessing and data manipulation. Numpy is used for preprocessing a given test image for prediction.

seaborn and matplotlib

Both are used to draw plots and represent statistical data.

imblearn

Used to handle under sampling as in the dataset the number of samples in some classes outweighs the other, thus improving efficiency and performance.

tensorflow and keras

Used for building machine learning models. Keras is also used for building neural network based models.

Importing libraries

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# for handling imbalancing
from imblearn.under_sampling import NearMiss
from keras.utils import np_utils

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report , confusion_matrix

import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormalization

import warnings
warnings.filterwarnings('ignore')
```

Data Preprocessing

Viewing the dataset

Data Exploring

```
[2] df = pd.read_csv('/content/drive/MyDrive/CI/A_Z Handwritten Data.csv')
```

```
[3] df.shape
```

```
(372450, 785)
```

```
df.head()
```

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	...	0.639	0.640	0.641	0.642	0.643	0.644	0.645	0.646	0.647	0.648
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	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

Dataset has 372450 samples of **28 x 28** images.

```
df.tail()
```

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	...	0.639	0.640	0.641	0.642	0.643	0.644	0.645	0.646	0.647	0.6
372445	25	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
372446	25	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
372447	25	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
372448	25	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
372449	25	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	

5 rows × 785 columns

Handling imbalanced data

Data preprocessing

```
[8]  # getting target variable
     y = df['0']
     del df['0']
```

Dealing with imbalanced target

```
[9]  x = y.replace([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25], ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
     'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'])
     x

0         A
1         A
2         A
3         A
4         A
         ..
372445    Z
372446    Z
372447    Z
372448    Z
372449    Z
Name: 0, Length: 372450, dtype: object
```

```
[14] nM = NearMiss()
     X_data, y_data = nM.fit_resample(df, y)
```

Replacing the integers with respective alphabets.

Here, the NearMiss algorithm is used to handle undersampling.

Normalization

Normalizing the values of images between 0 and 1 by dividing with 255. This has improved the performance of the model . Sometimes normalization leads to improved convergence rate of the machine learning algorithm during the training process.

Normalization

```
[18] X_data = X_data / 255
     X_data
```

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.10	...	0.639	0.640	0.641	0.642	0.643	0.644	0.645	0.646	0.647	0.648
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	0.0	0.0	0.0	0.0
1	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	0.0	0.0	0.0	0.0
2	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	0.0	0.0	0.0	0.0
3	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	0.0	0.0	0.0	0.0
4	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	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29115	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	0.0	0.0	0.0	0.0
29116	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	0.0	0.0	0.0	0.0
29117	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	0.0	0.0	0.0	0.0
29118	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	0.0	0.0	0.0	0.0
29119	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	0.0	0.0	0.0	0.0

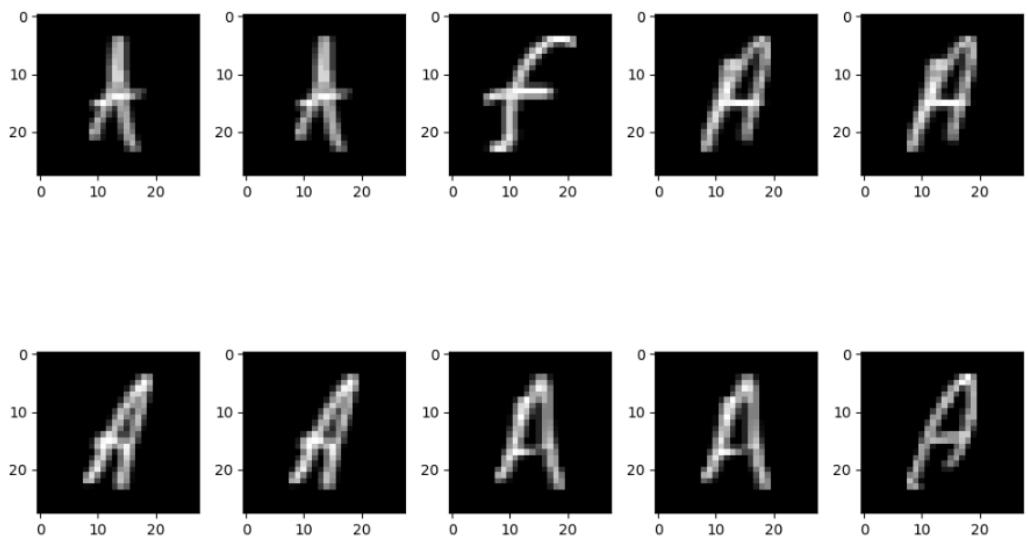
29120 rows × 784 columns

Visualizing data

Visualization

