In [3]:

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Activation, Dense, Flatten, BatchNormalization, Co
```

In [4]:

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_s
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', paddin
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation ="relu"))
model.add(Dense(128,activation ="relu"))
#model.add(Dropout(0.2))
model.add(Dense(128,activation ="relu"))
#model.add(Dropout(0.3))
model.add(Dense(10,activation ="softmax"))
```

In [5]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2  (None, 31, 31, 32)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 31, 31, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 15, 15, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_2 (Conv2D)           (None, 13, 13, 128)       73856     
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 128)         0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (None, 4608)              0         
                                                                 
 dense (Dense)               (None, 64)                294976    
                                                                 
 dense_1 (Dense)             (None, 128)               8320      
                                                                 
 dense_2 (Dense)             (None, 128)               16512     
                                                                 
 dense_3 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 414346 (1.58 MB)
Trainable params: 414346 (1.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [6]:
```
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crosse
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, m
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, ver



model.compile(optimizer=SGD(learning_rate=0.001), loss='categorical_crossen
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, m
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, ver
```

In [10]:

```
history2 = model.fit(train_batches, epochs=10, callbacks=[reduce_lr, early_
```

```
Epoch 1/10
252/252 [==============================] - 47s 180ms/step - loss: 2.8262e-
04 - accuracy: 1.0000 - val_loss: 3.7159e-04 - val_accuracy: 1.0000 - lr:
5.0000e-04
Epoch 2/10
252/252 [==============================] - 38s 151ms/step - loss: 2.6475e-
04 - accuracy: 1.0000 - val_loss: 3.4914e-04 - val_accuracy: 1.0000 - lr:
5.0000e-04
Epoch 3/10
252/252 [==============================] - 33s 129ms/step - loss: 2.4882e-
04 - accuracy: 1.0000 - val_loss: 3.2906e-04 - val_accuracy: 1.0000 - lr:
5.0000e-04
Epoch 4/10
252/252 [==============================] - 32s 128ms/step - loss: 2.3455e-
04 - accuracy: 1.0000 - val_loss: 3.1113e-04 - val_accuracy: 1.0000 - lr:
5.0000e-04
Epoch 5/10
252/252 [==============================] - 32s 128ms/step - loss: 2.2174e-
04 - accuracy: 1.0000 - val_loss: 2.9502e-04 - val_accuracy: 1.0000 - lr:
5.0000e-04
Epoch 6/10
252/252 [==============================] - 33s 132ms/step - loss: 2.1018e-
04 - accuracy: 1.0000 - val_loss: 2.8033e-04 - val_accuracy: 1.0000 - lr:
5.0000e-04
Epoch 7/10
252/252 [==============================] - 33s 131ms/step - loss: 1.9971e-
04 - accuracy: 1.0000 - val_loss: 2.6704e-04 - val_accuracy: 1.0000 - lr:
5.0000e-04
Epoch 8/10
252/252 [==============================] - 775s 3s/step - loss: 1.9018e-04
- accuracy: 1.0000 - val_loss: 2.5491e-04 - val_accuracy: 1.0000 - lr: 5.0
000e-04
Epoch 9/10
252/252 [==============================] - 37s 145ms/step - loss: 1.8148e-
04 - accuracy: 1.0000 - val_loss: 2.4376e-04 - val_accuracy: 1.0000 - lr:
5.0000e-04
Epoch 10/10
252/252 [==============================] - 35s 140ms/step - loss: 1.7351e-
04 - accuracy: 1.0000 - val_loss: 2.3352e-04 - val_accuracy: 1.0000 - lr:
5.0000e-04
```

In [9]:

```python
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Activation, Dense, Flatten, BatchNormalization, Co
from keras.optimizers import Adam, SGD
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
import itertools
import random
import warnings
import numpy as np
import cv2
import matplotlib.pyplot as plt
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import ModelCheckpoint, EarlyStopping
warnings.simplefilter(action='ignore', category=FutureWarning)


train_path = r'D:\gesture\train'
test_path = r'D:\gesture\test'

train_batches = ImageDataGenerator(preprocessing_function=tf.keras.applicat
test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applicati

imgs, labels = next(train_batches)


#Plotting the images...
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 10, figsize=(30,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()


plotImages(imgs)
print(imgs.shape)
print(labels)

model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_s
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', paddin
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation ="relu"))
model.add(Dense(128,activation ="relu"))
#model.add(Dropout(0.2))
model.add(Dense(128,activation ="relu"))
#model.add(Dropout(0.3))
```

```
model.add(Dense(10,activation ="softmax"))


# In[23]:


model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crosse
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, m
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, ver



model.compile(optimizer=SGD(learning_rate=0.001), loss='categorical_crossen
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, m
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, ver


history2 = model.fit(train_batches, epochs=10, callbacks=[reduce_lr, early_
imgs, labels = next(train_batches) # For getting next batch of imgs...

imgs, labels = next(test_batches) # For getting next batch of imgs...
scores = model.evaluate(imgs, labels, verbose=0)
print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} o


#model.save('best_model_dataflair.h5')
model.save('best_model_dataflair3.h5')

print(history2.history)

imgs, labels = next(test_batches)

model = keras.models.load_model(r"best_model_dataflair3.h5")

scores = model.evaluate(imgs, labels, verbose=0)
print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} o

model.summary()

scores #[loss, accuracy] on test data...
model.metrics_names


word_dict = {0:'One',1:'Ten',2:'Two',3:'Three',4:'Four',5:'Five',6:'Six',7:

predictions = model.predict(imgs, verbose=0)
print("predictions on a small set of test data--")
print("")
for ind, i in enumerate(predictions):
    print(word_dict[np.argmax(i)], end='   ')

plotImages(imgs)
print('Actual labels')
for i in labels:
    print(word_dict[np.argmax(i)], end='   ')

print(imgs.shape)

history2.history
```

```
Found 2520 images belonging to 10 classes.
Found 490 images belonging to 10 classes.

Clipping input data to the valid range for imshow with RGB data ([0..1]
for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1]
for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1]
for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1]
for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1]
for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1]
for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1]
for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1]
for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1]
```

