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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import cv2
from tqdm.notebook import tqdm
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, Flatten, Conv2D, MaxPool2D
import seaborn as sns
import tensorflow.keras.backend as K
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from tensorflow.keras import optimizers
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
sns.set()
```

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.3,
                                   zoom_range=0.3,
                                   width_shift_range=0.3,
                                   height_shift_range=0.3)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory('/content/drive/MyDrive/emoji_data/data/Train',
                                                    target_size=(150, 150),
                                                    batch_size=2,
                                                    color_mode='grayscale',
                                                    class_mode="sparse",
                                                    shuffle=True)

validation_generator = test_datagen.flow_from_directory('/content/drive/MyDrive/emoji_data/data/Valid',
                                                        target_size=(150, 150),
                                                        batch_size=2,
                                                        color_mode='grayscale',
                                                        class_mode="sparse")
```

```
Found 50 images belonging to 5 classes.
Found 50 images belonging to 5 classes.
```

```
import tensorflow as tf
def f1score(y, y_pred):
  return f1_score(y, tf.math.argmax(y_pred, axis=1), average='micro')
def custom_f1score(y, y_pred):
  return tf.py_function(f1score, (y, y_pred), tf.double)
```

```
class stop_training_callback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('val_custom_f1score') > 0.98 and logs.get('custom_f1score') > 0.98):
      self.model.stop_training = True
```

```
K.clear_session()
ip = Input(shape = (150,150,1))
z = Conv2D(filters = 32, kernel_size = (64,64), padding='same', input_shape = (150,150,1), activation='relu')(ip)
z = Conv2D(filters = 64, kernel_size = (16,16), padding='same', input_shape = (150,150,1), activation='relu')(z)
z = Conv2D(filters = 128, kernel_size = (8,8), padding='same', input_shape = (150,150,1), activation='relu')(z)
z = MaxPool2D(pool_size = (4,4))(z)
z = Flatten()(z)
z = Dense(32, activation='relu')(z)
op = Dense(5, activation='softmax')(z)
model = Model(inputs=ip, outputs=op)
model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizers.Adam(lr=0.00001), metrics=[custom_f1score])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.le
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 1)]     0

 conv2d (Conv2D)             (None, 150, 150, 32)      131104

 conv2d_1 (Conv2D)           (None, 150, 150, 64)      524352

 conv2d_2 (Conv2D)           (None, 150, 150, 128)     524416

 max_pooling2d (MaxPooling2  (None, 37, 37, 128)       0
 D)

 flatten (Flatten)           (None, 175232)            0

 dense (Dense)               (None, 32)                5607456

 dense_1 (Dense)             (None, 5)                 165

=================================================================
Total params: 6787493 (25.89 MB)
Trainable params: 6787493 (25.89 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
callbacks = [stop_training_callback()]
model.fit(train_generator,
          epochs=10,
          validation_data=validation_generator,
          callbacks=callbacks,
          verbose=1)
```

```
Epoch 1/10
25/25 [==============================] - 456s 18s/step - loss: 4.3229 - custom_f1score: 0.2000 - val_loss: 1.6299 - val_custom_f1score:
Epoch 2/10
25/25 [==============================] - 443s 18s/step - loss: 1.5836 - custom_f1score: 0.2800 - val_loss: 1.3615 - val_custom_f1score:
Epoch 3/10
25/25 [==============================] - 452s 18s/step - loss: 1.5170 - custom_f1score: 0.3000 - val_loss: 2.3227 - val_custom_f1score:
Epoch 4/10
25/25 [==============================] - 440s 18s/step - loss: 1.5016 - custom_f1score: 0.3600 - val_loss: 1.6133 - val_custom_f1score:
Epoch 5/10
25/25 [==============================] - 423s 17s/step - loss: 1.6112 - custom_f1score: 0.1000 - val_loss: 1.6116 - val_custom_f1score:
Epoch 6/10
25/25 [==============================] - 419s 17s/step - loss: 1.6105 - custom_f1score: 0.1800 - val_loss: 1.6112 - val_custom_f1score:
Epoch 7/10
25/25 [==============================] - 420s 17s/step - loss: 1.6080 - custom_f1score: 0.2600 - val_loss: 1.6321 - val_custom_f1score:
Epoch 8/10
25/25 [==============================] - 421s 17s/step - loss: 1.5178 - custom_f1score: 0.2800 - val_loss: 1.8813 - val_custom_f1score:
Epoch 9/10
25/25 [==============================] - 403s 16s/step - loss: 1.4059 - custom_f1score: 0.2800 - val_loss: 2.5547 - val_custom_f1score:
Epoch 10/10
25/25 [==============================] - 396s 16s/step - loss: 1.8196 - custom_f1score: 0.2600 - val_loss: 1.6110 - val_custom_f1score:
<keras.src.callbacks.History at 0x787f14ef2950>
```

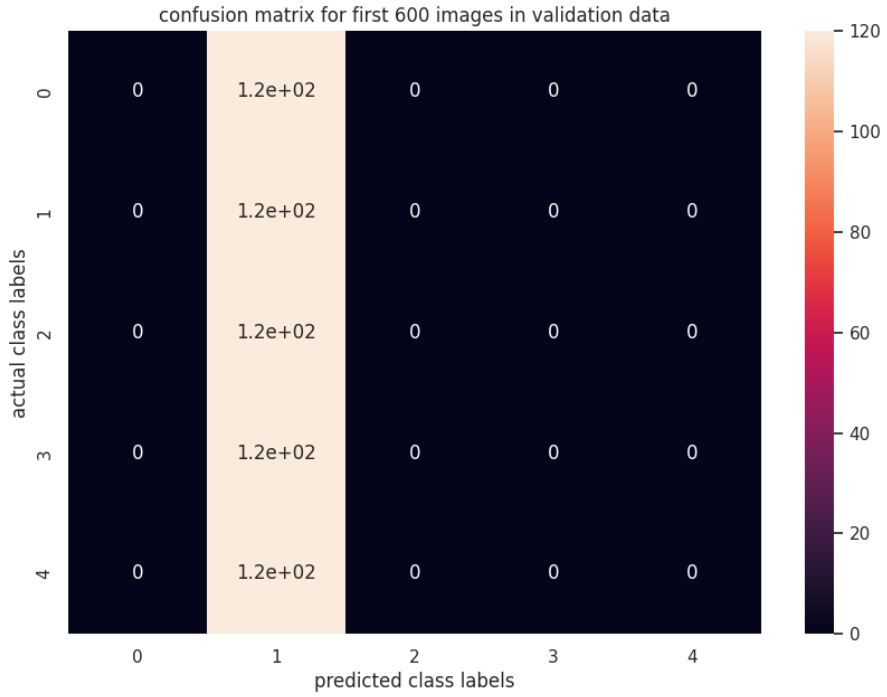
```
test_generator = test_datagen.flow_from_directory('/content/drive/MyDrive/emoji_data/data/Valid',
                                                  target_size=(150, 150),
                                                  batch_size=1,
                                                  color_mode='grayscale',
                                                  class_mode="sparse")
```

```
Found 50 images belonging to 5 classes.
```

```
num = 0
y_pred = []
y_true = []
for img, y_actual in test_generator:
    if num==10:
        break
    pred_label = model.predict(img).argmax()
    y_pred.append(pred_label)
    y_true.append(y_actual[0])
    num+=1
```

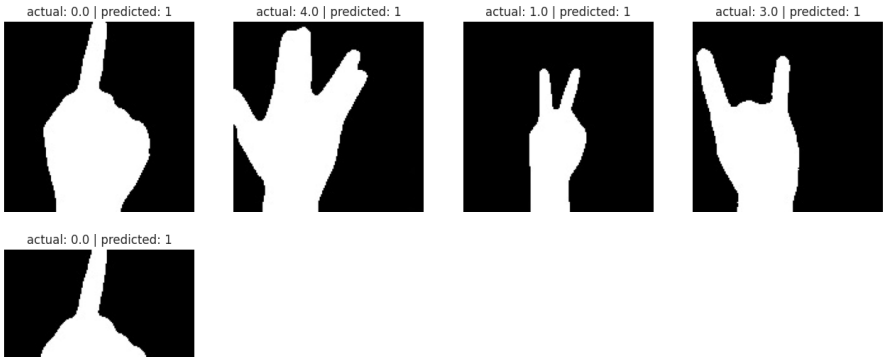
```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
```

```
cm = confusion_matrix(y_true, y_pred)
df_cm = pd.DataFrame(cm, index=np.arange(5), columns=np.arange(5))
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True)
plt.xlabel('predicted class labels')
plt.ylabel('actual class labels')
plt.title('confusion matrix for first 600 images in validation data')
plt.show()
```




```
plt.figure(figsize=(16,16))
for i,j in enumerate(test_generator):
    if i==5:
        break
    pred_label = model.predict(j[0]).argmax()
    actual_label = j[1][0]
    plt.subplot(4,4,i+1)
    img = j[0][0][:,:,0]
    plt.imshow(img, cmap='gray')
    plt.title(f'actual: {actual_label} | predicted: {pred_label}')
    plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
```




```
model.save_weights('model_weights.h5')
```

```
import cv2
import time
import numpy as np
import tensorflow.keras.backend as K
from sklearn.metrics import f1_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Input
from tensorflow.keras import optimizers
import tensorflow as tf
```

```
def create_model():
    K.clear_session()
    ip = Input(shape = (150,150,1))
    z = Conv2D(filters = 32, kernel_size = (64,64), padding='same', input_shape = (150,150,1), activation='relu')(ip)
    z = Conv2D(filters = 64, kernel_size = (16,16), padding='same', input_shape = (150,150,1), activation='relu')(z)
    z = Conv2D(filters = 128, kernel_size = (8,8), padding='same', input_shape = (150,150,1), activation='relu')(z)
