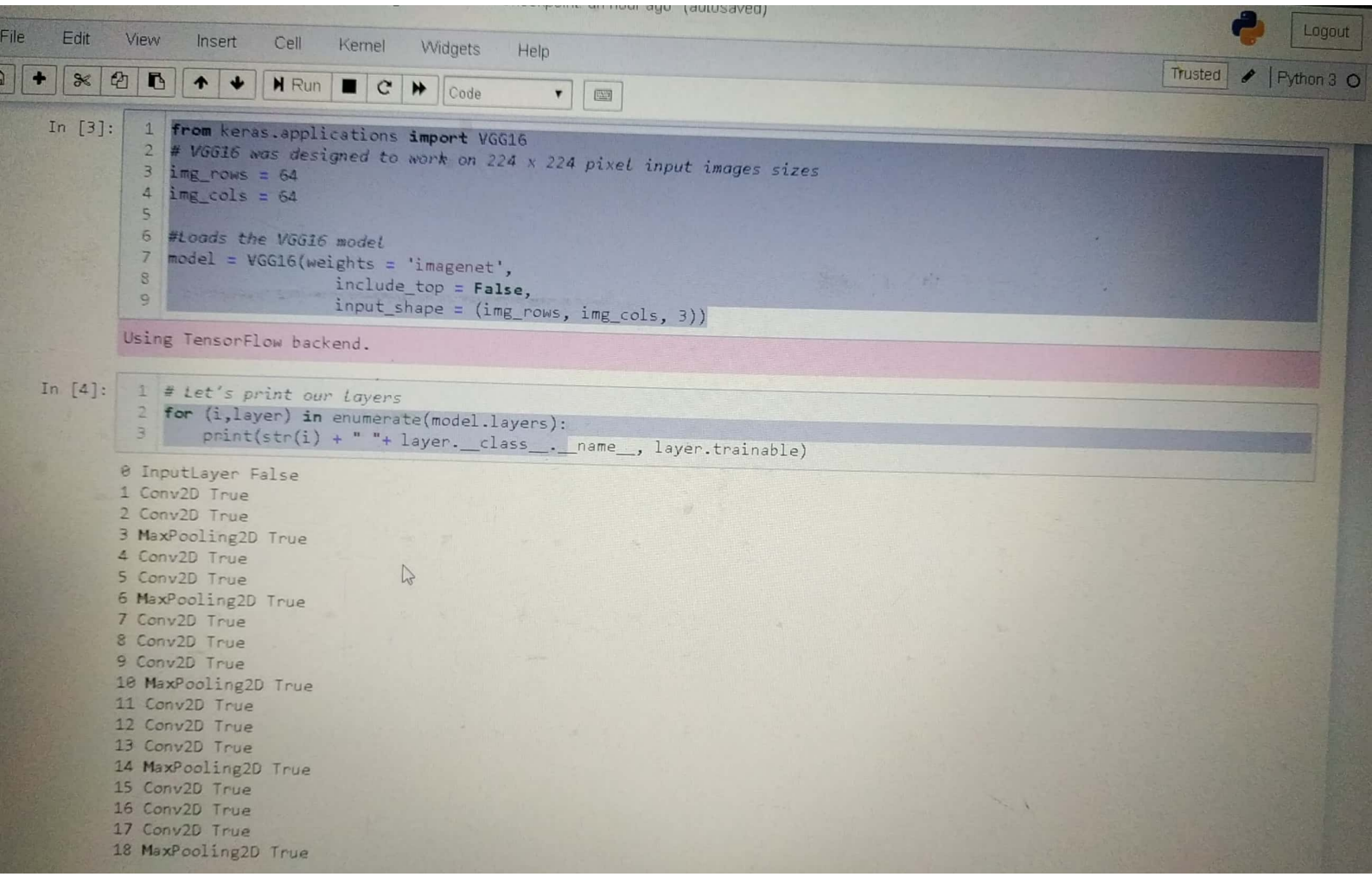
In [2]:

```python
import cv2
import numpy as np
# Load HAAR face classifier
face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
# Load functions
def face_extractor(img):
    # Function detects faces and returns the cropped face
    # If no face detected, it returns the input image
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray, 1.3, 5)
    if faces is ():
        return None
    # Crop all faces found
    for (x,y,w,h) in faces:
        cropped_face = img[y:y+h, x:x+w]
    return cropped_face
# Initialize Webcam
cap = cv2.VideoCapture(0)
count = 0
# Collect 100 samples of your face from webcam input
while True:
    ret, frame = cap.read()
    if face_extractor(frame) is not None:
        count += 1
        face = cv2.resize(face_extractor(frame), (200, 200))
        face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
        # Save file in specified directory with unique name
        file_name_path = 'C://Users//harsh//Desktop//MLOPS//self practice//train//harshita//image' + str(count) + '.jpg
        cv2.imwrite(file_name_path, face)
        # Put count on images and display live count
        cv2.putText(face, str(count), (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 2)
        cv2.imshow('Face Cropper', face)
    else:
        print("Face not found")
        pass
    if cv2.waitKey(1) == 13 or count == 100: #13 is the Enter Key
        break
cap.release()
cv2.destroyAllWindows()
print("Collecting Samples Complete")
```



File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run | Code

```
    top_model = bottom_model.output
    top_model=GlobalAveragePooling2D()(top_model)
    top_model=Dense(1024,activation='relu')(top_model)
    top_model=Dense(512,activation='relu')(top_model)
    top_model=Dense(248,activation='relu')(top_model)
    top_model=Dense(248,activation='relu')(top_model)
    top_model=Dense(128,activation='relu')(top_model)
    top_model=Dense(64,activation='relu')(top_model)
    top_model = Dense(num_classes, activation = "softmax")(top_model)
    return top_model
```

In [6]:

```
#Adding fully connected layer in VGG
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten,GlobalAveragePooling2D
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.models import Model
num_classes = 4

FC_Head = addTopModel(model, num_classes)

modelnew = Model(inputs=model.input, outputs=FC_Head)

print(modelnew.summary())
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64, 64, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
```

jupyter Deep Learning face recognition Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [7]:

```
#Loading Dataset
from keras.preprocessing.image import ImageDataGenerator

train_data_dir = 'C://Users//harsh//Desktop//MLOPS//self practice//train//'
validation_data_dir = 'C://Users//harsh//Desktop//MLOPS//self practice//validation//'

train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=20,
      width_shift_range=0.2,
      height_shift_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# Change the batchsize according to your system RAM
train_batchsize = 16
val_batchsize = 16

train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_rows, img_cols),
        batch_size=train_batchsize,
        class_mode='categorical')

validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_rows, img_cols),
        batch_size=val_batchsize,
        class_mode='categorical',
        shuffle=False)
```

```
Found 400 images belonging to 4 classes.
Found 40 images belonging to 4 classes.
```

Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Logout

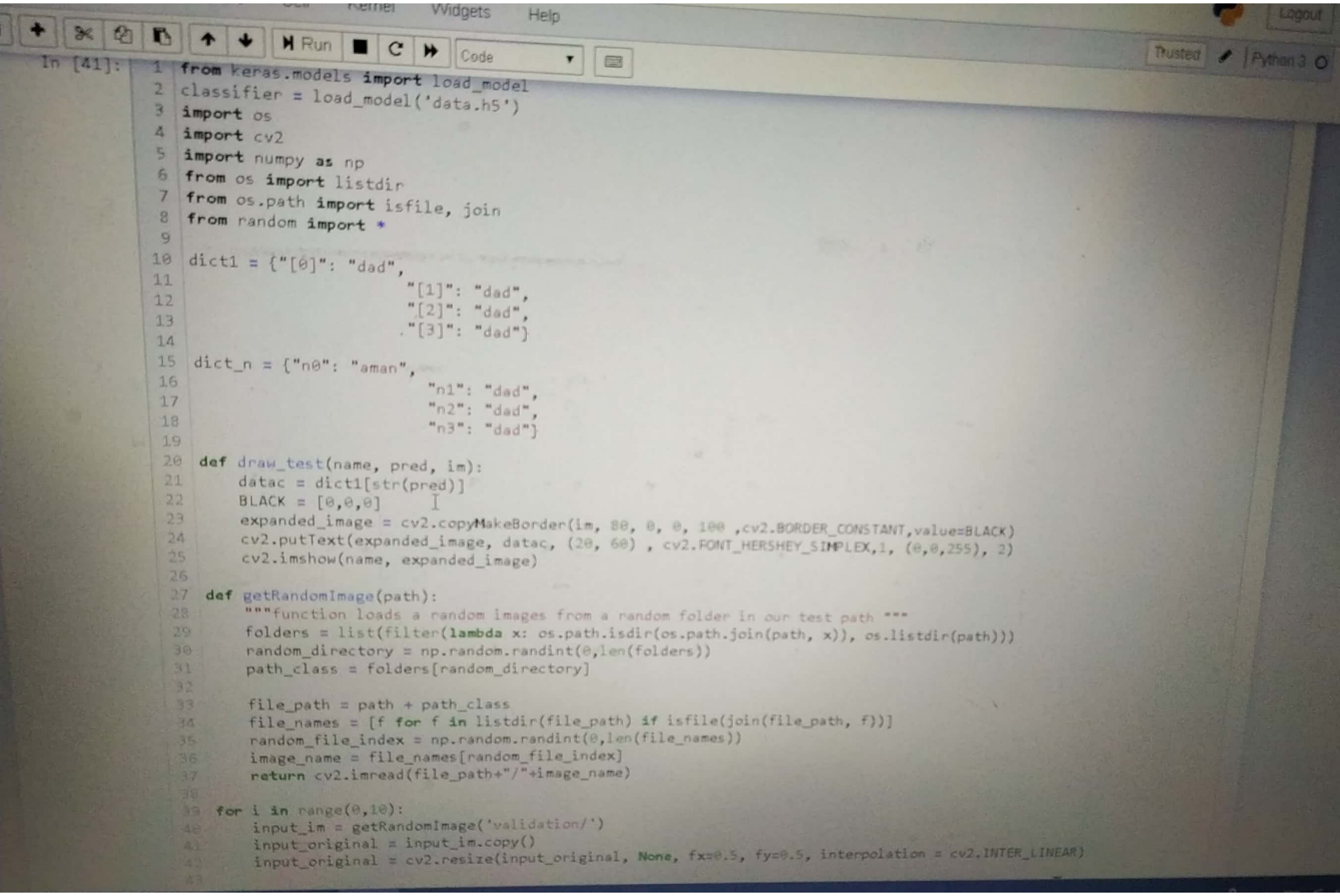
Run Code Trusted Python 3

Found 40 images belonging to 4 classes.

