Distributed Data Analytics

Lab 7

Submitted By- Abdullah Al Foysal

Id:277458

Exercise 1: Convolutional Neural Networks with Relu activation function

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



# This function returns the number of labels in the cifar10 dataset

def _load_label_names():

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

# This function takes the batches from the cifar10 folder and convert each batch size 3072 to the transformation of depth,length,
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:
        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

#This function is used to display the image statistics based on the batch_id and sample_id
#If the batch _id and sample_id is 4 and 1 respectively, it prints the first mage from training batch 4 along with its statistics
```

```python
def display_stats(cifar10_dataset_folder_path, batch_id, sample_id):

    batch_ids = list(range(1, 6))

    if batch_id not in batch_ids:
        print('Batch Id out of Range. Possible Batch Ids: {}'.format(batch_ids))
        return None

    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, 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]
    label_names = _load_label_names()

    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)

#This function is used to do preproccesing stuff like normalizing the features and one hot encoding the labels and the result is
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'))
```

```python
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)
        validation_count = int(len(features) * 0.1)

        # Prprocess and save a batch of training data
        _preprocess_and_save(
            normalize,
            one_hot_encode,
            features[:-validation_count],
            labels[:-validation_count],
            'preprocess_batch_' + str(batch_i) + '.p')

        # Use a portion of training batch for validation
        valid_features.extend(features[-validation_count:])
        valid_labels.extend(labels[-validation_count:])

    # Preprocess and Save all validation data
    _preprocess_and_save(
        normalize,
        one_hot_encode,
        np.array(valid_features),
        np.array(valid_labels),
        'preprocess_validation.p')

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

```python
    # load 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_test.p')


def batch_features_labels(features, labels, batch_size):

    for start in range(0, len(features), batch_size):
        end = min(start + batch_size, len(features))
        yield features[start:end], labels[start:end]

#load the preprocessed training batch
def load_preprocess_training_batch(batch_id, batch_size):

    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)
```

```python
#Display the prediction for the random test data based on the model saved after training it.
def display_image_predictions(features, labels, predictions):
    #it is the number of classes
    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=4, ncols=2)
    fig.tight_layout()
    fig.suptitle('Softmax Predictions', fontsize=20, y=1.1)
    #Top 3 probability predictions are shown for each random predicted test data.
    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, predi
        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])

#This is the folder path where i downloaded and extracted my cifar10 dataset
cifar10_dataset_folder_path = 'D:\\Uni Hildesheim\\4th semester\\DDA Lab\\lab 7\\cifar-10-python\\cifar-10-batches-py'
def normalize(x):

    maximum = np.max(x)
    minimum = np.min(x)
    return (x - minimum) / (maximum - minimum)
```

```
#Display the prediction for the random test data based on the model saved after training it.
def display_image_predictions(features, labels, predictions):
    #it is the number of classes
    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=4, ncols=2)
    fig.tight_layout()
    fig.suptitle('Softmax Predictions', fontsize=20, y=1.1)
    #Top 3 probability predictions are shown for each random predicted test data.
    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, predi
        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])

#This is the folder path where i downloaded and extracted my cifar10 dataset
cifar10_dataset_folder_path = 'D:\\Uni Hildesheim\\4th semester\\DDA Lab\\lab 7\\cifar-10-python\\cifar-10-batches-py'
def normalize(x):

    maximum = np.max(x)
    minimum = np.min(x)
    return (x - minimum) / (maximum - minimum)
```

```
#Defining the convolutional neural network with number of output, kernel size, strides, and pool side and padding
def conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, pool_strides):


    input_depth = x_tensor.get_shape().as_list()[-1]
    #weights
    W = tf.Variable(tf.random_normal(
        [conv_ksize[0], conv_ksize[1], input_depth, conv_num_outputs],
        stddev=0.1
    ))
    #bias
    b = tf.Variable(tf.zeros(conv_num_outputs))
    conv = tf.nn.conv2d(x_tensor, W, [1, conv_strides[0], conv_strides[1], 1], 'SAME') + b
    conv = tf.nn.relu(conv)
    return tf.nn.max_pool(
        conv,
        [1, pool_ksize[0], pool_ksize[1], 1],
        [1, pool_strides[0], pool_strides[1], 1],
        'SAME')

def flatten(x_tensor):

    shape = x_tensor.get_shape().as_list()
    return tf.reshape(x_tensor, [-1, np.prod(shape[1:])])

#Defined a Fully connected layer with relu activation function

def fully_conn(x_tensor, num_outputs):

    shape = x_tensor.get_shape().as_list()
    W = tf.Variable(tf.random_normal([shape[-1], num_outputs], stddev=0.1))
    b = tf.Variable(tf.zeros(num_outputs)) + 0.11
    return tf.nn.relu(tf.add(tf.matmul(x_tensor, W), b))

#defining the function for output of each network
def output(x_tensor, num_outputs):

    shape = x_tensor.get_shape().as_list()
    W = tf.Variable(tf.random_normal([shape[-1], num_outputs]))
    b = tf.Variable(tf.zeros(num_outputs))
    return tf.add(tf.matmul(x_tensor, W), b)
```

```python
def conv_net(x):

    #    conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, pool_strides)
    tmp = conv2d_maxpool(x, 64, [3, 3], [1, 1], [3, 3], [2, 2])

    tmp = flatten(tmp)
    #one fully connected layer fc1
    tmp = fully_conn(tmp, 384)

    #second fully connected layer fc2
    tmp = fully_conn(tmp, 192)



    # TODO: return output
    return output(tmp, 10)

tf.reset_default_graph()

# Inputs
x = neural_net_image_input((32, 32, 3))
y = neural_net_label_input(10)


# Model
logits = conv_net(x)

# 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().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')
tf.summary.histogram("loss",cost)
tf.summary.scalar("loss", cost)
# Create a summary to monitor accuracy tensor
tf.summary.histogram("accuracy", accuracy)
tf.summary.scalar("accuracy", accuracy)
# Merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()
```

```python
def train_neural_network(session, optimizer,cost,merged_summary_op, feature_batch, label_batch,epoch):

    train_opt,train_cost,summary = session.run([optimizer,cost,merged_summary_op], feed_dict={x: feature_batch, y: label_batch})
    summary_writer.add_summary(summary,epoch)

def print_stats(session, feature_batch, label_batch, cost, accuracy):

    global valid_features, valid_labels
    validation_accuracy = session.run(
        accuracy,
        feed_dict={
            x: valid_features,
            y: valid_labels,

        }
    )
    cost = session.run(
        cost,
        feed_dict={
            x: feature_batch,
            y: label_batch,

        }
    )
    print('Cost = {0} - Validation Accuracy = {1}'.format(cost, validation_accuracy))

    # Parameters
epochs = 5
batch_size = 1024
```

