به نام خداوند بخشنده و مهربان

درس یادگیری ماشین

گزارش تمرین شش

استاد درس: دکتر سعید شیری قیداری

نام دانشجو:

روزبه قاسمی ۹۵۳۱۴۲۴

زمستان ۱۳۹۷

سوال۱

طبق مطالبی که در سایتی که معرفی کرده بودید یعنی سایت deeplearning.ir مطلب رو به رو کلینک کنید! را پیدا کردم که توضیح داده بود می‌توانیم به دیتابیس های lmdb یا hdf5 یا leveldb تبدیل کنیم.

سوال۲

در ابتدا کد را از منبع آن یعنی سایتی که مشخص کرده بودید دانلود کردم که کد آن به شکل زیر است:

```
   from urllib.request import urlretrieve
        from os.path import isfile, isdir
        from tqdm import tqdm
        import tarfile

        cifar10_dataset_folder_path = 'cifar-10-batches-py'

class DLProgress(tqdm):
        last_block = 0

        def hook(self, block_num=1, block_size=1, total_size=None):
        self.total = total_size
        self.update((block_num - self.last_block) * block_size)
        self.last_block = block_num

        if not isfile('cifar-10-python.tar.gz'):
        with DLProgress(unit='B', unit_scale=True, miniters=1, desc='CIFAR-10
Dataset') as pbar:
        urlretrieve(
        'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz',
        'cifar-10-python.tar.gz',
        pbar.hook)

        if not isdir(cifar10_dataset_folder_path):
        with tarfile.open('cifar-10-python.tar.gz') as tar:
        tar.extractall()
        tar.close()
```

سپس در مرحله بعد عکس های موجود در این مجموعه داده را برچسب گذاری می‌کنیم و مشخص می‌کنیم از هر برچسب چند تا عکس داریم و از هر کدام یه نمونه خروجی می‌گیریم که کد آن به شکل زیر است:

```
import pickle
import matplotlib.pyplot as plt

LABEL_NAMES = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']

def load_cfar10_batch(batch_id):
"""
Load a batch of the dataset
```

```
"""
with open(cifar10_dataset_folder_path + '/data_batch_'
+ str(batch_id), mode='rb') as file:
batch = pickle.load(file, encoding='latin1')

features = batch['data'].reshape((len(batch['data']), 3, 32, 32)).transpose(0, 2, 3,
1)
labels = batch['labels']

return features, labels


def display_stats(features, labels, sample_id):
"""
Display Stats of the the dataset
"""

if not (0 <= sample_id < len(features)):
print('{} samples in batch {}.  {} is out of range.'
.format(len(features), batch_id, sample_id))
return None

print('\nStats of batch {}:'.format(batch_id))
print('Samples: {}'.format(len(features)))
print('Label Counts: {}'.format(dict(zip(*np.unique(labels, return_counts=True)))))
print('First 20 Labels: {}'.format(labels[:20]))

sample_image = features[sample_id]
sample_label = labels[sample_id]

print('\nExample of Image {}:'.format(sample_id))
print('Image - Min Value: {} Max Value: {}'.format(sample_image.min(),
sample_image.max()))
print('Image - Shape: {}'.format(sample_image.shape))
print('Label - Label Id: {} Name: {}'.format(sample_label, LABEL_NAMES[sample_label]))
plt.axis('off')
plt.imshow(sample_image)
plt.show()

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np

for batch_id in range(1,6):
    features, labels = load_cfar10_batch(batch_id)
    for image_id in range(0,2):
        display_stats(features, labels, image_id)

del features, labels # free memory
```

که خروجی های آن مانند زیر است:

```
Stats of batch 1:
Samples: 10000
Label Counts: {0: 1005, 1: 974, 2: 1032, 3: 1016, 4: 999, 5: 937, 6: 1030, 7:
1001, 8: 1025, 9: 981}
First 20 Labels: [6, 9, 9, 4, 1, 1, 2, 7, 8, 3, 4, 7, 7, 2, 9, 9, 9, 3, 2, 6]
```

```
Example of Image 0:
Image - Min Value: 0 Max Value: 255
Image - Shape: (32, 32, 3)
Label - Label Id: 6 Name: frog
```

حال در ادامه می‌بینیم که پیاده سازی من سه لایه است اما ابتدا به قسمت پیش پردازش خواسته شده می‌پردازیم:

A)

حال عملیات هایی که برای پیش پردازش داده ها انجام دادم به شکل زیر است:

۱- نرمالایز کردن داده ها

```
def normalize(x):
    # Each pixel has three channels - Red, Green and Blue.
    # Each channel is an int between 0 and 255 (8-bit color scheme).
    return np.array(x) / 255.0
```

۲- One-hot-encoding

با توجه به نکته ای در این سایت one-hot-encoding گفته شده است اینکار را کردم:

```
def one_hot_encode(x):
    one_hot_encoded = np.zeros((len(x), 10))
```

```
    for i in range(len(x)):
    one_hot_encoded[i][x[i]] = 1.0
    return one_hot_encoded
```

۳- mirroring

از دیگر پیش پردازش ها این عمل بود:

```
    from PIL import Image

        def flip_image(image_path, saved_location):
        """
        Flip or mirror the image

@param image_path: The path to the image to edit
@param saved_location: Path to save the cropped image
        """
        image_obj = Image.open(image_path)
        rotated_image = image_obj.transpose(Image.FLIP_LEFT_RIGHT)
        rotated_image.save(saved_location)
        rotated_image.show()

        if name == 'main':
        image = 'mantis.png'
        flip_image(image, 'flipped_mantis.jpg')
```

(B

در این قسمت باید بگویم اگر مقدار iteration ها را زیاد یا کم کنیم چه اتفاقی می‌افتد. طبق مطالبی که در سایت های مختلف و یوتیوب دیدم این بود که اگر از یک مقداری بیش از حد شود باعث می‌شود Overfiting رخ دهد و اگر از یه حدی کم شود باعث می‌شود underfiting رخ دهد در نتیجه باید یک مقدر مشخصی باشد.

همچنین اگر مقدار batch size باید متناسب با مقدار RAM باشد و اگر بتواند RAM پشتیبانی کند اگر زیاد کنیم باعث می‌شود در هر مرحله مقدار بیشتری محاسبات می‌کند و به طبع سرعت بالاتری نیز دارد.

حال شبکه عصبی که طراحی کردم را ایجاد می‌کنم:

ورودی آن به شکل زیر است:

```
import tensorflow as tf

        def neural_net_image_input(image_shape):
        return tf.placeholder(tf.float32, shape=(None, image_shape[0], image_shape[1],
image_shape[2]), name='x')
```

```
        def neural_net_label_input(n_classes):
        return tf.placeholder(tf.float32, shape=(None, n_classes), name='y')

        def neural_net_keep_prob_input():
        return tf.placeholder(tf.float32, name='keep_prob')
```

حال لایه convolutional و max pooling آن را ایجاد کردم:

```
def conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize,
pool_strides):

    # Create filter dimensions
    filter_height, filter_width, in_channels, out_channels = \
    conv_ksize[0], conv_ksize[1],  x_tensor.get_shape().as_list()[3], conv_num_outputs
    conv_filter = [filter_height, filter_width, in_channels, out_channels]

    # Create weights and bias
    weights = tf.Variable(tf.truncated_normal(conv_filter, stddev=0.05))
    bias = tf.Variable(tf.truncated_normal([conv_num_outputs], stddev=0.05))

    # Create strides
    strides=(1,conv_strides[0], conv_strides[1], 1)

    # Bind all together to create the layer
    conv = tf.nn.conv2d(x_tensor, weights, strides, padding='SAME')
    conv = tf.nn.bias_add(conv, bias)

    # Create ksize
    ksize = (1, pool_ksize[0], pool_ksize[1], 1)

    # Create strides
    strides=(1,pool_strides[0], pool_strides[1], 1)

    pool = tf.nn.max_pool(conv, ksize, strides, padding='SAME')

    print('Convolutional layer with conv_num_outputs:',conv_num_outputs,
    'conv_ksize:', conv_ksize,
    'conv_strides:', conv_strides,
    'pool_ksize:',pool_ksize,
    'pool_strides', pool_strides)
    print('layer input shape', x_tensor.get_shape().as_list(),
    'layer output shape', pool.get_shape().as_list())

    return pool
```

حال لایه flatten و fully connected آن را ایجاد می‌کنیم:

```
def flatten(x_tensor):
    _, height, width, channels = x_tensor.get_shape().as_list()
```

```
    net = tf.reshape(x_tensor, shape=[-1, height * width * channels])
    print('flatten shape', net.get_shape().as_list())
    return net


    def fully_conn(x_tensor, num_outputs):
    _, size = x_tensor.get_shape().as_list()
    weights = tf.Variable(tf.truncated_normal([size, num_outputs], stddev=0.05))
    bias = tf.Variable(tf.truncated_normal([num_outputs], stddev=0.05))




    fully_connected = tf.add(tf.matmul(x_tensor, weights), bias)

    print('layer input shape', x_tensor.get_shape().as_list(),
    'layer output shape', fully_connected.get_shape().as_list())

    return fully_connected
```

حالا معماری شبکه عصبی را مشخص می‌کنیم:

```
def conv_net(input_x, keep_probability):

    net = conv2d_maxpool(input_x,   32, (7,7), (2,2), (2,2), (2,2))
    net = conv2d_maxpool(net, 64, (3,3), (1,1), (2,2), (2,2))
    net = conv2d_maxpool(net, 128, (2,2), (1,1), (2,2), (2,2))

    net = flatten(net)
    net = tf.nn.dropout(net, keep_probability)
    net = fully_conn(net, 1024)
    net = tf.nn.dropout(net, keep_probability)
    net = fully_conn(net, 128)
    net = fully_conn(net, 10)

    return net
```

حالا شبکه عصبی را می‌سازیم:

```
##############################
## Build the Neural Network ##
##############################

# Remove previous weights, bias, inputs, etc..
tf.reset_default_graph()
```

```
IMAGE_SHAPE = (32, 32, 3)
LABELS_COUNT = 10

# Inputs
x = neural_net_image_input(IMAGE_SHAPE)
y = neural_net_label_input(LABELS_COUNT)
keep_prob = neural_net_keep_prob_input()

# Model
logits = conv_net(x, keep_prob)

# Name logits Tensor, so that is can be loaded from disk after training
logits = tf.identity(logits, name='logits')

# Loss and Optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits,
labels=y))
optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize(cost)

# Accuracy
correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32), name='accuracy')
```

خروجی:

```
Convolutional layer with conv_num_outputs: 32 conv_ksize: (7, 7)
conv_strides: (2, 2) pool_ksize: (2, 2) pool_strides (2, 2)
layer input shape [None, 32, 32, 3] layer output shape [None, 8, 8, 32]
Convolutional layer with conv_num_outputs: 64 conv_ksize: (3, 3)
conv_strides: (1, 1) pool_ksize: (2, 2) pool_strides (2, 2)
layer input shape [None, 8, 8, 32] layer output shape [None, 4, 4, 64]
Convolutional layer with conv_num_outputs: 128 conv_ksize: (2, 2)
conv_strides: (1, 1) pool_ksize: (2, 2) pool_strides (2, 2)
layer input shape [None, 4, 4, 64] layer output shape [None, 2, 2, 128]
flatten shape [None, 512]
layer input shape [None, 512] layer output shape [None, 1024]
layer input shape [None, 1024] layer output shape [None, 128]
layer input shape [None, 128] layer output shape [None, 10]
```

سپس نتایج آنرا نشان می‌دهم:

```
valid_features, valid_labels = pickle.load(open('preprocess_validation.p', mode='rb'))

        def print_stats(session, feature_batch, label_batch, cost, accuracy):
        batch_loss = session.run(cost, feed_dict=\
        {x:feature_batch, y:label_batch, keep_prob:1.0})
        batch_accuracy = session.run(accuracy, feed_dict=\
```

```
        {x:valid_features, y:valid_labels, keep_prob:1.0})

        print('batch loss is : ', batch_loss)
        print('batch_accuracy accuracy is :',batch_accuracy)
```

حال من Hyperparameter های آنرا مشخص کردم:

```
#For my laptop I selected:
    epochs = 50
    batch_size = 256

    # keep probability of 70%
    keep_probability = 0.5
```

```
#a couple of helper functions for loading a single batch
    def batch_features_labels(features, labels, batch_size):
    """
    Split features and labels into batches
    """
    for start in range(0, len(features), batch_size):
    end = min(start + batch_size, len(features))
    yield features[start:end], labels[start:end]

    def load_preprocess_training_batch(batch_id, batch_size):
    """
    Load the Preprocessed Training data and return them in batches of <batch_size> or
less
    """
    filename = 'preprocess_batch_' + str(batch_id) + '.p'
    features, labels = pickle.load(open(filename, mode='rb'))

    # Return the training data in batches of size <batch_size> or less
    return batch_features_labels(features, labels, batch_size)
```

سوال ۳

در این سوال از ما خواسته شده بررسی کنیم که اگر تعداد پارامترها ثابت باشد، عمق شبکه را اگر زیاد یا کم کنیم چه تاثیری می‌گذارد. پاسخ در این است که اگر بیش از حد عمق را زیاد کنیم هم باعث overfitting می‌شود و هم این که باعث می‌شود محاسبات بیش از حد انجام بدهیم که این یعنی از منابع که شامل سخت افزار سیستم می‌شود، بیهوده استفاده کرده‌ایم.

(A

برای اینکار کافی است از سه لایه ای که من طراحی کنیم فقط یک لایه رو در نظر بگیریم و همین کد را دوباره اجرا کنیم.

(B

برای این قسمت گفته اید حداقل یک لایه و خب من همان سه لایه خودم را در نظر می‌گیریم.

(C

از نتایج بدست آمده نتیجه گرفتم accuracy به طور محسوسی کاهش یافت!

سوال ۴

در این سوال نیاز است لایه های مان را فریز کنیم:

دستورات آن به شکل زیر است:

```
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        from __future__ import print_function
        import keras
        from keras.preprocessing.image import ImageDataGenerator, load_img
```

حال برای اینکه بتوانیم دیتاست تصاویر را بخوانیم آن را در google drive خود قرار می‌دهم:

```
# i have got this code from this link(way number 3): https://towardsdatascience.com/3-ways-to-load-csv-files-into-colab-7c14fcbdcb92
        # Code to read csv file into Colaboratory:

        !pip install -U -q PyDrive
        from pydrive.auth import GoogleAuth
        from pydrive.drive import GoogleDrive
        from google.colab import auth
        from oauth2client.client import GoogleCredentials
```

خروجی:

```
  100% |████████████████████████████████| 993kB 20.3MB/s
Building wheel for PyDrive (setup.py) ... done
```

```
#Authenticate and create the PyDrive client.
    auth.authenticate_user()
    gauth = GoogleAuth()
    gauth.credentials = GoogleCredentials.get_application_default()
    drive = GoogleDrive(gauth)

link = 'https://drive.google.com/open?id=1t1-PPc5mO6OvlilY6ffkFrOJoqKSk3RJ'
fluff, id = link.split('=')

print (id) # Verify that you have everything after '='
```

حال آدرس فولدر های train و validation را به آن می‌دهیم:

```
        train_dir = ' https://drive.google.com/open?id=15COsuoR3zbfiToquwB49d5j-
epI5K0I2
       validation_dir = '
https://drive.google.com/open?id=1kNo13cgcaDsvt_NC71WQIbDuYwhE6wbY'
        image_size = 224
```

حال مدل مان را بدست می‌آوریم:

```
from keras.applications import VGG16

    #Load the VGG model
    vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(image_size,
image_size, 3))

    # Freeze all the layers
    for layer in vgg_conv.layers[:]:
    layer.trainable = False

    # Check the trainable status of the individual layers
    for layer in vgg_conv.layers:
    print(layer, layer.trainable)


    from keras import models
    from keras import layers
    from keras import optimizers

    # Create the model
    model = models.Sequential()

    # Add the vgg convolutional base model
    model.add(vgg_conv)

    # Add new layers
    model.add(layers.Flatten())
    model.add(layers.Dense(1024, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(3, activation='softmax'))

    # Show a summary of the model. Check the number of trainable parameters
    model.summary()
```

خروجی:

```
<keras.engine.topology.InputLayer object at 0x7f2eed2f01d0> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed2f0240> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed2f0390> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2eed2f0518> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed29fa58> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed29f860> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2eed23f3c8> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed260710> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed260048> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed203f28> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2eed225da0> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed1c8278> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed1c8898> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed1e97f0> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2eed17c240> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed19d860> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed19d128> False
<keras.layers.convolutional.Conv2D object at 0x7f2eed13d780> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2eed15fda0> False
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              25691136
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 3075
=================================================================
Total params: 40,408,899
Trainable params: 25,694,211
Non-trainable params: 14,714,688
_________________________________________________________________
```

حالا باید مدل را آموزش دهیم:

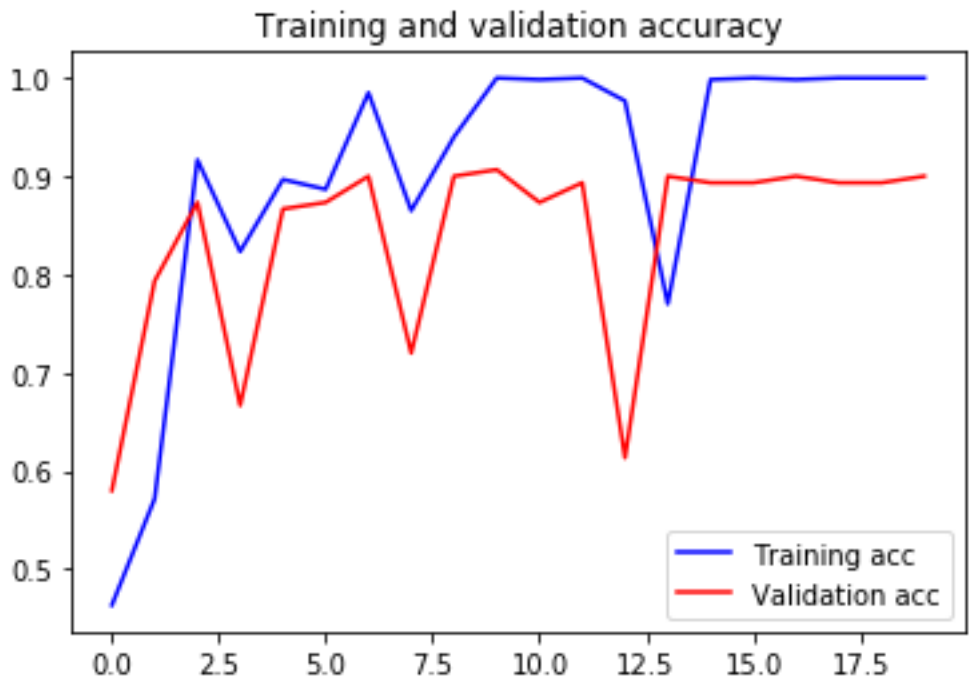
```
        # No Data augmentation
        train_datagen = ImageDataGenerator(rescale=1./255)
        validation_datagen = ImageDataGenerator(rescale=1./255)

        # Change the batchsize according to your system RAM
        train_batchsize = 100
        val_batchsize = 10

        # Data Generator for Training data
        train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(image_size, image_size),
        batch_size=train_batchsize,
```

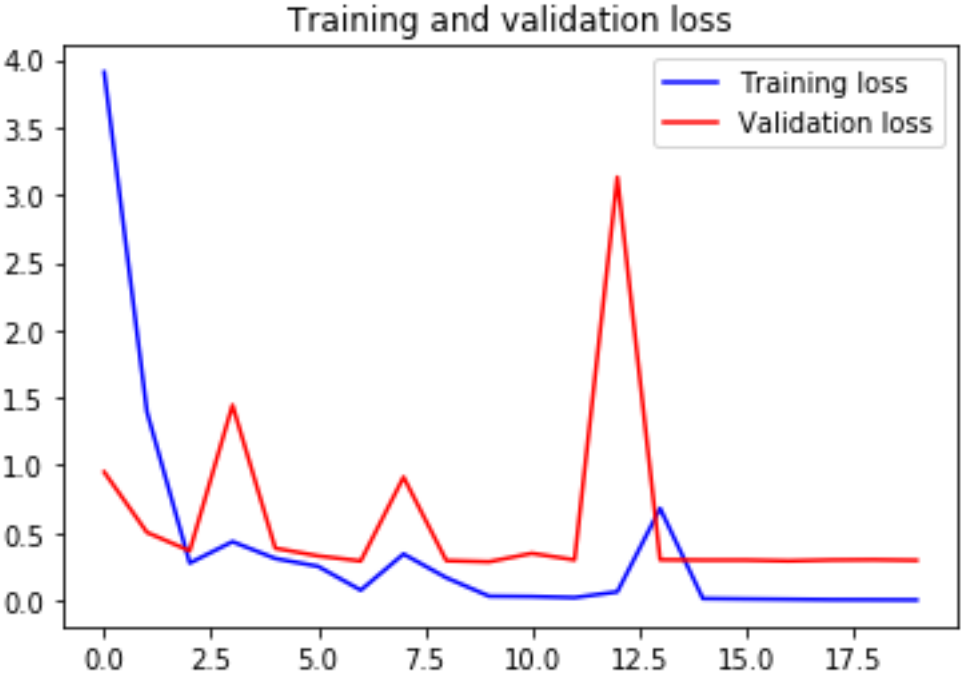
```
        class_mode='categorical')

        # Data Generator for Validation data
        validation_generator = validation_datagen.flow_from_directory(
        validation_dir,
        target_size=(image_size, image_size),
        batch_size=val_batchsize,
        class_mode='categorical',
        shuffle=False)

        # Compile the model
        model.compile(loss='categorical_crossentropy',
        optimizer=optimizers.RMSprop(lr=1e-4),
        metrics=['acc'])

        # Train the Model
        history = model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.samples/train_generator.batch_size ,
        epochs=20,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples/validation_generator.batch_size,
        verbose=1)

        # Save the Model
        model.save('all_freezed.h5')

        # Plot the accuracy and loss curves
        acc = history.history['acc']
        val_acc = history.history['val_acc']
        loss = history.history['loss']
        val_loss = history.history['val_loss']

        epochs = range(len(acc))

        plt.plot(epochs, acc, 'b', label='Training acc')
        plt.plot(epochs, val_acc, 'r', label='Validation acc')
        plt.title('Training and validation accuracy')
        plt.legend()

        plt.figure()

        plt.plot(epochs, loss, 'b', label='Training loss')
        plt.plot(epochs, val_loss, 'r', label='Validation loss')
        plt.title('Training and validation loss')
        plt.legend()

        plt.show()
```

خروجی:

```
Found 600 images belonging to 3 classes.
Found 150 images belonging to 3 classes.
Epoch 1/20
6/6 [==============================] - 5s - loss: 3.9083 - acc: 0.4633 -
val_loss: 0.9497 - val_acc: 0.5800
Epoch 2/20
6/6 [==============================] - 2s - loss: 1.3948 - acc: 0.5717 -
val_loss: 0.5033 - val_acc: 0.7933
```

```
Epoch 3/20
6/6 [==============================] - 2s - loss: 0.2767 - acc: 0.9167 -
val_loss: 0.3672 - val_acc: 0.8733
Epoch 4/20
6/6 [==============================] - 2s - loss: 0.4351 - acc: 0.8233 -
val_loss: 1.4450 - val_acc: 0.6667
Epoch 5/20
6/6 [==============================] - 2s - loss: 0.3103 - acc: 0.8967 -
val_loss: 0.3865 - val_acc: 0.8667
Epoch 6/20
6/6 [==============================] - 2s - loss: 0.2525 - acc: 0.8867 -
val_loss: 0.3298 - val_acc: 0.8733
Epoch 7/20
6/6 [==============================] - 2s - loss: 0.0763 - acc: 0.9850 -
val_loss: 0.2922 - val_acc: 0.9000
Epoch 8/20
6/6 [==============================] - 2s - loss: 0.3435 - acc: 0.8650 -
val_loss: 0.9127 - val_acc: 0.7200
Epoch 9/20
6/6 [==============================] - 2s - loss: 0.1685 - acc: 0.9400 -
val_loss: 0.2946 - val_acc: 0.9000
Epoch 10/20
6/6 [==============================] - 2s - loss: 0.0318 - acc: 1.0000 -
val_loss: 0.2864 - val_acc: 0.9067
Epoch 11/20
6/6 [==============================] - 2s - loss: 0.0291 - acc: 0.9983 -
val_loss: 0.3485 - val_acc: 0.8733
Epoch 12/20
6/6 [==============================] - 2s - loss: 0.0215 - acc: 1.0000 -
val_loss: 0.3003 - val_acc: 0.8933
Epoch 13/20
6/6 [==============================] - 2s - loss: 0.0640 - acc: 0.9767 -
val_loss: 3.1314 - val_acc: 0.6133
Epoch 14/20
6/6 [==============================] - 2s - loss: 0.6810 - acc: 0.7700 -
val_loss: 0.3014 - val_acc: 0.9000
Epoch 15/20
6/6 [==============================] - 2s - loss: 0.0139 - acc: 0.9983 -
val_loss: 0.2988 - val_acc: 0.8933
Epoch 16/20
6/6 [==============================] - 2s - loss: 0.0109 - acc: 1.0000 -
val_loss: 0.2988 - val_acc: 0.8933
Epoch 17/20
6/6 [==============================] - 2s - loss: 0.0085 - acc: 0.9983 -
val_loss: 0.2928 - val_acc: 0.9000
Epoch 18/20
6/6 [==============================] - 2s - loss: 0.0057 - acc: 1.0000 -
val_loss: 0.2994 - val_acc: 0.8933
Epoch 19/20
6/6 [==============================] - 2s - loss: 0.0049 - acc: 1.0000 -
val_loss: 0.3011 - val_acc: 0.8933
Epoch 20/20
6/6 [==============================] - 2s - loss: 0.0037 - acc: 1.0000 -
val_loss: 0.2972 - val_acc: 0.9000
```


Training and validation accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Training acc
Validation acc


Training and validation loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Training loss
Validation loss

حال برای آزمایش ۴ لایه آخر لایه های convolutional را در نظر می‌گیریم:

```
from keras.applications import VGG16

    #Load the VGG model
    vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(image_size,
image_size, 3))

    # Freeze all the layers
    for layer in vgg_conv.layers[:-4]:
    layer.trainable = False

    # Check the trainable status of the individual layers
    for layer in vgg_conv.layers:
    print(layer, layer.trainable)


    from keras import models
    from keras import layers
    from keras import optimizers

    # Create the model
    model = models.Sequential()

    # Add the vgg convolutional base model
    model.add(vgg_conv)

    # Add new layers
    model.add(layers.Flatten())
    model.add(layers.Dense(1024, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(3, activation='softmax'))

    # Show a summary of the model. Check the number of trainable parameters
    model.summary()
```

خروجی:

```
<keras.engine.topology.InputLayer object at 0x7f2eec8eeba8> False
<keras.layers.convolutional.Conv2D object at 0x7f2eec8ee320> False
<keras.layers.convolutional.Conv2D object at 0x7f2eec8eed30> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2eeb9551d0> False
<keras.layers.convolutional.Conv2D object at 0x7f2eeb8f9dd8> False
<keras.layers.convolutional.Conv2D object at 0x7f2eec5e3a58> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2eeb906b00> False
<keras.layers.convolutional.Conv2D object at 0x7f2eea3b61d0> False
<keras.layers.convolutional.Conv2D object at 0x7f2eea3b6518> False
<keras.layers.convolutional.Conv2D object at 0x7f2eea3a3b70> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2eea3df630> False
<keras.layers.convolutional.Conv2D object at 0x7f2eea3d6a20> False
<keras.layers.convolutional.Conv2D object at 0x7f2eea3d6828> False
<keras.layers.convolutional.Conv2D object at 0x7f2eea25a908> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2eea27bef0> False
<keras.layers.convolutional.Conv2D object at 0x7f2eea220438> True
<keras.layers.convolutional.Conv2D object at 0x7f2eea220f60> True
<keras.layers.convolutional.Conv2D object at 0x7f2eea1c0cc0> True
<keras.layers.pooling.MaxPooling2D object at 0x7f2eea1e2fd0> True
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_4 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_7 (Dense)              (None, 1024)              25691136
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 3075
=================================================================
Total params: 40,408,899
Trainable params: 32,773,635
Non-trainable params: 7,635,264
_________________________________________________________________
```

در نهایت دوباره مدل را آموزش می‌دهیم و از آن تست می‌گیریم:

```
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 = 50
val_batchsize = 10

# Data Generator for Training data
train_generator = train_datagen.flow_from_directory(
train_dir,
target_size=(image_size, image_size),
batch_size=train_batchsize,
class_mode='categorical')

# Data Generator for Validation data
validation_generator = validation_datagen.flow_from_directory(
validation_dir,
target_size=(image_size, image_size),
batch_size=val_batchsize,
class_mode='categorical',
shuffle=False)

# Compile the model
```