In [15]:

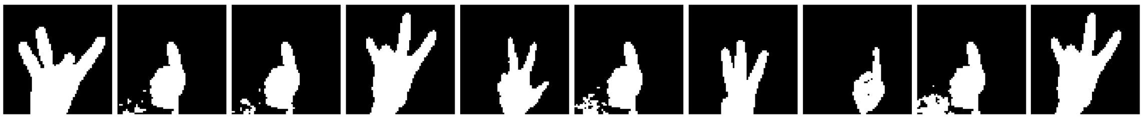
```python
# For getting next batch of testing imgs...
imgs, labels = next(test_batches)

scores = model.evaluate(imgs, labels, verbose=0)
print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} o


#Once the model is fitted we save the model using model.save()  function.


model.save(r'C:\Users\vishwanth\Downloads\sign-language-recognition-project
```

```
loss of 0.00024940239381976426; accuracy of 100.0%
```

In [13]:

```
word_dict = {0:'One',1:'Ten',2:'Two',3:'Three',4:'Four',5:'Five',6:'Six',7:

predictions = model.predict(imgs, verbose=0)
print("predictions on a small set of test data--")
print("")
for ind, i in enumerate(predictions):
    print(word_dict[np.argmax(i)], end='   ')

plotImages(imgs)
print('Actual labels')
for i in labels:
    print(word_dict[np.argmax(i)], end='   ')
```

```
predictions on a small set of test data--

Eight    Ten    Ten    Seven    Three    Ten    Six    One    Ten    Seven

Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```



```
Actual labels
Eight    Ten    Ten    Seven    Three    Ten    Six    One    Ten    Seven
```

In [21]:

```
import numpy as np
import cv2
import keras
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf

model = keras.models.load_model(r'C:\Users\vishwanth\Downloads\sign-languag


background = None
accumulated_weight = 0.5

ROI_top = 100
ROI_bottom = 300
ROI_right = 150
ROI_left = 350



def cal_accum_avg(frame, accumulated_weight):

    global background

    if background is None:
        background = frame.copy().astype("float")
        return None

    cv2.accumulateWeighted(frame, background, accumulated_weight)



def segment_hand(frame, threshold=25):
    global background

    diff = cv2.absdiff(background.astype("uint8"), frame)


    _ , thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY

    #Fetching contours in the frame (These contours can be of hand or any o
    contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXT


    # If length of contours list = 0, means we didn't get any contours...
    if len(contours) == 0:
        return None
    else:
        # The largest external contour should be the hand
        hand_segment_max_cont = max(contours, key=cv2.contourArea)

        # Returning the hand segment(max contour) and the thresholded image
        return (thresholded, hand_segment_max_cont)

cam = cv2.VideoCapture(0)
num_frames =0
while True:
    ret, frame = cam.read()

    # filpping the frame to prevent inverted image of captured frame...
    frame = cv2.flip(frame, 1)
    frame_copy = frame.copy()

    # ROI from the frame
    roi = frame[ROI_top:ROI_bottom, ROI_right:ROI_left]

    gray_frame = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    gray_frame = cv2.GaussianBlur(gray_frame, (9, 9), 0)


    if num_frames < 70:

        cal_accum_avg(gray_frame, accumulated_weight)

        cv2.putText(frame_copy, "FETCHING BACKGROUND...PLEASE WAIT", (80, 4

    else:
        # segmenting the hand region
        hand = segment_hand(gray_frame)


        # Checking if we are able to detect the hand...
        if hand is not None:

            thresholded, hand_segment = hand

            # Drawing contours around hand segment
            cv2.drawContours(frame_copy, [hand_segment + (ROI_right, ROI_to

            cv2.imshow("Thesholded Hand Image", thresholded)

            thresholded = cv2.resize(thresholded, (64, 64))
            thresholded = cv2.cvtColor(thresholded, cv2.COLOR_GRAY2RGB)
            thresholded = np.reshape(thresholded, (1,thresholded.shape[0],t

            pred = model.predict(thresholded)
            cv2.putText(frame_copy, word_dict[np.argmax(pred)], (170, 45),

    # Draw ROI on frame_copy
    cv2.rectangle(frame_copy, (ROI_left, ROI_top), (ROI_right, ROI_bottom),

    # incrementing the number of frames for tracking
    num_frames += 1

    # Display the frame with segmented hand
    cv2.putText(frame_copy, "DataFlair hand sign recognition_ _ _", (10, 20
    cv2.imshow("Sign Detection", frame_copy)


    # Close windows with Esc
    k = cv2.waitKey(1) & 0xFF

    if k == 27:
        break

# Release the camera and destroy all the windows
cam.release()
cv2.destroyAllWindows()
```

```
1/1 [==============================] - 0s 228ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 73ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 73ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 63ms/step
```

In [ ]:

In [ ]: