Image Classification with CIFAR-10 dataset

Some of the code and description of this notebook is borrowed by this repo (https://github.com/udacity/deep-learning/tree/master/image-classification)

Get the Data

Run the following cell to download the CIFAR-10 dataset for python (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz).

In [1]:

```
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 DownloadProgress(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

"""
    check if the data (zip) file is already downloaded
    if not, download it from "https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz" and save
 as cifar-10-python.tar.gz
"""
if not isfile('cifar-10-python.tar.gz'):
    with DownloadProgress(unit='B', unit_scale=True, miniters=1, desc='CIFAR-10 Dataset') as pba
r:
        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()
```

```
CIFAR-10 Dataset: 171MB [00:37, 4.60MB/s]
```

In [1]:

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
```

In [0]:

```
def load_label_names():
    return ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'tr
uck']
```

How to reshape into a such form?

The row vector (3072) has the exact same number of elements if you calculate 32*32*3==3072. In order to reshape the row vector, (3072)

1. divide the row vector (3072) into 3 pieces. Each piece corresponds to the each channels.
    - this results in (3 x 1024) dimension of tensor
2. divide the resulting tensor from the previous step with 32. 32 here means width of an image.
    - this results in (3 x 32 x 32)

In [0]:

```
def load_cfar10_batch(cifar10_dataset_folder_path, batch_id):
    with open(cifar10_dataset_folder_path + '/data_batch_' + str(batch_id), mode='rb') as file:
        # note the encoding type is 'latin1'
        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
```

Explore the Data

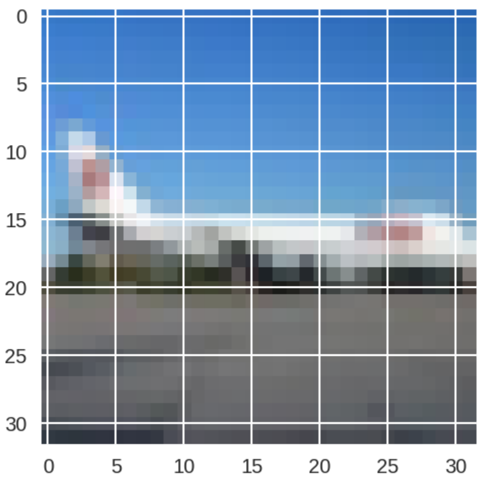
In [0]:

```
def display_stats(cifar10_dataset_folder_path, batch_id, sample_id):
    features, labels = load_cfar10_batch(cifar10_dataset_folder_path, batch_id)

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

    print('\nStats of batch #{}:'.format(batch_id))
    print('# of Samples: {}\n'.format(len(features)))

    label_names = load_label_names()
    label_counts = dict(zip(*np.unique(labels, return_counts=True)))
    for key, value in label_counts.items():
        print('Label Counts of [{}]({}) : {}'.format(key, label_names[key].upper(), value))

    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.imshow(sample_image)
```

In [26]:

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

import numpy as np

# Explore the dataset
batch_id = 3
sample_id = 7000
display_stats(cifar10_dataset_folder_path, batch_id, sample_id)
```

```
Stats of batch #3:
# of Samples: 10000

Label Counts of [0](AIRPLANE) : 994
Label Counts of [1](AUTOMOBILE) : 1042
Label Counts of [2](BIRD) : 965
Label Counts of [3](CAT) : 997
Label Counts of [4](DEER) : 990
Label Counts of [5](DOG) : 1029
Label Counts of [6](FROG) : 978
Label Counts of [7](HORSE) : 1015
Label Counts of [8](SHIP) : 961
Label Counts of [9](TRUCK) : 1029

Example of Image 7000:
Image - Min Value: 24 Max Value: 252
Image - Shape: (32, 32, 3)
Label - Label Id: 0 Name: airplane
```



Implement Preprocess Functions

In [0]:

```
def normalize(x):
    """
        argument
            - x: input image data in numpy array [32, 32, 3]
        return
            - normalized x
    """
    min_val = np.min(x)
    max_val = np.max(x)
    x = (x-min_val) / (max_val-min_val)
    return x
```

One-hot encode

In [0]:

```
def one_hot_encode(x):
    """
        argument
            - x: a list of labels
        return
            - one hot encoding matrix (number of labels, number of class)
    """
    encoded = np.zeros((len(x), 10))

    for idx, val in enumerate(x):
        encoded[idx][val] = 1

    return encoded
```

Preprocess all the data and save it

The code cell below uses the previously implemented functions, normalize and one_hot_encode, to preprocess the given dataset. Running the code cell below will preprocess all the CIFAR-10 data and save it to file.

In [0]:

```
def _preprocess_and_save(normalize, one_hot_encode, features, labels, filename):
    features = normalize(features)
    labels = one_hot_encode(labels)

    pickle.dump((features, labels), open(filename, 'wb'))


def preprocess_and_save_data(cifar10_dataset_folder_path, normalize, one_hot_encode):
    n_batches = 5
    valid_features = []
    valid_labels = []

    for batch_i in range(1, n_batches + 1):
        features, labels = load_cfar10_batch(cifar10_dataset_folder_path, batch_i)

        # find index to be the point as validation data in the whole dataset of the batch (10%)
        index_of_validation = int(len(features) * 0.1)

        # preprocess the 90% of the whole dataset of the batch
        # - normalize the features
        # - one_hot_encode the lables
        # - save in a new file named, "preprocess_batch_" + batch_number
        # - each file for each batch
        _preprocess_and_save(normalize, one_hot_encode,
                             features[:-index_of_validation], labels[:-index_of_validation],
                             'preprocess_batch_' + str(batch_i) + '.p')

        # unlike the training dataset, validation dataset will be added through all batch datase
t
        # - take 10% of the whold dataset of the batch
        # - add them into a list of
        #   - valid_features
        #   - valid_labels
        valid_features.extend(features[-index_of_validation:])
        valid_labels.extend(labels[-index_of_validation:])

    # preprocess the all stacked validation dataset
    _preprocess_and_save(normalize, one_hot_encode,
                         np.array(valid_features), np.array(valid_labels),
                         'preprocess_validation.p')

    # load the test dataset
    with open(cifar10_dataset_folder_path + '/test_batch', mode='rb') as file:
        batch = pickle.load(file, encoding='latin1')

    # preprocess the testing data
    test_features = batch['data'].reshape((len(batch['data']), 3, 32, 32)).transpose(0, 2, 3, 1)
    test_labels = batch['labels']

    # Preprocess and Save all testing data
    _preprocess_and_save(normalize, one_hot_encode,
                         np.array(test_features), np.array(test_labels),
                         'preprocess_training.p')
```

In [0]:

```
preprocess_and_save_data(cifar10_dataset_folder_path, normalize, one_hot_encode)
```

Checkpoint

In [0]:

```
import pickle

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

Prepare Input for the Model

In [0]:

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

# Inputs
x = tf.placeholder(tf.float32, shape=(None, 32, 32, 3), name='input_x')
y =  tf.placeholder(tf.float32, shape=(None, 10), name='output_y')
keep_prob = tf.placeholder(tf.float32, name='keep_prob')
```

Create Convolutional Model

The entire model consists of 9 layers in total. In addition to layers below lists what techniques are applied to build the model.

1. Convolution with 64 different filters in size of (3x3)
2. Max Pooling
    - ReLU activation function
    - Batch Normalization
3. Convolution with 128 different filters in size of (3x3)
    - Dropout
4. Max Pooling
    - ReLU activation function
    - Batch Normalization
5. Convolution with 256 different filters in size of (3x3)
6. Max Pooling
    - ReLU activation function
    - Batch Normalization
7. Convolution with 512 different filters in size of (3x3)
    - Dropout
8. Max Pooling
    - ReLU activation function
    - Batch Normalization
9. Flattening the 3-D output of the last convolutional operations.

In [0]:

```
import tensorflow as tf

def conv_net(x, keep_prob):
    conv1_filter = tf.Variable(tf.truncated_normal(shape=[3, 3, 3, 64], mean=0, stddev=0.08))
    conv2_filter = tf.Variable(tf.truncated_normal(shape=[3, 3, 64, 128], mean=0, stddev=0.08))
    conv3_filter = tf.Variable(tf.truncated_normal(shape=[5, 5, 128, 256], mean=0, stddev=0.08))
    conv4_filter = tf.Variable(tf.truncated_normal(shape=[5, 5, 256, 512], mean=0, stddev=0.08))

    # 1, 2
    conv1 = tf.nn.conv2d(x, conv1_filter, strides=[1,1,1,1], padding='SAME')
    conv1 = tf.nn.relu(conv1)
    conv1_pool = tf.nn.max_pool(conv1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
    conv1_bn = tf.layers.batch_normalization(conv1_pool)
    conv1_drop = tf.nn.dropout(conv1_bn, keep_prob)

    # 3, 4
    conv2 = tf.nn.conv2d(conv1_drop, conv2_filter, strides=[1,1,1,1], padding='SAME')
    conv2 = tf.nn.relu(conv2)
    conv2_pool = tf.nn.max_pool(conv2, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
    conv2_bn = tf.layers.batch_normalization(conv2_pool)
    conv2_drop = tf.nn.dropout(conv2_bn, keep_prob)

    # 5, 6
    conv3 = tf.nn.conv2d(conv2_bn, conv3_filter, strides=[1,1,1,1], padding='SAME')
    conv3 = tf.nn.relu(conv3)
    conv3_pool = tf.nn.max_pool(conv3, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
    conv3_bn = tf.layers.batch_normalization(conv3_pool)
    conv3_drop = tf.nn.dropout(conv3_bn, keep_prob)

    # 7, 8
    conv4 = tf.nn.conv2d(conv3_bn, conv4_filter, strides=[1,1,1,1], padding='SAME')
    conv4 = tf.nn.relu(conv4)
    conv4_pool = tf.nn.max_pool(conv4, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
    conv4_bn = tf.layers.batch_normalization(conv4_pool)
    conv4_drop = tf.nn.dropout(conv4_bn, keep_prob)

    # 9
    flat = tf.contrib.layers.flatten(conv4_drop)

    # 10
    out = tf.contrib.layers.fully_connected(inputs=flat, num_outputs=10, activation_fn=None)
    return out
```

Hyperparameters

- `epochs` : number of iterations until the network stops learning or start overfitting
- `batch_size` : highest number that your machine has memory for. Most people set them to common sizes of memory:
- `keep_probability` : probability of keeping a node using dropout
- `learning_rate` : number how fast the model learns

In [0]:

```
epochs = 10
batch_size = 128
keep_probability = 0.7
learning_rate = 0.001
```

In [35]:

```
logits = conv_net(x, keep_prob)
model = tf.identity(logits, name='logits') # Name logits Tensor, so that can be loaded from disk
 after training

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

```
WARNING:tensorflow:From <ipython-input-35-bc29ffa8bb57>:5: softmax_cross_entropy_w
ith_logits (from tensorflow.python.ops.nn_ops) is deprecated and will be removed i
n a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.
```

Train the Neural Network

In [0]:

```
def train_neural_network(session, optimizer, keep_probability, feature_batch, label_batch):
    session.run(optimizer,
                feed_dict={
                    x: feature_batch,
                    y: label_batch,
                    keep_prob: keep_probability
                })
```

Show Stats

In [0]:

```
def print_stats(session, feature_batch, label_batch, cost, accuracy):
    loss = sess.run(cost,
                    feed_dict={
                        x: feature_batch,
                        y: label_batch,
                        keep_prob: 1.
                    })
    valid_acc = sess.run(accuracy,
                         feed_dict={
                             x: valid_features,
                             y: valid_labels,
                             keep_prob: 1.
                         })

    print('Loss: {:>10.4f} Validation Accuracy: {:.6f}'.format(loss, valid_acc))
```

Fully Train the Model

In [0]:

```
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)
```

In [39]:

```
save_model_path = './image_classification'

print('Training...')
with tf.Session() as sess:
    # Initializing the variables
    sess.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):
                train_neural_network(sess, optimizer, keep_probability, batch_features, batch_labels)

            print('Epoch {:>2}, CIFAR-10 Batch {}:  '.format(epoch + 1, batch_i), end='')
            print_stats(sess, batch_features, batch_labels, cost, accuracy)

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

```
Training...
Epoch  1, CIFAR-10 Batch 1:  Loss:     2.2495 Validation Accuracy: 0.256200
Epoch  1, CIFAR-10 Batch 2:  Loss:     2.2704 Validation Accuracy: 0.182000
Epoch  1, CIFAR-10 Batch 3:  Loss:     1.9188 Validation Accuracy: 0.321800
Epoch  1, CIFAR-10 Batch 4:  Loss:     1.8917 Validation Accuracy: 0.382600
Epoch  1, CIFAR-10 Batch 5:  Loss:     1.4161 Validation Accuracy: 0.500200
Epoch  2, CIFAR-10 Batch 1:  Loss:     1.4422 Validation Accuracy: 0.560000
Epoch  2, CIFAR-10 Batch 2:  Loss:     1.4898 Validation Accuracy: 0.466400
Epoch  2, CIFAR-10 Batch 3:  Loss:     1.2394 Validation Accuracy: 0.527000
Epoch  2, CIFAR-10 Batch 4:  Loss:     1.1714 Validation Accuracy: 0.565000
Epoch  2, CIFAR-10 Batch 5:  Loss:     1.0176 Validation Accuracy: 0.585200
Epoch  3, CIFAR-10 Batch 1:  Loss:     0.9847 Validation Accuracy: 0.677800
Epoch  3, CIFAR-10 Batch 2:  Loss:     0.9526 Validation Accuracy: 0.594000
Epoch  3, CIFAR-10 Batch 3:  Loss:     0.7789 Validation Accuracy: 0.588400
Epoch  3, CIFAR-10 Batch 4:  Loss:     0.7376 Validation Accuracy: 0.627000
Epoch  3, CIFAR-10 Batch 5:  Loss:     0.6382 Validation Accuracy: 0.663000
Epoch  4, CIFAR-10 Batch 1:  Loss:     0.5730 Validation Accuracy: 0.706600
Epoch  4, CIFAR-10 Batch 2:  Loss:     0.6230 Validation Accuracy: 0.682600
Epoch  4, CIFAR-10 Batch 3:  Loss:     0.5219 Validation Accuracy: 0.646800
Epoch  4, CIFAR-10 Batch 4:  Loss:     0.3458 Validation Accuracy: 0.707600
Epoch  4, CIFAR-10 Batch 5:  Loss:     0.3809 Validation Accuracy: 0.682000
Epoch  5, CIFAR-10 Batch 1:  Loss:     0.2401 Validation Accuracy: 0.735000
Epoch  5, CIFAR-10 Batch 2:  Loss:     0.2884 Validation Accuracy: 0.744000
Epoch  5, CIFAR-10 Batch 3:  Loss:     0.2694 Validation Accuracy: 0.674400
Epoch  5, CIFAR-10 Batch 4:  Loss:     0.1538 Validation Accuracy: 0.728600
Epoch  5, CIFAR-10 Batch 5:  Loss:     0.2628 Validation Accuracy: 0.695400
Epoch  6, CIFAR-10 Batch 1:  Loss:     0.1464 Validation Accuracy: 0.726800
Epoch  6, CIFAR-10 Batch 2:  Loss:     0.1494 Validation Accuracy: 0.728400
Epoch  6, CIFAR-10 Batch 3:  Loss:     0.1376 Validation Accuracy: 0.685800
Epoch  6, CIFAR-10 Batch 4:  Loss:     0.0682 Validation Accuracy: 0.745600
Epoch  6, CIFAR-10 Batch 5:  Loss:     0.2747 Validation Accuracy: 0.705000
Epoch  7, CIFAR-10 Batch 1:  Loss:     0.0956 Validation Accuracy: 0.732600
Epoch  7, CIFAR-10 Batch 2:  Loss:     0.2054 Validation Accuracy: 0.718400
Epoch  7, CIFAR-10 Batch 3:  Loss:     0.0969 Validation Accuracy: 0.710200
Epoch  7, CIFAR-10 Batch 4:  Loss:     0.0837 Validation Accuracy: 0.721800
Epoch  7, CIFAR-10 Batch 5:  Loss:     0.0483 Validation Accuracy: 0.748200
Epoch  8, CIFAR-10 Batch 1:  Loss:     0.0640 Validation Accuracy: 0.724000
Epoch  8, CIFAR-10 Batch 2:  Loss:     0.1334 Validation Accuracy: 0.701600
Epoch  8, CIFAR-10 Batch 3:  Loss:     0.0178 Validation Accuracy: 0.731600
Epoch  8, CIFAR-10 Batch 4:  Loss:     0.0879 Validation Accuracy: 0.729800
Epoch  8, CIFAR-10 Batch 5:  Loss:     0.0383 Validation Accuracy: 0.743800
Epoch  9, CIFAR-10 Batch 1:  Loss:     0.0255 Validation Accuracy: 0.745800
Epoch  9, CIFAR-10 Batch 2:  Loss:     0.1279 Validation Accuracy: 0.697600
Epoch  9, CIFAR-10 Batch 3:  Loss:     0.0145 Validation Accuracy: 0.719200
Epoch  9, CIFAR-10 Batch 4:  Loss:     0.0295 Validation Accuracy: 0.711200
Epoch  9, CIFAR-10 Batch 5:  Loss:     0.0109 Validation Accuracy: 0.740200
Epoch 10, CIFAR-10 Batch 1:  Loss:     0.0104 Validation Accuracy: 0.748200
Epoch 10, CIFAR-10 Batch 2:  Loss:     0.0817 Validation Accuracy: 0.696200
Epoch 10, CIFAR-10 Batch 3:  Loss:     0.0507 Validation Accuracy: 0.737200
Epoch 10, CIFAR-10 Batch 4:  Loss:     0.0159 Validation Accuracy: 0.728000
Epoch 10, CIFAR-10 Batch 5:  Loss:     0.0050 Validation Accuracy: 0.741800
```

Test Model

In [0]:

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer

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 display_image_predictions(features, labels, predictions, top_n_predictions):
    n_classes = 10
    label_names = load_label_names()
    label_binarizer = LabelBinarizer()
    label_binarizer.fit(range(n_classes))
    label_ids = label_binarizer.inverse_transform(np.array(labels))

    fig, axies = plt.subplots(nrows=top_n_predictions, ncols=2, figsize=(20, 10))
    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, labe
l_ids, predictions.indices, predictions.values)):
        if (image_i < top_n_predictions):
            pred_names = [label_names[pred_i] for pred_i in pred_indicies]
            correct_name = label_names[label_id]

            axies[image_i][0].imshow((feature*255).astype(np.int32, copy=False))
            axies[image_i][0].set_title(correct_name)
            axies[image_i][0].set_axis_off()

            axies[image_i][1].barh(ind + margin, pred_values[:3], 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])
```

In [42]:

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

import tensorflow as tf
import pickle
import random

save_model_path = './image_classification'
batch_size = 64
n_samples = 10
top_n_predictions = 5

def test_model():
    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('input_x:0')
        loaded_y = loaded_graph.get_tensor_by_name('output_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_ke
ep_prob: 1.0})
            test_batch_count += 1

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



test_model()
```

```
INFO:tensorflow:Restoring parameters from ./image_classification
Testing Accuracy: 0.7360668789808917
```