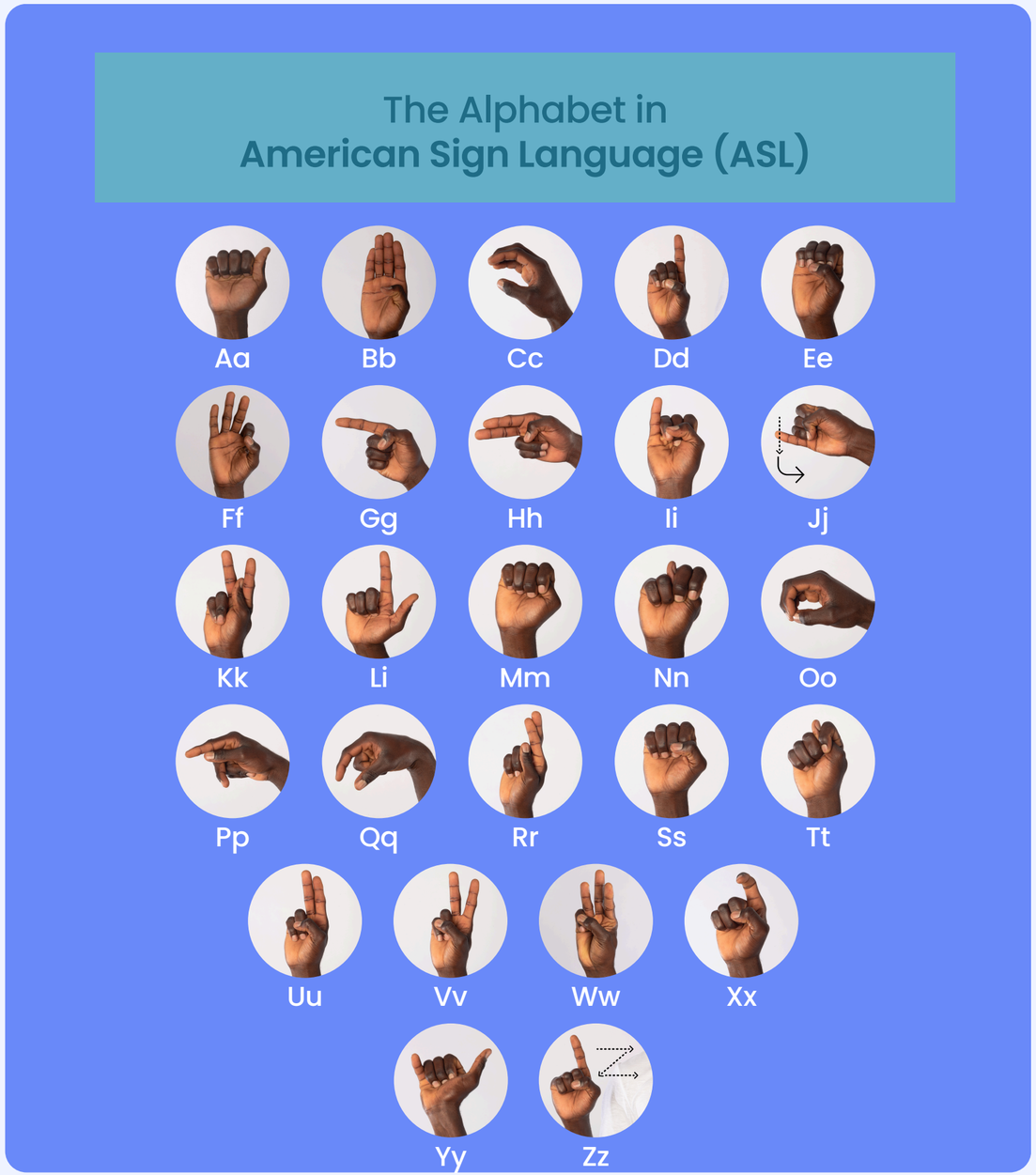
American sign language (ASI)

Under Supervision of
Dr. Ibrahim Zaghloul
En. Mahmoud Mansour



The aim of ASL project

neural network technology to interpret and generate ASL gestures. we strive to empower the deaf and hard-of-hearing community, facilitating seamless interaction and understanding for all.

ASL dataset

01. The data set is a collection of images of alphabets from the American Sign Language, separated in 29 folders which represent the various classes.

02. The training data set contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING.

03.
1-Train :Each class in folder of train contain 2500 image

2-Test :Each class in folder of test contain 300 image

3-Validation :Each class in folder of validation contain 300 image

4-Predict:We have 897 images divided into all Classes

first step: We start to import the library and Took the path of directory and created variables containing this path

01.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
```

02.

```
from tensorflow.keras import models, layers
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Flatten, Dense
import os
import seaborn as sns
import cv2
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score ,recall_score , f1_score,precision_score
```

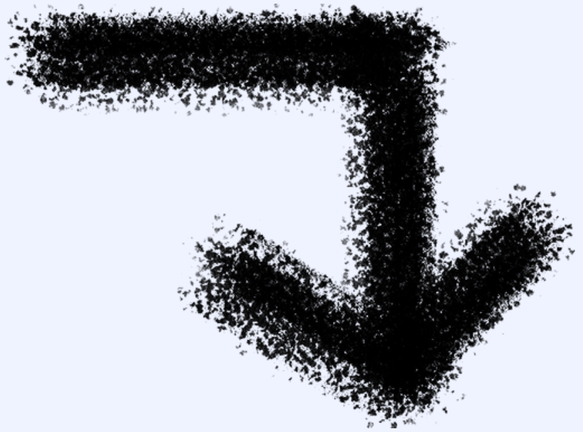
03.

```
train_dir = 'datasets/asl_alphabet/train/'
val_dir = 'datasets/asl_alphabet/val/'
test_dir  = 'datasets/asl_alphabet/test/'
```

we import the ImageDataGenerator to change the data augmentation

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

batch_size = 32
target_size = (32,32) # dataset pic = 200x200

train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True)
val_datagen   = ImageDataGenerator(rescale=1./255, horizontal_flip=True)
test_datagen  = ImageDataGenerator(rescale=1./255, horizontal_flip=True)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=target_size,
        batch_size=batch_size,
        color_mode="grayscale",
        class_mode='categorical',
        shuffle=True)

val_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=target_size,
        batch_size=batch_size,
        color_mode="grayscale",
        class_mode='categorical',
        shuffle=False)

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=target_size,
        batch_size=batch_size,
        color_mode="grayscale",
        class_mode='categorical',
        shuffle=True)
```

the output is



```
Found 71950 images belonging to 29 classes.
Found 8881 images belonging to 29 classes.
Found 8813 images belonging to 29 classes.
```

in this code we Iterate over the generator to collect data

```python
images_list = []
labels_list = []

# Iterate over the generator to collect data
for i in range(len(test_generator)):
    batch_images, batch_labels = test_generator[i]
    images_list.append(batch_images)
    labels_list.append(batch_labels)

# Concatenate the data arrays
images = np.concatenate(images_list)
y_test = np.concatenate(labels_list)

print(y_test)
```

the output is

```
[[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.]]
```

In this code, I will start storing the file names in a variable

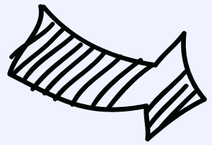
```python
labels = list(train_generator.class_indices.keys())
print(labels)
```

```
['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', 'del', 'nothing', 'sp
ace']
```

This code allows me to display 8 images for 10 classes

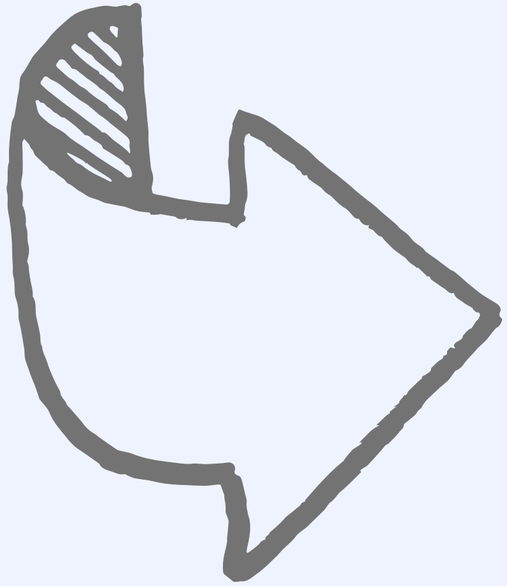
```python
TRAIN_PATH=train_dir
def sample_images(labels):
    plt.figure(figsize=(20, 25))  # Adjusted figsize
    for i, label in enumerate(labels):
        label_path = os.path.join(TRAIN_PATH, label)
        count = 0  # Counter for images of each label
        for img in os.listdir(label_path):
            if count == 8:  # Display only eight images per label
                break
            plt.subplot(10, 8, i*8 + count + 1)  # Adjusted subplot for 10 labels
            img_arr = cv2.imread(os.path.join(label_path, img))
            img_arr = cv2.cvtColor(img_arr, cv2.COLOR_BGR2RGB)
            plt.imshow(img_arr)
            plt.axis('off')
            plt.title(label)
            count += 1  # Increment the counter
    # Title
    plt.suptitle("Sample Images in ASL Alphabet Dataset", x=0.55, y=0.92)
    plt.show()
```