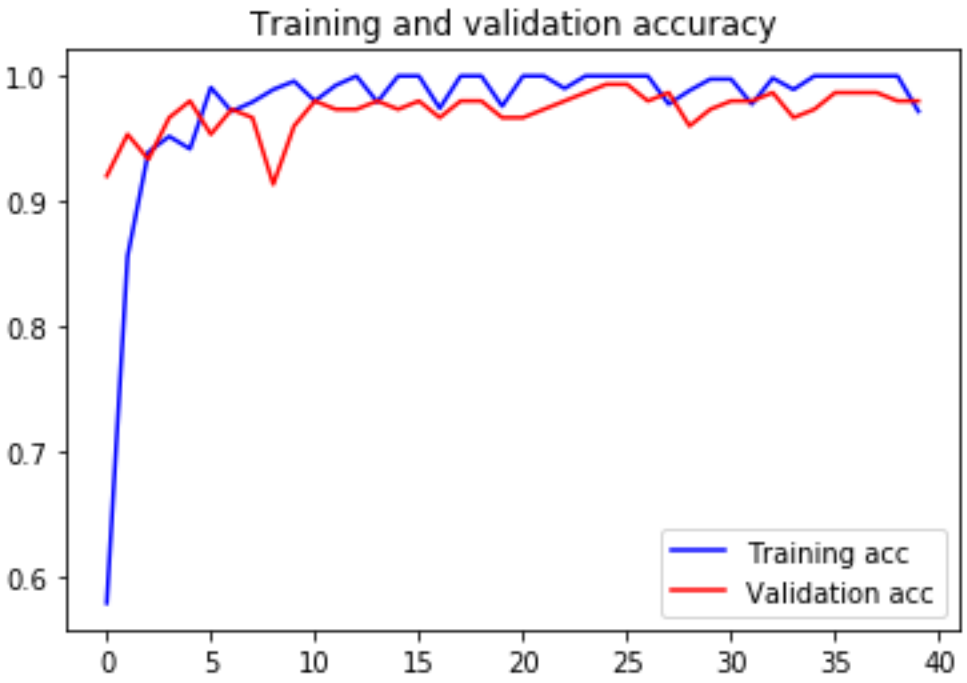
```
        model.compile(loss='categorical_crossentropy',
        optimizer=optimizers.RMSprop(lr=1e-4),
        metrics=['acc'])

        # Train the Model
        # NOTE that we have multiplied the steps_per_epoch by 2. This is because we
are using data augmentation.
        history = model.fit_generator(
        train_generator,
        steps_per_epoch=2*train_generator.samples/train_generator.batch_size ,
        epochs=40,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples/validation_generator.batch_size,
        verbose=1)

        # Save the Model
        model.save('da_last4_layers.h5')

        # Plot the accuracy and loss curves
        acc = history.history['acc']
        val_acc = history.history['val_acc']
        loss = history.history['loss']
        val_loss = history.history['val_loss']

        epochs = range(len(acc))

        plt.plot(epochs, acc, 'b', label='Training acc')
        plt.plot(epochs, val_acc, 'r', label='Validation acc')
        plt.title('Training and validation accuracy')
        plt.legend()

        plt.figure()

        plt.plot(epochs, loss, 'b', label='Training loss')
        plt.plot(epochs, val_loss, 'r', label='Validation loss')
        plt.title('Training and validation loss')
        plt.legend()

        plt.show()
```

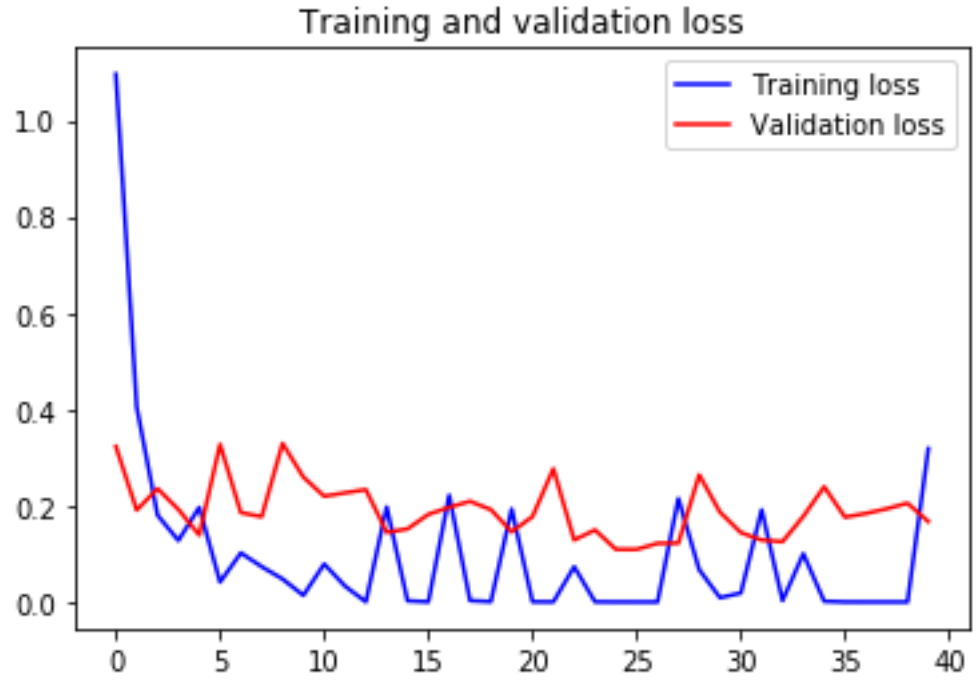
خروجی:

```
Found 600 images belonging to 3 classes.
Found 150 images belonging to 3 classes.
Epoch 1/40
24/24 [==============================] - 8s - loss: 1.0960 - acc: 0.5783 -
val_loss: 0.3230 - val_acc: 0.9200
Epoch 2/40
24/24 [==============================] - 7s - loss: 0.4048 - acc: 0.8558 -
val_loss: 0.1907 - val_acc: 0.9533
Epoch 3/40
24/24 [==============================] - 8s - loss: 0.1802 - acc: 0.9392 -
val_loss: 0.2353 - val_acc: 0.9333
Epoch 4/40
24/24 [==============================] - 7s - loss: 0.1277 - acc: 0.9517 -
val_loss: 0.1922 - val_acc: 0.9667
Epoch 5/40
24/24 [==============================] - 8s - loss: 0.1963 - acc: 0.9417 -
val_loss: 0.1390 - val_acc: 0.9800
```

```
Epoch 6/40
24/24 [==============================] - 7s - loss: 0.0414 - acc: 0.9908 -
val_loss: 0.3272 - val_acc: 0.9533
Epoch 7/40
24/24 [==============================] - 7s - loss: 0.1018 - acc: 0.9717 -
val_loss: 0.1854 - val_acc: 0.9733
Epoch 8/40
24/24 [==============================] - 7s - loss: 0.0733 - acc: 0.9792 -
val_loss: 0.1771 - val_acc: 0.9667
Epoch 9/40
24/24 [==============================] - 7s - loss: 0.0478 - acc: 0.9892 -
val_loss: 0.3292 - val_acc: 0.9133
Epoch 10/40
24/24 [==============================] - 7s - loss: 0.0138 - acc: 0.9958 -
val_loss: 0.2593 - val_acc: 0.9600
Epoch 11/40
24/24 [==============================] - 7s - loss: 0.0788 - acc: 0.9800 -
val_loss: 0.2195 - val_acc: 0.9800
Epoch 12/40
24/24 [==============================] - 7s - loss: 0.0329 - acc: 0.9925 -
val_loss: 0.2266 - val_acc: 0.9733
Epoch 13/40
24/24 [==============================] - 7s - loss: 2.2249e-04 - acc: 1.0000
- val_loss: 0.2333 - val_acc: 0.9733
Epoch 14/40
24/24 [==============================] - 7s - loss: 0.1973 - acc: 0.9792 -
val_loss: 0.1446 - val_acc: 0.9800
Epoch 15/40
24/24 [==============================] - 7s - loss: 0.0024 - acc: 1.0000 -
val_loss: 0.1517 - val_acc: 0.9733
Epoch 16/40
24/24 [==============================] - 7s - loss: 2.6369e-04 - acc: 1.0000
- val_loss: 0.1819 - val_acc: 0.9800
Epoch 17/40
24/24 [==============================] - 8s - loss: 0.2217 - acc: 0.9742 -
val_loss: 0.1965 - val_acc: 0.9667
Epoch 18/40
24/24 [==============================] - 7s - loss: 0.0028 - acc: 1.0000 -
val_loss: 0.2090 - val_acc: 0.9800
Epoch 19/40
24/24 [==============================] - 8s - loss: 6.4382e-04 - acc: 1.0000
- val_loss: 0.1917 - val_acc: 0.9800
Epoch 20/40
24/24 [==============================] - 7s - loss: 0.1931 - acc: 0.9758 -
val_loss: 0.1457 - val_acc: 0.9667
Epoch 21/40
24/24 [==============================] - 7s - loss: 4.4634e-04 - acc: 1.0000
- val_loss: 0.1774 - val_acc: 0.9667
Epoch 22/40
24/24 [==============================] - 7s - loss: 1.3386e-04 - acc: 1.0000
- val_loss: 0.2766 - val_acc: 0.9733
Epoch 23/40
24/24 [==============================] - 7s - loss: 0.0733 - acc: 0.9900 -
val_loss: 0.1287 - val_acc: 0.9800
Epoch 24/40
24/24 [==============================] - 7s - loss: 2.8142e-04 - acc: 1.0000
- val_loss: 0.1496 - val_acc: 0.9867
```

```
Epoch 25/40
24/24 [==============================] - 7s - loss: 4.9226e-05 - acc: 1.0000
- val_loss: 0.1093 - val_acc: 0.9933
Epoch 26/40
24/24 [==============================] - 7s - loss: 1.9475e-06 - acc: 1.0000
- val_loss: 0.1091 - val_acc: 0.9933
Epoch 27/40
24/24 [==============================] - 7s - loss: 5.5070e-06 - acc: 1.0000
- val_loss: 0.1217 - val_acc: 0.9800
Epoch 28/40
24/24 [==============================] - 7s - loss: 0.2148 - acc: 0.9775 -
val_loss: 0.1222 - val_acc: 0.9867
Epoch 29/40
24/24 [==============================] - 7s - loss: 0.0671 - acc: 0.9883 -
val_loss: 0.2639 - val_acc: 0.9600
Epoch 30/40
24/24 [==============================] - 7s - loss: 0.0088 - acc: 0.9975 -
val_loss: 0.1864 - val_acc: 0.9733
Epoch 31/40
24/24 [==============================] - 8s - loss: 0.0186 - acc: 0.9975 -
val_loss: 0.1440 - val_acc: 0.9800
Epoch 32/40
24/24 [==============================] - 7s - loss: 0.1914 - acc: 0.9775 -
val_loss: 0.1286 - val_acc: 0.9800
Epoch 33/40
24/24 [==============================] - 8s - loss: 0.0027 - acc: 0.9983 -
val_loss: 0.1252 - val_acc: 0.9867
Epoch 34/40
24/24 [==============================] - 7s - loss: 0.1001 - acc: 0.9892 -
val_loss: 0.1778 - val_acc: 0.9667
Epoch 35/40
24/24 [==============================] - 7s - loss: 0.0014 - acc: 1.0000 -
val_loss: 0.2393 - val_acc: 0.9733
Epoch 36/40
24/24 [==============================] - 7s - loss: 4.4375e-05 - acc: 1.0000
- val_loss: 0.1761 - val_acc: 0.9867
Epoch 37/40
24/24 [==============================] - 7s - loss: 1.3121e-06 - acc: 1.0000
- val_loss: 0.1838 - val_acc: 0.9867
Epoch 38/40
24/24 [==============================] - 7s - loss: 7.2186e-07 - acc: 1.0000
- val_loss: 0.1935 - val_acc: 0.9867
Epoch 39/40
24/24 [==============================] - 7s - loss: 8.2086e-07 - acc: 1.0000
- val_loss: 0.2051 - val_acc: 0.9800
Epoch 40/40
24/24 [==============================] - 7s - loss: 0.3185 - acc: 0.9717 -
val_loss: 0.1670 - val_acc: 0.9800
```


Training and validation accuracy
1.0
0.9
0.8
0.7
0.6
0
5
10
15
20
25
30
35
40
Training acc
Validation acc


Training and validation loss
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
35
40
Training loss
Validation loss

حال با تست می‌کنیم که عکس هارا درست پیش بینی می‌کند یا نه:

```
# Create a generator for prediction
        validation_generator = validation_datagen.flow_from_directory(
        validation_dir,
        target_size=(image_size, image_size),
        batch_size=val_batchsize,
        class_mode='categorical',
        shuffle=False)

        # Get the filenames from the generator
        fnames = validation_generator.filenames

        # Get the ground truth from generator
        ground_truth = validation_generator.classes

        # Get the label to class mapping from the generator
        label2index = validation_generator.class_indices

        # Getting the mapping from class index to class label
idx2label = dict((v,k) for k,v in label2index.items())

        # Get the predictions from the model using the generator
        predictions = model.predict_generator(validation_generator,
steps=validation_generator.samples/validation_generator.batch_size,verbose=1)
        predicted_classes = np.argmax(predictions,axis=1)

        errors = np.where(predicted_classes != ground_truth)[0]
        print("No of errors = {}/{}".format(len(errors),validation_generator.samples))

        # Show the errors
        for i in range(len(errors)):
        pred_class = np.argmax(predictions[errors[i]])
        pred_label = idx2label[pred_class]

        title = 'Original label:{}, Prediction :{}, confidence : {:.3f}'.format(
        fnames[errors[i]].split('/')[0],
        pred_label,
        predictions[errors[i]][pred_class])

        original = load_img('{}/{}'.format(validation_dir,fnames[errors[i]]))
        plt.figure(figsize=[7,7])
        plt.axis('off')
        plt.title(title)
        plt.imshow(original)
        plt.show()
```

خروجی:

```
Found 150 images belonging to 3 classes.
15/15 [==============================] - 0s
No of errors = 3/150
```

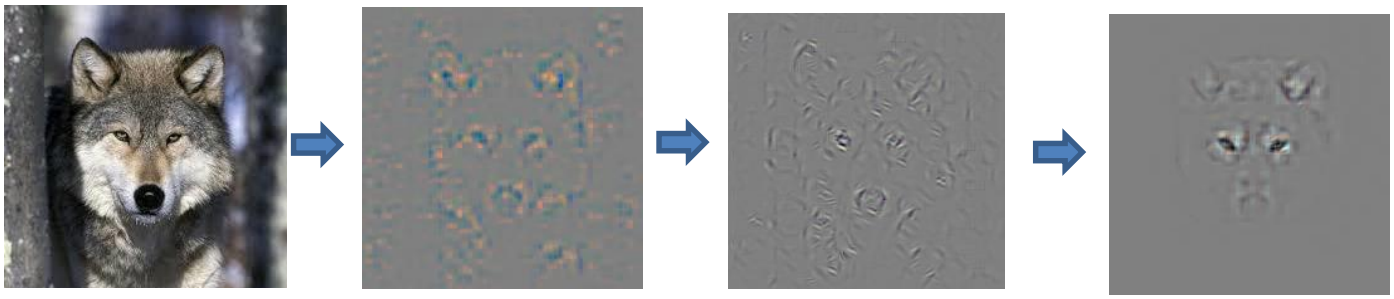
Original label:pumpkin, Prediction :watermelon, confidence : 0.999

Original label:tomato, Prediction :watermelon, confidence : 1.000

Original label:watermelon, Prediction :tomato, confidence : 0.432

سوال ۵

برای این سوال یک ارائه به صورت pdf تهیه کردم و آنرا مطالعه کردم و از نتایج آن فهمیدم بر روی عکس ها پردازش می‌کند تا feature هایی که میخواهد را بدست بیارد که نتایج آن مثل شکل های زیر است:



سوال ۶

در این سوال کافی است دستورات زیر اجرا شود:

```
print('Checking the Training on a Single Batch...')
with tf.Session() as session:
    # Initializing the variables
    session.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(epochs):
    batch_i = 1
    for batch_features, batch_labels in \
    load_preprocess_training_batch(batch_i, batch_size):
    session.run(optimizer, feed_dict=\
    {x:batch_features, y:batch_labels, keep_prob:keep_probability})
    print('Epoch {:>2}, CIFAR-10 Batch {}:  '.format(epoch + 1, batch_i), end='')
    print_stats(session, batch_features, batch_labels, cost, accuracy)
```

خروجی آن نیز به شکل زیر است:

```
Checking the Training on a Single Batch...
Epoch  1, CIFAR-10 Batch 1:  batch loss is :  2.04934
batch_accuracy accuracy is : 0.3524
```

```
Epoch  2, CIFAR-10 Batch 1:  batch loss is :  1.80641
batch_accuracy accuracy is : 0.4126
Epoch  3, CIFAR-10 Batch 1:  batch loss is :  1.54692
batch_accuracy accuracy is : 0.4486
Epoch  4, CIFAR-10 Batch 1:  batch loss is :  1.27744
batch_accuracy accuracy is : 0.4792
Epoch  5, CIFAR-10 Batch 1:  batch loss is :  1.13442
batch_accuracy accuracy is : 0.4928
Epoch  6, CIFAR-10 Batch 1:  batch loss is :  0.92034
batch_accuracy accuracy is : 0.496
Epoch  7, CIFAR-10 Batch 1:  batch loss is :  0.806121
batch_accuracy accuracy is : 0.4896
Epoch  8, CIFAR-10 Batch 1:  batch loss is :  0.745746
batch_accuracy accuracy is : 0.5048
Epoch  9, CIFAR-10 Batch 1:  batch loss is :  0.615836
batch_accuracy accuracy is : 0.531
Epoch 10, CIFAR-10 Batch 1:  batch loss is :  0.497367
batch_accuracy accuracy is : 0.5288
Epoch 11, CIFAR-10 Batch 1:  batch loss is :  0.442382
batch_accuracy accuracy is : 0.531
Epoch 12, CIFAR-10 Batch 1:  batch loss is :  0.381339
batch_accuracy accuracy is : 0.55
Epoch 13, CIFAR-10 Batch 1:  batch loss is :  0.365281
batch_accuracy accuracy is : 0.5634
Epoch 14, CIFAR-10 Batch 1:  batch loss is :  0.307827
batch_accuracy accuracy is : 0.5432
Epoch 15, CIFAR-10 Batch 1:  batch loss is :  0.26708
batch_accuracy accuracy is : 0.571
Epoch 16, CIFAR-10 Batch 1:  batch loss is :  0.222317
batch_accuracy accuracy is : 0.577
Epoch 17, CIFAR-10 Batch 1:  batch loss is :  0.191946
batch_accuracy accuracy is : 0.5728
Epoch 18, CIFAR-10 Batch 1:  batch loss is :  0.142695
batch_accuracy accuracy is : 0.5794
Epoch 19, CIFAR-10 Batch 1:  batch loss is :  0.176674
batch_accuracy accuracy is : 0.5508
Epoch 20, CIFAR-10 Batch 1:  batch loss is :  0.163542
batch_accuracy accuracy is : 0.579
Epoch 21, CIFAR-10 Batch 1:  batch loss is :  0.171183
batch_accuracy accuracy is : 0.5622
Epoch 22, CIFAR-10 Batch 1:  batch loss is :  0.132356
batch_accuracy accuracy is : 0.5728
Epoch 23, CIFAR-10 Batch 1:  batch loss is :  0.10712
batch_accuracy accuracy is : 0.5704
Epoch 24, CIFAR-10 Batch 1:  batch loss is :  0.101802
batch_accuracy accuracy is : 0.5618
Epoch 25, CIFAR-10 Batch 1:  batch loss is :  0.108635
batch_accuracy accuracy is : 0.5678
Epoch 26, CIFAR-10 Batch 1:  batch loss is :  0.112882
batch_accuracy accuracy is : 0.5596
Epoch 27, CIFAR-10 Batch 1:  batch loss is :  0.0682271
batch_accuracy accuracy is : 0.5884
Epoch 28, CIFAR-10 Batch 1:  batch loss is :  0.0640762
batch_accuracy accuracy is : 0.5856
Epoch 29, CIFAR-10 Batch 1:  batch loss is :  0.0570013
batch_accuracy accuracy is : 0.5866
Epoch 30, CIFAR-10 Batch 1:  batch loss is :  0.0763902
```

```
batch_accuracy accuracy is : 0.583
Epoch 31, CIFAR-10 Batch 1:  batch loss is :  0.0564564
batch_accuracy accuracy is : 0.571
Epoch 32, CIFAR-10 Batch 1:  batch loss is :  0.0532774
batch_accuracy accuracy is : 0.588
Epoch 33, CIFAR-10 Batch 1:  batch loss is :  0.0437081
batch_accuracy accuracy is : 0.5838
Epoch 34, CIFAR-10 Batch 1:  batch loss is :  0.0328673
batch_accuracy accuracy is : 0.5664
Epoch 35, CIFAR-10 Batch 1:  batch loss is :  0.0359792
batch_accuracy accuracy is : 0.5884
Epoch 36, CIFAR-10 Batch 1:  batch loss is :  0.0376761
batch_accuracy accuracy is : 0.5898
Epoch 37, CIFAR-10 Batch 1:  batch loss is :  0.0326239
batch_accuracy accuracy is : 0.5694
Epoch 38, CIFAR-10 Batch 1:  batch loss is :  0.0366496
batch_accuracy accuracy is : 0.589
Epoch 39, CIFAR-10 Batch 1:  batch loss is :  0.0224001
batch_accuracy accuracy is : 0.582
Epoch 40, CIFAR-10 Batch 1:  batch loss is :  0.0224597
batch_accuracy accuracy is : 0.5896
Epoch 41, CIFAR-10 Batch 1:  batch loss is :  0.0199231
batch_accuracy accuracy is : 0.5834
Epoch 42, CIFAR-10 Batch 1:  batch loss is :  0.0231632
batch_accuracy accuracy is : 0.5878
Epoch 43, CIFAR-10 Batch 1:  batch loss is :  0.0191447
batch_accuracy accuracy is : 0.5932
Epoch 44, CIFAR-10 Batch 1:  batch loss is :  0.0206479
batch_accuracy accuracy is : 0.5776
Epoch 45, CIFAR-10 Batch 1:  batch loss is :  0.0263408
batch_accuracy accuracy is : 0.5718
Epoch 46, CIFAR-10 Batch 1:  batch loss is :  0.0125673
batch_accuracy accuracy is : 0.5802
Epoch 47, CIFAR-10 Batch 1:  batch loss is :  0.0197601
batch_accuracy accuracy is : 0.5828
Epoch 48, CIFAR-10 Batch 1:  batch loss is :  0.0156334
batch_accuracy accuracy is : 0.5908
Epoch 49, CIFAR-10 Batch 1:  batch loss is :  0.011323
batch_accuracy accuracy is : 0.5846
Epoch 50, CIFAR-10 Batch 1:  batch loss is :  0.0132246
batch_accuracy accuracy is : 0.5784
```

حال مدل fully trained آن را بدست می‌آورم:

```
save_model_path = './image_classification'

        print('Training...')
        with tf.Session() as session:
        # Initializing the variables
        session.run(tf.global_variables_initializer())

        # Training cycle
        for epoch in range(epochs):
        # Loop over all batches
        n_batches = 5
```

```
        for batch_i in range(1, n_batches + 1):
        for batch_features, batch_labels in \
        load_preprocess_training_batch(batch_i, batch_size):
        session.run(optimizer, feed_dict=\
        {x:batch_features, y:batch_labels, keep_prob:keep_probability})

        # Save Model
        saver = tf.train.Saver()
        save_path = saver.save(session, save_model_path)
```

