به نام خدا

تشخیص تومور های مغزی با استفاده از شبکه های CNN و تکنیک Data Augmentation

دانشجو: آرش رحمتی نام استاد: دکتر خدایی مهر

شماره دانشجو: ۴۰۰۳۰۸۶۴

مقدمه

تومور های مغزی در سال ۲۰۲۰ حدود ۳۰۸ هزار نفر را درگیر کرده اند و تشخیص زود هنگام آن ها در درمان تومور نقش اساسی دارد. این تومور ها ۱۰ اُمین دلیل مرگ ها در سراسر جهان هستند. در سال های اخیر "یادگیری ژرف" و شبکه هایCNN کمک بسیاری در این زمینه به صنعت پزشکی کرده اند و نتایج قابل قبولی داشته اند. در این مقاله می خواهیم با استفاده از یک شبکه عصبی VGG16 و کمی تغییرات یک شبکه ی عصبی عمیق بسازیم که با دقت بالا بتواند یک عکس image.jpg را با ابعاد ۲۲۴ در ۲۲۴ دریافت کند و نهایتاً پاسخ دهد که صاحب عکس آیا تومور مغزی دارد یا خیر.

دیتاست

در این مقاله ما از دیتاست تصاویر MRI از مغز استفاده می کنیم که در سایت kaggle.com با عنوان زیر دسترسی آزاد برای عموم دارد:

Brain MRI Images for Brain Tumor Detection

همچنین لینک را اینجا قرار می دهم. اما این دیتاست همه یک فرمت نیست من لینک دیتاست تمیز شده که باهاش کار کردم را در ایمیل برای شما میفرستم همچنین اینجا قرار می دهم.

این دیتاست شامل ۲۵۳ عکس MRI از مغز است که ۱۵۵ تا از عکس ها برچسب yes (وجود تومور مغزی) و ۹۸ تا عکس دیگر برچسب no (سالم بودن مغز) دارند. ما قصد داریم بخشی از داده را برای آموزش و بخشی برای آزمون استفاده کنیم. همچنین چون تعداد داده ها کم است لازم است از تکنیک Data Augmentation استفاده کنیم تا سایز مجموعه ی آموزش را بتوانیم بیشتر کنیم و در نتیجه مدل بهتری بدست آوریم.

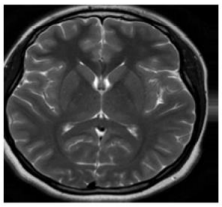
توضیح Data Augmentation

این تکنیک می تواند از یک عکس در داده ی آموزش چندین عکس جدید ایجاد کند. چندین روش Data Augmentation که در این مقاله استفاده شده اند عبارت اند از:

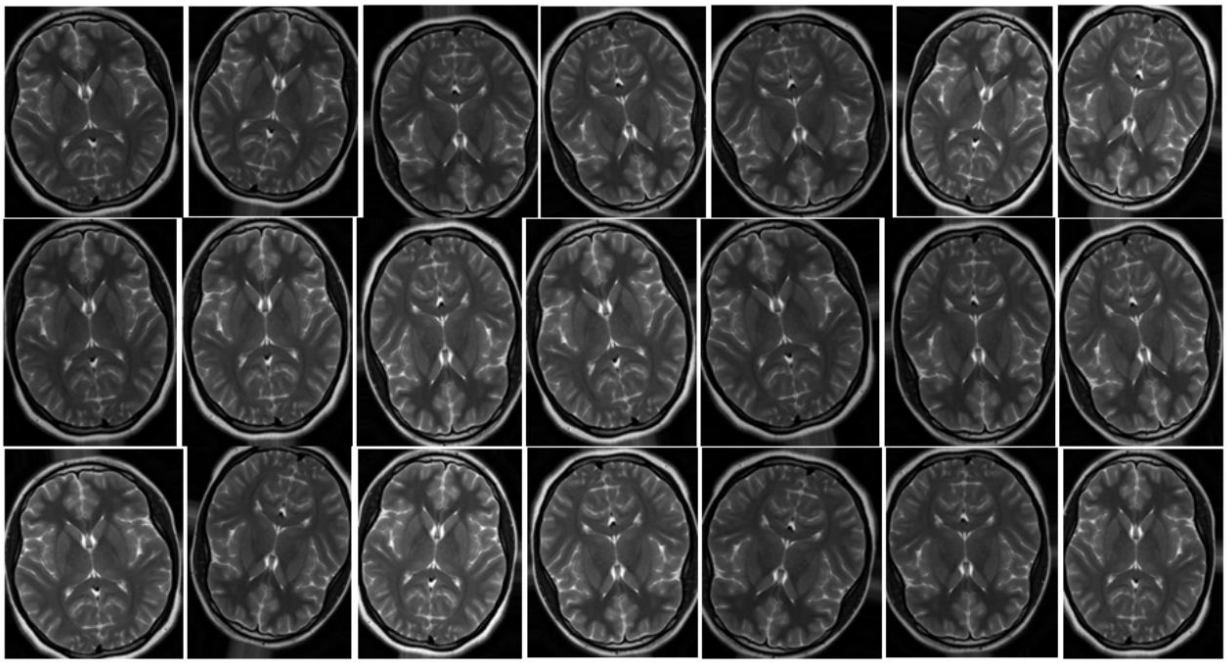
دوران (Rotation) تصویر آینه ای (Flip Horizontal/Vertical) جابجایی (translation)

به عنوان مثال یک نمونه از Data Augmentation را از یک عکس در داده ی آموزش در زیر می بینیم:

Original image

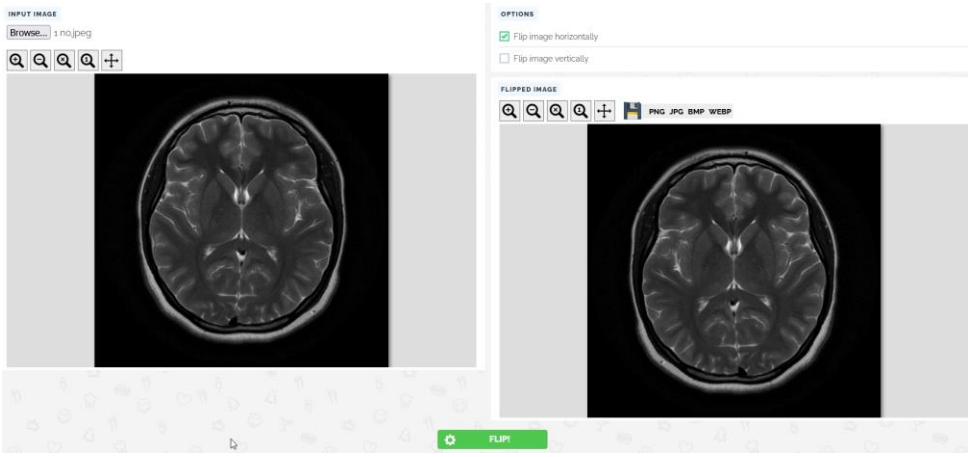


Augemented image

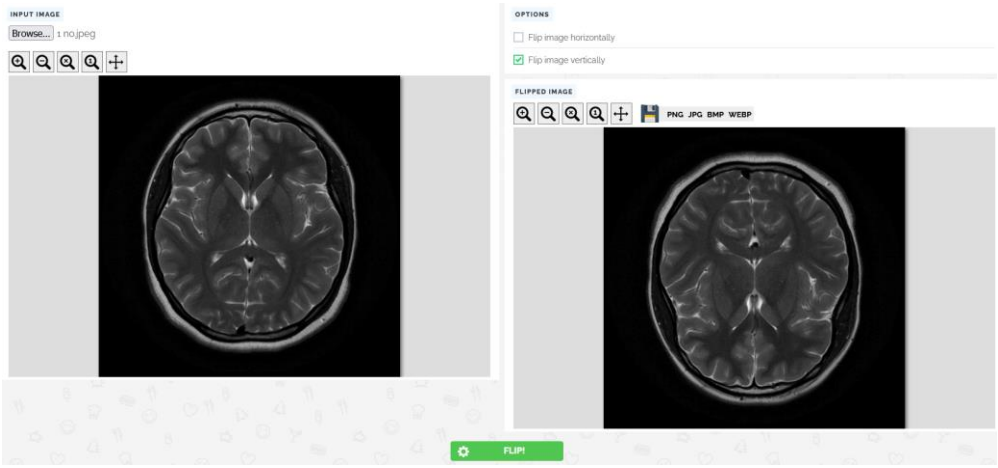


مثال دقیق تر دوران – جابجایی – تصویر آینه ای:

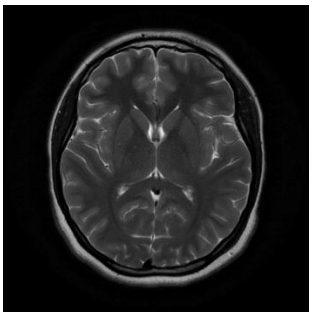
Horizontal Flip:

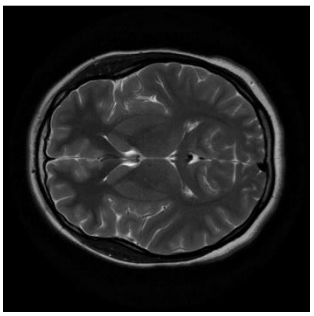


Vertical Flip



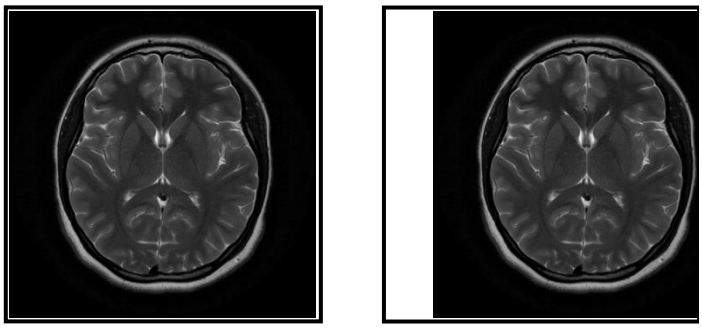
Rotate 90 degrees:





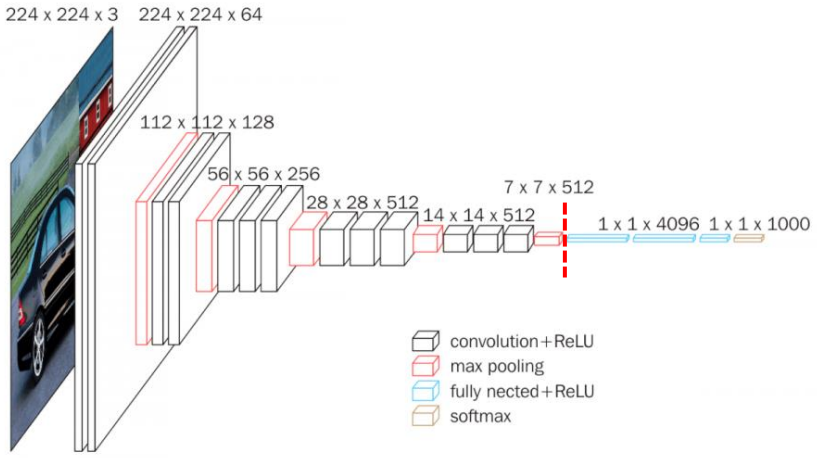
در دوران و جابجایی ممکن است پیکسل هایی از دست بروند. می توانم با نزدیک ترین پیکسل آن را پر کرد یا آن را سیاه در نظر گرفت.

Translation (x-axis)

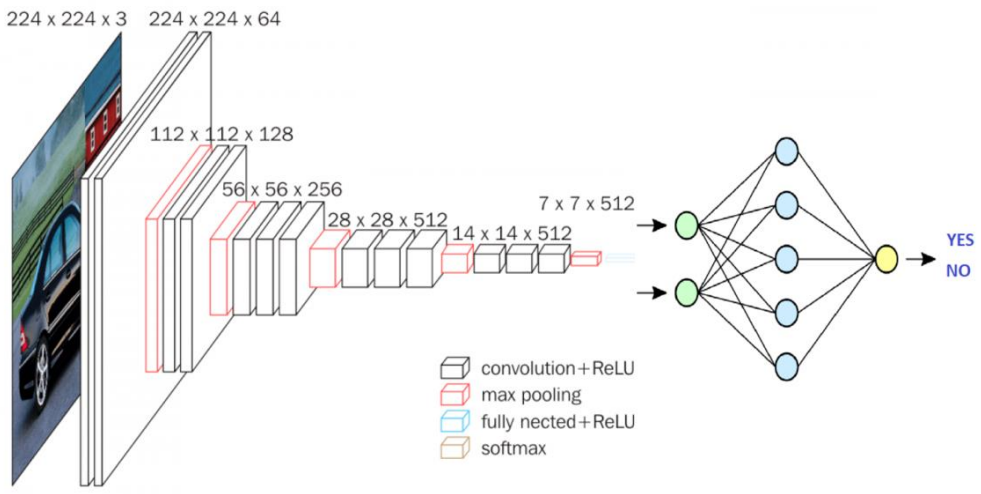


روش کار Methodology

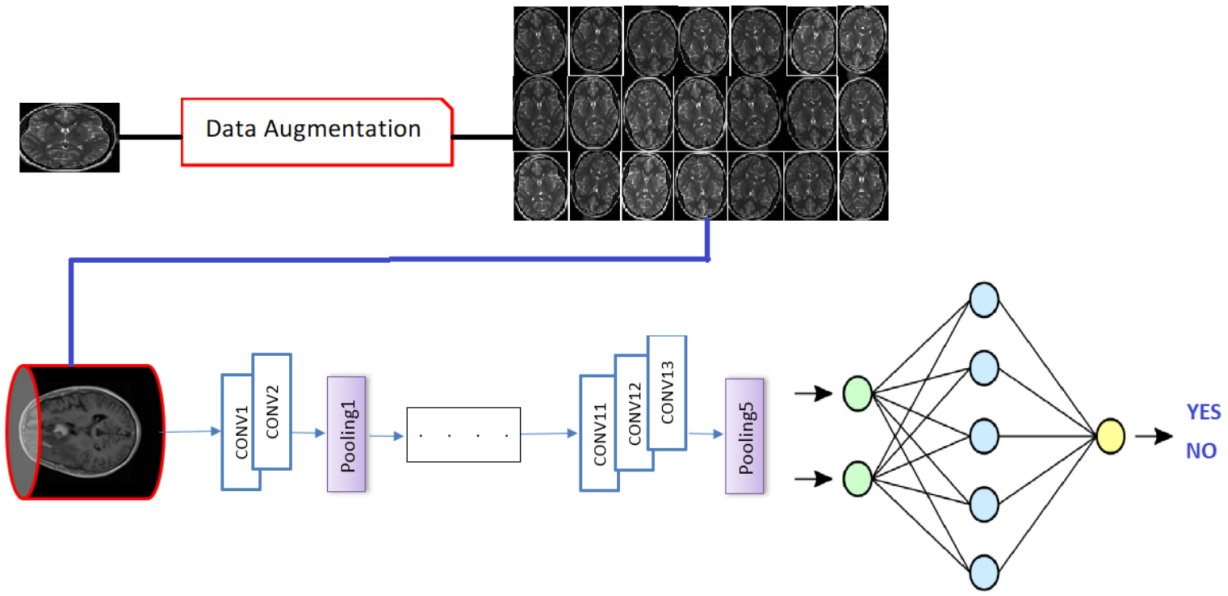
در این مقاله می خواهیم از شبکه از قبل آموزش دیده VGG16 روی داده های Image net استفاده کنیم و آن را کمی تغییر دهیم. یک شبکه ی VGG16 بصورت کلی بفرم زیر است:



اما ما فقط تا آخرین لایه ی pooling را بر می داریم و از آنجا به بعد از یک شبکه کوچک fully connected دیگر استفاده می کنیم:



در این شبکه عصبی بجای softmax از sigmoid استفاده می کینم زیرا در اینجا مسئله Image net نیست بلکه binary classification برای تومور های مغزی است. در ۲ لایه های دیگر اضافه شده هم از تابع فعالساز ReLU استفاده می کنیم. پس با استفاده از Data Augmentation مدل نهایی بفرم زیر خواهد بود:



همچنین نکته ی قابل توجه این است که ما وزن های شبکه ی VGG16 را freeze می کنیم و تغییرات را فقط در شبکه ی اضافه شده ی خودمان انجام می دهیم. این کار به بهبود سرعت آموزش شبکه بسیار کمک می کند.

نهایتاً برای آموزش با استفاده از کتاب خانه ی ImageDataGenerator برای تکنیک Data Augmentation استفاده می کنیم. این کتاب خانه قادر است Real-time تصاویر جدیدی تولید کند و نیازی نیست ما چندین هزار عکس را در کامپیوتر خود ذخیره کنیم. عکس های جدید در زمان نیاز تولید شده و بعد از استفاده بطور اتوماتیک حذف خواهند شد.

همچنین در هنگام آموزش شبکه از دو callback استفاده می کنیم:

اول: ModelCheckPoint

دوم: Earlystopping

اولی برای این است که بهترین دقت مدل را ذخیره کند. (این دقت می تواند آموزش یا آزمون باشد اما اکثراً دقت آزمون مانیتور می شود.) دومی برای جلوگیری از تکرار اضافی استفاده می شود. این callback می تواند یک پارامتر مثلاً تابع هزینه را مانیتور کند و اگر patience=10 بار متوالی کاهشی نداشته باشد آموزش را متوقف کند. در کد من این معیار که انتظار داریم حداقل هر ۱۰ تکرار بهبودی داشته باشد معیار دقت آموزش train_accuracy می باشد.

استفاده از GPU:

برای سریع تر کردن محاسبات لازم است از CUDA core های GPU استفاده شود این کار با نصب موارد زیر در Anaconda قابل انجام است:

cuDNN

cudatoolkit

tensorflow-gpu

tensorflow-estimator

پیاده سازی:

کد را جدا ارسال می کنم ولی مواردی که توضیح بخواهد را اینجا روی کد می نویسم:

```
import numpy as np
import tensorflow as tf

np.random.seed(30)
tf.random.set_seed(30)

from tensorflow.keras.models import load_model
from tensorflow import keras
from tensorflow.keras.applications.vgg16 import VGG16 , preprocess_input
```

```
vgg16 = VGG16( weights='imagenet', include_top=False, input_shape=(224,224,3))
```

```
vgg16.summary()
```

در این قسمت مدل معروف VGG16 وارد می شود. اما این یک مدل آموزش دیده pre trained هست و آموزش آن هم روی دیتاست معروف imagenet انجام شده است پس این مدل همینطوری هم خیلی از عکس ها را تشخیص می دهد. Include_top=False زیرا می خواهیم لایه های آخر وارد نشوند و لایه های دلخواه خودمان برای binary classification را طراحی کنیم.

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
```

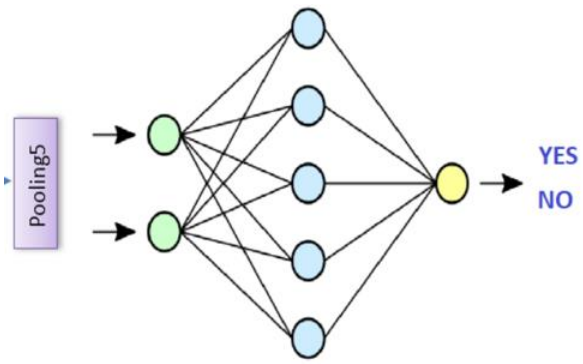
```
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

همان طور که می بینید این مدل **۱۴ میلیون پارامتر** دارد که برای ما زیاد هست پس ما همه را **freeze** می کنیم. ما ۳ لایه به آن اضافه می کنیم و این وزن ها را آپدیت می کنیم.

```
vgg16.trainable=False

myModel = tf.keras.Sequential()
myModel.add (vgg16)
myModel.add (tf.keras.layers.Flatten())
myModel.add (keras.layers.Dense(2,activation='relu'))
myModel.add (keras.layers.Dense(5, activation='relu'))
myModel.add (keras.layers.Dense(1,activation='sigmoid'))
```

```
myModel.summary()
```

دستور vgg16.trainable=False برای freeze کردن تمامی وزن های VGG16 می باشد. Flatten به این دلیل است که لایه ی آخر VGG16 ابعاد (7, 7, 15) دارد و ما باید آن را یک بعدی کنیم و به لایه fully connected بدهیم. بقیه لایه ها متناسب با مقاله اضافه شده اند.



```
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
myModel.compile(optimizer=opt,
                loss='binary_crossentropy',
                metrics=["accuracy"])
```

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

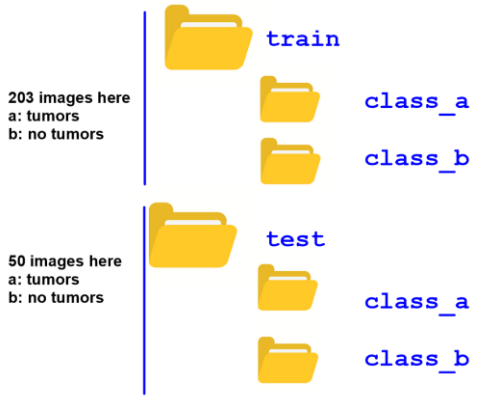
سپس پارامتر های مدل مانند optimizer, loss, metric تعیین می شوند. کتابخانه ی ImageDataGenerator هم وارد شد البته در مرحله اول ما نمی خواهیم Data Augmentation انجام دهیم و روی خود دیتا کار می کنیم و از این کتابخانه فقط برای normalize کردن تصاویر استفاده می کنیم.

without augmentation

```
train_datagen = ImageDataGenerator(rescale=1./255)

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
No_train_samples=203
No_test_samples=50
epochs=30
batch_size=32
train_dir='train'
test_dir='test'
```

در این قسمت توجه می کنیم که فولدر های train و test در محیط برنامه environment بفرمت زیر وجود دارند:



در غیر اینصورت کتابخانه ی ImageDataGenerator نمیتواند با دیتا کار کند.

```
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(224,224),
                                                    batch_size=batch_size,
                                                    class_mode='binary'
                                                    ,seed=100)

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  target_size=(224,224),
                                                  batch_size=batch_size,
                                                  class_mode='binary',
                                                  shuffle=False,
                                                  seed=100)
```

```
Found 203 images belonging to 2 classes.
Found 50 images belonging to 2 classes.
```

کتاب خانه ی ImageDataGenerator بدرستی عکس ها را تشخیص داده است.

```
temp = tf.keras.callbacks.ModelCheckpoint(
    filepath="presentOne",
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

temp2 = tf.keras.callbacks.EarlyStopping(
    monitor="accuracy",
    patience=10)
```

در مورد این callback ها پیش تر توضیح داده ایم.

```
history = myModel.fit(train_generator,
                              epochs=epochs,
                              steps_per_epoch=No_train_samples//batch_size,
                              validation_data=test_generator, callbacks=[temp2, temp])
```

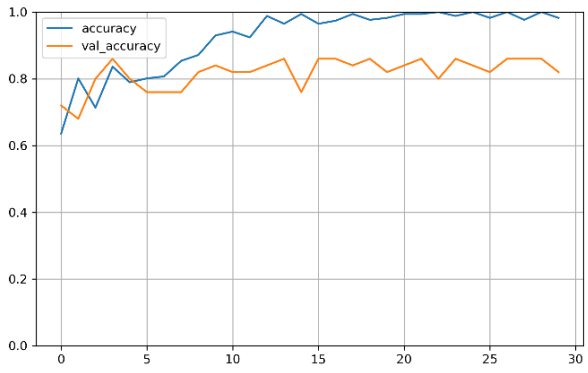
نتیجه (بدون Augmentation):

```
Epoch 1/30
6/6 [==============================] - ETA: 0s - loss: 0.6053 - accuracy:
0.6354
INFO:tensorflow:Assets written to: presentOne\assets
6/6 [==============================] - 11s 2s/step - loss: 0.6053 - accuracy:
0.6354 - val_loss: 0.6019 - val_accuracy: 0.7200
Epoch 2/30
6/6 [==============================] - 2s 411ms/step - loss: 0.4990 -
accuracy: 0.8012 - val_loss: 0.5933 - val_accuracy: 0.6800
Epoch 3/30
```

```
6/6 [==============================] - ETA: 0s - loss: 0.5129 - accuracy:
0.7135
INFO:tensorflow:Assets written to: presentOne\assets
6/6 [==============================] - 10s 2s/step - loss: 0.5129 - accuracy:
0.7135 - val_loss: 0.5807 - val_accuracy: 0.8000
Epoch 4/30
6/6 [==============================] - ETA: 0s - loss: 0.5094 - accuracy:
0.8363
INFO:tensorflow:Assets written to: presentOne\assets
6/6 [==============================] - 9s 2s/step - loss: 0.5094 - accuracy:
0.8363 - val_loss: 0.5558 - val_accuracy: 0.8600
Epoch 5/30
6/6 [==============================] - 3s 414ms/step - loss: 0.4447 -
accuracy: 0.7895 - val_loss: 0.5859 - val_accuracy: 0.8000
Epoch 6/30
6/6 [==============================] - 2s 447ms/step - loss: 0.4076 -
accuracy: 0.8012 - val_loss: 0.5416 - val_accuracy: 0.7600
Epoch 7/30
6/6 [==============================] - 3s 416ms/step - loss: 0.4155 -
accuracy: 0.8070 - val_loss: 0.5495 - val_accuracy: 0.7600
Epoch 8/30
6/6 [==============================] - 2s 447ms/step - loss: 0.3488 -
accuracy: 0.8538 - val_loss: 0.6594 - val_accuracy: 0.7600
Epoch 9/30
6/6 [==============================] - 2s 448ms/step - loss: 0.3620 -
accuracy: 0.8713 - val_loss: 0.5878 - val_accuracy: 0.8200
Epoch 10/30
6/6 [==============================] - 2s 412ms/step - loss: 0.3319 -
accuracy: 0.9298 - val_loss: 0.5602 - val_accuracy: 0.8400
Epoch 11/30
6/6 [==============================] - 2s 452ms/step - loss: 0.3159 -
accuracy: 0.9415 - val_loss: 0.5743 - val_accuracy: 0.8200
Epoch 12/30
6/6 [==============================] - 2s 412ms/step - loss: 0.3103 -
accuracy: 0.9240 - val_loss: 0.5677 - val_accuracy: 0.8200
Epoch 13/30
6/6 [==============================] - 2s 452ms/step - loss: 0.2838 -
accuracy: 0.9883 - val_loss: 0.5723 - val_accuracy: 0.8400
Epoch 14/30
6/6 [==============================] - 3s 416ms/step - loss: 0.2576 -
accuracy: 0.9649 - val_loss: 0.5336 - val_accuracy: 0.8600
Epoch 15/30
6/6 [==============================] - 2s 416ms/step - loss: 0.2830 -
accuracy: 0.9942 - val_loss: 0.6300 - val_accuracy: 0.7600
Epoch 16/30
6/6 [==============================] - 2s 411ms/step - loss: 0.2566 -
accuracy: 0.9649 - val_loss: 0.5480 - val_accuracy: 0.8600
Epoch 17/30
6/6 [==============================] - 3s 448ms/step - loss: 0.2562 -
accuracy: 0.9740 - val_loss: 0.5828 - val_accuracy: 0.8600
Epoch 18/30
6/6 [==============================] - 2s 414ms/step - loss: 0.2567 -
accuracy: 0.9942 - val_loss: 0.6130 - val_accuracy: 0.8400
Epoch 19/30
6/6 [==============================] - 2s 450ms/step - loss: 0.2442 -
accuracy: 0.9766 - val_loss: 0.5624 - val_accuracy: 0.8600
Epoch 20/30
```

```
6/6 [==============================] - 3s 412ms/step - loss: 0.2580 -
accuracy: 0.9825 - val_loss: 0.6218 - val_accuracy: 0.8200
Epoch 21/30
6/6 [==============================] - 2s 416ms/step - loss: 0.2499 -
accuracy: 0.9942 - val_loss: 0.6132 - val_accuracy: 0.8400
Epoch 22/30
6/6 [==============================] - 3s 415ms/step - loss: 0.2365 -
accuracy: 0.9942 - val_loss: 0.5947 - val_accuracy: 0.8600
Epoch 23/30
6/6 [==============================] - 3s 415ms/step - loss: 0.2364 -
accuracy: 1.0000 - val_loss: 0.6649 - val_accuracy: 0.8000
Epoch 24/30
6/6 [==============================] - 2s 416ms/step - loss: 0.2270 -
accuracy: 0.9883 - val_loss: 0.5559 - val_accuracy: 0.8600
Epoch 25/30
6/6 [==============================] - 2s 414ms/step - loss: 0.2255 -
accuracy: 1.0000 - val_loss: 0.6498 - val_accuracy: 0.8400
Epoch 26/30
6/6 [==============================] - 3s 415ms/step - loss: 0.2273 -
accuracy: 0.9825 - val_loss: 0.5267 - val_accuracy: 0.8200
Epoch 27/30
6/6 [==============================] - 2s 450ms/step - loss: 0.2188 -
accuracy: 1.0000 - val_loss: 0.5910 - val_accuracy: 0.8600
Epoch 28/30
6/6 [==============================] - 2s 413ms/step - loss: 0.2267 -
accuracy: 0.9766 - val_loss: 0.6123 - val_accuracy: 0.8600
Epoch 29/30
6/6 [==============================] - 2s 413ms/step - loss: 0.2162 -
accuracy: 1.0000 - val_loss: 0.5404 - val_accuracy: 0.8600
Epoch 30/30
6/6 [==============================] - 2s 450ms/step - loss: 0.2229 -
accuracy: 0.9825 - val_loss: 0.7063 - val_accuracy: 0.8200
```

همان طور که میبینیم آخرین ذخیره سازی بهترین مدل در فولدری به نام presentOne برای دقت آزمون ۸۶ درصد انجام شده و از آنجا به بعد دچار overfitting شده ایم. دقت آموزش تا ۱۰۰ هم رفته ولی دقت آزمون از ۸۶ بیشتر نشده است. اگر Data Augmentation استفاده می کردیم این اتفاق نمی افتاد. نتیجه:



```python
from sklearn.metrics import classification_report, confusion_matrix

def rounder(a):
    k=[]
    for t in range(0,len(a)):
       a[t][0]=round(a[t][0])
    return np.array(a)
```

```python
temp = load_model ('presentOne')
```

```python
print('acc: ', temp.evaluate(test_generator))

predictions = temp.predict_generator(test_generator, steps=len(test_generator))
y=rounder(predictions)
print('scores: ')
report = classification_report (y_true= test_generator.classes, y_pred=y.reshape(50), \
                                target_names=test_generator.class_indices)
print(report)
```

سپس با دستور های بالا F1-score محاسبه شده است. دقت کنیم که تابع rounder را خودم متناسب با مسئله تعریف کرده ام. چون خروجی سیگموید بین ۰ و ۱ است مجبور بودم با یک تابع جدا آن ها را به ۰ و ۱ تبدیل کنم. (بالای 0.5 به یک و پایین تر از 0.5 به صفر تبدیل شده است. نتیجه:

```
2/2 [==============================] - 1s 240ms/step - loss: 0.5288 -
accuracy: 0.8600

acc:  [0.5288028120994568, 0.8600000143051147]

  predictions = temp.predict_generator(test_generator,
steps=len(test_generator))
scores:
              precision    recall  f1-score   support

     class_a       0.79      1.00      0.89        27
     class_b       1.00      0.70      0.82        23

    accuracy                           0.86        50
   macro avg       0.90      0.85      0.85        50
weighted avg       0.89      0.86      0.86        50
```

همانطور که می بینیم f1_score=0.86 است ولی مقاله به 0.97 رسیده است (with augmentation):

حال همین کار را با استفاده از Data Augmentation انجام می دهیم:

with augmentation

```
train_datagen2 = ImageDataGenerator(rescale=1./255,
                                    rotation_range=15,
                                    width_shift_range=0.1,
                                    height_shift_range=0.1,
                                    shear_range=0.1,
                                    brightness_range=[0.5, 1.5],
                                    horizontal_flip=True,
                                    vertical_flip=True,
                                   preprocessing_function=preprocess_input)

test_datagen2 = ImageDataGenerator(preprocessing_function=preprocess_input)
```

تفاوت در این است که train data را data augmentation انجام داده ایم ولی test data فقط همان nomarlize شده است. (دقت کنید که test data نباید تغییر کند چون می خواهیم دقت مدل را با آن بررسی کنیم پس باید خود عکس های طبیعی و تغییر نیافته باشند.)

```
train_generator2 = train_datagen2.flow_from_directory(train_dir,
                                                      target_size=(224,224),
                                                      batch_size=batch_size,
                                                      class_mode='binary'
                                                      , seed=100)

test_generator2 = test_datagen2.flow_from_directory(test_dir,
                                                    target_size=(224,224),
                                                    batch_size=batch_size,
                                                    class_mode='binary',
                                                    shuffle=False,
                                                      seed=100)
```

```
Found 203 images belonging to 2 classes.
Found 50 images belonging to 2 classes.
```

```
temp3 = tf.keras.callbacks.ModelCheckpoint(
    filepath="presentTwo",
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

temp4 = tf.keras.callbacks.EarlyStopping(
    monitor="accuracy",
    patience=10)
```

کال بک ها مانند قبل است فقط اینبار مدل در فولدر presentTwo ذخیره می شود که مشخص باشد و مدل قبلی هم پاک نشود.

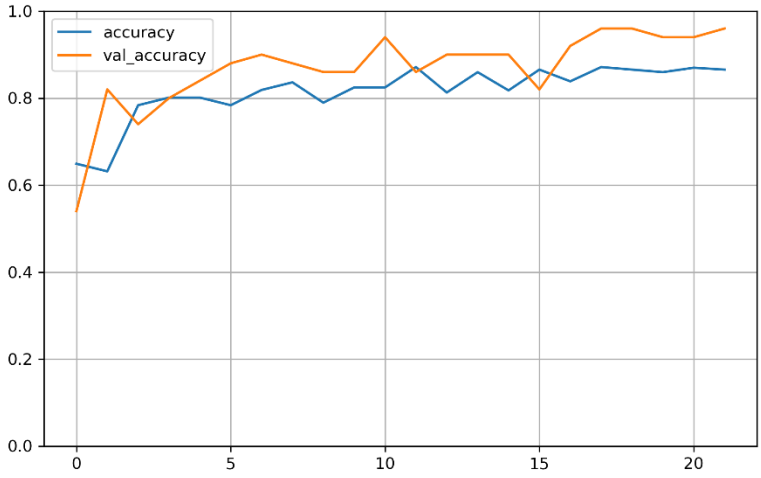
```
history2 = myModel.fit_generator(train_generator2,
                        epochs=epochs,
                    steps_per_epoch=No_train_samples//batch_size,
                   validation_data=test_generator2, callbacks=[temp3, temp4])
```

نتیجه:

```
Epoch 1/30
6/6 [==============================] - ETA: 0s - loss: 0.6087 - accuracy:
0.6491
INFO:tensorflow:Assets written to: presentTwo\assets
6/6 [==============================] - 13s 2s/step - loss: 0.6087 - accuracy:
0.6491 - val_loss: 0.7044 - val_accuracy: 0.5400
Epoch 2/30
6/6 [==============================] - ETA: 0s - loss: 0.5482 - accuracy:
0.6316
INFO:tensorflow:Assets written to: presentTwo\assets
6/6 [==============================] - 11s 2s/step - loss: 0.5482 - accuracy:
0.6316 - val_loss: 1.3651 - val_accuracy: 0.8200
Epoch 3/30
6/6 [==============================] - 4s 636ms/step - loss: 0.5426 -
accuracy: 0.7836 - val_loss: 2.6699 - val_accuracy: 0.7400
Epoch 4/30
6/6 [==============================] - 4s 634ms/step - loss: 0.5046 -
accuracy: 0.8012 - val_loss: 1.1276 - val_accuracy: 0.8000
Epoch 5/30
6/6 [==============================] - ETA: 0s - loss: 0.5122 - accuracy:
0.8012
INFO:tensorflow:Assets written to: presentTwo\assets
6/6 [==============================] - 11s 2s/step - loss: 0.5122 - accuracy:
0.8012 - val_loss: 0.7337 - val_accuracy: 0.8400
Epoch 6/30
6/6 [==============================] - ETA: 0s - loss: 0.5090 - accuracy:
0.7836
INFO:tensorflow:Assets written to: presentTwo\assets
6/6 [==============================] - 11s 2s/step - loss: 0.5090 - accuracy:
0.7836 - val_loss: 0.7741 - val_accuracy: 0.8800
Epoch 7/30
6/6 [==============================] - ETA: 0s - loss: 0.4616 - accuracy:
0.8187
INFO:tensorflow:Assets written to: presentTwo\assets
6/6 [==============================] - 11s 2s/step - loss: 0.4616 - accuracy:
0.8187 - val_loss: 0.5030 - val_accuracy: 0.9000
Epoch 8/30
6/6 [==============================] - 4s 592ms/step - loss: 0.4789 -
accuracy: 0.8363 - val_loss: 0.6324 - val_accuracy: 0.8800
Epoch 9/30
6/6 [==============================] - 4s 644ms/step - loss: 0.4570 -
accuracy: 0.7895 - val_loss: 0.6735 - val_accuracy: 0.8600
Epoch 10/30
6/6 [==============================] - 4s 637ms/step - loss: 0.4765 -
accuracy: 0.8246 - val_loss: 0.5497 - val_accuracy: 0.8600
Epoch 11/30
```

```
6/6 [==============================] - ETA: 0s - loss: 0.4557 - accuracy:
0.8246
INFO:tensorflow:Assets written to: presentTwo\assets
6/6 [==============================] - 11s 2s/step - loss: 0.4557 - accuracy:
0.8246 - val_loss: 0.5055 - val_accuracy: 0.9400
Epoch 12/30
6/6 [==============================] - 4s 631ms/step - loss: 0.4413 -
accuracy: 0.8713 - val_loss: 0.7911 - val_accuracy: 0.8600
Epoch 13/30
6/6 [==============================] - 4s 643ms/step - loss: 0.4358 -
accuracy: 0.8129 - val_loss: 0.6230 - val_accuracy: 0.9000
Epoch 14/30
6/6 [==============================] - 4s 641ms/step - loss: 0.4231 -
accuracy: 0.8596 - val_loss: 0.8491 - val_accuracy: 0.9000
Epoch 15/30
6/6 [==============================] - 4s 687ms/step - loss: 0.4159 -
accuracy: 0.8177 - val_loss: 0.7893 - val_accuracy: 0.9000
Epoch 16/30
6/6 [==============================] - 4s 717ms/step - loss: 0.4135 -
accuracy: 0.8655 - val_loss: 1.3159 - val_accuracy: 0.8200
Epoch 17/30
6/6 [==============================] - 4s 682ms/step - loss: 0.4414 -
accuracy: 0.8385 - val_loss: 0.6280 - val_accuracy: 0.9200
Epoch 18/30
6/6 [==============================] - ETA: 0s - loss: 0.3969 - accuracy:
0.8713
INFO:tensorflow:Assets written to: presentTwo\assets
6/6 [==============================] - 11s 2s/step - loss: 0.3969 - accuracy:
0.8713 - val_loss: 0.4967 - val_accuracy: 0.9600
Epoch 19/30
6/6 [==============================] - 4s 632ms/step - loss: 0.4029 -
accuracy: 0.8655 - val_loss: 0.4611 - val_accuracy: 0.9600
Epoch 20/30
6/6 [==============================] - 4s 650ms/step - loss: 0.3808 -
accuracy: 0.8596 - val_loss: 0.4133 - val_accuracy: 0.9400
Epoch 21/30
6/6 [==============================] - 4s 680ms/step - loss: 0.3969 -
accuracy: 0.8698 - val_loss: 0.5445 - val_accuracy: 0.9400
Epoch 22/30
6/6 [==============================] - 4s 609ms/step - loss: 0.3842 -
accuracy: 0.8655 - val_loss: 0.4070 - val_accuracy: 0.9600
```

همانطور که انتظار داشتیم دقت آزمون تا ۰,۹۶ درصد افزایش یافت همان دقت خود مقاله. نتیجه را در نمودار می بینیم:



در این حالت Data Augmentation به ما کمک کرد تا از overfitting جلوگیری شود.

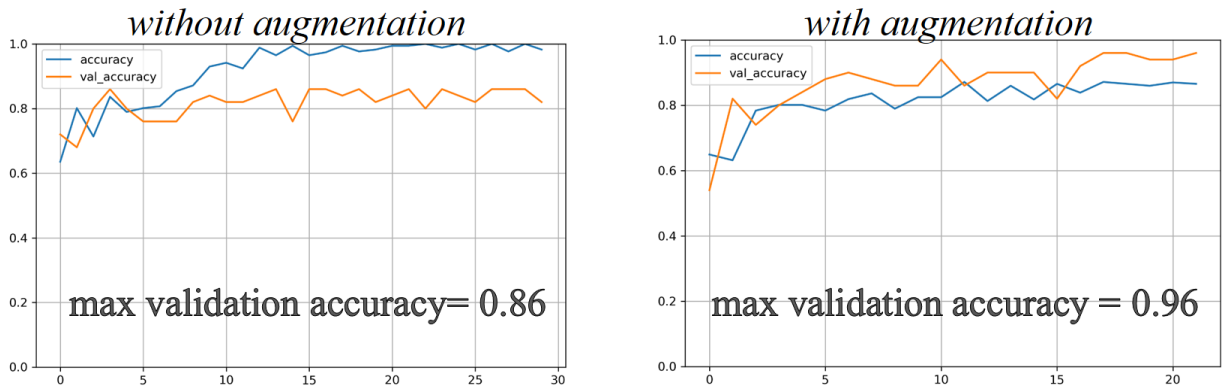
```
temp = load_model ('presentTwo')
```

```
print('acc: ', temp.evaluate(test_generator2))

predictions = temp.predict_generator(test_generator2, steps=len(test_generator2))
y=rounder(predictions)
print('scores: ')
report = classification_report (y_true= test_generator2.classes, y_pred=y.reshape(50), \
                                target_names=test_generator2.class_indices)
print(report)
```

```
2/2 [==============================] - 1s 239ms/step - loss: 0.4967 -
accuracy: 0.9600

scores:
              precision    recall  f1-score   support

     class_a       0.96      0.96      0.96        27
     class_b       0.96      0.96      0.96        23

    accuracy                           0.96        50
   macro avg       0.96      0.96      0.96        50
weighted avg       0.96      0.96      0.96        50
```

f1-score هم مطابقاً افزایش داشته اما مقاله به f1-score=0.97 رسیده که بنظرم بدلیل انتخاب داده ها به عنوان آموزش و آزمون هست زیرا در مقاله بیان نشده کدام دیتا ها آموزش و کدام دیتا ها آزمون بوده اند. همچنین در سایت

kaggle.com فقط دیتا ها بر اساس yes و no دسته بندی شده اند. در هر صورت تاثیر Data Augmentation در این فرایند واضح است.



در انتهای کد نیز یک عکس را data augmentation انجام داده ایم که ببینیم چطوری یک عکس به چند عکس تبدیل می شود.

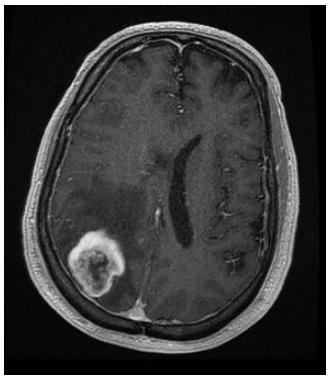
```
dgen = ImageDataGenerator(
                        rotation_range=15,
                        width_shift_range=0.1,
                        height_shift_range=0.1,
                        shear_range=0.1,
                        brightness_range=[0.5, 1.5],
                        horizontal_flip=True,
                        vertical_flip=True)
```

```
from numpy import expand_dims
from matplotlib import pyplot as plt
import numpy as np
from tensorflow.keras.utils import img_to_array
from keras.preprocessing import image
```

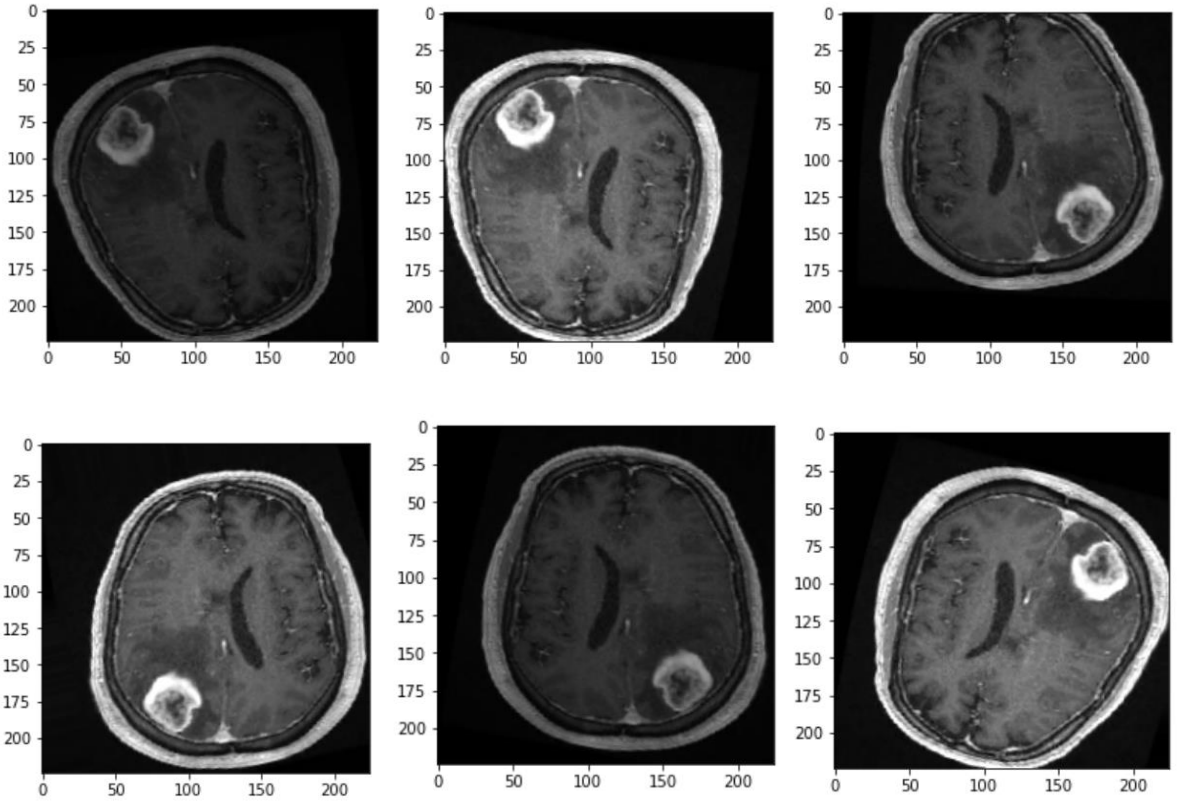
```
myimage = tf.keras.utils.load_img("yes.jpg",target_size = (224,224))
data = img_to_array(myimage)
samples = data.reshape(1,224,224,3)
```

```
it = dgen.flow(samples, batch_size=1)
le = []

for i in range(6):
    batch = it.next()
    image = batch[0].astype('uint8')
    le.append(image)
```

عکس اصلی:



عکس های تولید شده از عکس اصلی:



کدها جدا ارسال می شوند.

پایان