```
import tensorflow as tf
tf.test.gpu_device_name()
```

```
{"type":"string"}
```

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core import Flatten, Dense, Dropout
from keras.layers.convolutional import Convolution2D, MaxPooling2D,
ZeroPadding2D
from keras.optimizers import SGD
import cv2

from keras.utils import to_categorical

X_train = []
y_train = []
X_test = []
y_test = []
df= pd.read_csv('/content/fer2013.csv')
for index, row in df.iterrows():
    k = row['pixels'].split(" ")
    if row['Usage'] == 'Training':
        X_train.append(np.array(k))
        y_train.append(row['emotion'])
    elif row['Usage'] == 'PublicTest':
        X_test.append(np.array(k))
        y_test.append(row['emotion'])

X_train = np.array(X_train, dtype = 'uint8')
y_train = np.array(y_train, dtype = 'uint8')
X_test = np.array(X_test, dtype = 'uint8')
y_test = np.array(y_test, dtype = 'uint8')

import keras
from keras.utils import to_categorical
y_train= to_categorical(y_train, num_classes=7)
y_test = to_categorical(y_test, num_classes=7)

X_train = X_train.reshape(X_train.shape[0], 48, 48, 1)
X_test = X_test.reshape(X_test.shape[0], 48, 48, 1)

from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range = 10,
    horizontal_flip = True,
    width_shift_range=0.1,
```

```
    height_shift_range=0.1,
    fill_mode = 'nearest')
testgen = ImageDataGenerator(rescale=1./255)
datagen.fit(X_train)
batch_size = 64

train_flow = datagen.flow(X_train, y_train, batch_size=batch_size)
test_flow = testgen.flow(X_test, y_test, batch_size=batch_size)

from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input, Dense, Flatten, Dropout,
BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.layers import concatenate
from keras.optimizers import Adam, SGD
from keras.regularizers import l1, l2
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix

def FER_Model(input_shape=(48,48,1)):
    # first input model
    visible = Input(shape=input_shape, name='input')
    num_classes = 7
    #the 1-st block
    conv1_1 = Conv2D(64, kernel_size=3, activation='relu',
padding='same', name = 'conv1_1')(visible)
    conv1_1 = BatchNormalization()(conv1_1)
    conv1_2 = Conv2D(64, kernel_size=3, activation='relu',
padding='same', name = 'conv1_2')(conv1_1)
    conv1_2 = BatchNormalization()(conv1_2)
    pool1_1 = MaxPooling2D(pool_size=(2,2), name = 'pool1_1')(conv1_2)
    drop1_1 = Dropout(0.3, name = 'drop1_1')(pool1_1)#the 2-nd block
    conv2_1 = Conv2D(128, kernel_size=3, activation='relu',
padding='same', name = 'conv2_1')(drop1_1)
    conv2_1 = BatchNormalization()(conv2_1)
    conv2_2 = Conv2D(128, kernel_size=3, activation='relu',
padding='same', name = 'conv2_2')(conv2_1)
    conv2_2 = BatchNormalization()(conv2_2)
    conv2_3 = Conv2D(128, kernel_size=3, activation='relu',
padding='same', name = 'conv2_3')(conv2_2)
    conv2_2 = BatchNormalization()(conv2_3)
    pool2_1 = MaxPooling2D(pool_size=(2,2), name = 'pool2_1')(conv2_3)
    drop2_1 = Dropout(0.3, name = 'drop2_1')(pool2_1)#the 3-rd block
    conv3_1 = Conv2D(256, kernel_size=3, activation='relu',
padding='same', name = 'conv3_1')(drop2_1)
    conv3_1 = BatchNormalization()(conv3_1)
    conv3_2 = Conv2D(256, kernel_size=3, activation='relu',
padding='same', name = 'conv3_2')(conv3_1)
```

```
    conv3_2 = BatchNormalization()(conv3_2)
    conv3_3 = Conv2D(256, kernel_size=3, activation='relu',
padding='same', name = 'conv3_3')(conv3_2)
    conv3_3 = BatchNormalization()(conv3_3)
    conv3_4 = Conv2D(256, kernel_size=3, activation='relu',
padding='same', name = 'conv3_4')(conv3_3)
    conv3_4 = BatchNormalization()(conv3_4)
    pool3_1 = MaxPooling2D(pool_size=(2,2), name = 'pool3_1')(conv3_4)
    drop3_1 = Dropout(0.3, name = 'drop3_1')(pool3_1)#the 4-th block
    conv4_1 = Conv2D(256, kernel_size=3, activation='relu',
padding='same', name = 'conv4_1')(drop3_1)
    conv4_1 = BatchNormalization()(conv4_1)
    conv4_2 = Conv2D(256, kernel_size=3, activation='relu',
padding='same', name = 'conv4_2')(conv4_1)
    conv4_2 = BatchNormalization()(conv4_2)
    conv4_3 = Conv2D(256, kernel_size=3, activation='relu',
padding='same', name = 'conv4_3')(conv4_2)
    conv4_3 = BatchNormalization()(conv4_3)
    conv4_4 = Conv2D(256, kernel_size=3, activation='relu',
padding='same', name = 'conv4_4')(conv4_3)
    conv4_4 = BatchNormalization()(conv4_4)
    pool4_1 = MaxPooling2D(pool_size=(2,2), name = 'pool4_1')(conv4_4)
    drop4_1 = Dropout(0.3, name = 'drop4_1')(pool4_1)

    #5th block
    conv5_1 = Conv2D(512, kernel_size=3, activation='relu',
padding='same', name = 'conv5_1')(drop4_1)
    conv5_1 = BatchNormalization()(conv5_1)
    conv5_2 = Conv2D(512, kernel_size=3, activation='relu',
padding='same', name = 'conv5_2')(conv5_1)
    conv5_2 = BatchNormalization()(conv5_2)
    conv5_3 = Conv2D(512, kernel_size=3, activation='relu',
padding='same', name = 'conv5_3')(conv5_2)
    conv5_3 = BatchNormalization()(conv5_3)
    conv5_4 = Conv2D(512, kernel_size=3, activation='relu',
padding='same', name = 'conv5_4')(conv5_3)
    conv5_3 = BatchNormalization()(conv5_3)
    pool5_1 = MaxPooling2D(pool_size=(2,2), name = 'pool5_1')(conv5_4)
    drop5_1 = Dropout(0.3, name = 'drop5_1')(pool5_1)#Flatten and
output
    flatten = Flatten(name = 'flatten')(drop5_1)
    ouput = Dense(num_classes, activation='softmax', name = 'output')
(flatten)# create model
    model = Model(inputs =visible, outputs = ouput)
    # summary layers
    print(model.summary())

    return model
```

```
model = FER_Model()
opt = Adam(lr=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=opt,
metrics=['accuracy'])
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 48, 48, 1)]       0

 conv1_1 (Conv2D)            (None, 48, 48, 64)        640

 batch_normalization_17 (Bat  (None, 48, 48, 64)       256
 chNormalization)

 conv1_2 (Conv2D)            (None, 48, 48, 64)        36928

 batch_normalization_18 (Bat  (None, 48, 48, 64)       256
 chNormalization)

 pool1_1 (MaxPooling2D)      (None, 24, 24, 64)        0

 drop1_1 (Dropout)           (None, 24, 24, 64)        0

 conv2_1 (Conv2D)            (None, 24, 24, 128)       73856

 batch_normalization_19 (Bat  (None, 24, 24, 128)      512
 chNormalization)

 conv2_2 (Conv2D)            (None, 24, 24, 128)       147584

 batch_normalization_20 (Bat  (None, 24, 24, 128)      512
 chNormalization)

 conv2_3 (Conv2D)            (None, 24, 24, 128)       147584

 pool2_1 (MaxPooling2D)      (None, 12, 12, 128)       0

 drop2_1 (Dropout)           (None, 12, 12, 128)       0

 conv3_1 (Conv2D)            (None, 12, 12, 256)       295168

 batch_normalization_22 (Bat  (None, 12, 12, 256)      1024
 chNormalization)

 conv3_2 (Conv2D)            (None, 12, 12, 256)       590080

 batch_normalization_23 (Bat  (None, 12, 12, 256)      1024
 chNormalization)
```

```
conv3_3 (Conv2D)             (None, 12, 12, 256)       590080

batch_normalization_24 (Bat  (None, 12, 12, 256)       1024
chNormalization)

conv3_4 (Conv2D)             (None, 12, 12, 256)       590080

batch_normalization_25 (Bat  (None, 12, 12, 256)       1024
chNormalization)

pool3_1 (MaxPooling2D)       (None, 6, 6, 256)         0

drop3_1 (Dropout)            (None, 6, 6, 256)         0

conv4_1 (Conv2D)             (None, 6, 6, 256)         590080

batch_normalization_26 (Bat  (None, 6, 6, 256)         1024
chNormalization)

conv4_2 (Conv2D)             (None, 6, 6, 256)         590080

batch_normalization_27 (Bat  (None, 6, 6, 256)         1024
chNormalization)

conv4_3 (Conv2D)             (None, 6, 6, 256)         590080

batch_normalization_28 (Bat  (None, 6, 6, 256)         1024
chNormalization)

conv4_4 (Conv2D)             (None, 6, 6, 256)         590080

batch_normalization_29 (Bat  (None, 6, 6, 256)         1024
chNormalization)

pool4_1 (MaxPooling2D)       (None, 3, 3, 256)         0

drop4_1 (Dropout)            (None, 3, 3, 256)         0

conv5_1 (Conv2D)             (None, 3, 3, 512)         1180160

batch_normalization_30 (Bat  (None, 3, 3, 512)         2048
chNormalization)

conv5_2 (Conv2D)             (None, 3, 3, 512)         2359808

batch_normalization_31 (Bat  (None, 3, 3, 512)         2048
chNormalization)
```

```
 conv5_3 (Conv2D)            (None, 3, 3, 512)         2359808

 batch_normalization_32 (Bat  (None, 3, 3, 512)        2048
 chNormalization)

 conv5_4 (Conv2D)            (None, 3, 3, 512)         2359808

 pool5_1 (MaxPooling2D)      (None, 1, 1, 512)         0

 drop5_1 (Dropout)           (None, 1, 1, 512)         0

 flatten (Flatten)           (None, 512)               0

 output (Dense)              (None, 7)                 3591

=================================================================
Total params: 13,111,367
Trainable params: 13,103,431
Non-trainable params: 7,936
_________________________________________________________________
None

/usr/local/lib/python3.9/dist-packages/keras/optimizers/legacy/
adam.py:117: UserWarning: The `lr` argument is deprecated, use
`learning_rate` instead.
  super().__init__(name, **kwargs)

num_epochs = 20
history = model.fit_generator(train_flow,
                    steps_per_epoch=len(X_train) / batch_size,
                    epochs=num_epochs,
                    verbose=1,
                    validation_data=test_flow,
                    validation_steps=len(X_test) / batch_size)

Epoch 1/20

<ipython-input-29-a6284849a290>:2: UserWarning: `Model.fit_generator`
is deprecated and will be removed in a future version. Please use
`Model.fit`, which supports generators.
  history = model.fit_generator(train_flow,

9/9 [==============================] - 4s 232ms/step - loss: 2.5422 -
accuracy: 0.1447
Epoch 2/20
9/9 [==============================] - 1s 85ms/step - loss: 2.4343 -
accuracy: 0.1768
Epoch 3/20
9/9 [==============================] - 1s 85ms/step - loss: 2.4635 -
accuracy: 0.1736
```

```
Epoch 4/20
9/9 [==============================] - 1s 87ms/step - loss: 2.3531 -
accuracy: 0.1833
Epoch 5/20
9/9 [==============================] - 1s 86ms/step - loss: 2.3582 -
accuracy: 0.1656
Epoch 6/20
9/9 [==============================] - 1s 94ms/step - loss: 2.2823 -
accuracy: 0.1592
Epoch 7/20
9/9 [==============================] - 1s 93ms/step - loss: 2.2623 -
accuracy: 0.1865
Epoch 8/20
9/9 [==============================] - 1s 85ms/step - loss: 2.2433 -
accuracy: 0.2058
Epoch 9/20
9/9 [==============================] - 1s 85ms/step - loss: 2.2314 -
accuracy: 0.1640
Epoch 10/20
9/9 [==============================] - 1s 84ms/step - loss: 2.1718 -
accuracy: 0.1768
Epoch 11/20
9/9 [==============================] - 1s 85ms/step - loss: 2.2141 -
accuracy: 0.1785
Epoch 12/20
9/9 [==============================] - 1s 84ms/step - loss: 2.1275 -
accuracy: 0.1801
Epoch 13/20
9/9 [==============================] - 1s 84ms/step - loss: 2.1281 -
accuracy: 0.2074
Epoch 14/20
9/9 [==============================] - 1s 85ms/step - loss: 2.1827 -
accuracy: 0.1704
Epoch 15/20
9/9 [==============================] - 1s 85ms/step - loss: 2.0485 -
accuracy: 0.2026
Epoch 16/20
9/9 [==============================] - 1s 86ms/step - loss: 2.0855 -
accuracy: 0.1833
Epoch 17/20
9/9 [==============================] - 1s 92ms/step - loss: 2.0009 -
accuracy: 0.2010
Epoch 18/20
9/9 [==============================] - 1s 94ms/step - loss: 1.9528 -
accuracy: 0.2010
Epoch 19/20
9/9 [==============================] - 1s 87ms/step - loss: 1.9734 -
accuracy: 0.1945
Epoch 20/20
```

```
9/9 [==============================] - 1s 85ms/step - loss: 1.9950 -
accuracy: 0.1897
```

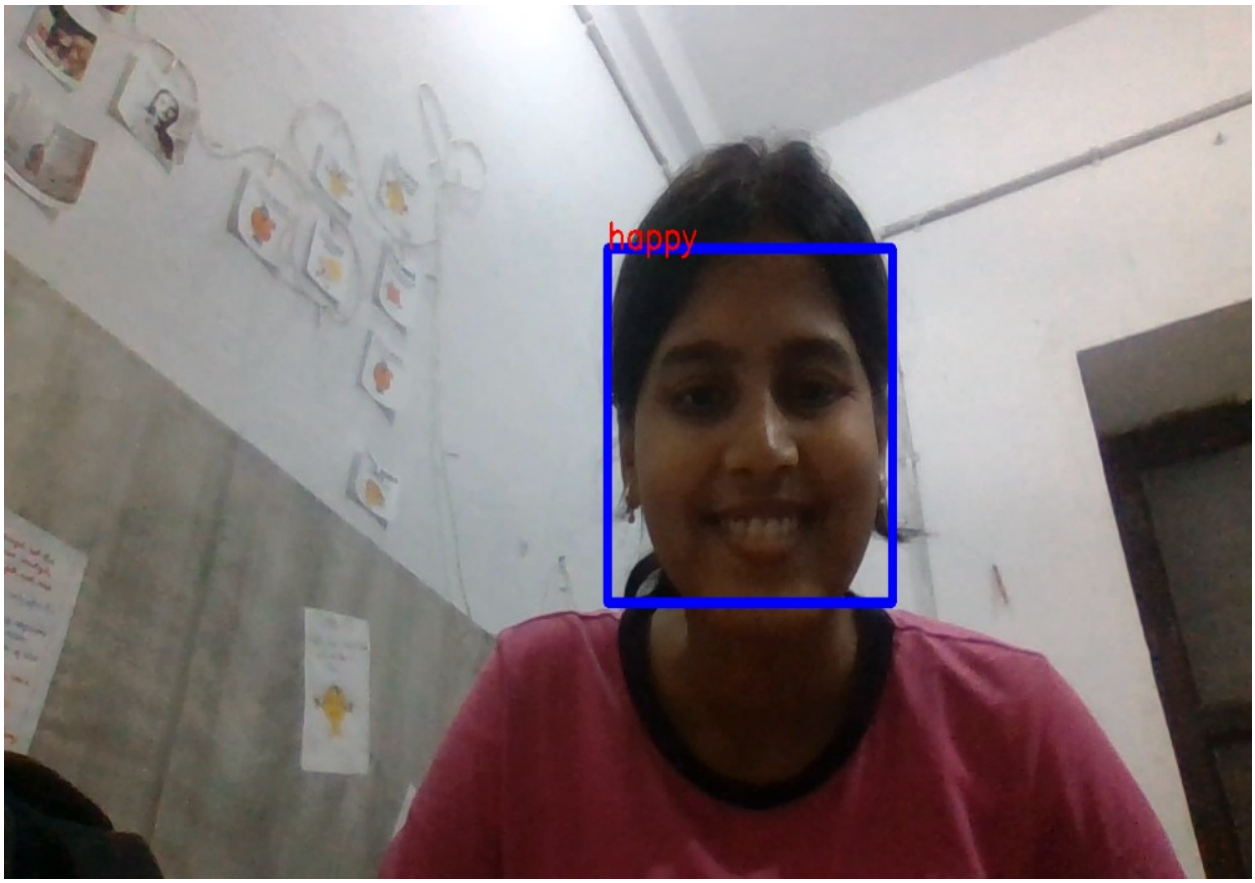
```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

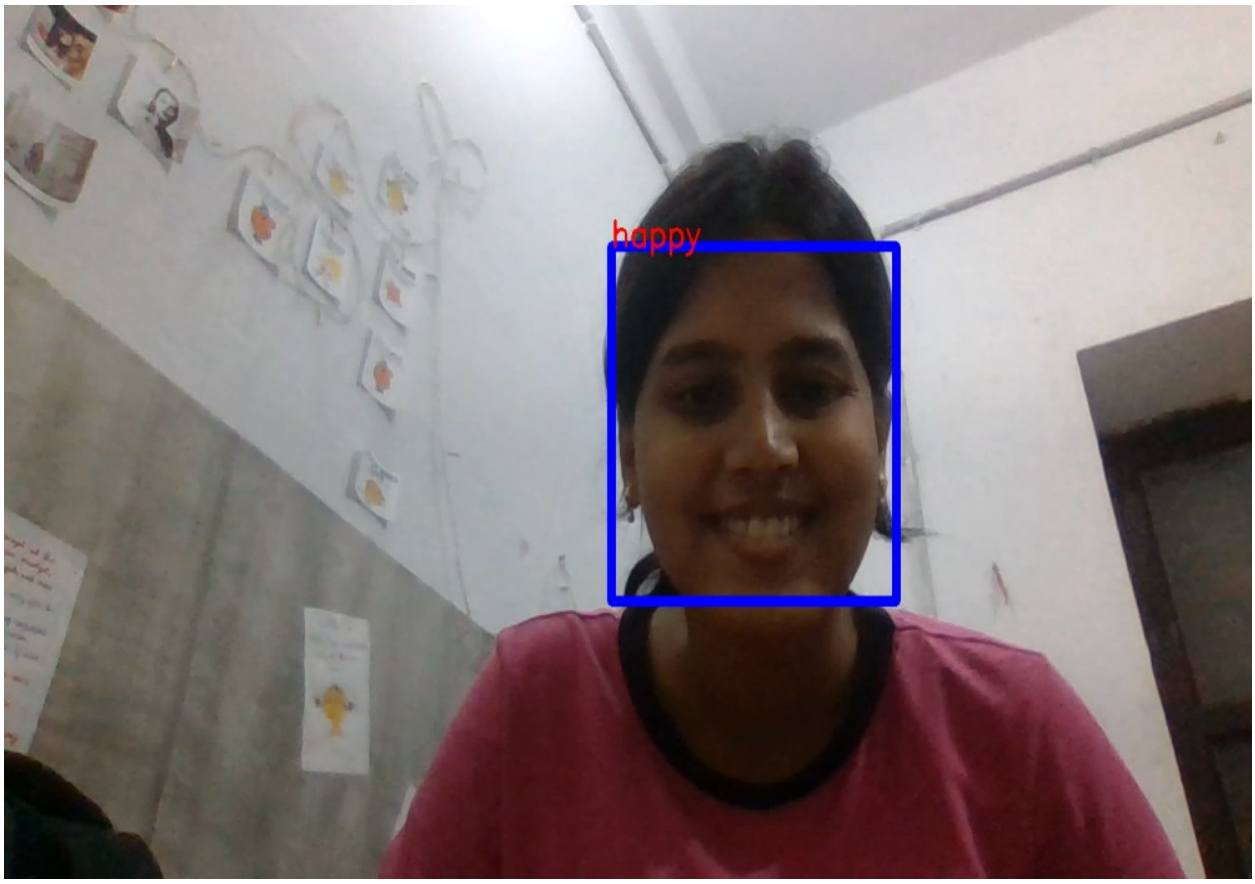
```
import os
import cv2
import numpy as np
from keras.models import model_from_json
#from keras.preprocessing import image
import keras.utils as image

model = model_from_json(open("model.json", "r").read())
model.load_weights('model.h5')
face_haar_cascade =
cv2.CascadeClassifier('/content/haarcascade_frontalface_default.xml')

from google.colab.patches import cv2_imshow


cap = cv2.VideoCapture('/content/WIN_20230401_21_02_16_Pro.mp4')
while True:
    ret,test_img=cap.read()# captures frame and returns boolean value
and captured image
    if not ret:
        continue
    gray_img= cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)

    faces_detected = face_haar_cascade.detectMultiScale(gray_img,
1.32, 5)


    for (x,y,w,h) in faces_detected:
        cv2.rectangle(test_img,(x,y),(x+w,y+h),(255,0,0),thickness=7)
        roi_gray=gray_img[y:y+w,x:x+h]#cropping region of interest
i.e. face area from  image
        roi_gray=cv2.resize(roi_gray,(48,48))
        img_pixels = image.img_to_array(roi_gray)
        img_pixels = np.expand_dims(img_pixels, axis = 0)
        img_pixels /= 255

        predictions = model.predict(img_pixels)

        #find max indexed array
        max_index = np.argmax(predictions[0])
```

```
        emotions = ('angry', 'disgust', 'fear', 'happy', 'sad',
'surprise', 'neutral')
        predicted_emotion = emotions[max_index]

        cv2.putText(test_img, predicted_emotion, (int(x), int(y)),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

    resized_img = cv2.resize(test_img, (1000, 700))
    #cv2.imshow('Facial emotion analysis ',resized_img)
    cv2_imshow(resized_img)



    if cv2.waitKey(10) == ord('q'):#wait until 'q' key is pressed
        break

cap.release()
cv2.destroyAllWindows
```

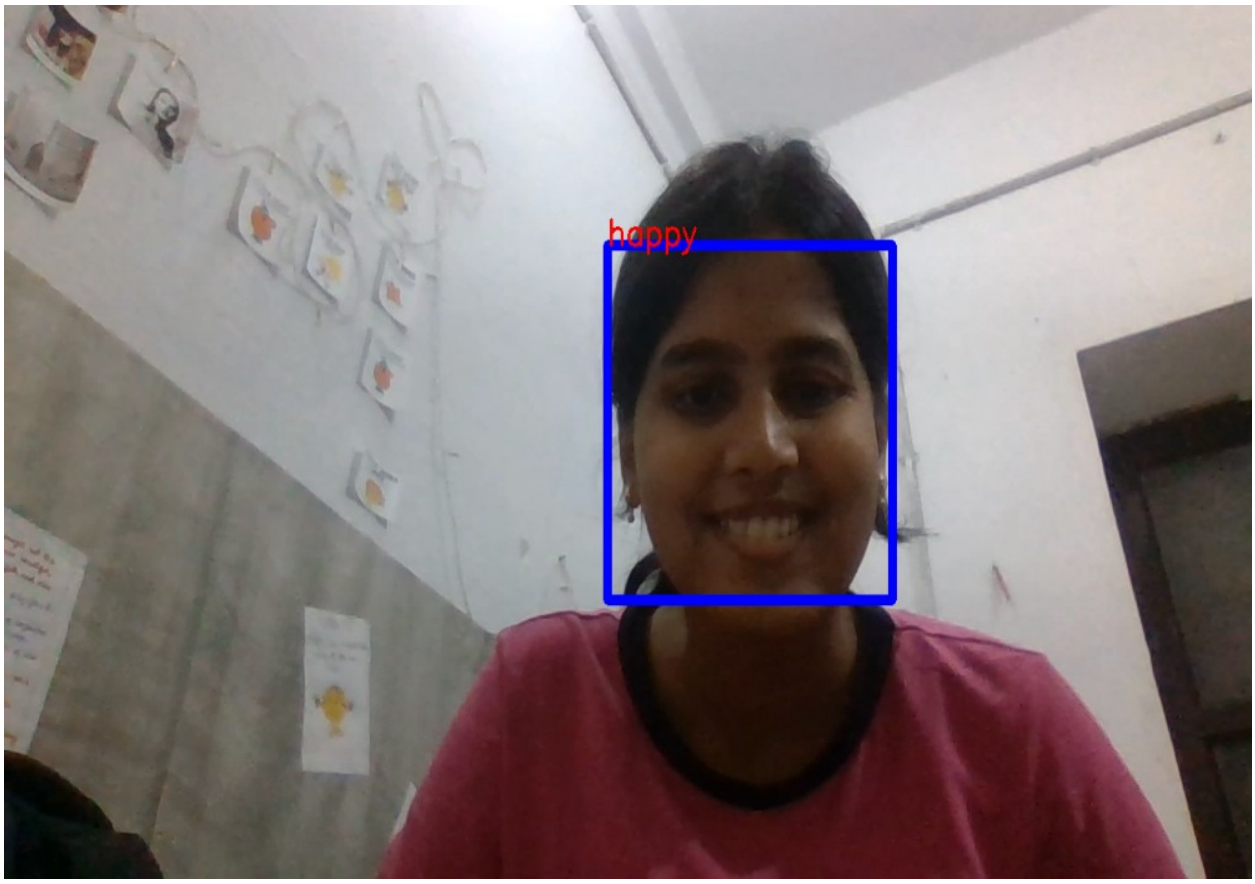
```
1/1 [==============================] - 1s 645ms/step
```



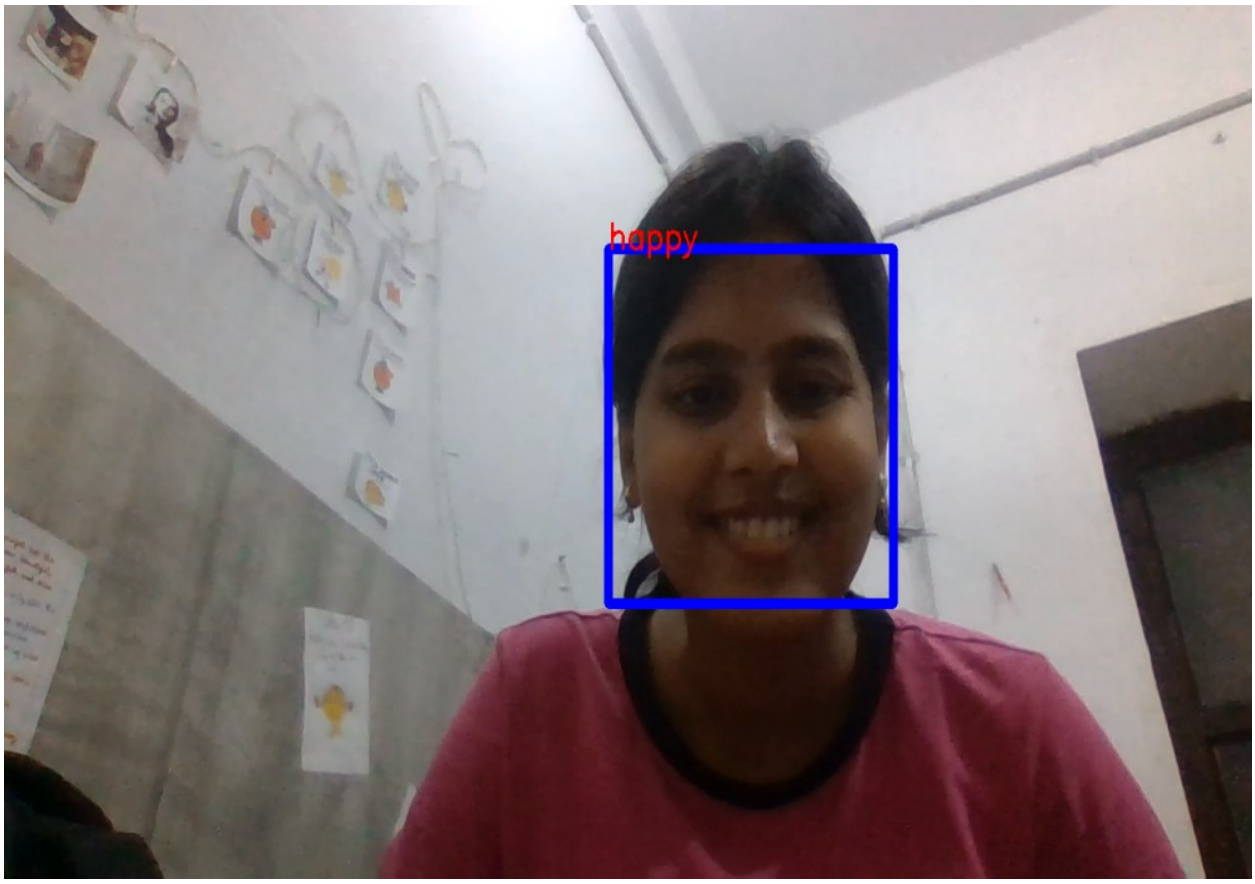

```
1/1 [==============================] - 0s 28ms/step
```



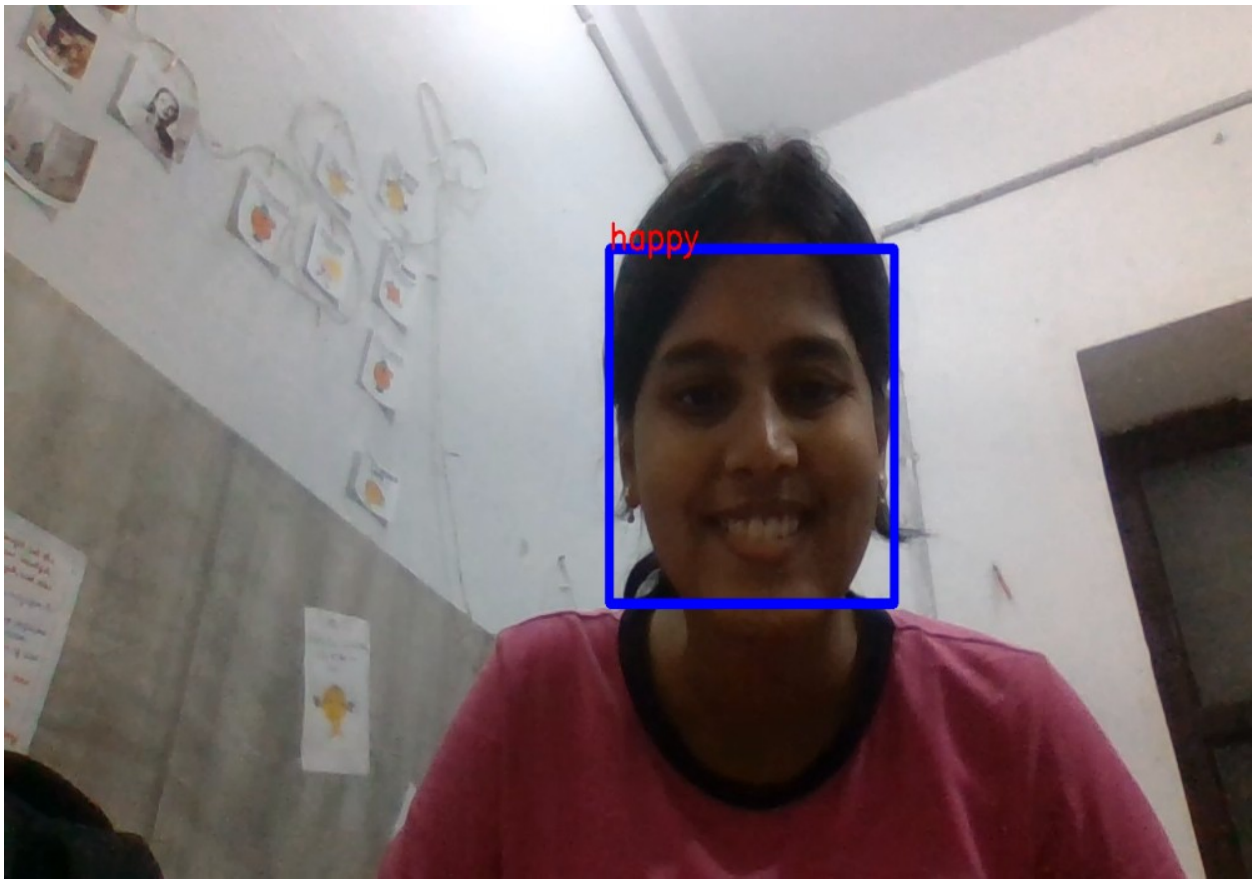

```
1/1 [==============================] - 0s 23ms/step
```



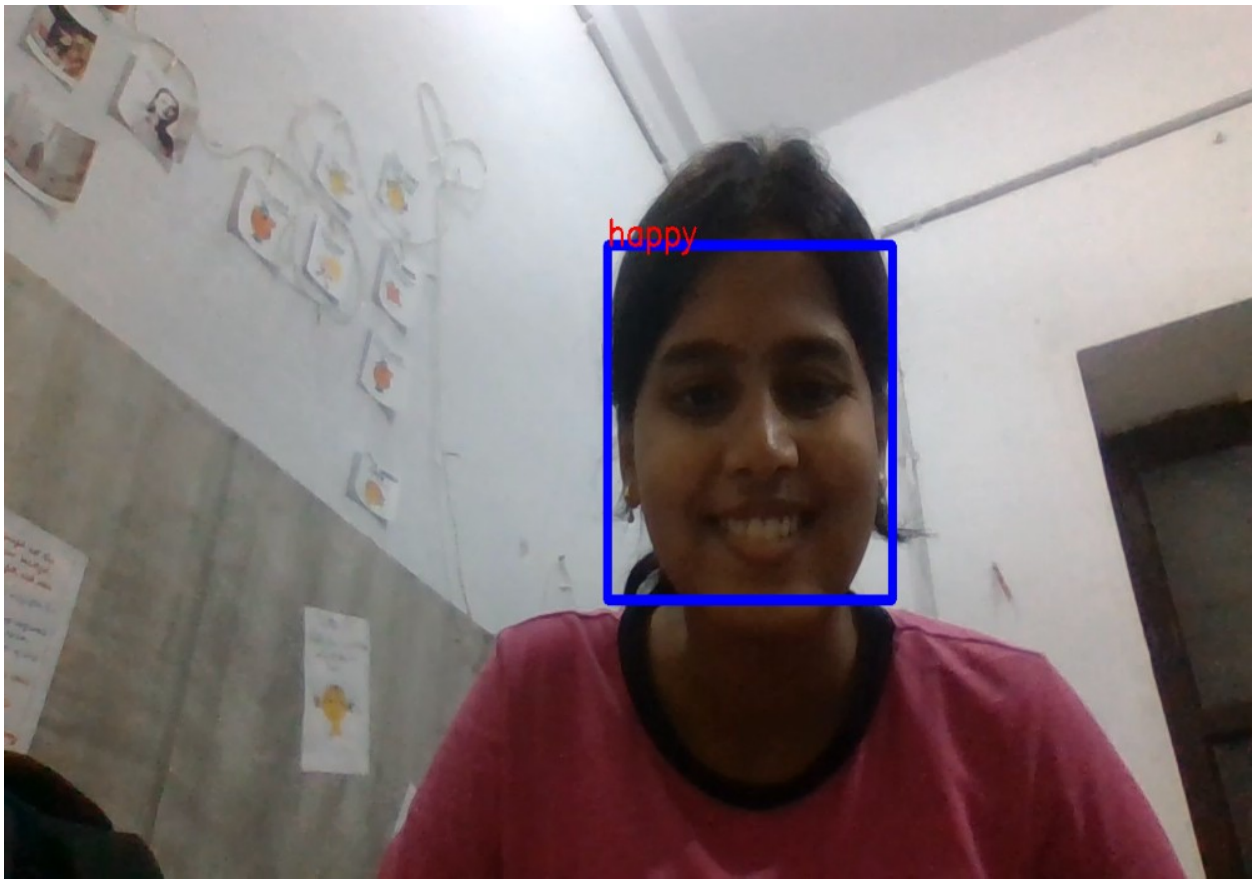

```
1/1 [==============================] - 0s 36ms/step
```



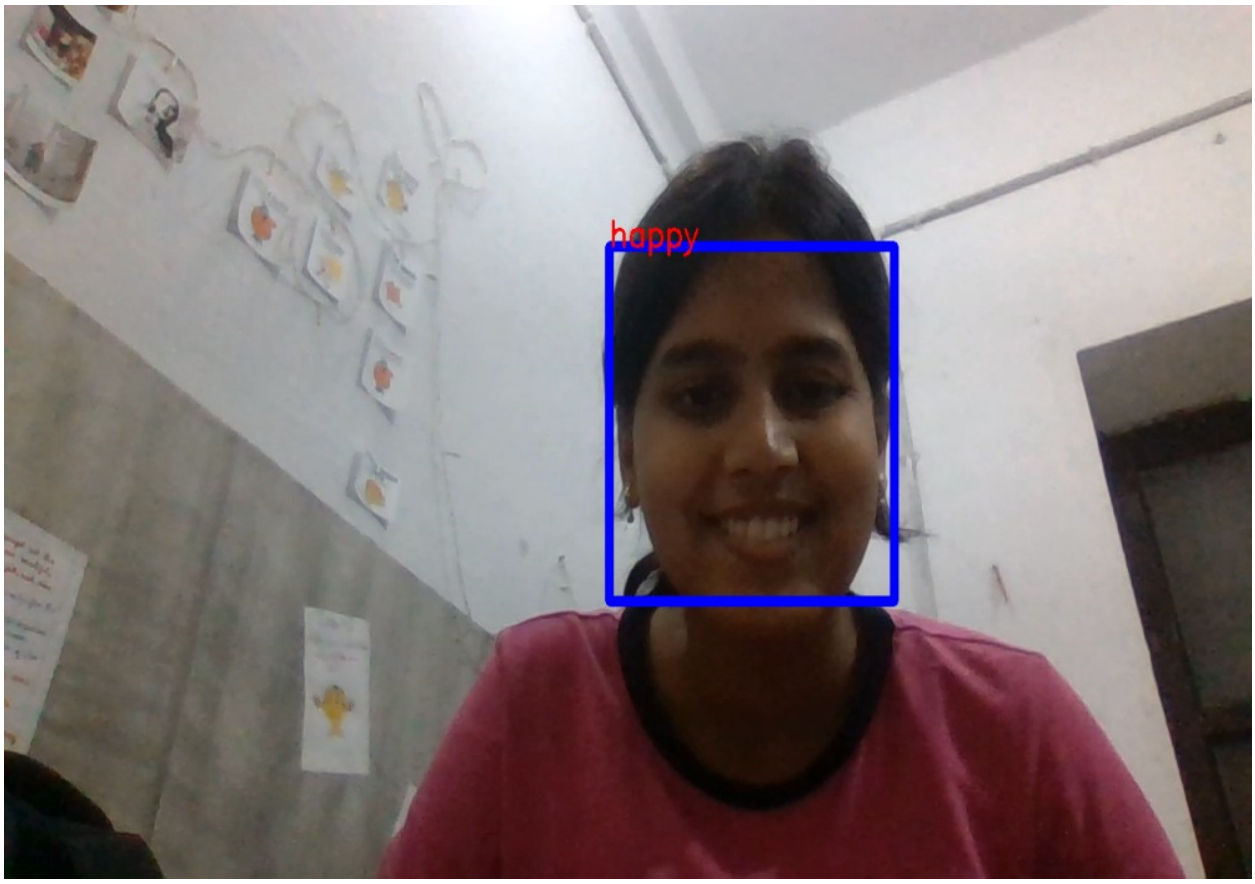

```
1/1 [==============================] - 0s 19ms/step
```



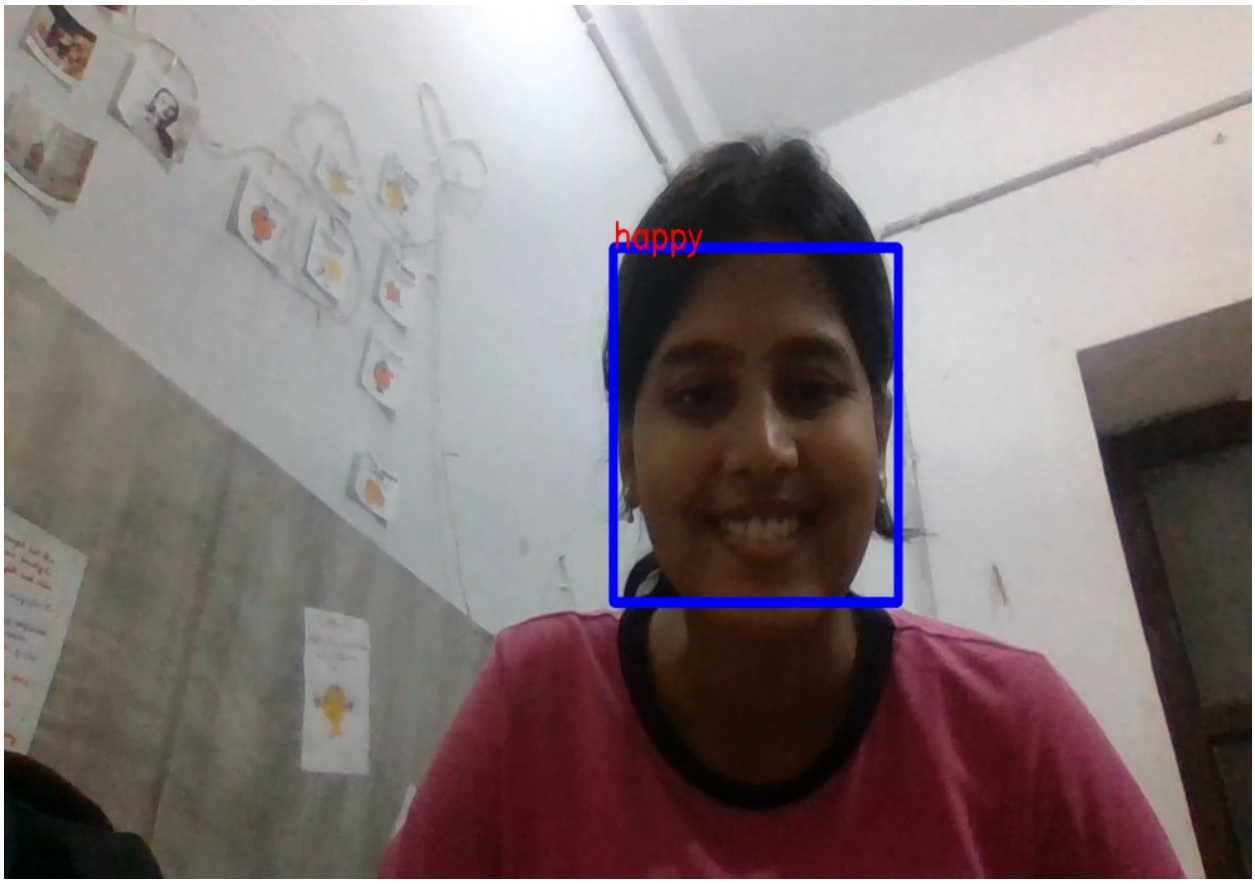

```
1/1 [==============================] - 0s 24ms/step
```



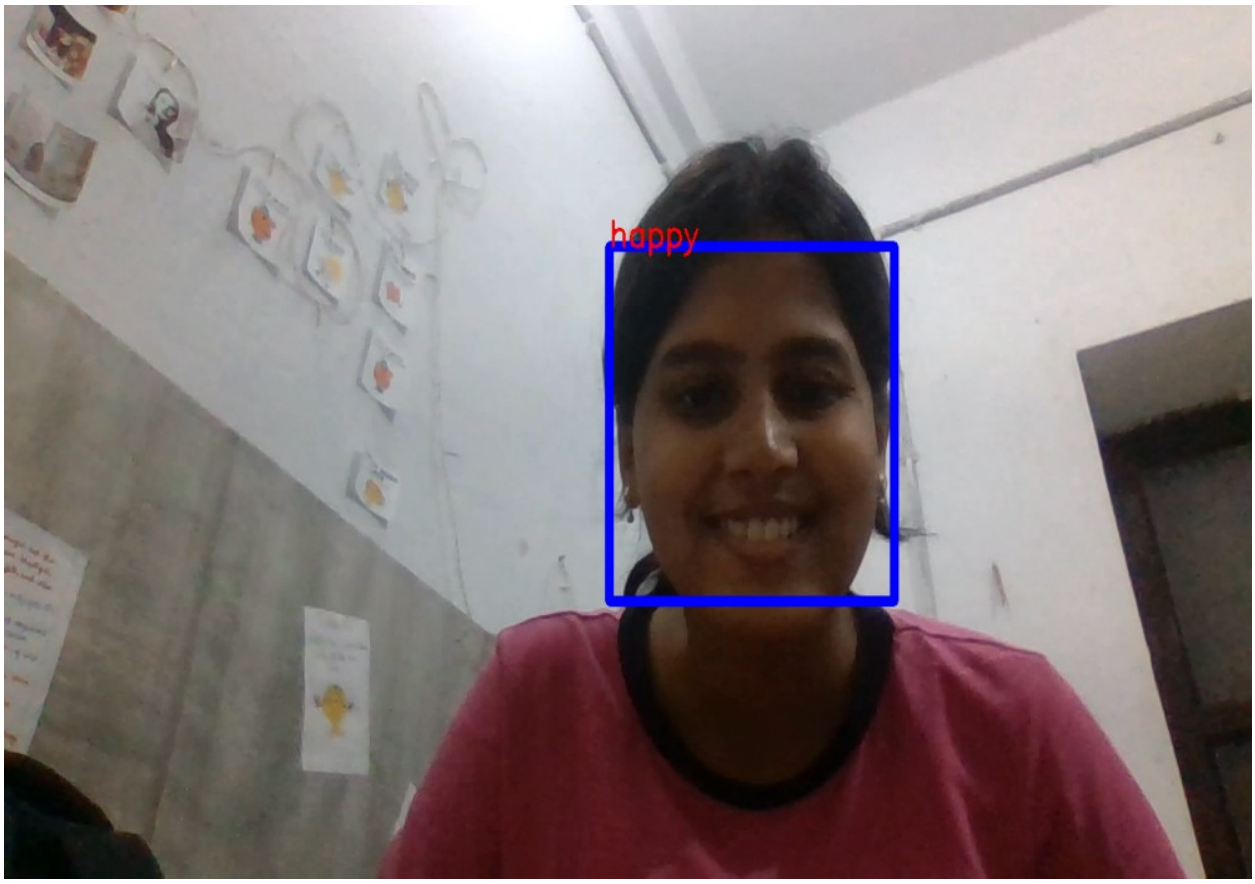

```
1/1 [==============================] - 0s 19ms/step
```



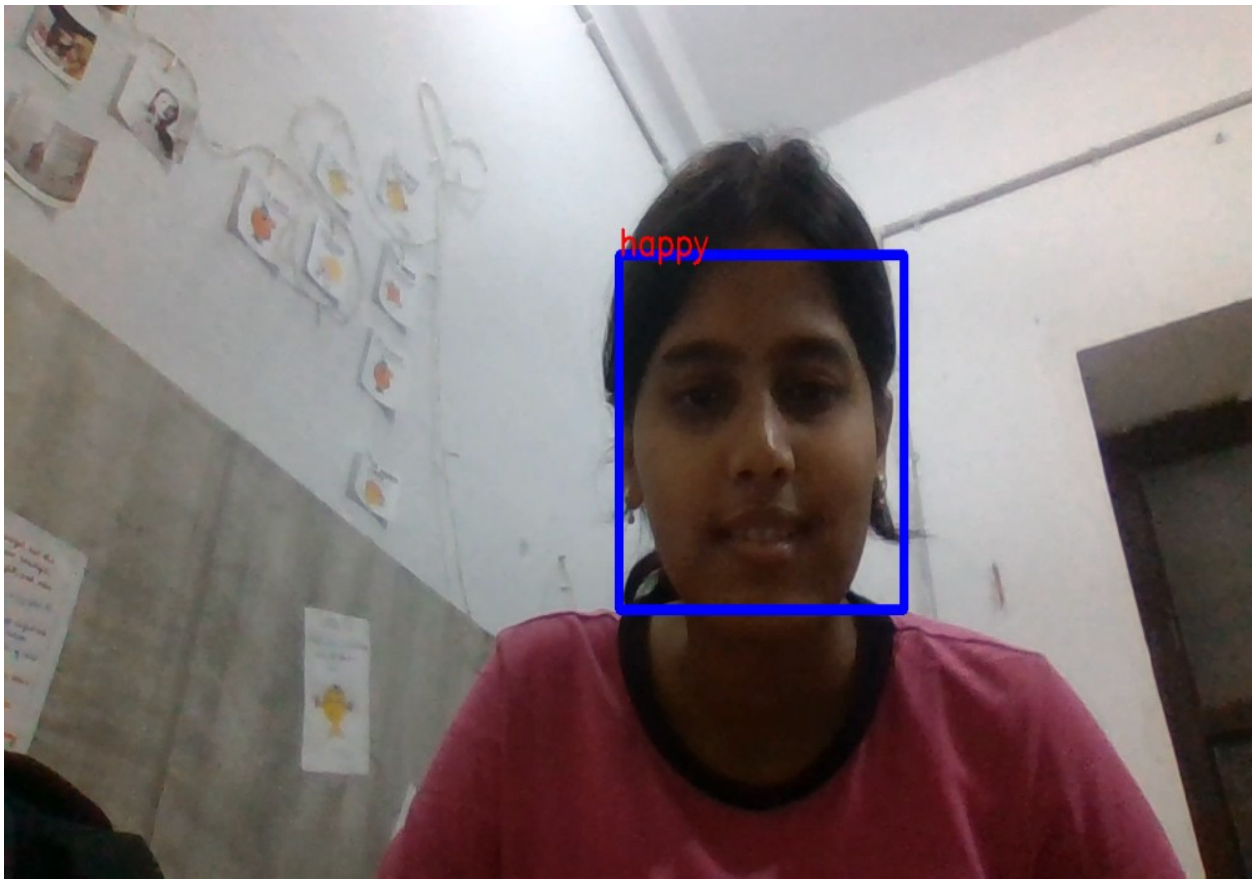

```
1/1 [==============================] - 0s 31ms/step
```



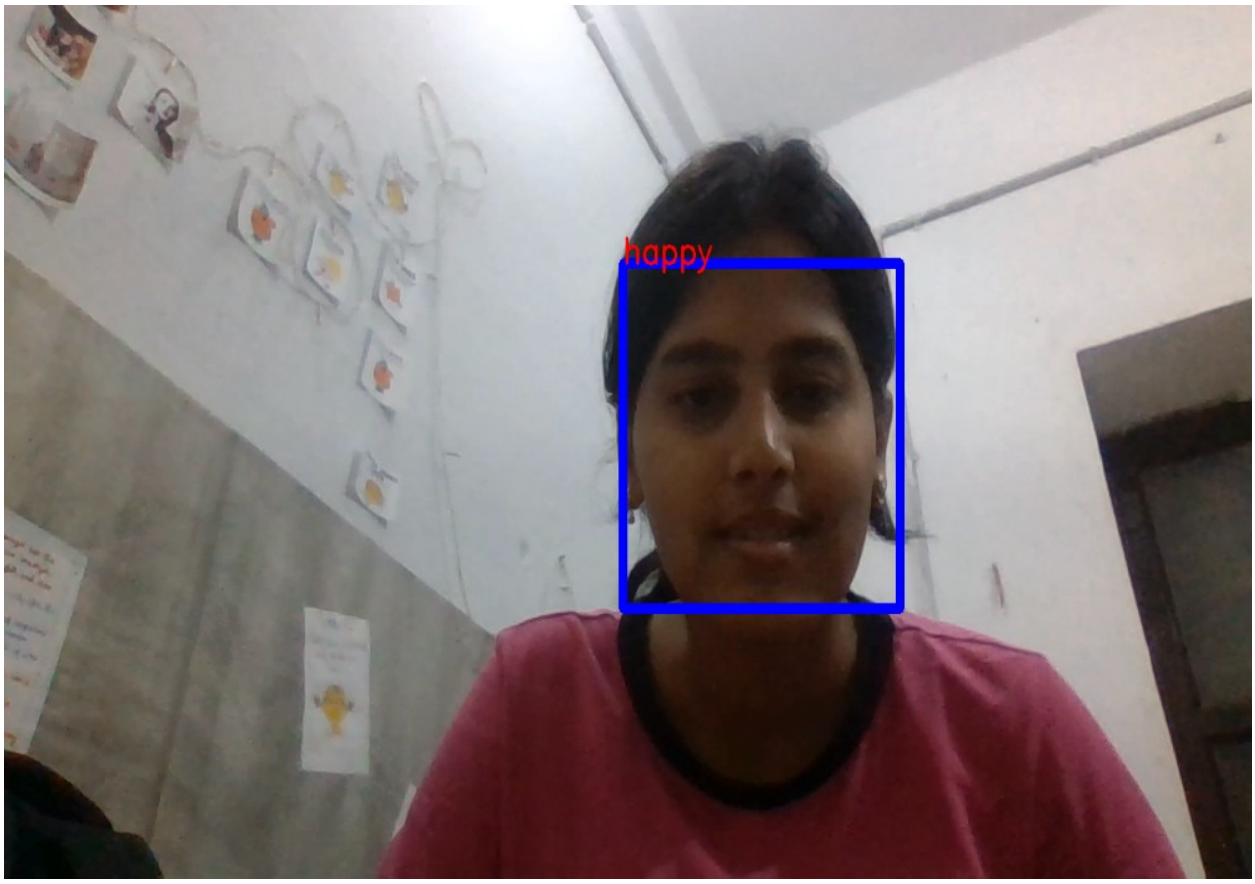

```
1/1 [==============================] - 0s 29ms/step
```



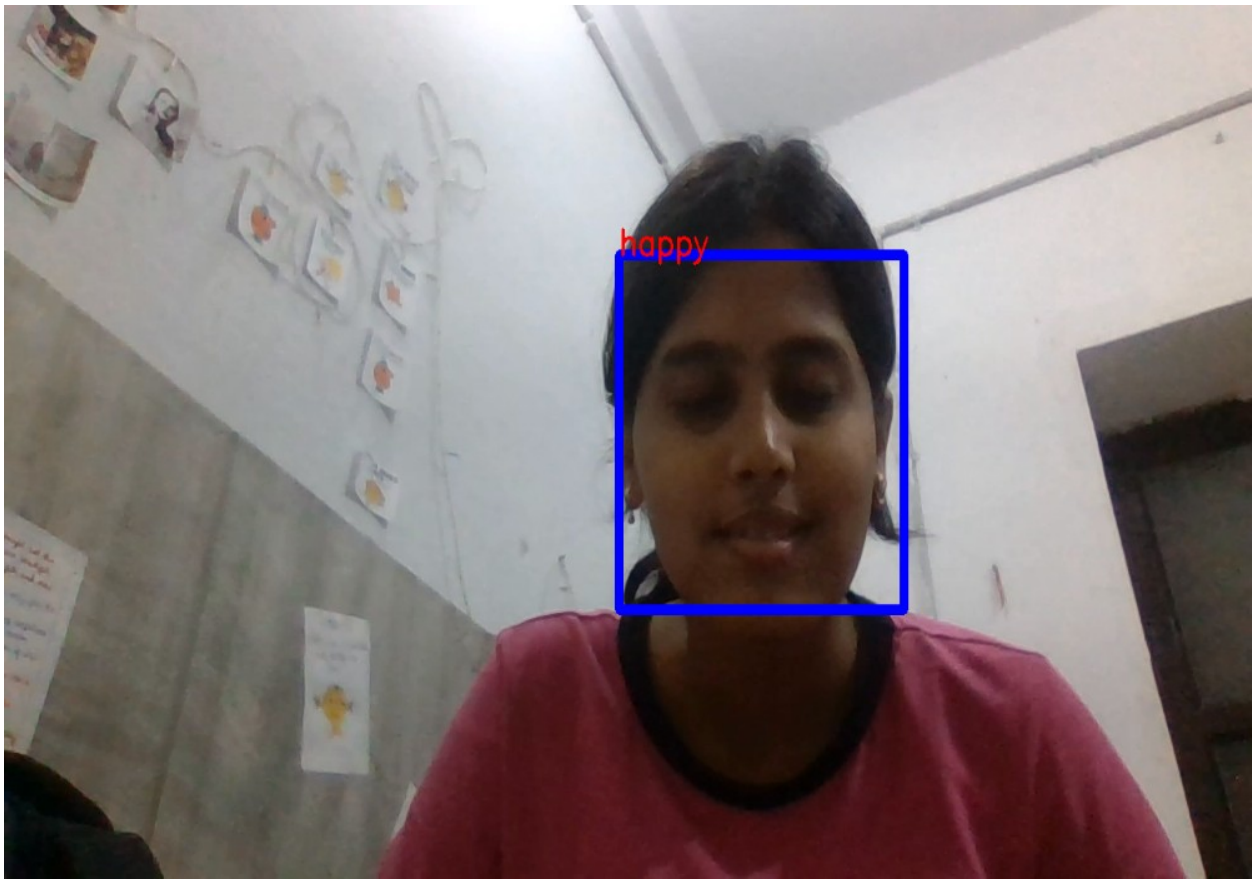

```
1/1 [==============================] - 0s 33ms/step
```



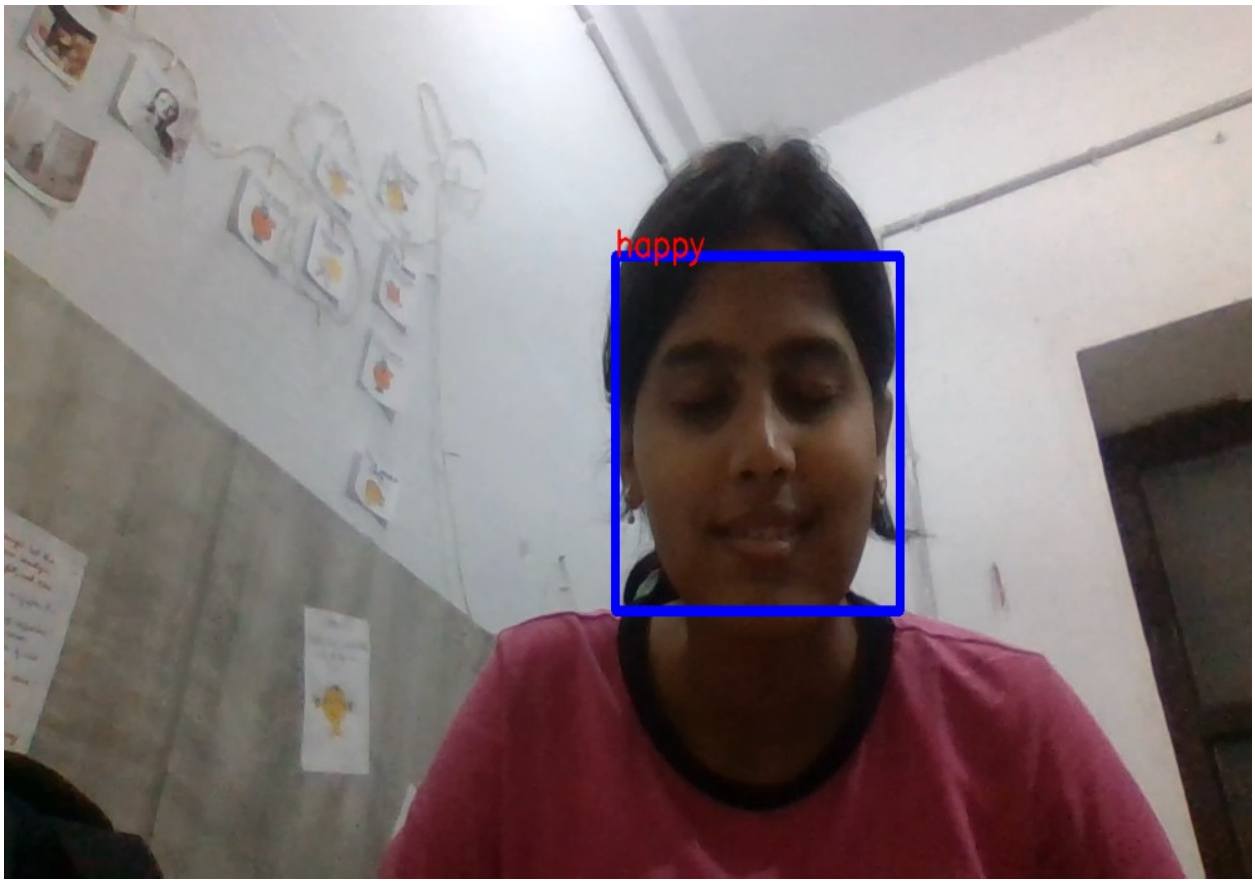

```
1/1 [==============================] - 0s 33ms/step
```



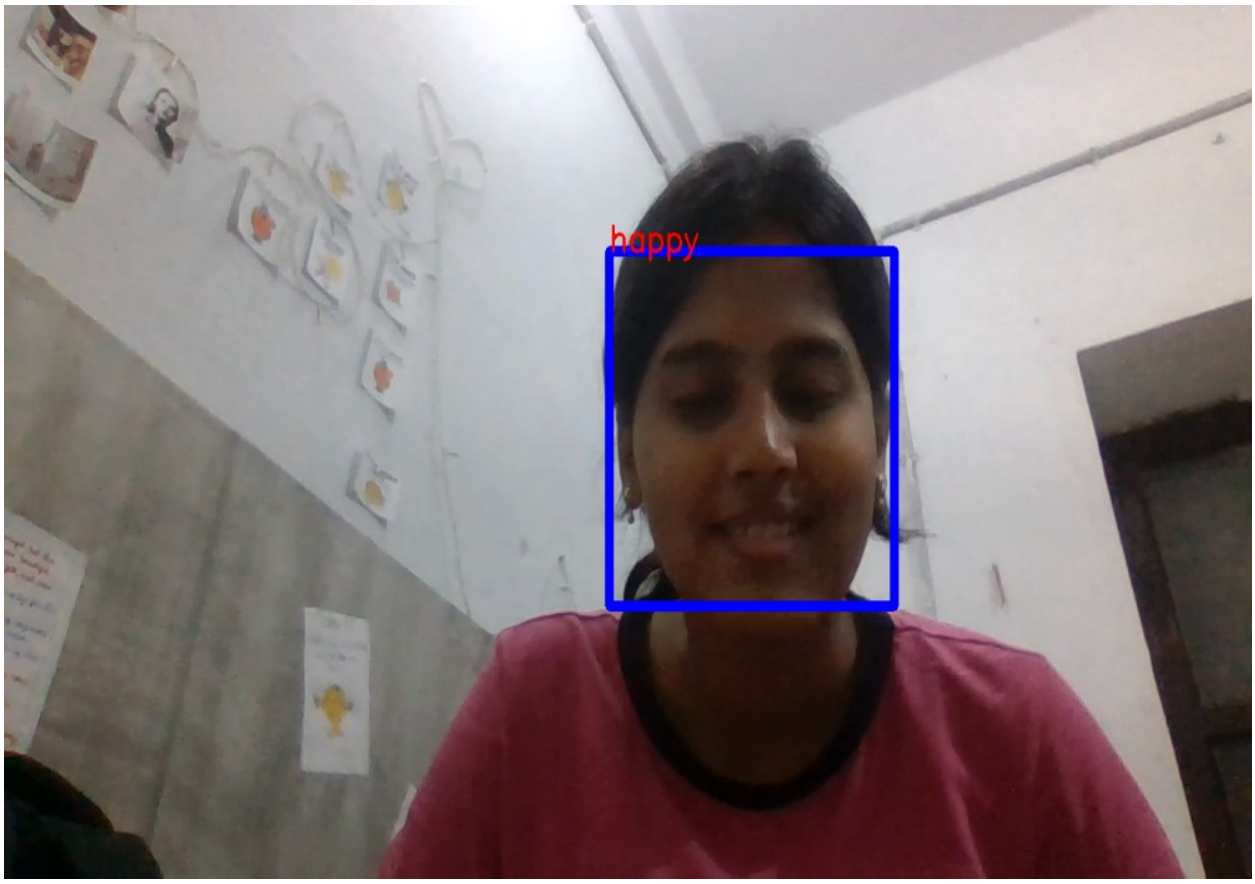

```
1/1 [==============================] - 0s 23ms/step
```



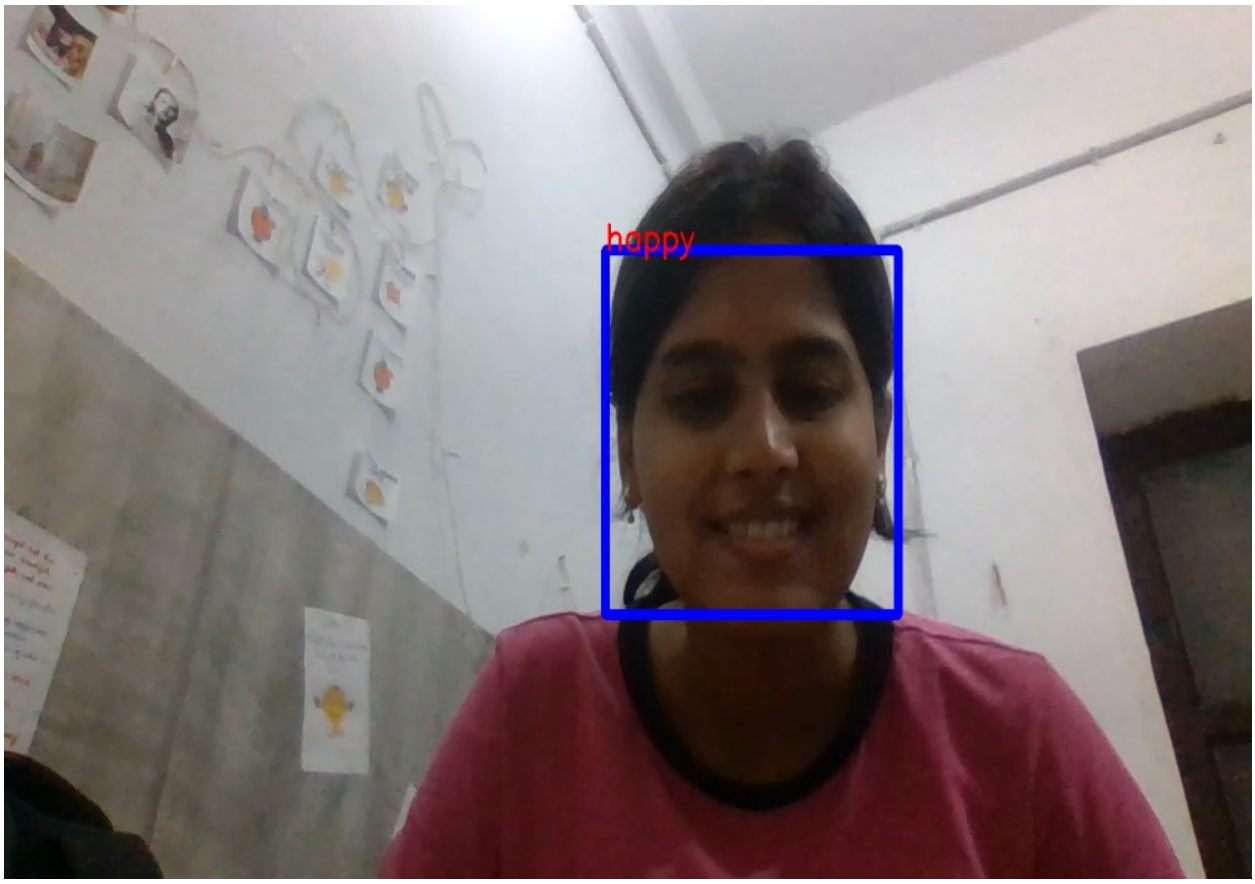

```
1/1 [==============================] - 0s 38ms/step
```



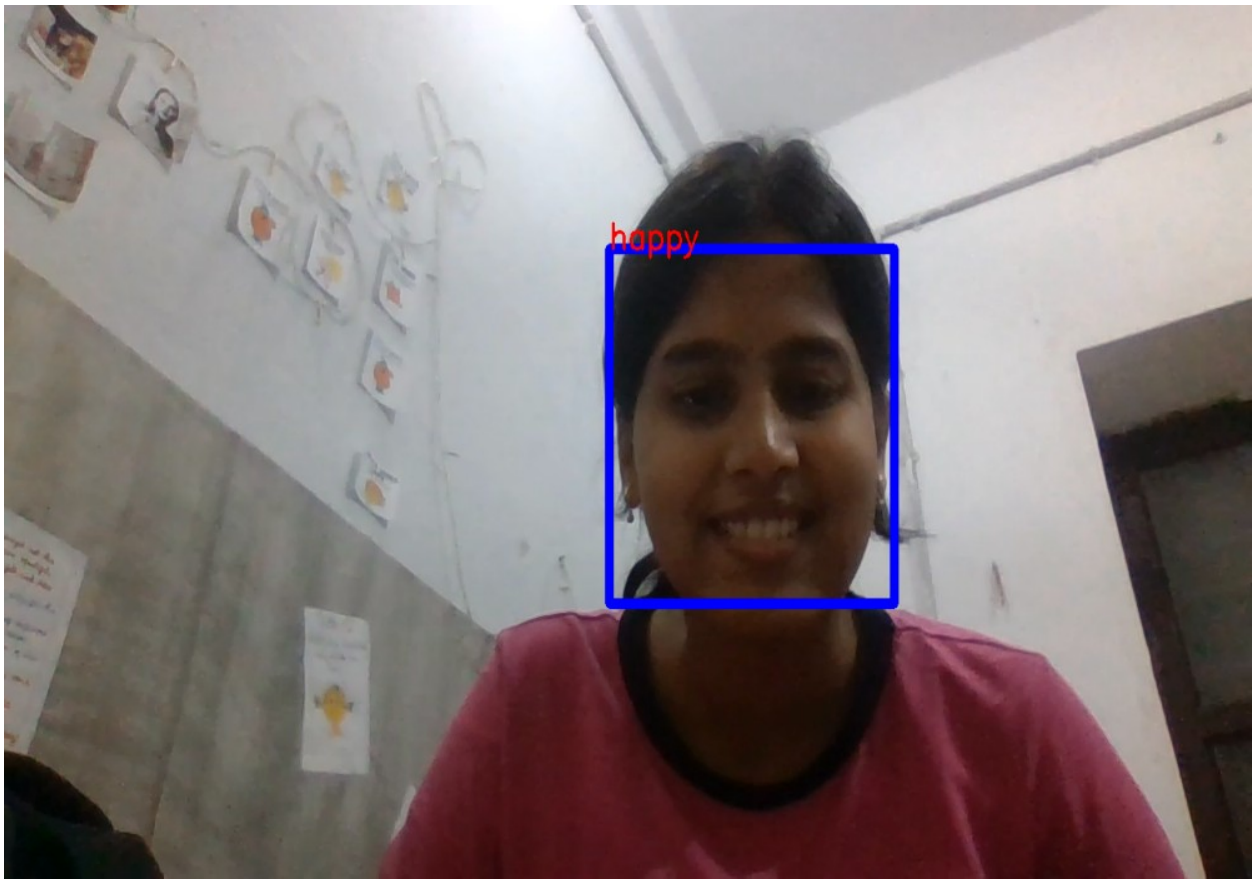

```
1/1 [==============================] - 0s 38ms/step
```



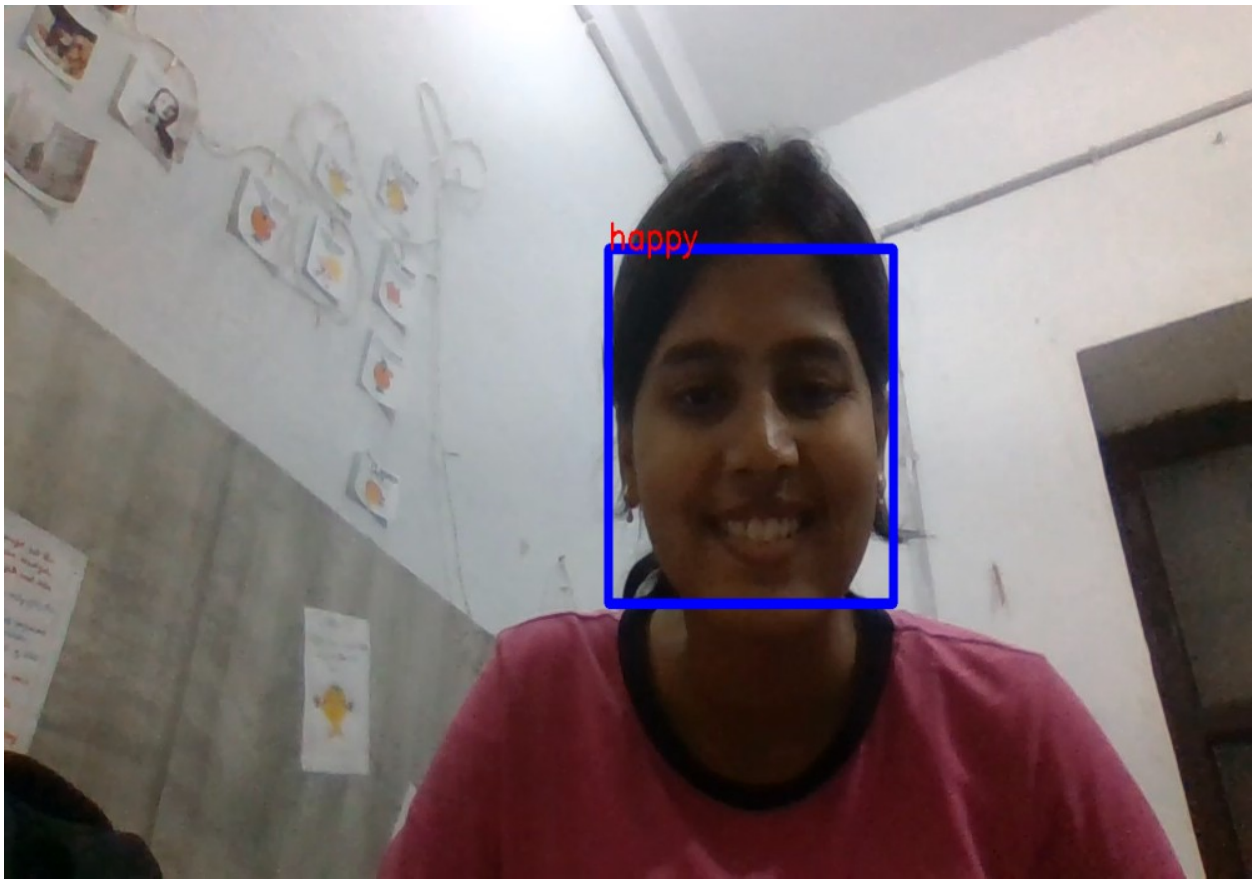

```
1/1 [==============================] - 0s 36ms/step
```



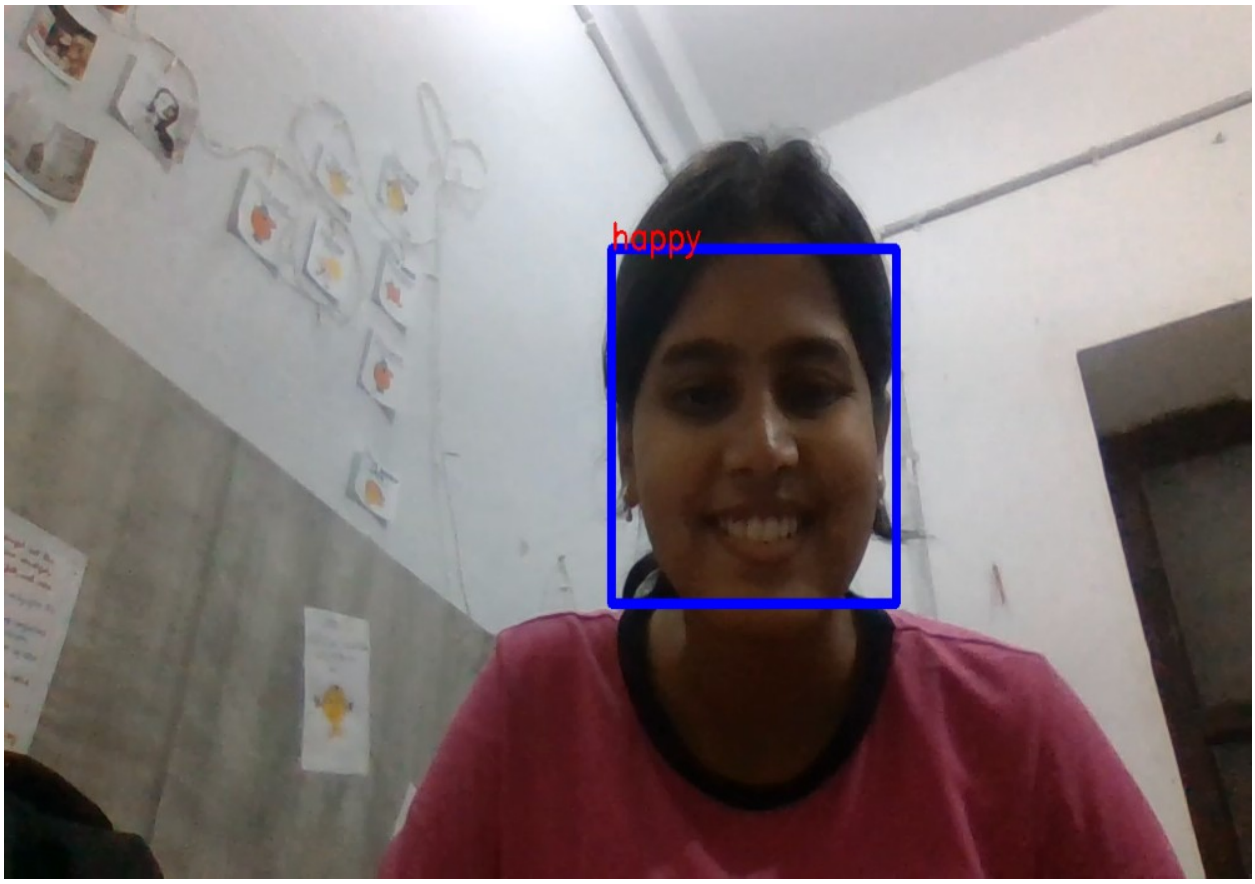

```
1/1 [==============================] - 0s 44ms/step
```



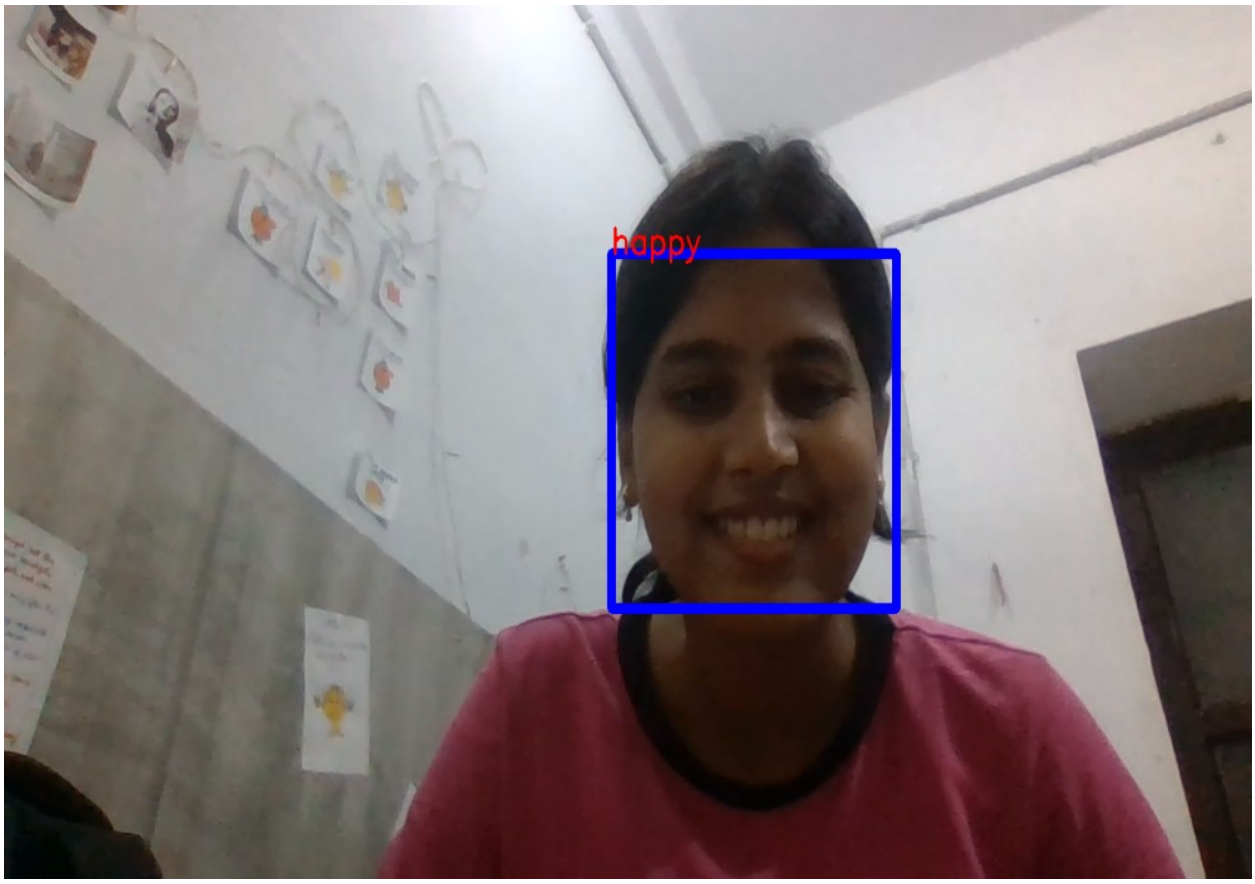

```
1/1 [==============================] - 0s 41ms/step
```



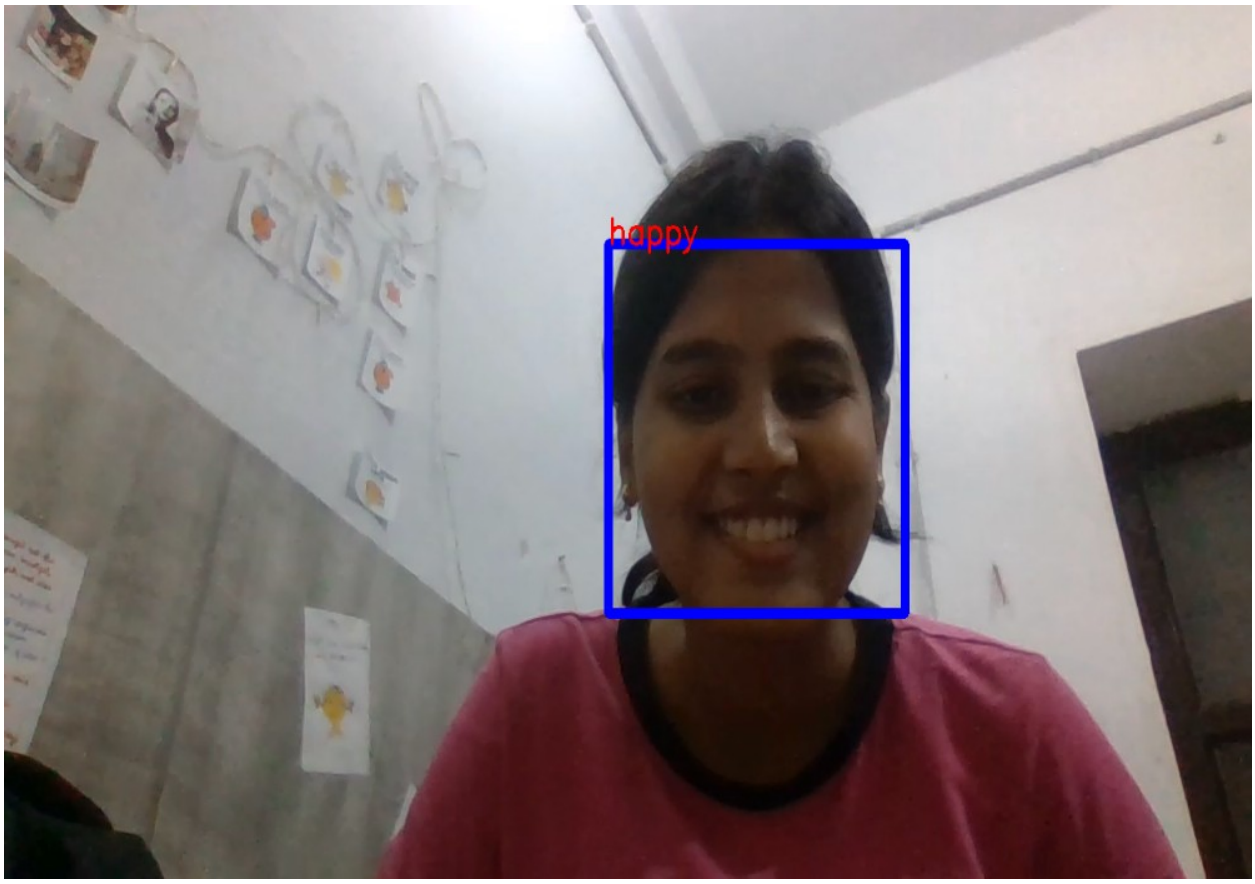

```
1/1 [==============================] - 0s 54ms/step
```