```
X_data = np.array(X_data)
X_data = X_data.reshape(-1,28,28,1)
# Showing few images

f, ax = plt.subplots(2,5)
f.set_size_inches(10,10)
k = 0
for i in range(2):
    for j in range(5):
        ax[i,j].imshow(X_data[k].reshape(28,28), cmap='gray')
        k += 1
    plt.tight_layout()
```



Splitting into Test and Train data.

Here, we are spitting the given dataset into train and test data by splitting it in the ratio of 80% : 20%.

Train test split

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X_data, y, test_size=0.2 ,random_state=102)
```

```
[ ] X_train.shape, X_test.shape, y_train.shape, y_test.shape

    ((23296, 28, 28, 1), (5824, 28, 28, 1), (23296, 26), (5824, 26))
```

Model1

Architecture and Summary

Model 1

```
#Build an ordinary "Deep Learning" model with CNN and maxpooling by using Keras.
model = Sequential()
model.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
#Choose an optimizer and compile the model.
model.compile(optimizer = Adam(learning_rate = 0.01), loss = 'categorical_crossentropy', metrics = ['accuracy'])
#And print the summary of the model.
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 32)        832

 max_pooling2d (MaxPooling2D  (None, 12, 12, 32)       0
 )

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 128)               589952

 dense_1 (Dense)             (None, 26)                3354

=================================================================
Total params: 594,138
Trainable params: 594,138
Non-trainable params: 0
_________________________________________________________________
None
```

This is the first model we have implemented. It has five layers and a total of 594138 trainable parameters.

Training Model1

```
[ ] history = model.fit(X_train,y_train,epochs=15, batch_size=128, validation_data=(X_test,y_test))

    Epoch 1/15
    182/182 [==============================] - 13s 6ms/step - loss: 0.4713 - accuracy: 0.8688 - val_loss: 0.1716 - val_accura
    Epoch 2/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.1152 - accuracy: 0.9664 - val_loss: 0.1535 - val_accurac
    Epoch 3/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.0731 - accuracy: 0.9778 - val_loss: 0.1231 - val_accurac
    Epoch 4/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.0495 - accuracy: 0.9847 - val_loss: 0.1096 - val_accurac
    Epoch 5/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.0400 - accuracy: 0.9879 - val_loss: 0.1285 - val_accurac
    Epoch 6/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.0403 - accuracy: 0.9883 - val_loss: 0.1109 - val_accurac
    Epoch 7/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.0491 - accuracy: 0.9849 - val_loss: 0.1449 - val_accurac
    Epoch 8/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.0464 - accuracy: 0.9867 - val_loss: 0.1511 - val_accurac
    Epoch 9/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.0340 - accuracy: 0.9910 - val_loss: 0.1843 - val_accurac
    Epoch 10/15
    182/182 [==============================] - 1s 5ms/step - loss: 0.0405 - accuracy: 0.9880 - val_loss: 0.1539 - val_accurac
    Epoch 11/15
    182/182 [==============================] - 1s 6ms/step - loss: 0.0297 - accuracy: 0.9911 - val_loss: 0.1859 - val_accurac
    Epoch 12/15
    182/182 [==============================] - 1s 6ms/step - loss: 0.0362 - accuracy: 0.9903 - val_loss: 0.1874 - val_accurac
    Epoch 13/15
    182/182 [==============================] - 1s 5ms/step - loss: 0.0314 - accuracy: 0.9916 - val_loss: 0.1606 - val_accurac
    Epoch 14/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.0285 - accuracy: 0.9927 - val_loss: 0.2444 - val_accurac
    Epoch 15/15
    182/182 [==============================] - 1s 4ms/step - loss: 0.0388 - accuracy: 0.9907 - val_loss: 0.2313 - val_accurac
```