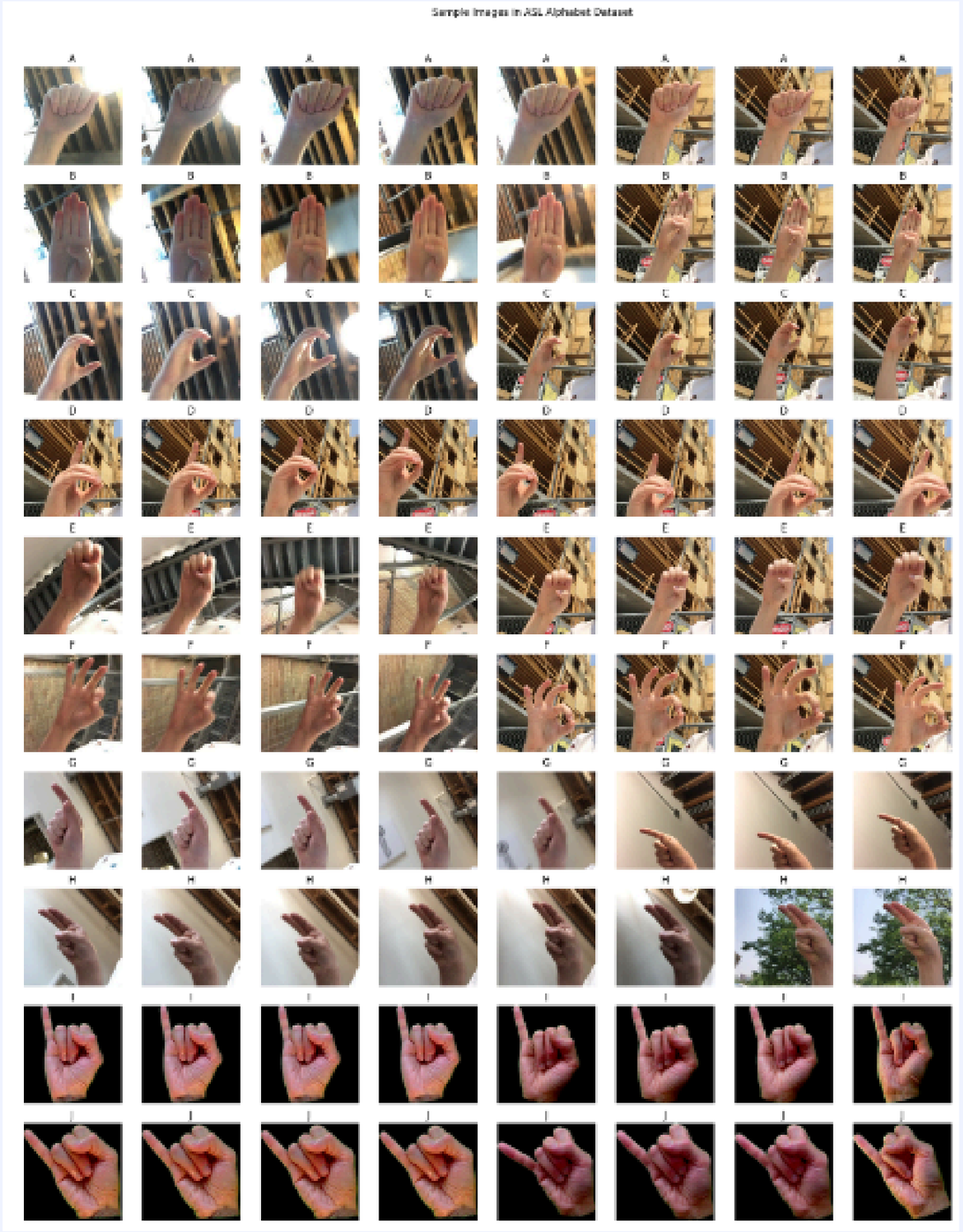
here we print first 10 labels of the image



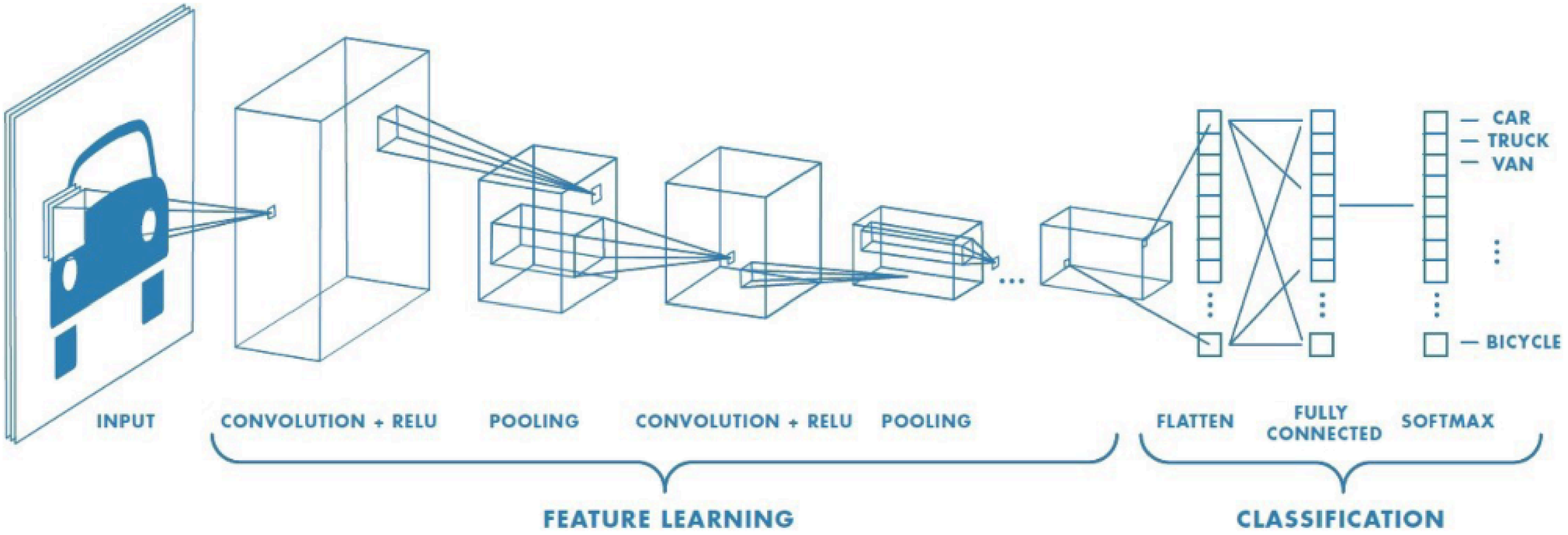
```python
sample_images(labels[:10])
```

The output is:

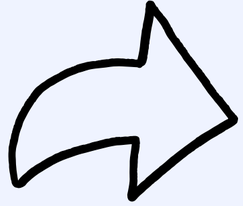




CNN


CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

make variable to take length of label resize shape

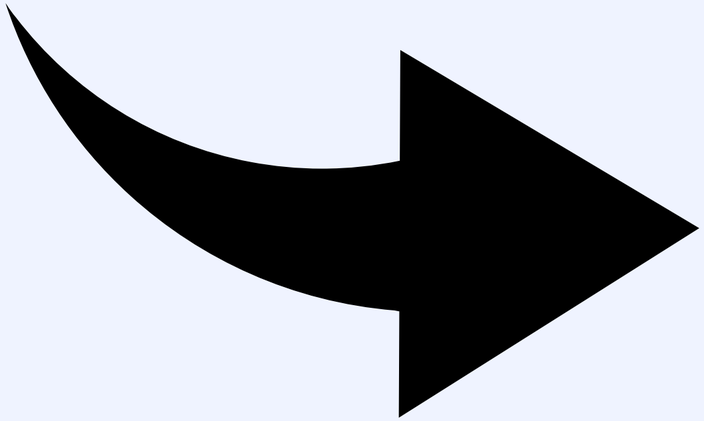


```
num_classes = len(labels)
input_shape = (32,32,1)
```

we start to build model ,We used CNN model with 3 convolution layer and fully-connected layers(contain flatten,Dense,output layer).

```
 Build Model
 model = models.Sequential()
# 1st convolution layer
 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
 model.add(BatchNormalization())
#  2nd convolution layer
 model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'))
 model.add(MaxPooling2D(pool_size=(2, 2)))
 model.add(BatchNormalization())
 # 3rd convolution layer
 model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'))
 model.add(MaxPooling2D(pool_size=(2, 2)))
 model.add(BatchNormalization())
# fully-connected layers
model.add(layers.Flatten())
 model.add(layers.Dense(128, activation='relu'))
 model.add(layers.Dense(num_classes, activation='softmax'))

 model.summary()
```

The output is



Model: "sequential"

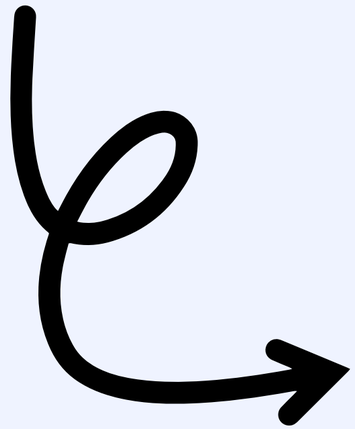
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	320
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
batch_normalization (BatchNormalization)	(None, 16, 16, 32)	128
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 8, 8, 64)	256
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
batch_normalization_2 (BatchNormalization)	(None, 4, 4, 128)	512
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262,272
dense_1 (Dense)	(None, 29)	3,741

Total params: 359,581 (1.37 MB)

Trainable params: 359,133 (1.37 MB)

Non-trainable params: 448 (1.75 KB)

Create ModelCheckpoint callback



```python
from tensorflow.keras.callbacks import ModelCheckpoint

# Define checkpoint path
checkpoint_path = "best_model_final.keras"

# Create ModelCheckpoint callback
checkpoint = ModelCheckpoint(checkpoint_path,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

Start to Compile Model

```python
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

We start to train the model

```
[*]: history = model.fit(train_generator, validation_data=val_generator, epochs=20, callbacks=[checkpoint])
```

```
Epoch 1/20
C:\Users\omar maged\anaconda3\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:120: UserWarning: Your `PyD
all `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do
nts to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
2246/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.5344 - loss: 1.5845
Epoch 1: val_accuracy improved from -inf to 0.79462, saving model to best_mod_final.keras
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 105s 46ms/step - accuracy: 0.5347 - loss: 1.5832 - val_accuracy: 0.7946 - val_loss: 0.6860
Epoch 2/20
2247/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.9220 - loss: 0.2367
Epoch 2: val_accuracy improved from 0.79462 to 0.85520, saving model to best_mod_final.keras
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 81s 36ms/step - accuracy: 0.9221 - loss: 0.2366 - val_accuracy: 0.8552 - val_loss: 0.5142
Epoch 3/20
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.9532 - loss: 0.1385
Epoch 3: val_accuracy improved from 0.85520 to 0.87254, saving model to best_mod_final.keras
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 79s 35ms/step - accuracy: 0.9532 - loss: 0.1385 - val_accuracy: 0.8725 - val_loss: 0.4045
Epoch 4/20
2247/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9679 - loss: 0.0976
Epoch 4: val_accuracy did not improve from 0.87254
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 77s 34ms/step - accuracy: 0.9679 - loss: 0.0976 - val_accuracy: 0.8527 - val_loss: 0.4725
Epoch 5/20
2247/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9753 - loss: 0.0761
Epoch 5: val_accuracy improved from 0.87254 to 0.89033, saving model to best_mod_final.keras
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 77s 34ms/step - accuracy: 0.9753 - loss: 0.0761 - val_accuracy: 0.8903 - val_loss: 0.4009
Epoch 6/20
1610/2249 ━━━━━━━━━━━━━━━━━━━━ 20s 32ms/step - accuracy: 0.9810 - loss: 0.0600
```