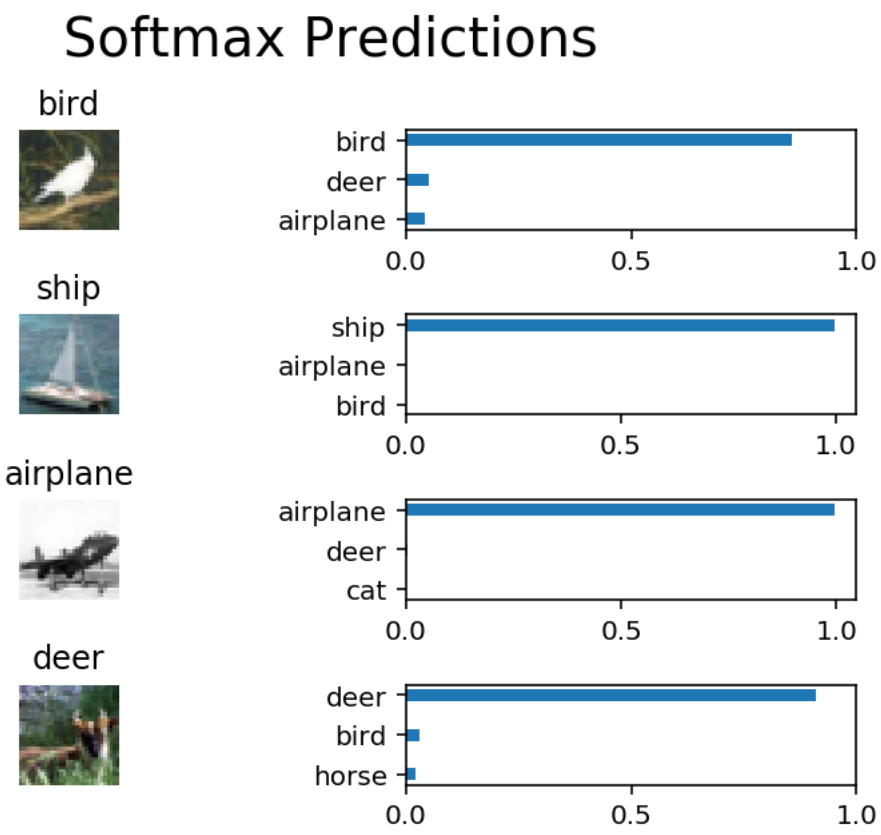
حال نوبت به آن رسید که تست کنیم تصاویری که می‌دهیم درست تشخیص می‌دهد یا نه.

```
        %matplotlib inline
        %config InlineBackend.figure_format = 'retina'

        import tensorflow as tf
        import pickle
        import random
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.preprocessing import LabelBinarizer

        def display_image_predictions(features, labels, predictions):
        label_binarizer = LabelBinarizer()
        label_binarizer.fit(range(LABELS_COUNT))
        label_ids = label_binarizer.inverse_transform(np.array(labels))

        fig, axies = plt.subplots(nrows=4, ncols=2)
        fig.tight_layout()
        fig.suptitle('Softmax Predictions', fontsize=20, y=1.1)

        n_predictions = 3
        margin = 0.05
        ind = np.arange(n_predictions)
        width = (1. - 2. * margin) / n_predictions

        for image_i, (feature, label_id, pred_indicies, pred_values) \
        in enumerate(zip(features, label_ids, predictions.indices,
predictions.values)):
        pred_names = [LABEL_NAMES[pred_i] for pred_i in pred_indicies]
        correct_name = LABEL_NAMES[label_id]

        axies[image_i][0].imshow(feature)
        axies[image_i][0].set_title(correct_name)
        axies[image_i][0].set_axis_off()

        axies[image_i][1].barh(ind + margin, pred_values[::-1], width)
        axies[image_i][1].set_yticks(ind + margin)
        axies[image_i][1].set_yticklabels(pred_names[::-1])
        axies[image_i][1].set_xticks([0, 0.5, 1.0])

        save_model_path = './image_classification'
        n_samples = 4
        top_n_predictions = 3

        def test_model():
```

```
        """
        Test the saved model against the test dataset
        """

        test_features, test_labels = pickle.load(open('preprocess_training.p',
mode='rb'))
        loaded_graph = tf.Graph()

        with tf.Session(graph=loaded_graph) as sess:
        # Load model
        loader = tf.train.import_meta_graph(save_model_path + '.meta')
        loader.restore(sess, save_model_path)

        # Get Tensors from loaded model
        loaded_x = loaded_graph.get_tensor_by_name('x:0')
        loaded_y = loaded_graph.get_tensor_by_name('y:0')
        loaded_keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')
        loaded_logits = loaded_graph.get_tensor_by_name('logits:0')
        loaded_acc = loaded_graph.get_tensor_by_name('accuracy:0')

        # Get accuracy in batches for memory limitations
        test_batch_acc_total = 0
        test_batch_count = 0

        for train_feature_batch, train_label_batch in \
        batch_features_labels(test_features, test_labels, batch_size):
        test_batch_acc_total += sess.run(
        loaded_acc,
        feed_dict={loaded_x: train_feature_batch, \
        loaded_y: train_label_batch, loaded_keep_prob: 1.0})
        test_batch_count += 1

        print('Testing Accuracy: {}\n'.format(test_batch_acc_total/test_batch_count))

        # Print Random Samples
        random_test_features, random_test_labels = \
        tuple(zip(*random.sample(list(zip(test_features, test_labels)), n_samples)))

        random_test_predictions = sess.run(
        tf.nn.top_k(tf.nn.softmax(loaded_logits), top_n_predictions),
        feed_dict={loaded_x: random_test_features, \
        loaded_y: random_test_labels, loaded_keep_prob: 1.0})
        display_image_predictions(random_test_features, \
        random_test_labels, random_test_predictions)


        test_model()
```

نتیجه ای که بدست آمد به شرح زیر است:

```
Testing Accuracy: 0.6734375
```



به صد دفتر نشاید گفت شرح الحالِ مشتاقی
به پایان آمد این دفتر، حکایت همچنان باقی

ان افعل ما تری آنی علی عهدی و میثاقی
کتاب بالغُ منی حیبا معرضاً عنی

که خود را بر تو می‌بندم به سالوسی و زرّاقی
نگویم نسبتی دارم به نزدیکانِ درگاهت

مریض العشق لایبری و لایشکو الی الرّاقی
اخلایی و احبابی ذروا من حبه مابی

تو را گر خواب می گیرد نه صاحب دردِ عشّاقی | نشانِ عاشق آن باشد که شب با روز پیوندد

اما انت الذی تسقی فعین السّم تریاقی | قم املا و اسقنی کأسا و دع ما فیه مسموما

مرا بگذار تا حیران بماند چشم در ساقی | قدح چون دورِ ما باشد به هشیارانِ مجلس ده

انا المجنون لا اعبا باحراقِ و اغراقِ | سعی فی هتکی الشانی و لما یدر ماشانی

مگر نفسِ ملک باشد بدین پاکیزه اخلاقی | مگر شمسِ فلک باشد بدین فرخنده دیداری

و هذا الظبی فی شیراز یسبینی باحداقِ | لقیت الاسد فی الغابات لاتقوی علی صیدی

بمیرد تشنه مستسقی و دریا همچنان باقی | نه حسنت آخری دارد، نه سعدی را سخن پایان

پایان