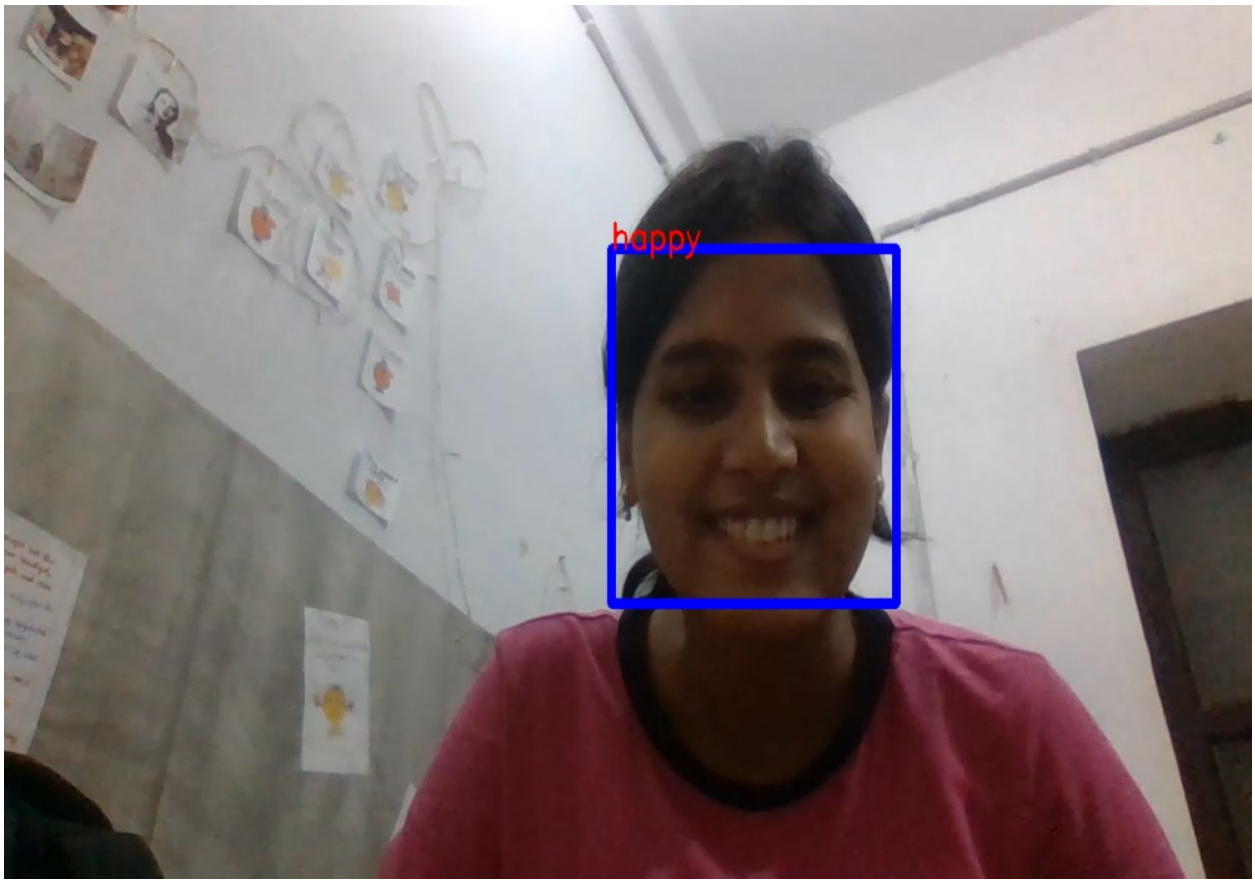

```
1/1 [==============================] - 0s 37ms/step
```



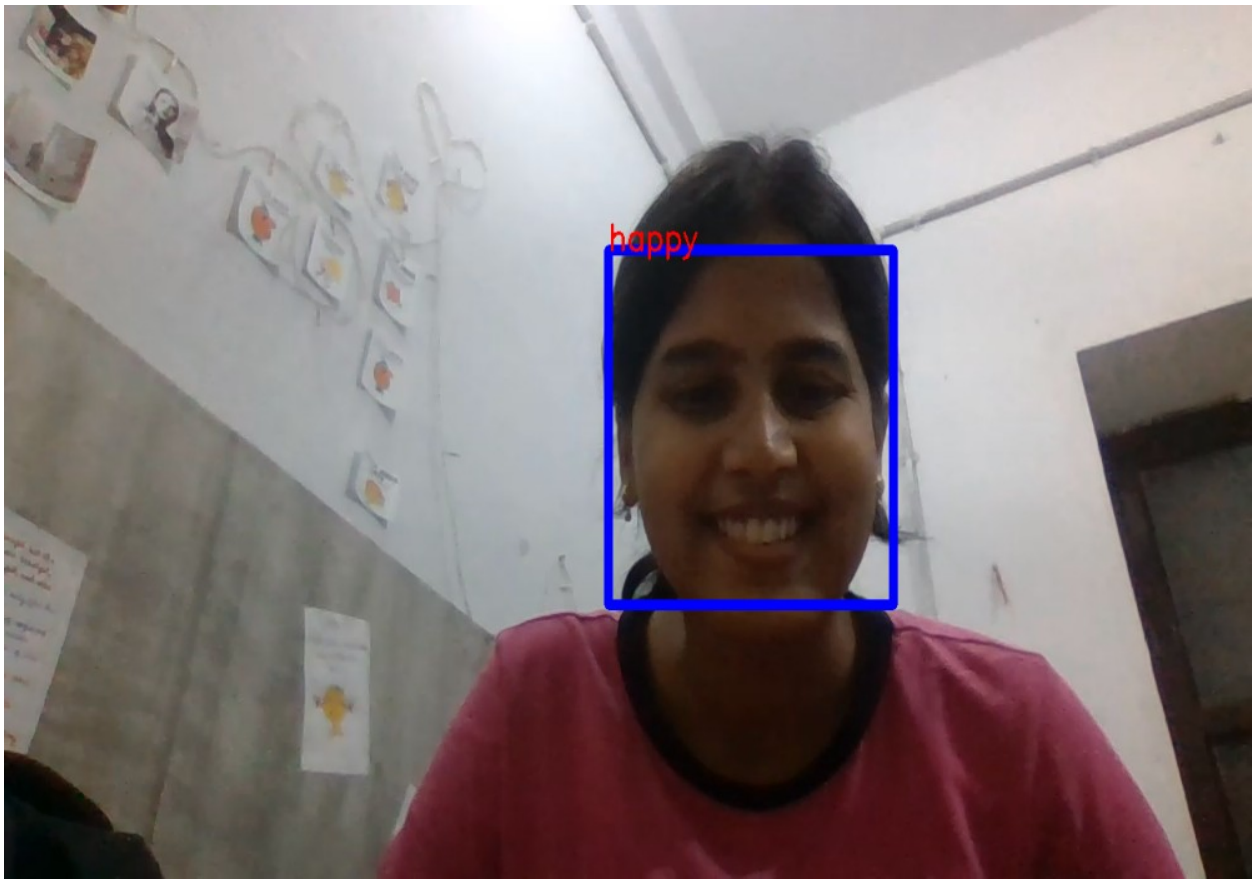

```
1/1 [==============================] - 0s 85ms/step
```



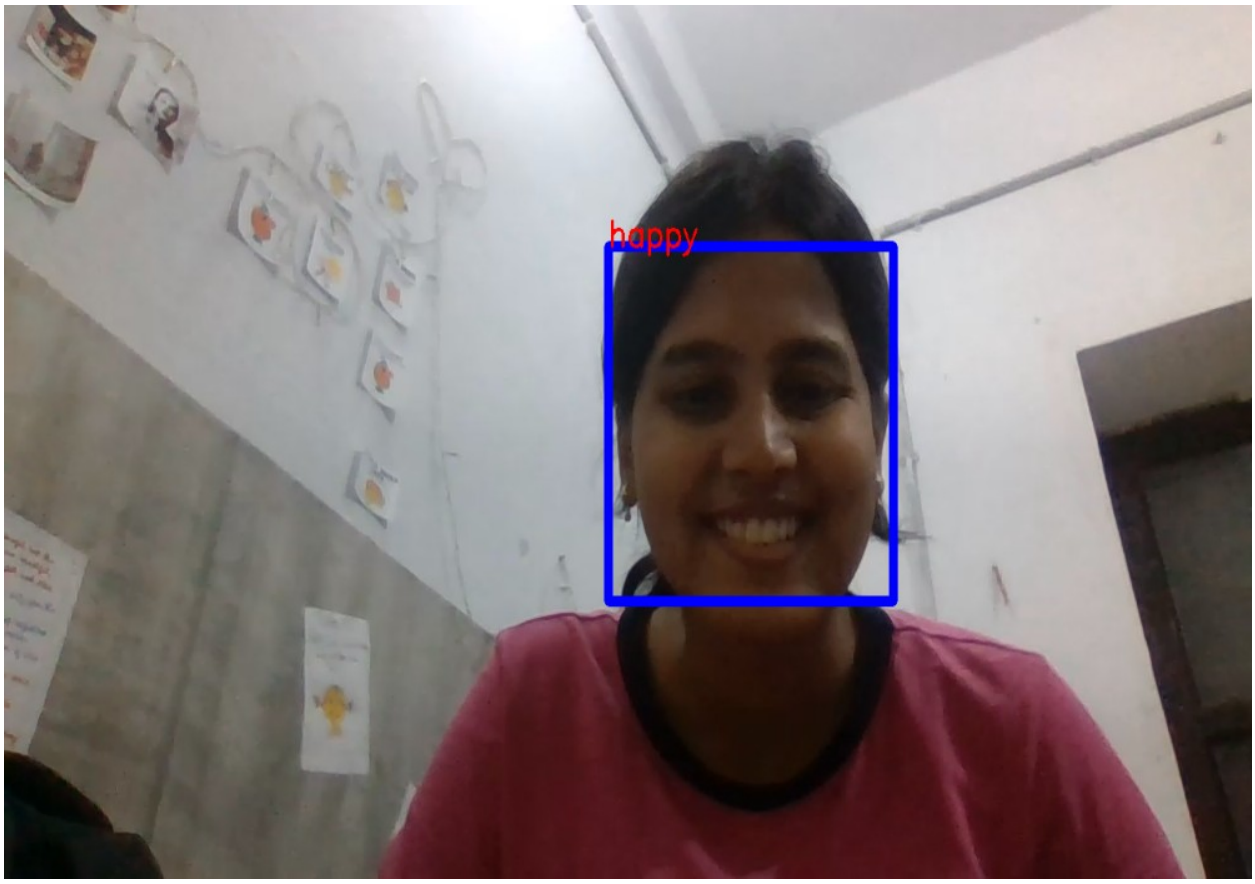

```
1/1 [==============================] - 0s 48ms/step
```



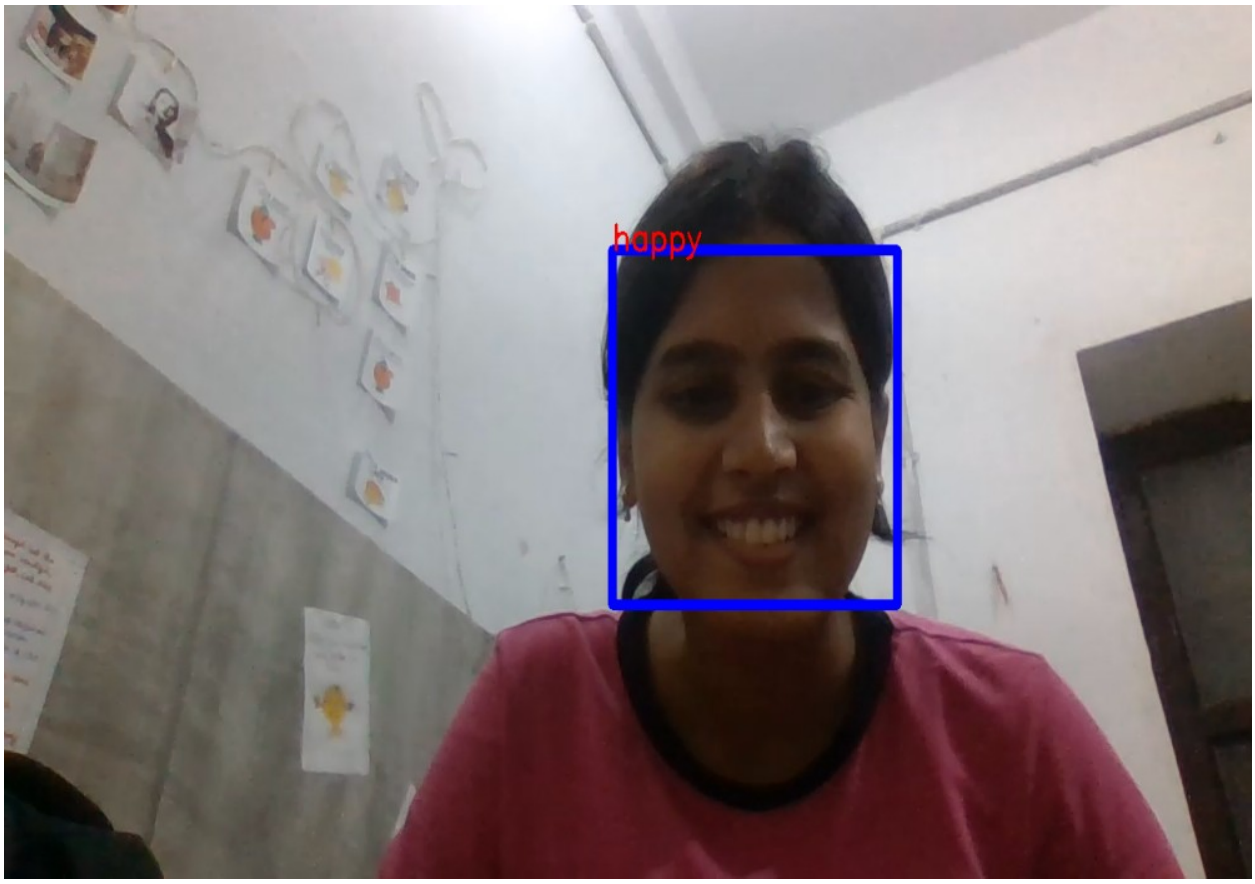

```
1/1 [==============================] - 0s 30ms/step
```



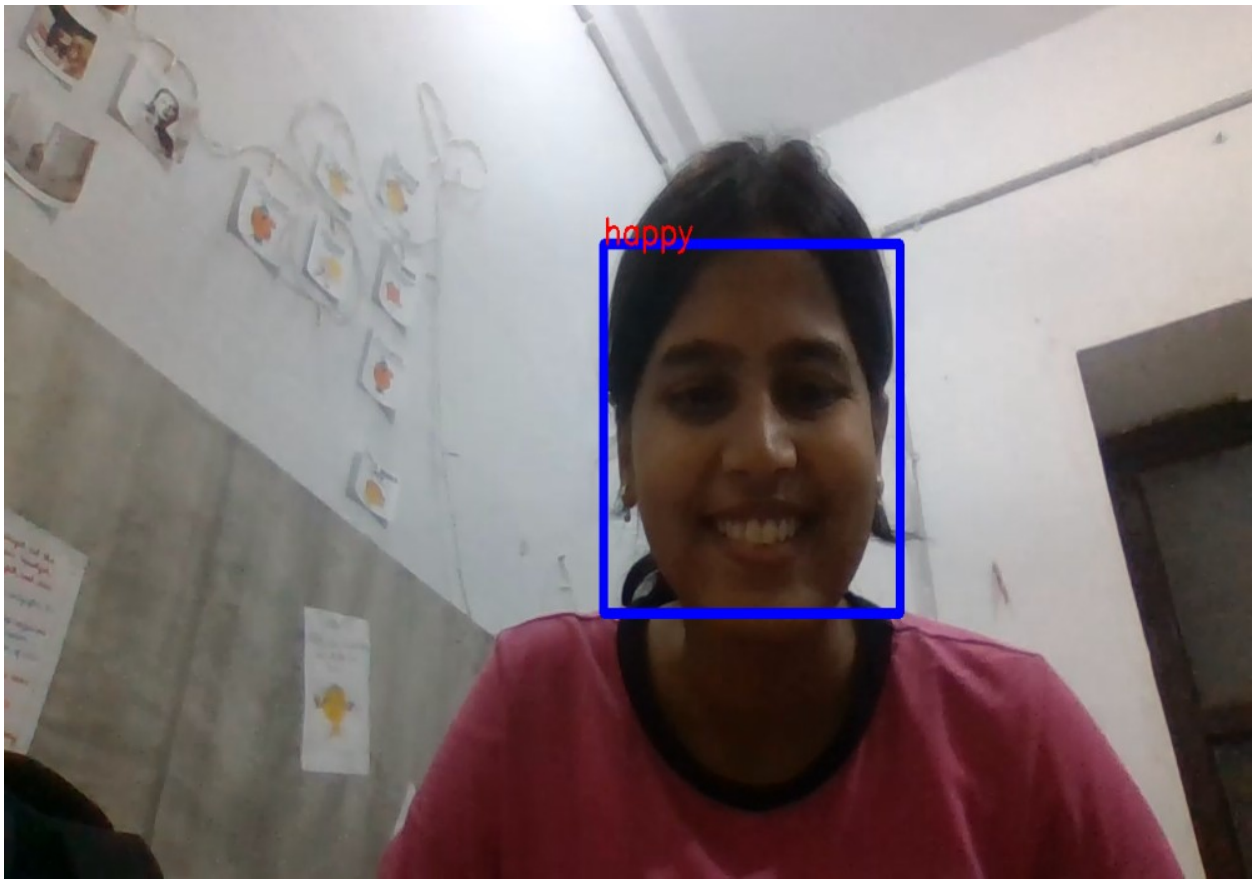

```
1/1 [==============================] - 0s 28ms/step
```



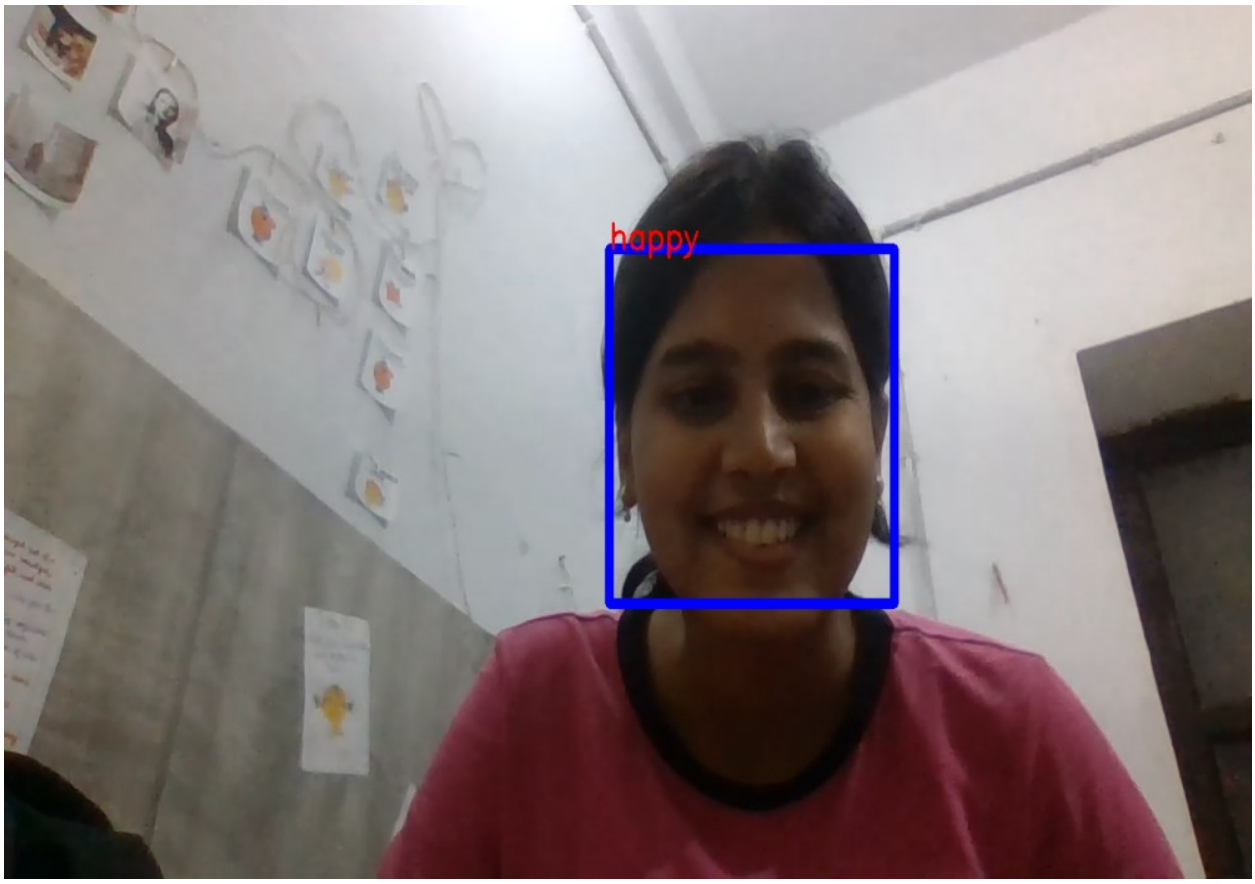

```
1/1 [==============================] - 0s 31ms/step
```



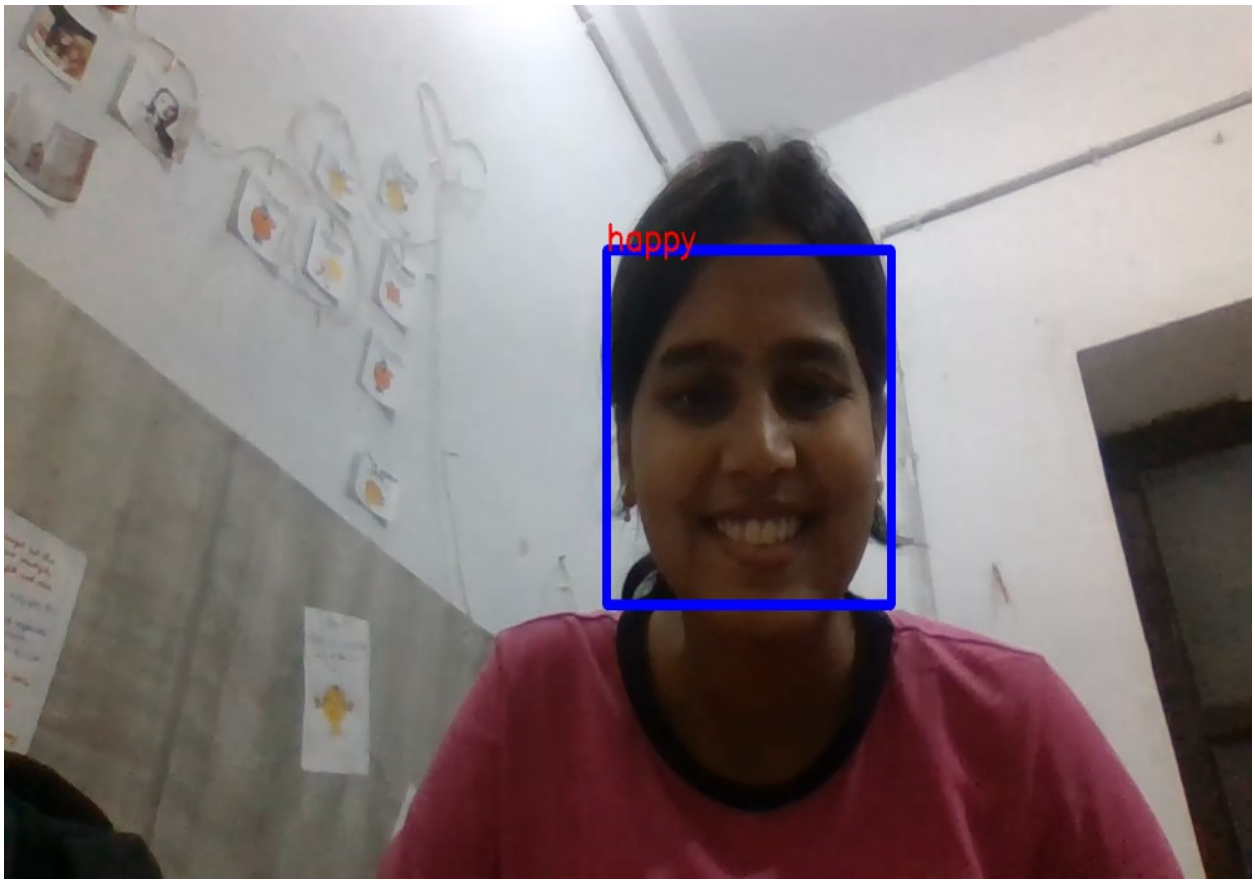

```
1/1 [==============================] - 0s 29ms/step
```



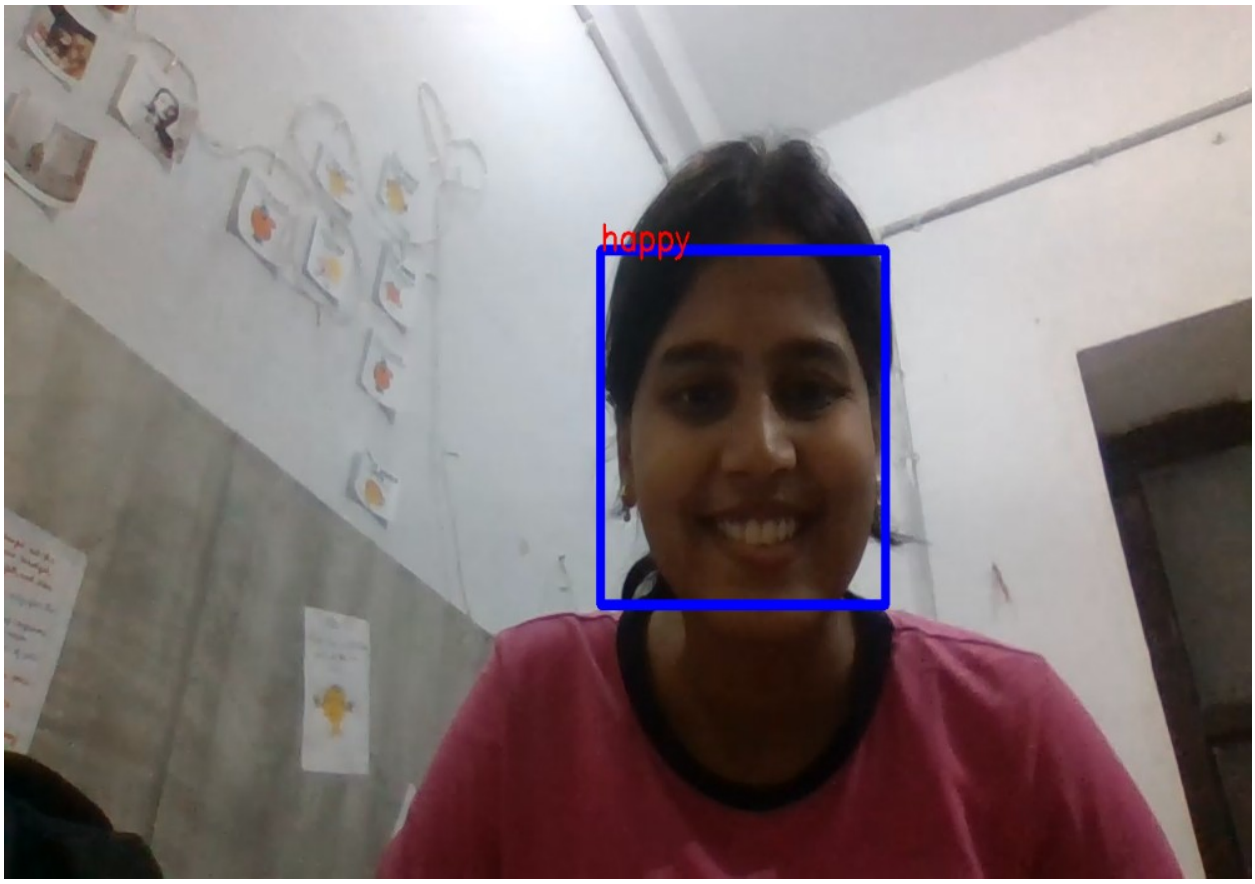

```
1/1 [==============================] - 0s 36ms/step
```




```
1/1 [==============================] - 0s 31ms/step
```




```
1/1 [==============================] - 0s 28ms/step
```




```
1/1 [==============================] - ETA: 0s
```