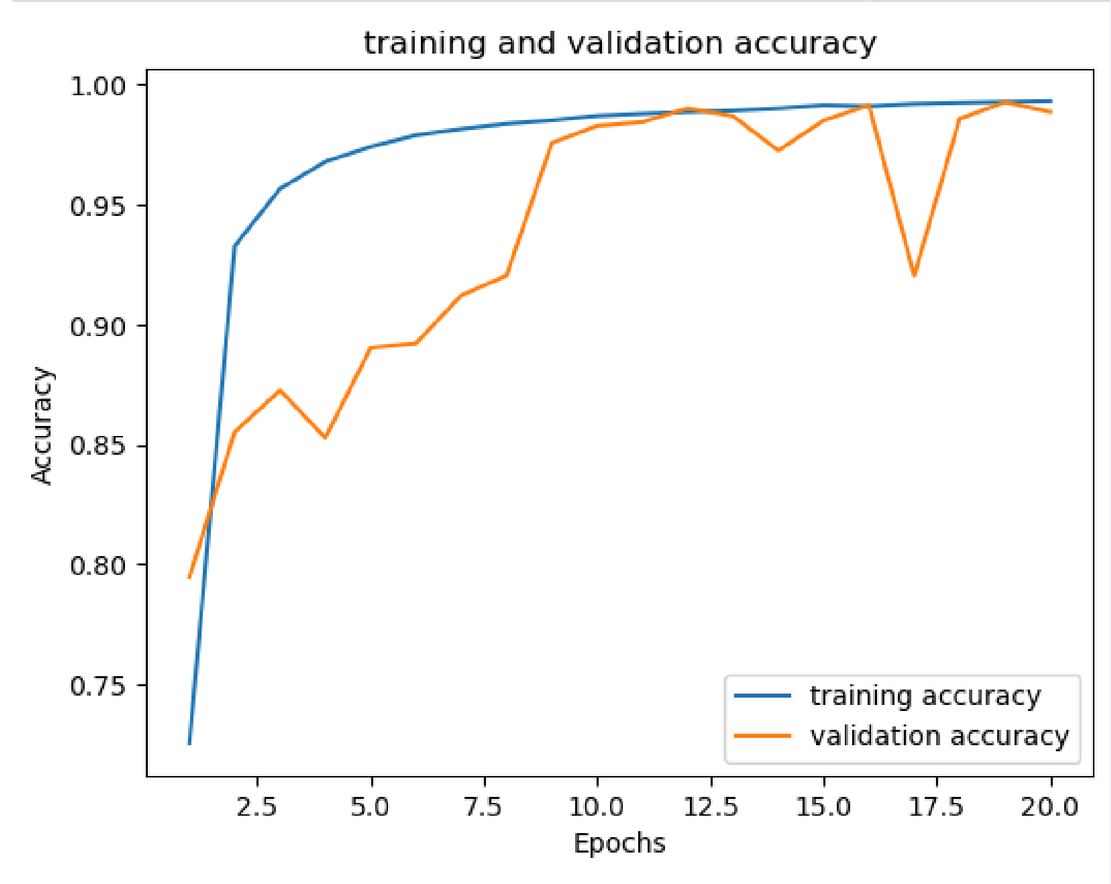
The Final Epoch

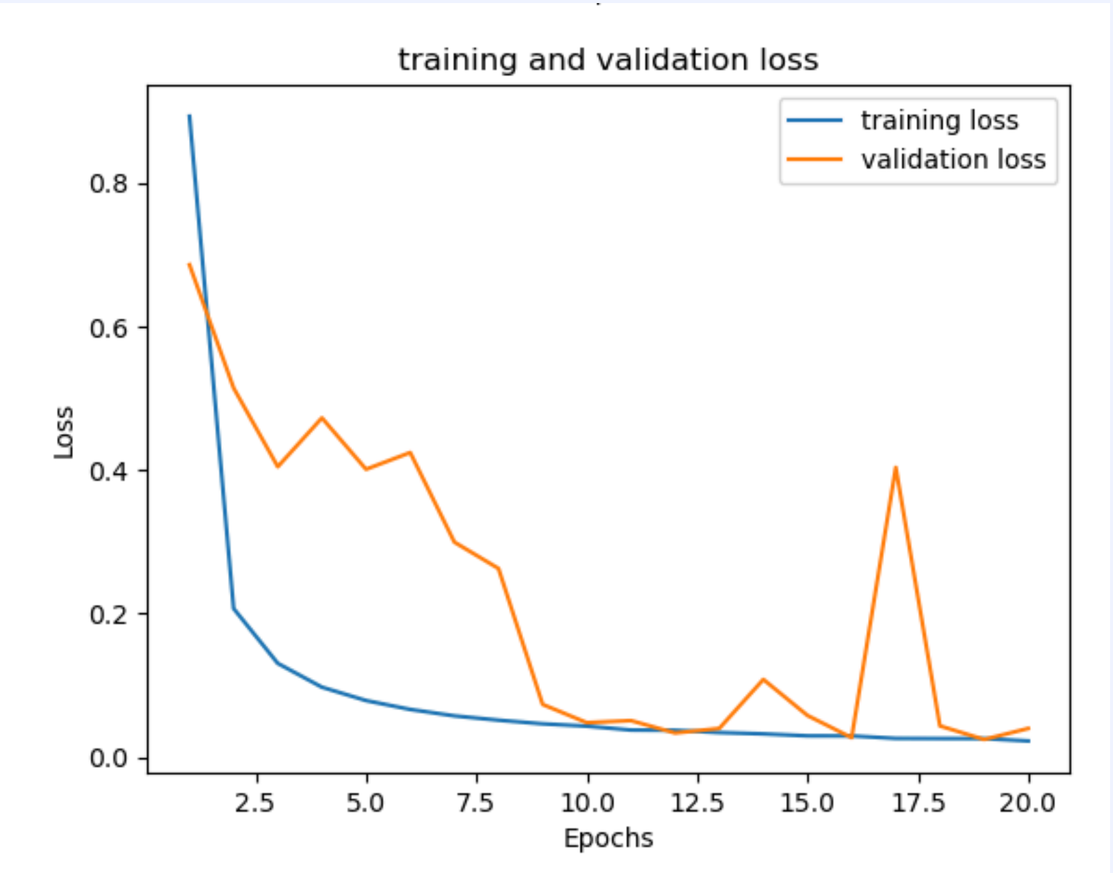
```
Epoch 15/20
2246/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.9926 - loss: 0.0274
Epoch 15: val_accuracy did not improve from 0.99020
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 79s 35ms/step - accuracy: 0.9926 - loss: 0.0274 - val_accuracy: 0.9851 - val_loss: 0.0578
Epoch 16/20
2247/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.9912 - loss: 0.0284
Epoch 16: val_accuracy improved from 0.99020 to 0.99155, saving model to best_mod_final.keras
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 78s 35ms/step - accuracy: 0.9912 - loss: 0.0284 - val_accuracy: 0.9916 - val_loss: 0.0269
Epoch 17/20
2248/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.9922 - loss: 0.0248
Epoch 17: val_accuracy did not improve from 0.99155
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 79s 35ms/step - accuracy: 0.9922 - loss: 0.0248 - val_accuracy: 0.9204 - val_loss: 0.4035
Epoch 18/20
2248/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.9933 - loss: 0.0219
Epoch 18: val_accuracy did not improve from 0.99155
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 79s 35ms/step - accuracy: 0.9933 - loss: 0.0219 - val_accuracy: 0.9858 - val_loss: 0.0433
Epoch 19/20
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9931 - loss: 0.0237
Epoch 19: val_accuracy improved from 0.99155 to 0.99279, saving model to best_mod_final.keras
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 78s 34ms/step - accuracy: 0.9931 - loss: 0.0237 - val_accuracy: 0.9928 - val_loss: 0.0244
Epoch 20/20
2248/2249 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9931 - loss: 0.0219
Epoch 20: val_accuracy did not improve from 0.99279
2249/2249 ━━━━━━━━━━━━━━━━━━━━ 77s 34ms/step - accuracy: 0.9931 - loss: 0.0219 - val_accuracy: 0.9887 - val_loss: 0.0397
```

this code start to train an validation accuracy and validation loss

```
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(1,len(acc)+1)

#train an validation accuracy
plt.plot(epochs,acc,label='training accuracy')
plt.plot(epochs,val_acc,label='validation accuracy')
plt.title('training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.figure()



#train and validation loss
plt.plot(epochs,loss,label='training loss')
plt.plot(epochs,val_loss,label='validation loss')
plt.title('training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

The output is



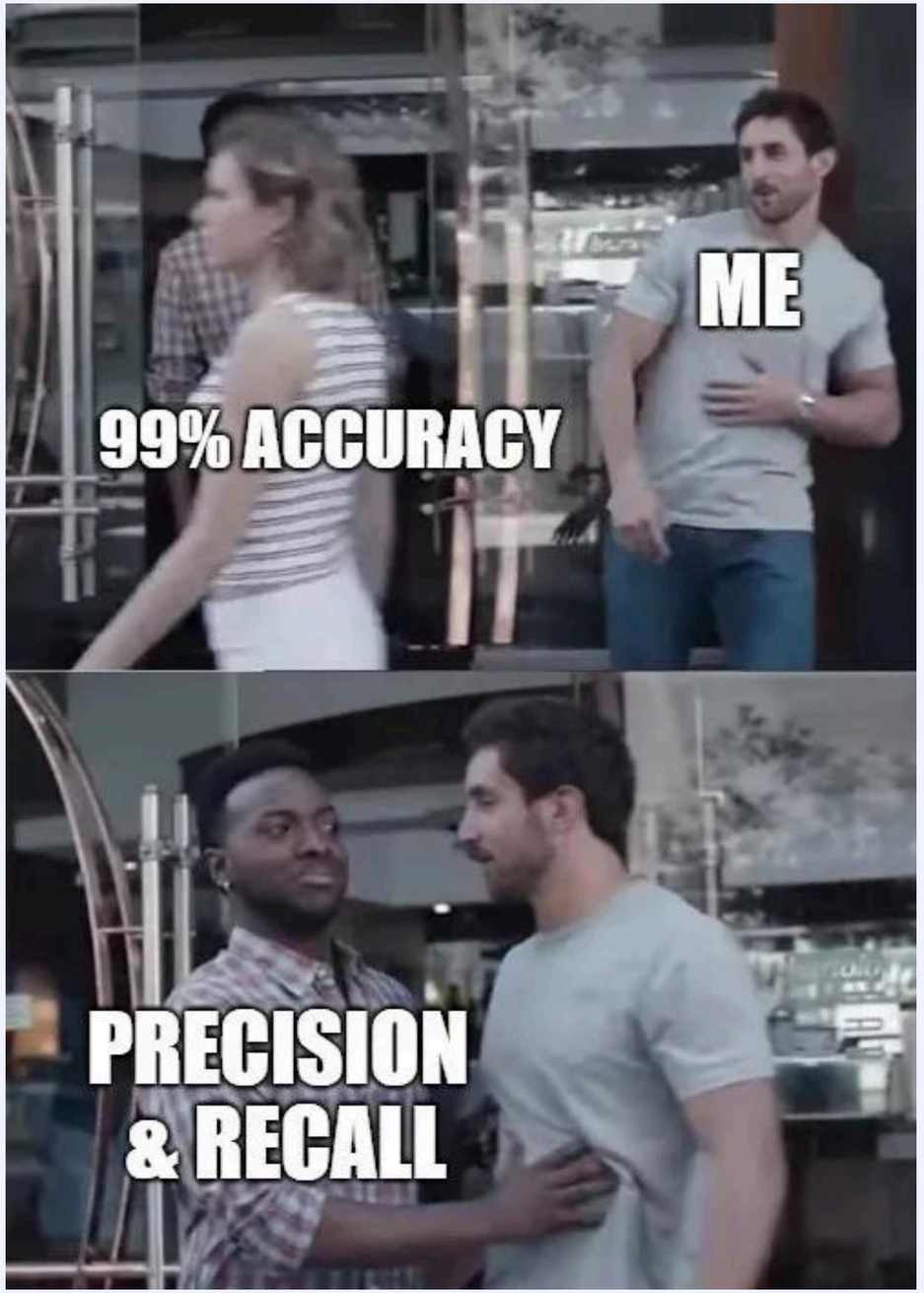


this code tell us the percentage accuracy and loss

