CV-Tricks.com
Best of Deep Learning & Computer vision (http://cv-tricks.com/)

Tensorflow Tutorial 2: image classifier using convolutional neural network

by ANKIT SACHAN

In this Tensorflow tutorial, we shall build a convolutional neural network based image classifier using Tensorflow. If you are just getting started with Tensorflow, then it would be a good idea to read the basic Tensorflow tutorial here (http://cv-tricks.com/artificial-intelligence/deep-learning/deep-learning-frameworks/tensorflow/tensorflow-tutorial/).

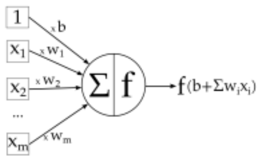
To demonstrate how to build a convolutional neural network based image classifier, we shall build a 6 layer neural network that will identify and separate images of dogs from that of cats. This network that we shall build is a very small network that you can run on a CPU as well. Traditional neural networks that are very good at doing image classification have many more paramters and take a lot of time if trained on CPU. However, in this post, my objective is to show you how to build a real-world convolutional neural network using Tensorflow rather than participating in ILSVRC (http://image-net.org/challenges/LSVRC/). Before we start with Tensorflow tutorial, let's cover basics of convolutional neural network. If you are already familiar with conv-nets(and call them conv-nets), you can move to part-2 i.e. Tensorflow tutorial.

Part-1: Basics of Convolutional Neural networks:

Neural Networks are essentially mathematical models to solve an optimization problem. They are made of neurons, the basic computation unit of neural networks. A neuron takes an input(say x), do some computation on it(say: multiply it with a variable w and adds another variable b) to produce a value (say; z= wx+b). This value is passed to a non-linear function called **activation function(f)** to produce the final output(activation) of a neuron. There are many kinds of activation functions. One of the popular activation function is **Sigmoid,** which is:

$$y = \frac{1}{1+e^{-z}}$$

The neuron which uses sigmoid function as an activation function will be called **Sigmoid neuron**. Depending on the activation functions, neurons are named and there are many kinds of them like **RELU, TanH** etc(remember this). One neuron can be connected to multiple neurons, like this:

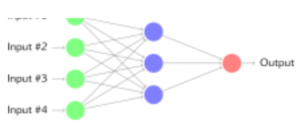


In this example, you can see that the weights are the property of the connection, i.e. each connection has a different weight value while bias is the property of the neuron. This is the complete picture of a sigmoid neuron which produces output y:

$$z = b + \sum_i x_i w_i \qquad y = \frac{1}{1+e^{-z}}$$

Layers:

If you stack neurons in a single line, it's called a layer; which is the next building block of neural networks.



As you can see above, the neurons in green make 1 layer which is the first layer of the network through which input data is passed to the network. Similarly, the last layer is called output layer as shown in red. The layers in between input and output layer are called hidden layers. In this example, we have only 1 hidden layer shown in blue. The networks which have many hidden layers tend to be more accurate and are called deep network and hence machine learning algorithms which uses these deep networks are called deep learning.

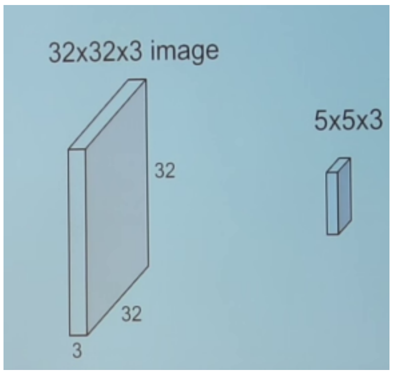
Types of layers:

Typically, all the neurons in one layer, do similar kind of mathematical operations and that's how that a layer gets its name(Except for input and output layers as they do little mathematical operations). Here are the most popular kinds of layers you should know about:

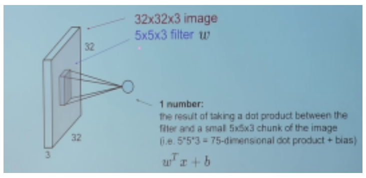
I.

Convolutional Layer:

Convolution is a mathematical operation that's used in single processing to filter signals, find patterns in signals etc. In a convolutional layer, all neurons apply convolution operation to the inputs, hence they are called convolutional neurons. The most important parameter in a convolutional neuron is the filter size, let's say we have a layer with filter size 5*5*3. Also, assume that the input that's fed to convolutional neuron is an input image of size of 32*32 with 3 channels.

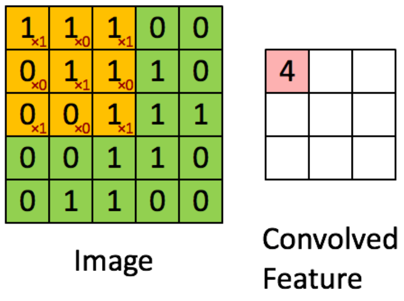


Let's pick one 5*5*3(3 for number of channels in a colored image) sized chunk from image and calculate convolution(dot product) with our filter(w). This one convolution operation will result in a single number as output. We shall also add the bias(b) to this output.


32x32x3 image
5x5x3 filter w
32
32
3
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)
$w^T x + b$

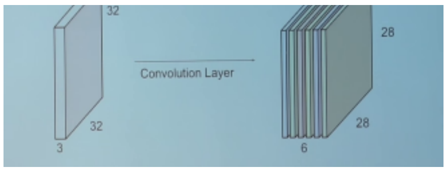
In order to calculate the dot product, it's mandatory for the 3rd dimension of the filter to be same as the number of channels in the input. i.e. when we calculate the dot product it's a matrix multiplication of 5*5*3 sized chunk with 5*5*3 sized filter.

We shall slide convolutional filter over whole input image to calculate this output across the image as shown by a schematic below:



In this case, we slide our window by 1 pixel at a time. If some cases, people slide the windows by more than 1 pixel. This number is called ***stride*.**

If you concatenate all these outputs in 2D, we shall have an output ***activation map*** of size 28*28(can you think of why 28*28 from 32*32 with the filter of 5*5 and stride of 1). Typically, we use more than 1 filter in one convolution layer. If we have 6 filters in our example, we shall have an output of size 28*28*6.

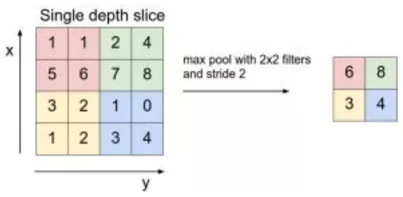


As you can see, after each convolution, the output reduces in size(as in this case we are going from 32*32 to 28*28). In a deep neural network with many layers, the output will become very small this way, which doesn't work very well. So, it's a standard practice to add zeros on the boundary of the input layer such that the output is the same size as input layer. So, in this example, if we add a padding of size 2 on both sides of the input layer, the size of the output layer will be 32*32*6 which works great from the implementation purpose as well. Let's say you have an input of size N*N, filter size is F, you are using S as stride and input is added with 0 pad of size P. Then, the output size will be:

(N-F+2P)/S +1

2. Pooling Layer:

Pooling layer is mostly used immediately after the convolutional layer to reduce the spatial size(only width and height, not depth). This reduces the number of parameters, hence computation is reduced. Also, less number of parameters avoid overfitting(don't worry about it now, will describe it little later). The most common form of pooling is **Max pooling** where we take a filter of size F*F and apply the maximum operation over the F*F sized part of the image.



If you take the average in place of taking maximum, it will be called average pooling, but it's not very popular.

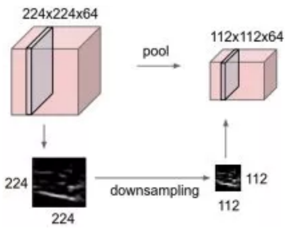
If your input is of size w1*h1*d1 and the size of the filter is f*f with stride S. Then the output sizes w2*h2*d2 will be:

w2= (w1-f)/S +1

h2=(h1-f)/S +1

d2=d1

Most common pooling is done with the filter of size 2*2 with a stride of 2. As you can calculate using the above formula, it essentially reduces the size of input by half.



3. Fully Connected Layer:

If each neuron in a layer receives input from all the neurons in the previous layer, then this layer is called fully connected layer. The output of this layer is computed by matrix multiplication followed by bias offset.

Understanding Training process:

Deep neural networks are nothing but mathematical models of intelligence which to a certain extent mimic human brains. When we are trying to train a neural network, there are two fundamental things we need to do:

I.
The Architecture of the network:

When designing the architecture of a neural network you have to decide on: How do you arrange layers? which layers to use? how many neurons to use in each layer etc.? Designing the architecture is slightly complicated and advanced topic and takes a lot of research. There are many standard architectures which work great for many standard problems. Examples being AlexNet, GoogleNet, InceptionResnet, VGG etc. In the beginning, you should only use the standard network architectures.

II.

Correct weights/parameters:

Once you have decided the architecture of the network; the second biggest variable is the weights(w) and biases(b) or the parameters of the network. The **objective of the training** is to get the best possible values of the all these parameters which solve the problem reliably. For example, when we are trying to build the classifier between dog and cat, we are looking to find parameters such that output layer gives out probability of dog as 1(or at least higher than cat) for all images of dogs and probability of cat as 1((or at least higher than dog) for all images of cats.

You can find the best set of parameters using a process called **Backward propagation**, i.e. you start with a random set of parameters and keep changing these weights such that for every training image we get the correct output. There are many optimizer methods to change the weights that are mathematically quick in finding the correct weights. GradientDescent is one such method(Backward propagation and optimizer methods to change the gradient is a very complicated topic. But we don't need to worry about it now as Tensorflow takes care of it).

So, let's say, we start with some initial values of parameters and feed 1 training image(in reality multiple images are fed together) of dog and we calculate the output of the network as 0.1 for it being a dog and 0.9 of it being a cat. Now, we do backward propagation to **slowly change** the parameters such that the probability of this image being a dog increases in the next iteration. There is a variable that is used to govern how fast do we change the parameters of the network during training, it's called **learning rate**. If you think about it, we want to maximise the total correct classifications by the network i.e. we care for the whole training set; we want to make these changes such that the number of correct classifications by the network increases. So we define a single number called **cost** which indicates if the training is going in the right direction. **Typically cost is defined in such a way that; as the cost is reduced, the accuracy of the network increases.** So, we keep an eye on the cost and we keep doing many iterations of forward and backward propagations(10s of thousands sometimes) till cost stops decreasing. There are many ways to define cost. One of the simple one is mean root square cost. Let's say $y_{prediction}$ is the vector containing the output of the network for all the training images and y_{actual} is the vector containing actual values(also called **ground truth**) of these labeled images. So, if we minimize the distance between these two variables, it would be a good indicator of the training. So, we define the cost as the average of these distances for all the images:

$$cost = 0.5\sum_{i=0}^{n}(y_{actual} - y_{prediction})^2$$

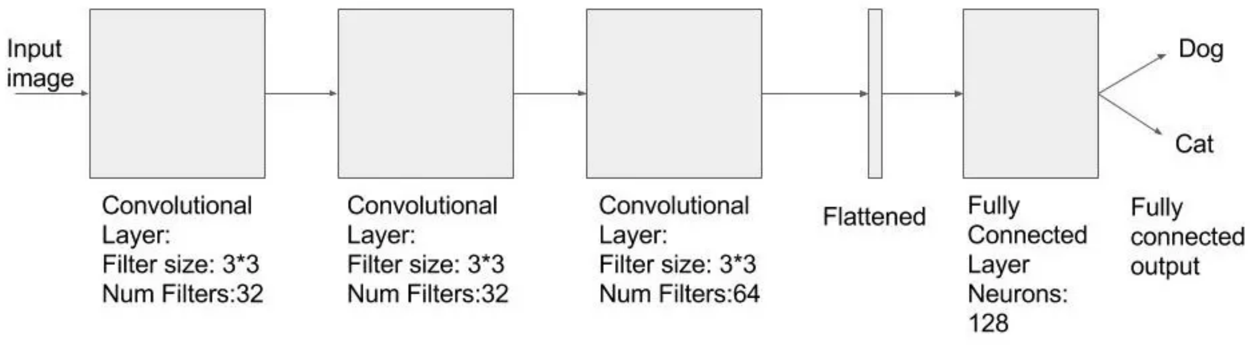
This is a very simple example of cost, but in actual training, we use much more complicated cost measures, like cross-entropy cost. But Tensorflow implements many of these costs so we don't need to worry about the details of these costs at this point in time.

After training is done, these parameters and architecture will be saved in a binary file(called **model**). In production set-up when we get a new image of dog/cat to classify, we load this model in the same network architecture and calculate the probability of the new image being a cat/dog. This is called ***inference*** or ***prediction***.

For computational simplicity, not all training data is fed to the network at once. Rather, let's say we have total 1600 images, we divide them in small batches say of size 16 or 32 called ***batch-size***. Hence, it will take 100 or 50 rounds(**iterations**) for complete data to be used for training. This is called **one *epoch,*** *i.e.* in one epoch the networks sees all the training images once. There are a few more things that are done to improve accuracy but let's not worry about everything at once.

Part-2: Tensorflow tutorial-> Building a small Neural network based image classifier:

Network that we will implement in this tutorial is smaller and simpler (than the ones that are used to solve real-world problems) so that you can train this on your cpu as well. While training, images from both the classes(dogs/cats) are fed to a convolutional layer which is followed by 2 more convolutional layers. After convolutional layers, we flatten the output and add two fully connected layer in the end. The second fully connected layer has only two outputs which represent the probability of an image being a cat or a dog.


Input image
Convolutional Layer: Filter size: 3*3 Num Filters:32
Convolutional Layer: Filter size: 3*3 Num Filters:32
Convolutional Layer: Filter size: 3*3 Num Filters:64
Flattened
Fully Connected Layer Neurons: 128
Dog
Cat
Fully connected output

a) Pre-requisites:

i) Shape function:

if you have multi-dimensional Tensor in TF, you can get the shape of it by doing this:

```
sess = tf.Session()
sess.run(tf.initialize_all_variables())
sess.run(tf.shape(a))
```

output will be: array([16, 128, 128,   3], dtype=int32)

You can reshape this to a new 2D Tensor of shape[16 128*128*3]= [16 49152].

```

b=tf.reshape(a,[16,49152])
sess.run(tf.shape(b))
```

Output: array([16, 49152], dtype=int32)

ii) **Softmax**: is a function that converts K-dimensional vector 'x' containing real values to the same shaped vector of real values in the range of (0,1), whose sum is 1. We shall apply the softmax function to the output of our convolutional neural network in order to, convert the output to the probability for each class.

$$o(x)_j = \frac{e^{x_i}}{\sum_{n=1}^{N} e^{x_n}} \; for\, j = 1 \ldots . N$$

b) Reading inputs:

I have used 500 images of dogs and cats each from Kaggle dataset (https://www.kaggle.com/c/dogs-vs-cats) but you could use any two image folders on your computer which contain two different kinds of objects. We divide our input data into 3 parts:

I. **Training data**: we shall use 80% i.e. 800 images for training.

II. **Validation data**: 20% images will be used for validation. These images are taken out of training data to calculate accuracy independently during the training process.

III. **Test set**: separate independent data for testing which has around 400 images. Sometimes due to something called **Overfitting;** after training, neural networks start working very well on the training data(and very similar images) i.e. the cost becomes very small, but they fail to work well for other images. For example, if you are training a classifier between dogs and cats and you get training data from someone who takes all images with white backgrounds. It's possible that your network works very well on this validation data-set, but if you try to run it on an image with a cluttered background, it will most likely fail. So, that's why we try to get our test-set from an independent source.

```

classes = ['dogs', 'cats']
num_classes = len(classes)

train_path='training_data'
test_path='testing_data'

# validation split
validation_size = 0.2

# batch size
batch_size = 16

data = dataset.read_train_sets(train_path, img_size, classes, validation_size=validation_size)
test_images, test_ids = dataset.read_test_set(test_path, img_size)
```

dataset is a class that I have created to read the input data. This is a simple python code that reads images from the provided training and testing data folders.

The objective of our training is to learn the correct values of weights/biases for all the neurons in the network that work to do classification between dog and cat. The Initial value of these weights can be taken anything but it works better if you take normal distributions(with mean zero and small variance). There are other methods to initialize the network but normal distribution is more prevalent. Let's create functions to create initial weights quickly just by specifying the shape(Remember we talked about truncated_normal function in the earlier post (http://cv-tricks.com/artificial-intelligence/deep-learning/deep-learning-frameworks/tensorflow/tensorflow-tutorial/)).

```

def new_weights(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.05))

def new_biases(length):
    return tf.Variable(tf.constant(0.05, shape=[length]))
```

c) Creating network layers:

i) Building convolution layer in TensorFlow:

tf.nn.conv2d function can be used to build a convolutional layer which takes these inputs:

input= the output(activation) from the previous layer. This should be a 4-D tensor. Typically, in the first convolutional layer, you pass n images of size width*height*num_channels, then this has the size [n width height num_channels] part of network design. If your filter is of size filter_size and input fed has num_input_channels and you have num_filters filters in your current layer, then **filter** will have following shape:

[filter_size filter_size num_input_channels num_filters]

strides= defines how much you move your filter when doing convolution. In this function, it needs to be a Tensor of size>=4 i.e. [batch_stride x_stride y_stride depth_stride]. batch_stride is always 1 as you don't want to skip images in your batch. x_stride and y_stride are same mostly and the choice is part of network design and we shall use them as 1 in our example. depth_stride is always set as 1 as you don't skip along the depth.

padding=SAME means we shall 0 pad the input such a way that output x,y dimensions are same as that of input.

After convolution, we add the biases of that neuron, which are also learnable/trainable. Again we start with random normal distribution and learn these values during training.

Now, we apply max-pooling using tf.nn.max_pool function that has a very similar signature as that of conv2d function.

```

tf.nn.max_pool(value=layer,
                               ksize=[1, 2, 2, 1],
                               strides=[1, 2, 2, 1],
                               padding='SAME')
```

Notice that we are using k_size/filter_size as 2*2 and stride of 2 in both x and y direction. If you use the formula (w2= (w1-f)/S +1; h2=(h1-f)/S +1) mentioned earlier we can see that output is exactly half of input. These are most commonly used values for max pooling.

Finally, we use a RELU as our activation function which simply takes the output of max_pool and applies RELU using ***tf.nn.relu***

All these operations are done in a single convolution layer. Let's create a function to define a complete convolutional layer.

```

def new_conv_layer(input, num_input_channels, filter_size, num_filters, use_pooling=True):

    shape = [filter_size, filter_size,  num_input_channels, num_filters]

    weights = new_weights(shape=shape)

    biases = new_biases(length=num_filters)

    layer = tf.nn.conv2d(input=input,
                     filter=weights,
                     strides=[1, 1, 1, 1],
                     padding='SAME')

    layer += biases

    if use_pooling:
        layer = tf.nn.max_pool(value=layer,
                               ksize=[1, 2, 2, 1],
                               strides=[1, 2, 2, 1],
                               padding='SAME')
    layer = tf.nn.relu(layer)
    return layer, weights
```

ii) Flattening layer:

The Output of a convolutional layer is a multi-dimensional Tensor. We want to convert this into a one-dimensional tensor. This is done in the Flattening layer. We simply use the reshape operation to create a single dimensional tensor as defined below:

```

def flatten_layer(layer):
    layer_shape = layer.get_shape()
    num_features = layer_shape[1:4].num_elements()

    layer_flat = tf.reshape(layer, [-1, num_features])
    return layer_flat, num_features
```

iii) Fully connected layer:

Now, let's define a function to create a fully connected layer. Just like any other layer, we declare weights and biases as random normal distributions. In fully connected layer, we take all the inputs, do the standard z=wx+b operation on it. Also sometimes you would want to add a non-linearity(**RELU**) to it. So, let's add a condition that allows the caller to add RELU to the layer.

```
             num_inputs,
             num_outputs,
             use_relu=True):

    weights = new_weights(shape=[num_inputs, num_outputs])
    biases = new_biases(length=num_outputs)

    layer = tf.matmul(input, weights) + biases
    if use_relu:
        layer = tf.nn.relu(layer)

    return layer
```

So, we have finished defining the building blocks of the network.

iv) Placeholders and input:

Now, let's create a placeholder that will hold the input training images. All the input images are read in dataset.py file and resized to 128*128*3 size. While reading images, instead of reading them in a multi-dimensional Tensor, we read them into a single dimensional Tensor of size 49152(128*128*3) for simplicity. So the input placeholder x is created in the shape of [None, 49152]. The first dimension being None means you can pass any number of images to it. For this program, we shall pass images in the batch of 16 i.e. shape will be [16 49152]. After this, we reshape this into [16 128 128 3]. Similarly, we create a placeholder y_true for storing the predictions. For each image, we have two outputs i.e. probabilities for each class. Hence y_pred is of the shape [None 2] (for batch size 16 it will be [16 2].

```

img_size_flat= img_size * img_size * num_channels
x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='x')
x_image = tf.reshape(x, [-1, img_size, img_size, num_channels])

y_true = tf.placeholder(tf.float32, shape=[None, num_classes], name='y_true')
y_true_cls = tf.argmax(y_true, dimension=1)
```

v) network design:

We use the functions defined above to create various layers of the network.

```

layer_conv1, weights_conv1 = \
new_conv_layer(input=x_image,
               num_input_channels=num_channels,
               filter_size=filter_size1,
               num_filters=num_filters1,
               use_pooling=True)

layer_conv2, weights_conv2 = \
new_conv_layer(input=layer_conv1,
               num_input_channels=num_filters1,
               filter_size=filter_size2,
               num_filters=num_filters2,
               use_pooling=True)

layer_conv3, weights_conv3 = \
new_conv_layer(input=layer_conv2,
               num_input_channels=num_filters2,
               filter_size=filter_size3,
               num_filters=num_filters3,
               use_pooling=True)

layer_flat, num_features = flatten_layer(layer_conv3)
layer_fc1 = new_fc_layer(input=layer_flat,
                     num_inputs=num_features,
                     num_outputs=fc_size,
                     use_relu=True)

layer_fc2 = new_fc_layer(input=layer_fc1,
                     num_inputs=fc_size,
                     num_outputs=num_classes,
                     use_relu=False)
```

vi) Predictions:

As mentioned above, you can get the probability of each class by applying softmax to the output of fully connected layer.

```
y_pred = tf.nn.softmax(layer_fc2)
```

y_pred contains the predicted probability of each class for each input image. The class having higher probability is the prediction of the network. `y_pred_cls = tf.argmax(y_pred, dimension=1)`

Now, let's define the cost that will be minimized to reach the optimum value of weights. We will use a simple cost that will be calculated using a Tensorflow function softmax_cross_entropy_with_logits which takes the output of last fully connected layer and actual labels to calculate cross_entropy whose average will give us the cost.

```

cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=layer_fc2,
                                                    labels=y_true)
cost = tf.reduce_mean(cross_entropy)
```

VII) Optimization:

trying to minimise cost with a learning rate of 0.0001.

```
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)
```

As you know, if we run optimizer operation inside session.run(), in order to calculate the value of cost, the whole network will have to be run and we will pass the training images in a feed_dict(Does that make sense? Think about, what variable would you need to calculate cost and keep going up in the code). Training images are passed in a batch of 16(train_batch_size) in each iteration.

```

batch_size = 16
img_size_flat = img_size * img_size * num_channels
x_batch, y_true_batch, _, cls_batch = data.train.next_batch(train_batch_size)
x_batch = x_batch.reshape(train_batch_size, img_size_flat)

feed_dict_train = {x: x_batch,
                           y_true: y_true_batch}

session.run(optimizer, feed_dict=feed_dict_train)
```

where next_batch is a simple python function in dataset.py file that returns the next 16 images to be passed for training. Similarly, we pass the validation batch of images independently to in another session.run() call.

```

x_valid_batch, y_valid_batch, _, valid_cls_batch = data.valid.next_batch(train_batch_size)

x_valid_batch = x_valid_batch.reshape(train_batch_size, img_size_flat)

feed_dict_validate = {x: x_valid_batch,
                      y_true: y_valid_batch}

val_loss = session.run(cost, feed_dict=feed_dict_validate)
```

Note that in this case, we are passing cost in the session.run() with a **batch of validation images** as opposed to training images. In order to calculate the cost, the whole network(3 convolution+1 flattening+2 fc layers) will have to be executed to produce layer_fc2(which is required to calculate cross_entropy, hence cost). However, as opposed to training, this time optimization `optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)`

will not be run(as we only have to calculate cost). This is what changes the gradients and weights and is very computationally expensive. We can calculate the accuracy on validataion set using true labels(y_true) and predicted labels(y_pred).

```

correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

We can calculate the validation accuracy by passing accuracy in session.run() and providing validation images in a feed_dict.

```
val_acc = session.run(accuracy,feed_dict=feed_dict_validate)
```

Similarly, we also report the accuracy for the training images.

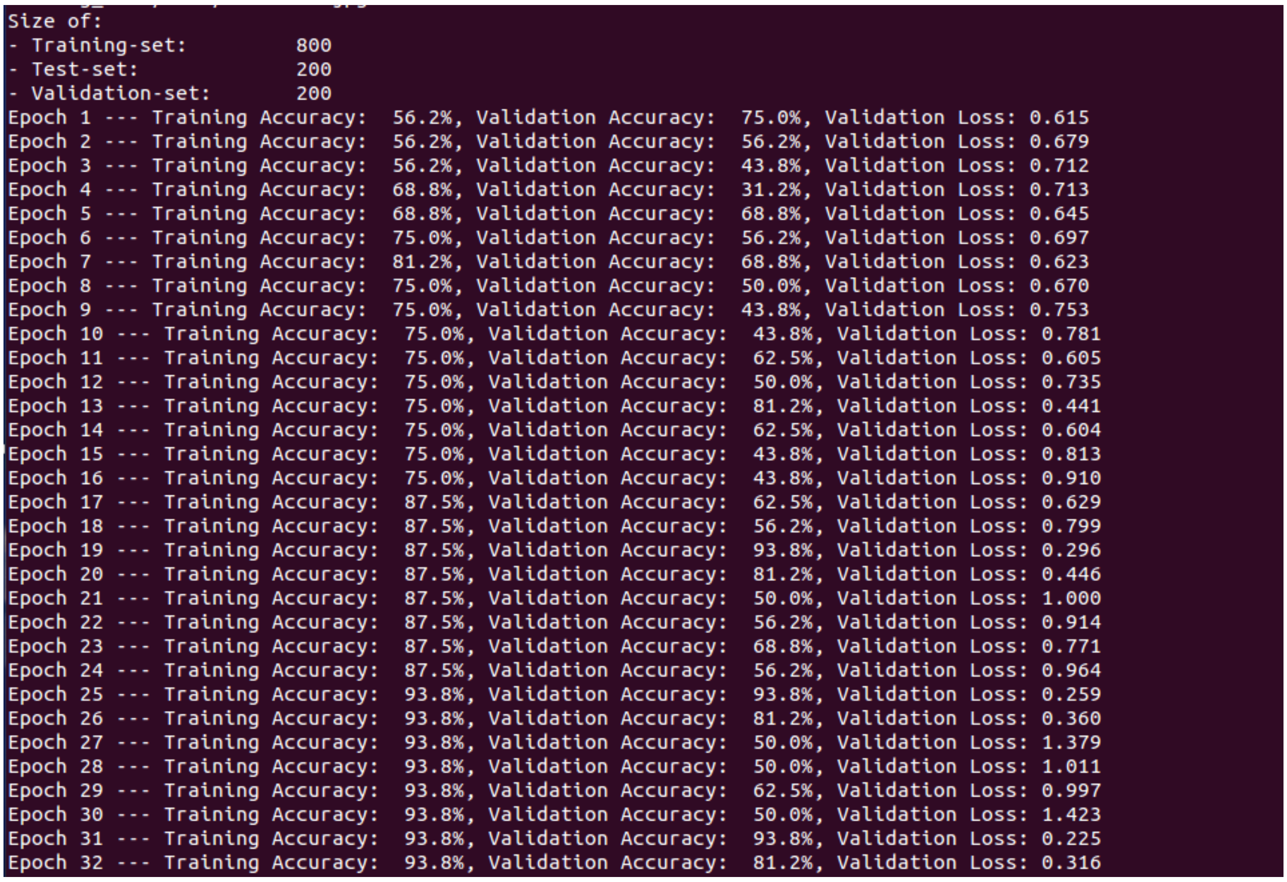
```
acc = session.run(accuracy, feed_dict=feed_dict_train)
```

As, training images along with labels are used for training, so training accuracy will be higher than validation. We report training accuracy to know that we are at least moving in the right direction and are at least improving accuracy in the training dataset.

So, this is how the complete optimize function looks like:

```

def optimize(num_iterations):
    global total_iterations

    best_val_loss = float("inf")
    patience = 0

    for i in range(total_iterations,
                   total_iterations + num_iterations):

        x_batch, y_true_batch, _, cls_batch = data.train.next_batch(train_batch_size)

        x_valid_batch, y_valid_batch, _, valid_cls_batch = data.valid.next_batch(train_batch_size)

        x_batch = x_batch.reshape(train_batch_size, img_size_flat)
        x_valid_batch = x_valid_batch.reshape(train_batch_size, img_size_flat)
        feed_dict_train = {x: x_batch,
                           y_true: y_true_batch}

        feed_dict_validate = {x: x_valid_batch,
                              y_true: y_valid_batch}

        session.run(optimizer, feed_dict=feed_dict_train)


        if i % int(data.train.num_examples/batch_size) == 0:
            val_loss = session.run(cost, feed_dict=feed_dict_validate)
            epoch = int(i / int(data.train.num_examples/batch_size))

            print_progress(epoch, feed_dict_train, feed_dict_validate, val_loss)

    total_iterations += num_iterations
```

This code is slightly long as it's a real world example. So, please go here (https://github.com/sankit1/cv-tricks.com), clone the code and run the train.py file to start the training. This is how the output will look like:



```
Size of:
- Training-set:         800
- Test-set:             200
- Validation-set:       200
Epoch 1 --- Training Accuracy:  56.2%, Validation Accuracy:  75.0%, Validation Loss: 0.615
Epoch 2 --- Training Accuracy:  56.2%, Validation Accuracy:  56.2%, Validation Loss: 0.679
Epoch 3 --- Training Accuracy:  56.2%, Validation Accuracy:  43.8%, Validation Loss: 0.712
Epoch 4 --- Training Accuracy:  68.8%, Validation Accuracy:  31.2%, Validation Loss: 0.713
Epoch 5 --- Training Accuracy:  68.8%, Validation Accuracy:  68.8%, Validation Loss: 0.645
Epoch 6 --- Training Accuracy:  75.0%, Validation Accuracy:  56.2%, Validation Loss: 0.697
Epoch 7 --- Training Accuracy:  81.2%, Validation Accuracy:  68.8%, Validation Loss: 0.623
Epoch 8 --- Training Accuracy:  75.0%, Validation Accuracy:  50.0%, Validation Loss: 0.670
Epoch 9 --- Training Accuracy:  75.0%, Validation Accuracy:  43.8%, Validation Loss: 0.753
Epoch 10 --- Training Accuracy:  75.0%, Validation Accuracy:  43.8%, Validation Loss: 0.781
Epoch 11 --- Training Accuracy:  75.0%, Validation Accuracy:  62.5%, Validation Loss: 0.605
Epoch 12 --- Training Accuracy:  75.0%, Validation Accuracy:  50.0%, Validation Loss: 0.735
Epoch 13 --- Training Accuracy:  75.0%, Validation Accuracy:  81.2%, Validation Loss: 0.441
Epoch 14 --- Training Accuracy:  75.0%, Validation Accuracy:  62.5%, Validation Loss: 0.604
Epoch 15 --- Training Accuracy:  75.0%, Validation Accuracy:  43.8%, Validation Loss: 0.813
Epoch 16 --- Training Accuracy:  75.0%, Validation Accuracy:  43.8%, Validation Loss: 0.910
Epoch 17 --- Training Accuracy:  87.5%, Validation Accuracy:  62.5%, Validation Loss: 0.629
Epoch 18 --- Training Accuracy:  87.5%, Validation Accuracy:  56.2%, Validation Loss: 0.799
Epoch 19 --- Training Accuracy:  87.5%, Validation Accuracy:  93.8%, Validation Loss: 0.296
Epoch 20 --- Training Accuracy:  87.5%, Validation Accuracy:  81.2%, Validation Loss: 0.446
Epoch 21 --- Training Accuracy:  87.5%, Validation Accuracy:  50.0%, Validation Loss: 1.000
Epoch 22 --- Training Accuracy:  87.5%, Validation Accuracy:  56.2%, Validation Loss: 0.914
Epoch 23 --- Training Accuracy:  87.5%, Validation Accuracy:  68.8%, Validation Loss: 0.771
Epoch 24 --- Training Accuracy:  87.5%, Validation Accuracy:  56.2%, Validation Loss: 0.964
Epoch 25 --- Training Accuracy:  93.8%, Validation Accuracy:  93.8%, Validation Loss: 0.259
Epoch 26 --- Training Accuracy:  93.8%, Validation Accuracy:  81.2%, Validation Loss: 0.360
Epoch 27 --- Training Accuracy:  93.8%, Validation Accuracy:  50.0%, Validation Loss: 1.379
Epoch 28 --- Training Accuracy:  93.8%, Validation Accuracy:  50.0%, Validation Loss: 1.011
Epoch 29 --- Training Accuracy:  93.8%, Validation Accuracy:  62.5%, Validation Loss: 0.997
Epoch 30 --- Training Accuracy:  93.8%, Validation Accuracy:  50.0%, Validation Loss: 1.423
Epoch 31 --- Training Accuracy:  93.8%, Validation Accuracy:  93.8%, Validation Loss: 0.225
Epoch 32 --- Training Accuracy:  93.8%, Validation Accuracy:  81.2%, Validation Loss: 0.316
```

This is a small network and is not ideal to build an image classifier. We see a lot of fluctuation in loss and accuracy. One thing that you can see that as loss goes down, accuracy goes up. We can save all the weights in a binary file and use that file to run a classifier for new images. This binary file is called trained model. In future, we shall learn how to save these models and use them in production system. For now, congratulations! you have learnt how to build and train an image classifier based on convolutional neural networks.

The complete code is available here (https://github.com/sankit1/cv-tricks.com). Please let me know your questions and feedback in the comments below. These comments and feedback are my motivation to create more tutorials ☺ .

#How-to (http://cv-tricks.com/tag/how-to/), #Image recognition (http://cv-tricks.com/tag/image-recognition/), #Tensorflow tutorial (http://cv-tricks.com/tag/tensorflow-tutorial/)

SHARE ON TWITTER (HTTPS://TWITTER.COM/SHARE?URL=HTTP://CV-TRICKS.COM/TENSORFLOW-TUTORIAL/TRAINING-CONVOLUTIONAL-NEURAL-NETW

SHARE ON PINTEREST (HTTP://PINTEREST.COM/PIN/CREATE/BUTTON/?URL=HTTP://CV-TRICKS.COM/TENSORFLOW-TUTORIAL/TRAINING-CONVOLUTIONAL

236 Shares

129

58

Featured Comment

Ankit Mod Kaushal Sharma • 3 months ago

Hi Kaushal/Yap Seng Kuang, this is what you do.

While saving the model:
a) Give y_pred an appropriate name(Otherwise, Tensorflow will give at a name):
y_pred = tf.nn.softmax(layer_fc2,name='y_pred')
b) Save using a saver, after training is done:
saver = tf.train.Saver()
saver.save(session, 'my_test_model')

Restore the model:

import tensorflow as tf
import dataset
import numpy as np
sess = tf.Session()
saver = tf.train.import_meta_graph('my_test_model.meta')
saver.restore(sess, tf.train.latest_checkpoint('./'))
graph = tf.get_default_graph()

see more

1 • Share ›

56 Comments **cv-tricks.com** Login

Recommend 6 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Niroj Pariyar • 5 days ago

Is there any way to input and image and classify it as dog or cat showing probability of each

• Reply • Share ›

yiyijanice • 7 days ago

Hey, i got the error when i tried to run the train.py. Anyone could me to resolve this issue?

File "train.py", line 294
saver = tf.train.Saver()
^
TabError: inconsistent use of tabs and spaces in indentation

• Reply • Share ›

Nishit • 15 days ago

File "<ipython-input-10-1f355614c454>", line 1, in <module>
runfile('C:/Users/Nishit/tutorial-2-image-classifier/train.py', wdir='C:/Users/Nishit/tutorial-2-image-classifier')

File "C:\Users\Nishit\Anaconda3\lib\site-packages\spyder\utils\site\sitecustomize.py", line 880, in runfile
execfile(filename, namespace)

File "C:\Users\Nishit\Anaconda3\lib\site-packages\spyder\utils\site\sitecustomize.py", line 102, in execfile
exec(compile(f.read(), filename, 'exec'), namespace)

File "C:/Users/Nishit/tutorial-2-image-classifier/train.py", line 309, in <module>

File "C:/Users/Nishit/tutorial-2-image-classifier/train.py", line 295, in optimize
saver.save(session, 'my_test_model')

File "C:\Users\Nishit\Anaconda3\lib\site-packages\tensorflow\python\training\saver.py", line 1382, in save
"Parent directory of {} doesn't exist, can't save.".format(save_path))

ValueError: Parent directory of my_test_model doesn't exist, can't save.

Please Help Me!!

• Reply • Share ›

Nikitha JV • 16 days ago

Hi can u tell how epochs and iterations are related? and whats the difference between them?

• Reply • Share ›

Sagun Shrestha • a month ago

What I want to do is give in an image of my own to the model and get the score back, for example I gave an image it returns the score as dog = 55.63%, cat = 44.37%, something like this. Can anyone have the code?

• Reply • Share ›

Jasson Hidalgo • a month ago

WOW, Now I got another error:

```
---------------------------------------------------------------------------
NameError Traceback (most recent call last)
<ipython-input-40-a7e03211cae3> in <module>()
22 batch_size = 16
23
---> 24 data = dataset.read_train_sets(train_path, img_size, classes, validation_size=validation_size)
25 test_images, test_ids = dataset.read_test_set(test_path, img_size)

NameError: name 'img_size' is not defined
```

WHAT VALUE DID YOU GUYS USE?

• Reply • Share ›

Jasson Hidalgo • a month ago

I downloaded the dataset.py and saved a copy in my working directory /Users/MyName. Now in my code I am importing it as import dataset, and I get the following error:

```
ImportError Traceback (most recent call last)
<ipython-input-36-b39e96b98dce> in <module>()
9
10 #let us train our data for the first time
---> 11 import dataset
12 classes = ['roses','sunflowers']
13 num_classes = len(classes)

/Users/jMyName/dataset.py in <module>()
7 import glob
8 import numpy as np
----> 9 import cv2
10 from sklearn.utils import shuffle
11

ImportError: No module named cv2
```

does cv2 has to be installed?

• Reply • Share ›

Jasson Hidalgo ➦ Jasson Hidalgo • a month ago

Never mind, I had to install it. These are the steps I took:

1. I opened a termina (0n my Mac laptop) and I typed: conda config --add channels menpo
2. Then I typed: conda install opencv
Note: you can also use conda install opencv3

3. To check if it was installed and to see the version, I used the following lines of code:

```
$ python
>>> import cv2
>>> cv2.__version__
```

This gave me: '2.4.11'

References:
https://stackoverflow.com/q...

• Reply • Share ›

Hi Ankit

This tutorial is really very helpful . I have just started to explore CNN. Can I use this code for a data set which has more than 2 classes.

Reply • Share ›

David • a month ago

Tensorflow is powerful but like to use user interface like https://vize.ai

Reply • Share ›

Erfan Ebrahimi • 2 months ago

Dear Ankit,

thanks for the great tutorial. I have started working with puthon for almost 2 days and still am struggling to figure out omage classification with tensorflow as well as the language itself. I am using windows 10 pro, visual studio 2017, python 3.6.2rcl. Can you please share your code for dataset class that you have used in reading in inputs? Also I really appreciate if you kindly mention what packages we need to import.

Once again thank you so much.

Reply • Share ›

Lanka • 2 months ago

Dear Ankit,
i have successfuly restored train mode as you explain here.i want to know how i predict just input image and its results.
input_image=cv2.imread('test.jpg') thats it.
Thanks.

Reply • Share ›

Ezer Miller • 3 months ago

Dear Ankit,
Thank you for this tutorial.
I feel there is lack in TF tutorial and this is a good one.
Question:
(1) Why do you flatten the images before set them in a placeholder? Can I read into the placeholder images as they are as multidimentional np.array?
(2) why do you use special functions for tf.varaibles and tf.constant instead of using them witihn the function of the layers?
I tried to do it and I got starnge error....
Thanks !

Reply • Share ›

Filip Novoselnik • 3 months ago

Hello,
When I run your code I get following error and I can't figure it out. Thanks!

```
Traceback (most recent call last):
File "/home/fnovoselnik/cv-tricks.com-master/Tensorflow-tutorials/tutorial-2-image-classifier/train.py", line 62, in <module>
data = dataset.read_train_sets(train_path, img_size, classes, validation_size=validation_size)
File "/home/fnovoselnik/cv-tricks.com-master/Tensorflow-tutorials/tutorial-2-image-classifier/dataset.py", line 131, in read_train_sets
images, labels, ids, cls = load_train(train_path, image_size, classes)
File "/home/fnovoselnik/cv-tricks.com-master/Tensorflow-tutorials/tutorial-2-image-classifier/dataset.py", line 22, in load_train
image = cv2.resize(image, (image_size, image_size), cv2.INTER_LINEAR)
TypeError: dst is not a numpy array, neither a scalar
```

Reply • Share ›

Ankit Mod ➦ Filip Novoselnik • 3 months ago

Please try this: Run the code from /home/fnovoselnik/cv-tricks.com-master/Tensorflow-tutorials/tutorial-2-image-classifier/ folder. So:

```
cd /home/fnovoselnik/cv-tricks.com-master/Tensorflow-tutorials/tutorial-2-image-classifier/
python train.py
```

Reply • Share ›

Filip Novoselnik ➦ Ankit • 3 months ago

Same error again.

All libraries are installed, i checked. I really can't figure it out.

Reply • Share ›

Lanka ➦ Filip Novoselnik • 2 months ago

change your code as below,

image = cv2.resize(image, (image_size, image_size),0,0, cv2.INTER_LINEAR) in every places.

then working perfectly.

Reply • Share ›

• Reply • Share ›

yap seng kuang • 3 months ago

hi. i follow your step and successful run the training. and i follow your another tutorial and save the model build. now my question is how to restore the model to continue train on another set of data. and furthermore, how to restore this model for predictions? thanks!

• Reply • Share ›

KKL1 ➦ yap seng kuang • 2 months ago

hi, did you find a solution to resume the training?

• Reply • Share ›

Kaushal Sharma ➦ yap seng kuang • 3 months ago

Hi Ankit, even I have the same issue. I have been able to save the model but I am not able to restore the model and test it on the testing set. Could you please tell the steps/script to restore the model. After saving, I have got these three files: my_test_model, my_test_model.meta and checkpoint.

• Reply • Share ›

Ankit Mod ➦ Kaushal Sharma • 3 months ago

Featured by cv-tricks.com

Hi Kaushal/Yap Seng Kuang, this is what you do.

While saving the model:
a) Give y_pred an appropriate name(Otherwise, Tensorflow will give at a name):
y_pred = tf.nn.softmax(layer_fc2,name='y_pred')
b) Save using a saver, after training is done:
saver = tf.train.Saver()
saver.save(session, 'my_test_model')

Restore the model:

```
import tensorflow as tf
import dataset
import numpy as np
sess = tf.Session()
saver = tf.train.import_meta_graph('my_test_model.meta')
saver.restore(sess, tf.train.latest_checkpoint('./'))
graph = tf.get_default_graph()
```

see more

1 • Reply • Share ›

Kaushal Sharma ➦ Ankit • 2 months ago

Thanks Ankit for the solution. I downloaded the updated train.py file where you have made all the changes mentioned in your answer (giving a proper name and saving the model). I have three issues:

1. I understand that I should only save the last model after the complete training not the ones after each epoch and therefore saver should be the very last step and not within the for loop. Is it correct?

2. How can I retrieve the performance indicators also (cross entropy, accuracy etc.) for the test images?

3. If I use the saver within the for loop, code is taking a really long time on CPU (days). Is it normal?

Thanks once again.

• Reply • Share ›

Ankit Mod ➦ Kaushal Sharma • 3 months ago

Hi, I have written very clear and step by instructions for restoring the model here http://cv-tricks.com/tensor... . Could you please let me know the specific problem you are having?

• Reply • Share ›

Gaùcho Crozos ➦ Ankit • 2 months ago

hi **@Ankit** , the function read_test_set(test_path

, img_size,classes) doesn't work with me, test images is always 0
i hope you give the solution
thank you

^ | v • Reply • Share ›

Kaushal Sharma ➦ Ankit • 3 months ago

General instructions to save and restore the mode are working fine (I followed your answer at http://stackoverflow.com/qu.... But how it should be implemented for this specific tutorial of cat-dog classification, where we have a testing set of 400 cat dog images and we want to get the predicted values for these images?

I trained the network using your code 'train.py' and now I am trying to test it on the images in 'testing_set' folder as follows:

```
import tensorflow as tf
import dataset
import numpy as np

img_size = 128
classes = ['dogs', 'cats']
num_classes = len(classes)
test_path='testing_data'
num_channels = 3
img_size_flat = img_size * img_size * num_channels

sess=tf.Session()
saver = tf.train.import_meta_graph('my_test_model.meta')
```

see more

^ | v • Reply • Share ›

Gaùcho Crozos ➦ Kaushal Sharma • 3 months ago

Hey Kaushal Sharma did you test the training example ?
please share the the full code
thank you

^ | v • Reply • Share ›

Kaushal Sharma ➦ Gaùcho Crozos • 3 months ago

Try this one: https://github.com/rdcolema...

It is working

^ | v • Reply • Share ›

Kaushal Sharma ➦ Gaùcho Crozos • 3 months ago

I am still struggling. I understand that once I restore the model, I just need to supply the test images and it should provide the predicted values. I am using these steps:

```
import tensorflow as tf
import dataset
import numpy as np

img_size = 128
classes = ['dogs', 'cats']
num_classes = len(classes)
test_path='testing_data'
num_channels = 3
img_size_flat = img_size * img_size * num_channels

#Restoring the model
sess=tf.Session()
saver = tf.train.import_meta_graph('my_test_model.meta')
saver.restore(sess,tf.train.latest_checkpoint('./'))
sess.run(tf.initialize_all_variables())
```

see more

^ | v • Reply • Share ›

You forgot to access y_pred after restoring. You need to access y_pred something like this after graph.tf.get_default_graph().

y_pred= graph.get_tensor_by_name("y_pred:0")

Reply • Share ›

yap seng kuang Ankit • 3 months ago

Hi Ankit,
I try y_pred=graph.get_tensor_by_name("y_pred:0") but i get an error as below:
KeyError: "The naume 'y_pred:0' refers to a Tensor which does not exist. The operation, 'y_pred', does not exist in the graph.

I use [n.name for n in tf.get_default_graph().as_graph_def().node] and i don't see 'y_pred' in it. Do we have to specify names for every tensor like below?

y_pred = tf.nn.softmax(layer_fc2, name='y_pred')

Thank you!

1 • Reply • Share ›

Ankit Mod yap seng kuang • 3 months ago

While saving the network, you need to give a good name to the ops which you want to restore. Here is the complete code:

```
import tensorflow as tf
import dataset
import numpy as np
sess = tf.Session()
saver = tf.train.import_meta_graph('my_test_model.meta')
saver.restore(sess, tf.train.latest_checkpoint('./'))
graph = tf.get_default_graph()

y_pred = graph.get_tensor_by_name("y_pred:0")

test_path='testing_data'
img_size = 128
classes = ['dogs', 'cats']
num_classes = len(classes)
test_path='testing_data'
num_channels = 3
img_size_flat = img_size * img_size * num_channels
test_images, test_ids = dataset.read_test_set(test_path, img_size,classes)
x= graph.get_tensor_by_name("x:0") ### You missed this.
x_batch = test_images.reshape(200, img_size_flat)
y_true = graph.get_tensor_by_name("y_true:0") ### You missed this.
y_test_images = np.zeros((200, 2))
feed_dict_testing = {x: x_batch, y_true: y_test_images}
print(sess.run(y_pred, feed_dict=feed_dict_testing))
```

Reply • Share ›

Mayur Rumalwala Ankit • a month ago

Hi Ankit,

Thanks for your code. It runs perfectly but when I run the program it takes 2 mins to import and restore the model. Can you please help me for this problem? I really appreciate that.

Thanks

Reply • Share ›

KKL1 Ankit • 2 months ago

i am using this code and i am still getting the same error of :KeyError: "The naume 'y_pred:0' refers to a Tensor which does not exist. The operation, 'y_pred', does not exist in the graph

Reply • Share ›

Gaùcho Crozos Ankit • 2 months ago

when i checked the file my_test_model.meta : i found the x , y_true, but y_pred doesn't exist, i think here is the issue

Reply • Share ›

Kaushal Sharma Ankit • 3 months ago

Hi Ankit, I tried the suggested solution also but it is not working. I am just running the train.py from your github repository, with few lines to save the model. Then I want to test how this network is performing on the test_images of cats and dogs from the directory testing_data. Code to train the network is there in your repository but code to test the model is not there. Could you please suggest how to do this or share the code to test the network?
Can we actually use this trained network (that we have got after running train.py) for testing?

Reply • Share ›

i am checking about the same issue, i didn't find a solution
any Code for testing data ?
please share it Kaushal Sharma
Reply • Share ›

Ankit Mod Kaushal Sharma • 3 months ago
Please check my new comment, I have provided the exact code. You could use this network for prediction but this is a very small network. For more accurate and better network architectures check http://cv-tricks.com/tensor...
Reply • Share ›

이재정 • 3 months ago
Hi I am following the steps after you but i just faced some problems
I just copied the code you uploaded on github and tried to run it with pycharm.
And i found that tensorflow and cv2(opencv) is incompatible.
through the train.py with opencv interpreter, it seemes that it tries to refer the dataset.py whose interpreter should be tensorflow.
but result was "no module named cv2".
is there any wrong point from what I wrote above? If not, could you please tell me how to fix it
Reply • Share ›

Ankit Mod 이재정 • 3 months ago
It appears that openCV is not installed. Run import cv2, cv2.__version__ commands to see if opencv is properly installed.
Reply • Share ›

이재정 Ankit • 3 months ago
Thanx a lot :) I just solved it
Reply • Share ›

Abeeha Muskan • 4 months ago
Hey , I have 1600 images with labels , I want to try it on my dataset , I am confused where to put labels?
Reply • Share ›

sword Abeeha Muskan • 3 months ago
You need to create folders with the label name. And only images of those categories will be in the folder. You can write a small piece of code to automate that.
Reply • Share ›

KKL1 • 5 months ago
hello, thank you for this tutorial, i was wondering how to test the trained model on our testing data and display the obtained results.
Reply • Share ›

Ankit Mod KKL1 • 5 months ago
The complete code with data-set is available here https://github.com/sankit1/... . You can replace the content of testing_data folder with your images and the code will display the results on your data-set.
Reply • Share ›

KKL1 Ankit • 5 months ago
this code does only work on the training data, but it does not display the prediction value on my testing data. I have trained this network successfully and saved the model,
Now i'm trying to display the prediction probalities on my testing data
Reply • Share ›

Ankit Mod KKL1 • 4 months ago
After training, you should create a feed_dict for your testing image/data-set (like I created feed_dict_validate) for feeding data to the network and then feed y_pred to the session.run. This will calculate the probability for each class.

If you want to run it after the training from the saved weights, you need to:
1. Load pre-trained model. Read this in case of confusion(http://cv-tricks.com/tensor...
2. Create the complete graph and pass the appropriate operation to session.run() to calculate the output(like y_pred in the above tutorial)
2 • Reply • Share ›

KKL1 Ankit • 4 months ago
Thank you!!
Reply • Share ›

Gaùcho Crozos KKL1 • 3 months ago
can you share the file of the testing data plz
1 • Reply • Share ›

Kaushal Sharma KKL1 • 3 months ago

• Reply • Share ›

Load more comments

ALSO ON **CV-TRICKS.COM**

Accelerating Convolutional Neural Networks on Raspberry Pi

2 comments • 4 months ago•

Avat **Koustubh Sinhal** — Meant similar devices like ODROID-C2. Have a look at this look about these types of devices.http://www.learnopencv.com/...

Image Segmentation using deconvolution layer in Tensorflow

5 comments • 3 months ago•

Avat **C Raymond** — I was wondering if you have a full example of an implementation of deconvolution for image segmentation?

TensorFlow Tutorial: 10 minutes Practical TensorFlow lesson for quick learners

5 comments • 6 months ago•

Avat **Hooman** — How can I see the "cost" and "w" values during the learning process ?

Quick complete Tensorflow tutorial to understand and run Alexnet, VGG, Inceptionv3, Resnet and squeezeNet …

5 comments • 5 months ago•

Avat **Lennon Lin** — Hi, I want to fine tune the model, how should I do?THX