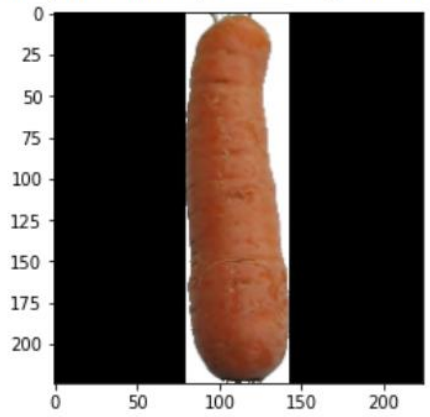
سوال سوم) الف :

```python
def resize_img(img, desired_size = 224):
    # write your code here
    new_img = img
    num = desired_size / max(img.shape[0], img.shape[1])
    ww = int(img.shape[1] * num)
    hh = int(img.shape[0] * num)
    new_img = cv2.resize(new_img, (ww, hh))
    border1, border2, border3 = np.int64(np.divide(np.subtract(desired_size, new_img.shape), 2))
    i, j, k = np.mod(new_img.shape, 2)
    new_img = cv2.copyMakeBorder(new_img, top= border1 + i, bottom = border1, left = border2, right = border2 + j, borderType = cv2.BORDER_CONSTANT)
    return new_img
```

برای این تابع ابتدا ماکسیمم بین طول و عرض تصویر را پیدا کردیم چون بین این دو آنی که کمتر است نیاز به پدینگ دارد و آنی که بیشتر است خودش desired_size را پر میکند. نسبت desired_size به ماکسیمم را بدست آورده و در طول و عرض تصویر ضرب میکنیمو به تابع cv2.resize میدهیم. سپس برای بعد کوچکتر تصویر حاشیه درنظر میگیریم طوری که با احتساب این حاشیه آن بعد تصویر هم با desired_size برابر شود.

```
<matplotlib.image.AxesImage at 0x7f7ee62d2a90>
```



ب) مدل رزنت با وزن های رندوم را ساختیم:

چون ما تصاویر ۱۰۰۰ کلاسه نداریم نیازی به لایه ی بالایی نیست. اما در پایان دو لایه ی dense که نیاز داشتیم اضافه کردیم.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten_2 (Flatten)         (None, 2048)              0

 dense_7 (Dense)             (None, 512)               1049088

 dense_8 (Dense)             (None, 24)                12312

=================================================================
Total params: 24,649,112
Trainable params: 24,595,992
Non-trainable params: 53,120
```

نتیجه ی آموزش:

```
Epoch 20/20
65/65 [==============================] - 65s 996ms/step - loss: 0.0763 - acc: 0.9811
<keras.callbacks.History at 0x7f7bfde72c70>
```

پ) این هم مثل قبلی است فقط به جای وزن های رندوم از وزن های آموزش دیده شده روی دیتای imagenet استفاده کردیم و برای منظم سازی از droupout با پارامتر 0,55 استفاده کردیم.

نتیجه ی آموزش:

```
Epoch 20/20
65/65 [==============================] - 35s 528ms/step - loss: 0.6832 - acc: 0.7594
<keras.callbacks.History at 0x7f7f92c9cbe0>
```

ت)

نتیجه ی تست مدل اول:

```
1 resnet.evaluate(test_generator)
```

```
33/33 [==============================] - 18s 520ms/step - loss: 26.6302 - acc: 0.0260
[26.630224227905273, 0.026045016944408417]
```

نتیجه ی تست مدل دوم:

```
1 fine_tune_resnet.evaluate(test_generator)
```

```
33/33 [==============================] - 18s 508ms/step - loss: 0.4166 - acc: 0.8714
[0.4166014790534973, 0.8713826537132263]
```

مدل دوم خیلی نتیجه ی بهتری روی داده ی تست داده است.

مدل اول روی داده ی آموزش دقت ۹۸ درصد دارد یعنی تقریبا همه ی داده ها را درست جواب میدهد و دقتش روی داده ی تست خیلی کم شده و این یعنی overfit که البته با توجه به تعداد خیلی زیاد پارامتر ها نسبت به فقط ۶۵ تا داده ی آموزشی و ۳۳ داده ی تست طبیعی است. زمان اجرای هر ایپاک به طور متوسط ۶۵ ثانیه طول کشید و این هم اصلا بهینه نبود.

مدل دوم دارای دقت ۷۵ درصد روی داده ی آموزشی و دقت ۸۷ روی داده تست است و این نتیجه خیلی عالی میباشد. همچنین زمان اجرای هر ایپاک هم نسبت به مدل قبل کاهش یافته و به طور میانگین به ۳۵ ثانیه رسید.

میتوان نتیجه گرفت استفاده از وزن ها در شبکه های از قبل آموزش دیده با داده های بیشتر خیلی میتواند به ما کمک کند که برای داده های کمتر هم نتیجه ی بهتری بگیریم هم در زمان صرفه جویی کنیم.