```
scores = new_model.evaluate(test_generator)
print('Test loss: ', scores[0])
print('Test accuracy: ', scores[1])
```

```
276/276 ━━━━━━━━━━━━━━━━━━━━ 24s 86ms/step - accuracy: 0.9791 - loss: 0.0871
Test loss:  0.08457118272781372
Test accuracy:  0.9794621467590332
```


99% TRAINING ACCURACY
39% TESTING ACCURACY


99% ACCURACY
ME
PRECISION & RECALL

Create a variable to store the testing generator in

```
import tensorflow
new_model = tensorflow.keras.models.load_model('asl_alphabet_cnn_final_maged.keras')
```

```
output = new_model.predict(test_generator)
pred_lbls = np.argmax(output,axis=1)
```

here we use the raws in confusion Matrix

```
accuracy = accuracy_score(test_generator.classes, pred_lbls) # labels
print("Accuracy = {}".format(accuracy))

# Calculate Precision score
precision=precision_score(test_generator.classes,pred_lbls,average="macro")
print("Precision = {}".format(precision))

recall = recall_score(test_generator.classes, pred_lbls, average="macro")
print("Recall = {}".format(recall))

f1 = f1_score(test_generator.classes, pred_lbls, average="macro")
print("F1 Score = {}".format(f1))
```

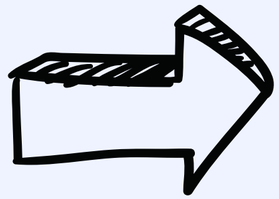
The output is

```
Accuracy = 0.9778706108163034
Precision = 0.9784281935772623
Recall = 0.9779221482247797
F1 Score = 0.9779217889010856
```

Accuracy Law

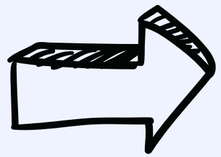
$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}}$$

Percision Law



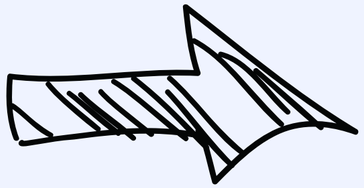
$$\underset{\text{Class A}}{\text{Precision}} = \frac{TP_{\text{Class A}}}{TP_{\text{Class A}} + FP_{\text{Class A}}}$$

Recall Law



$$\underset{\text{Class A}}{\text{Recall}} = \frac{TP_{\text{Class A}}}{TP_{\text{Class A}} + FN_{\text{Class A}}}$$

F1 Score



$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

The confusion matrix is like the blueprint that dissects your model's predictions. It consists of four key elements:

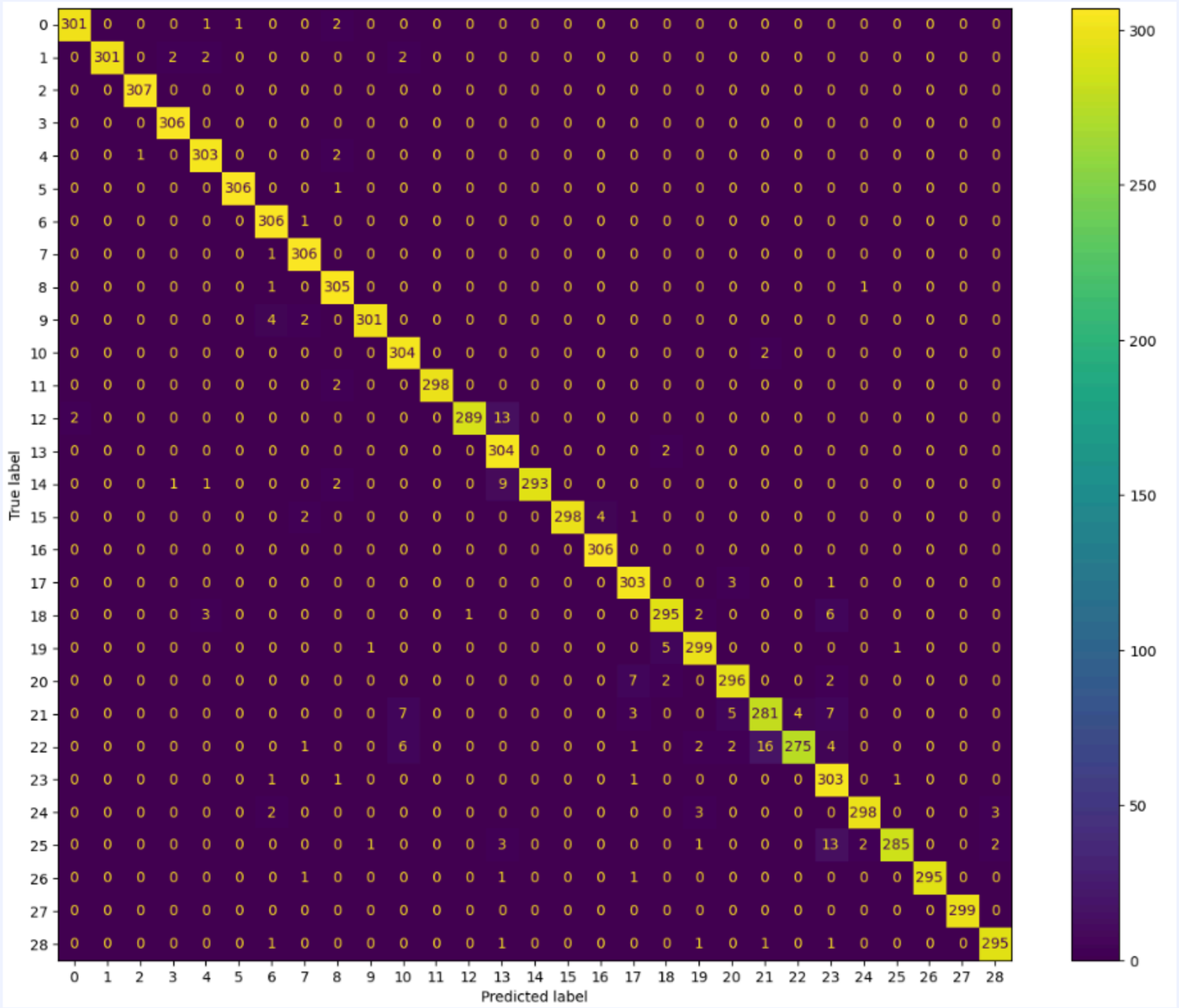
Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

display the Confusion Matrix

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
import seaborn as sns
import matplotlib.pyplot as plt

# Create the confusion matrix
#conf_matrix = confusion_matrix(true_labels, pred_lbls)
print("Length of y_test:", len(test_generator.classes))
print("Length of pred_lbls:", len(pred_lbls))

cm = confusion_matrix(test_generator.classes, pred_lbls, labels=np.unique(test_generator.classes))
# Display the confusion matrix
fig, ax = plt.subplots(figsize=(15, 12))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(test_generator.classes))
disp.plot(ax=ax)
plt.show()
```