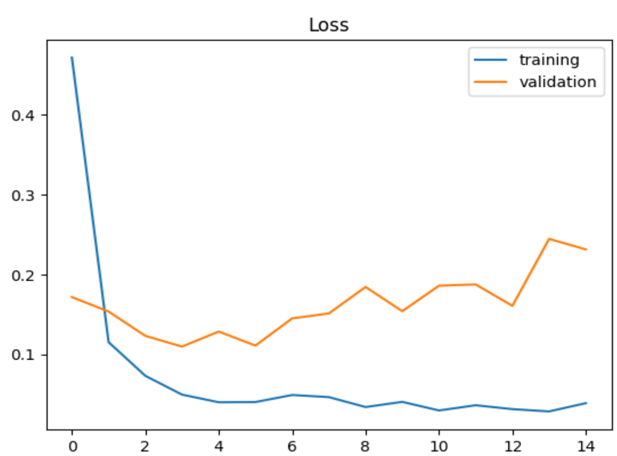
Accuracy = 99%

Loss = 0.0388

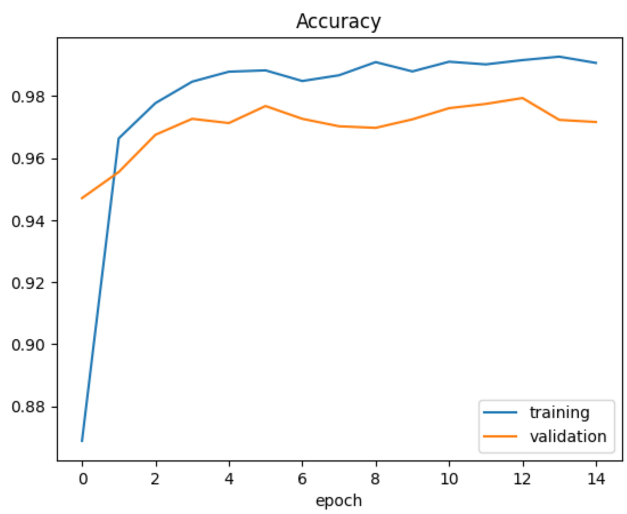
Plots for loss vs epoch

```
[ ] model.save('/content/drive/MyDrive/CI/model1.h5')
```

```
plt.figure(1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training','validation'])
plt.title('Loss')
plt.xlabel('epoch')
plt.figure(2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title('Accuracy')
plt.xlabel('epoch')
plt.show()
```



Plots for accuracy vs epoch



Model2

Architecture and Summary

· Model 2

```
model2 = Sequential()

model2.add(Conv2D(64, (5, 5), input_shape=(28, 28, 1), activation='relu', padding="same"))
model2.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1), activation='relu',padding="same"))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(128, (3, 3), activation='relu', padding="same"))
model2.add(Conv2D(128, (3, 3), activation='relu', padding="same"))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Dropout(0.2))

model2.add(Flatten())
model2.add(Dense(128, activation='relu'))
model2.add(Dense(num_classes, activation='softmax'))

model2.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
print(model2.summary())
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_33 (Conv2D)          (None, 28, 28, 64)        1664

 conv2d_34 (Conv2D)          (None, 28, 28, 32)        51232

 max_pooling2d_17 (MaxPoolin  (None, 14, 14, 32)       0
 g2D)

 conv2d_35 (Conv2D)          (None, 14, 14, 128)       36992

 conv2d_36 (Conv2D)          (None, 14, 14, 128)       147584

 max_pooling2d_18 (MaxPoolin  (None, 7, 7, 128)        0
 g2D)

 dropout_8 (Dropout)         (None, 7, 7, 128)         0

 flatten_9 (Flatten)         (None, 6272)              0

 dense_18 (Dense)            (None, 128)               802944

 dense_19 (Dense)            (None, 26)                3354

=================================================================
Total params: 1,043,770
Trainable params: 1,043,770
Non-trainable params: 0
_________________________________________________________________
None
```

This is the first model we have implemented. It has five layers and a total of 1,043,770 trainable parameters.