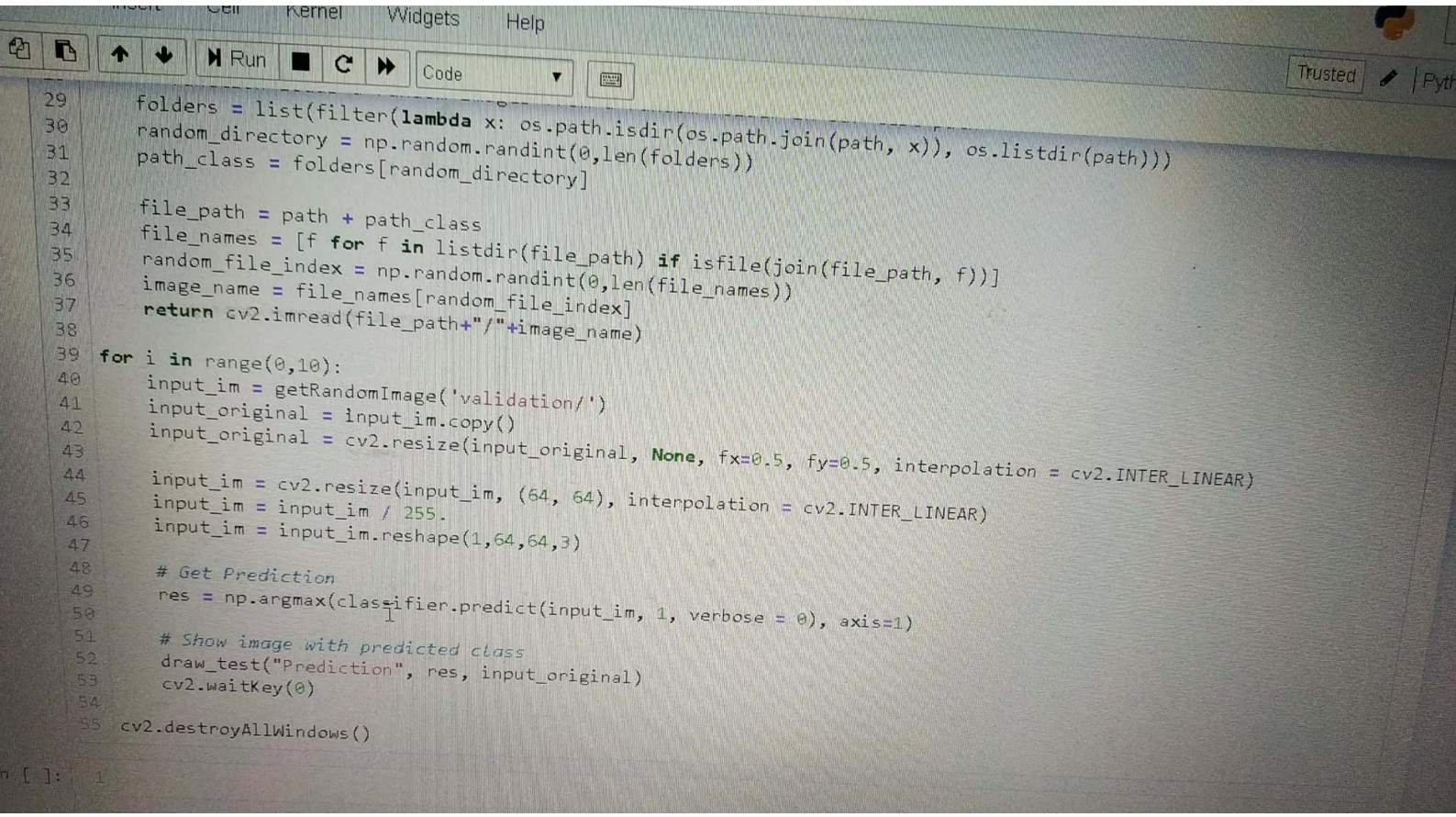
In [8]:

```
from keras.optimizers import RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint = ModelCheckpoint("data.h5",
                             monitor="val_loss",

                             mode="min",
                             save_best_only = True,
                             verbose=1)

earlystop = EarlyStopping(monitor = 'val_loss',
                          min_delta = 0,
                          patience = 3,
                          verbose = 1,
                          restore_best_weights = True)

# we put our call backs into a callback list
callbacks = [earlystop, checkpoint]

# Note we use a very small learning rate
modelnew.compile(loss = 'categorical_crossentropy',
              optimizer = RMSprop(lr = 0.001),
              metrics = ['accuracy'])

nb_train_samples = 400
nb_validation_samples = 100
epochs = 5
batch_size = 20
history = modelnew.fit_generator(
    train_generator,
    steps_per_epoch = nb_train_samples // batch_size,
    epochs = epochs,
    callbacks = callbacks,
    validation_data = validation_generator,
    validation_steps = nb_validation_samples // batch_size)
modelnew.save("data.h5")
```

```
Epoch 1/5
20/20 [==============================] - 52s 3s/step - loss: 373.6386 - accuracy: 0.2094 - val_loss: 1.3759 - val_accurac
y: 0.2778
```



```
In [41]:
from keras.models import load_model
classifier = load_model('data.h5')
import os
import cv2
import numpy as np
from os import listdir
from os.path import isfile, join
from random import *

dict1 = {"[0]": "dad",
                      "[1]": "dad",
                      "[2]": "dad",
                      "[3]": "dad"}

dict_n = {"n0": "aman",
                        "n1": "dad",
                        "n2": "dad",
                        "n3": "dad"}

def draw_test(name, pred, im):
    datac = dict1[str(pred)]
    BLACK = [0,0,0]
    expanded_image = cv2.copyMakeBorder(im, 80, 0, 0, 100 ,cv2.BORDER_CONSTANT,value=BLACK)
    cv2.putText(expanded_image, datac, (20, 60) , cv2.FONT_HERSHEY_SIMPLEX,1, (0,0,255), 2)
    cv2.imshow(name, expanded_image)

def getRandomImage(path):
    """function loads a random images from a random folder in our test path """
    folders = list(filter(lambda x: os.path.isdir(os.path.join(path, x)), os.listdir(path)))
    random_directory = np.random.randint(0,len(folders))
    path_class = folders[random_directory]

    file_path = path + path_class
    file_names = [f for f in listdir(file_path) if isfile(join(file_path, f))]
    random_file_index = np.random.randint(0,len(file_names))
    image_name = file_names[random_file_index]
    return cv2.imread(file_path+"/"+image_name)

for i in range(0,10):
    input_im = getRandomImage('validation/')
    input_original = input_im.copy()
    input_original = cv2.resize(input_original, None, fx=0.5, fy=0.5, interpolation = cv2.INTER_LINEAR)
```



```
    folders = list(filter(lambda x: os.path.isdir(os.path.join(path, x)), os.listdir(path)))
    random_directory = np.random.randint(0,len(folders))
    path_class = folders[random_directory]

    file_path = path + path_class
    file_names = [f for f in listdir(file_path) if isfile(join(file_path, f))]
    random_file_index = np.random.randint(0,len(file_names))
    image_name = file_names[random_file_index]
    return cv2.imread(file_path+"/"+image_name)

for i in range(0,10):
    input_im = getRandomImage('validation/')
    input_original = input_im.copy()
    input_original = cv2.resize(input_original, None, fx=0.5, fy=0.5, interpolation = cv2.INTER_LINEAR)

    input_im = cv2.resize(input_im, (64, 64), interpolation = cv2.INTER_LINEAR)
    input_im = input_im / 255.
    input_im = input_im.reshape(1,64,64,3)

    # Get Prediction
    res = np.argmax(classifier.predict(input_im, 1, verbose = 0), axis=1)

    # Show image with predicted class
    draw_test("Prediction", res, input_original)
    cv2.waitKey(0)

cv2.destroyAllWindows()
```