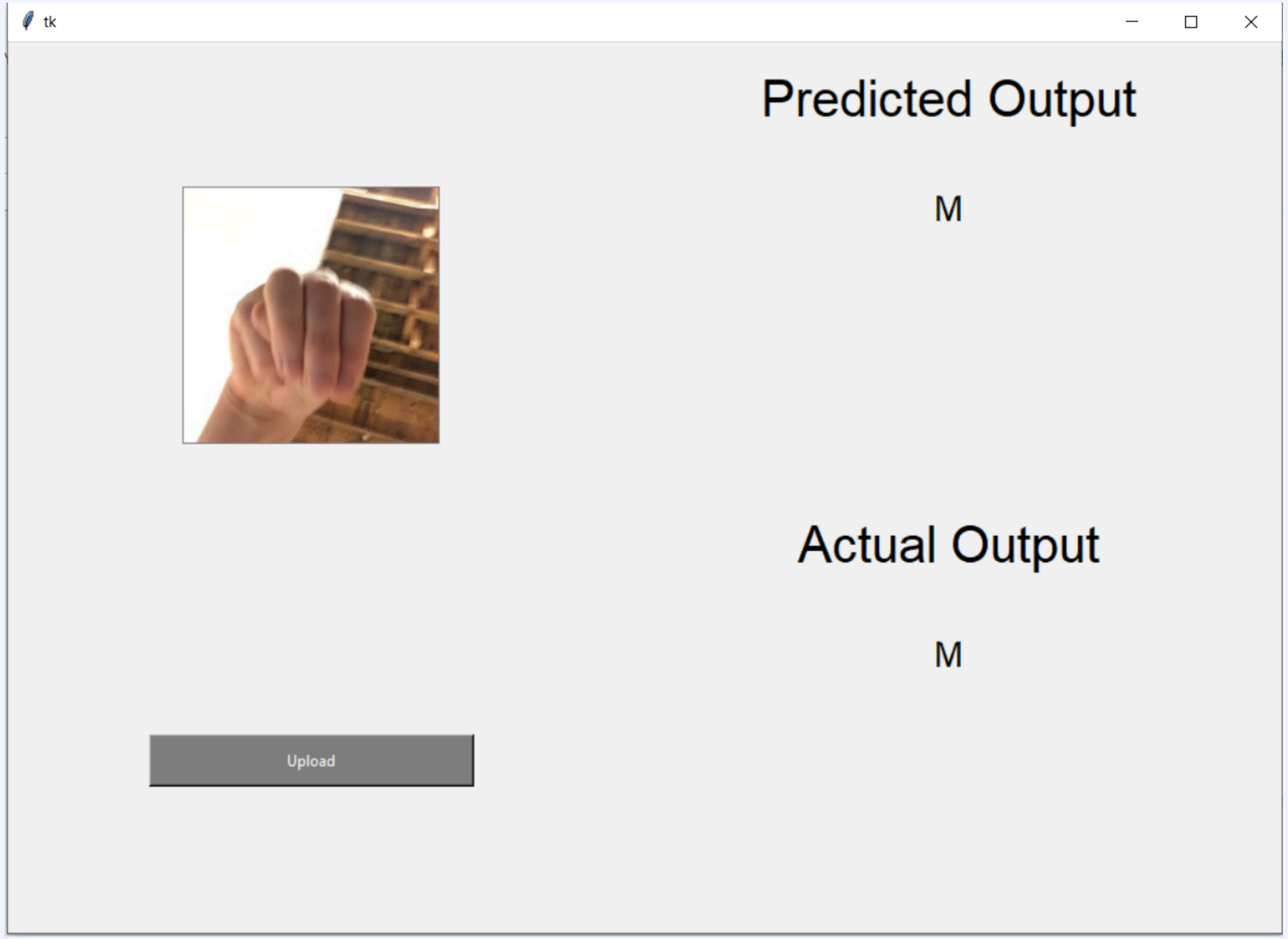
The output is



Length of y_test: 8813

Length of pred_lbls: 8813

GUI



Thank you very much!