Training Model2

```
history = model2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=256, verbose=2)

Epoch 1/10
91/91 - 6s - loss: 0.0201 - accuracy: 0.5848 - val_loss: 0.0067 - val_accuracy: 0.8838 - 6s/epoch - 70ms/step
Epoch 2/10
91/91 - 3s - loss: 0.0056 - accuracy: 0.9001 - val_loss: 0.0043 - val_accuracy: 0.9220 - 3s/epoch - 34ms/step
Epoch 3/10
91/91 - 3s - loss: 0.0041 - accuracy: 0.9247 - val_loss: 0.0035 - val_accuracy: 0.9341 - 3s/epoch - 32ms/step
Epoch 4/10
91/91 - 3s - loss: 0.0033 - accuracy: 0.9381 - val_loss: 0.0032 - val_accuracy: 0.9366 - 3s/epoch - 31ms/step
Epoch 5/10
91/91 - 3s - loss: 0.0029 - accuracy: 0.9431 - val_loss: 0.0030 - val_accuracy: 0.9394 - 3s/epoch - 33ms/step
Epoch 6/10
91/91 - 3s - loss: 0.0026 - accuracy: 0.9472 - val_loss: 0.0029 - val_accuracy: 0.9411 - 3s/epoch - 31ms/step
Epoch 7/10
91/91 - 3s - loss: 0.0024 - accuracy: 0.9507 - val_loss: 0.0027 - val_accuracy: 0.9433 - 3s/epoch - 33ms/step
Epoch 8/10
91/91 - 3s - loss: 0.0023 - accuracy: 0.9513 - val_loss: 0.0028 - val_accuracy: 0.9425 - 3s/epoch - 30ms/step
Epoch 9/10
91/91 - 3s - loss: 0.0023 - accuracy: 0.9513 - val_loss: 0.0027 - val_accuracy: 0.9447 - 3s/epoch - 30ms/step
Epoch 10/10
91/91 - 3s - loss: 0.0021 - accuracy: 0.9539 - val_loss: 0.0028 - val_accuracy: 0.9404 - 3s/epoch - 32ms/step
```