    z = MaxPool2D(pool_size = (4,4))(z)
    z = Flatten()(z)
    z = Dense(32, activation='relu')(z)
    op = Dense(5, activation='softmax')(z)
    model = Model(inputs=ip, outputs=op)
    return model
model = create_model()
model.load_weights('model_weights.h5')
```

```
def hand_detect(img, type='haar'):
    cv2.imshow('output', img)
    key = cv2.waitKey(20)
    if key == 27:  # exit on ESC
        cv2.destroyAllWindows()
    hand_img = img.copy()
    if type=='yolo':
        width, height, inference_time, results = yolo.inference(img)
    elif type=='haar':
        results = cascade.detectMultiScale(hand_img, scaleFactor=1.3, minNeighbors=7)
    if len(results) > 0:
        if type=='yolo':
            _,_,_, x, y, w, h = results[0]
            return x,y,150,150
        elif type=='haar':
            x, y, w, h = results[0]
            return x-50,y-70,150,150
    else:
        return []
```

```
def main():
    roi = []
    while(len(roi) == 0):
        ret, frame = cap.read()
        frame = cv2.flip(frame, 1)
        roi = hand_detect(frame)
```

```
    cv2.destroyAllWindows()
    ret = tracker.init(frame, roi)
    cv2.destroyAllWindows()
    c = 0
    d = c+1
    while True:
        ret, frame = cap.read()
        frame = cv2.flip(frame, 1)
        success, roi = tracker.update(frame)
        (x,y,w,h) = tuple(map(int, roi))
        if success:
            pt1 = (x,y)
            pt2 = (x+w, y+h)
            square  = frame[y:y+h, x:x+w]
            gray = cv2.cvtColor(square, cv2.COLOR_BGR2GRAY)
            gray_ = cv2.medianBlur(gray, 7)
            hand = contours(gray, th=150)[0]
            im = np.array([hand])
            im = im.reshape(-1,150,150,1)
            result = pred(im)
            cv2.rectangle(frame, pt1, pt2, (255,255,0), 3) .
            emo = icon(result)
            frame_copy = paste(frame, emo)
            (a,b) = hand.shape
            for i in range(3):
                hand = hand*255
                frame_copy[0:a, 0:b, i] = hand
        else:
            cv2.putText(frame, 'Failed to detect object', (100,200), cv2.FONT_HERSHEY_SIMPLEX, 2, 255)
            cv2.imshow('output', frame_copy)
            roi = []

            while(len(roi) == 0):
                ret, frame = cap.read()
                frame = cv2.flip(frame, 1)
                roi = hand_detect(frame)
            continue
        cv2.imshow('output', frame_copy)
        if cv2.waitKey(1) & 0xFF == 27:
          break
    cap.release()
    cv2.destroyAllWindows()
```

```
def contours(diff, th=100):
  thresholded = cv2.threshold(diff, th, 255, cv2.THRESH_BINARY_INV)
  contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
  if len(contours) == 0:
      return None
  else:
      hand_segment = max(contours, key=cv2.contourArea)
  return (thresholded, hand_segment)
```

```
def pred(img):
    return model.predict(img).argmax(axis=1)[0]
```

```
emoji = []
for i in range(1,6):
    img = cv2.imread(f'/content/drive/MyDrive/emoji_data/emoji/{i}.png', -1)
    emoji.append(img)

```

```
def icon(x, e=emoji):
    return e[x - 1]
```

```
def paste(frame, s_img, pt1, pt2):
    x, y = pt1
    x_w, y_h = pt2
    n = min(x_w-x, y_h-y)
    s_img = cv2.resize(s_img, dsize=(n,n))
    l_img = frame
    (x_offset,y_offset,_) = (frame.shape)
    (y_offset,x_offset) = (x_offset//2 - 72, y_offset//2 - 72)
    y1, y2 = y_offset, y_offset + s_img.shape[0]
    x1, x2 = x_offset, x_offset + s_img.shape[1]
```

```
11
12      alpha_s = s_img[:, :, 3] / 255.0
13      alpha_l = 1.0 - alpha_s
14
15      for c in range(0, 3):
16          l_img[y:y_h, x:x_w, c] = (alpha_s * s_img[:, :, c] + alpha_l * l_img[y:y_h, x:x_w, c])
17      return l_img
```