Accuracy = 95%

Loss = 0.95

Plots for loss vs epoch and accuracy vs epoch

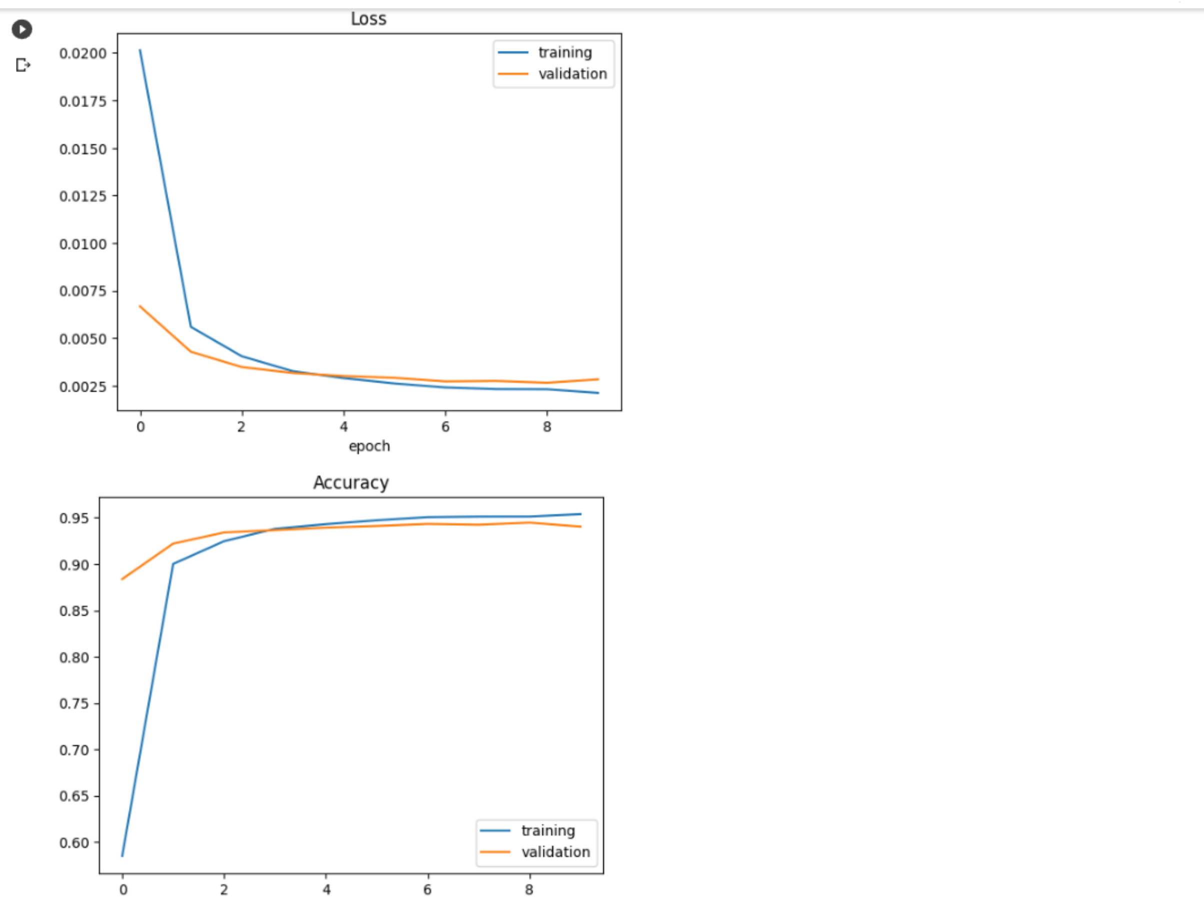
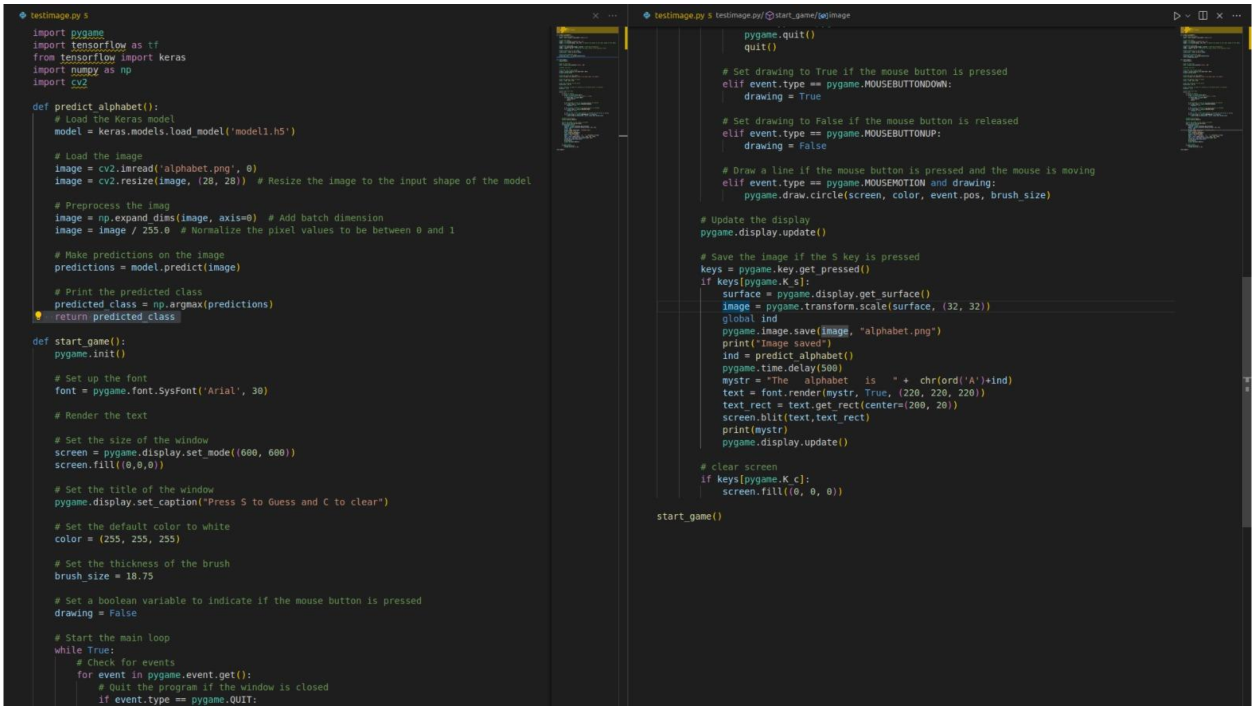


Image Prediction



A pygame based app to take image input and then test the input.


from tensorflow import keras
import numpy as np
import cv2
def predict_alphabet
# Load the Keras
model = keras.mo
# Load the image
image = cv2.imre
image = cv2.resi
out shape of t
# Preprocess the
image = np.expan
image = image /
0 and 1
# Make predictio
predictions = mo
# Print the pred
predicted_class
Press S to Guess and C to clear
The alphabet is X
OUTPUT
04-19 02:01:04.352150
.cc:38] TF-TRT
saved
=================================] - 0s 111ms/step
alphabet is X


Thank You!