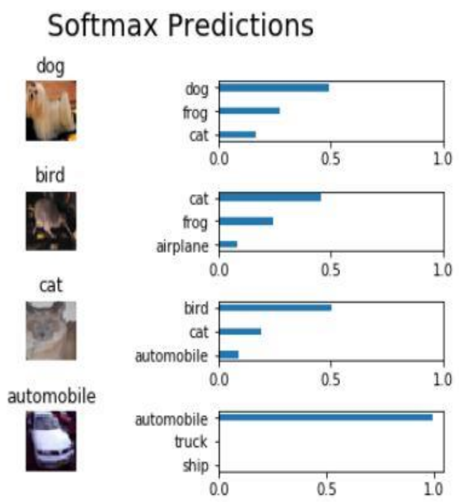
```python
#This the model path where i save my trained result which can be later used for predicting the results

save_model_path = 'D:\\Uni Hildesheim\\4th semester\\DDA Lab\\lab 7\\cifar-10-python\\cifar-10-batches-py\\image_classification'

print('Training...')
with tf.Session() as sess:
    # Initializing the variables
    sess.run(tf.global_variables_initializer())
    summary_writer = tf.summary.FileWriter("D:\\Uni Hildesheim\\4th semester\\DDA Lab\\lab 7\\CNN-relu\\train\\", graph=tf.get_de

    # 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,cost,merged_summary_op,batch_features, batch_labels,epoch)
            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)



# # I have ran each training batch for 5 epoch, as i dont have GPU and my CPU takes more time for running.The model which i have
```

```
WARNING:tensorflow:From <ipython-input-1-032265a8074a>:277: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_op
s) is deprecated and will be removed in 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}.

Training...
Epoch  1, CIFAR-10 Batch 1:  Cost = 31.28889274597168 - Validation Accuracy = 0.1242000013589859
Epoch  1, CIFAR-10 Batch 2:  Cost = 9.799338340759277 - Validation Accuracy = 0.1518000066280365
Epoch  1, CIFAR-10 Batch 3:  Cost = 4.63309907913208 - Validation Accuracy = 0.18140000104904175
Epoch  1, CIFAR-10 Batch 4:  Cost = 3.597994565963745 - Validation Accuracy = 0.21199999749660492
Epoch  1, CIFAR-10 Batch 5:  Cost = 2.816688060760498 - Validation Accuracy = 0.22939999401569366
Epoch  2, CIFAR-10 Batch 1:  Cost = 2.629655599594116 - Validation Accuracy = 0.28200000524520874
Epoch  2, CIFAR-10 Batch 2:  Cost = 2.3413617610931396 - Validation Accuracy = 0.30079999566078186
Epoch  2, CIFAR-10 Batch 3:  Cost = 2.0733025074005127 - Validation Accuracy = 0.3393999934196472
Epoch  2, CIFAR-10 Batch 4:  Cost = 1.9147813320159912 - Validation Accuracy = 0.3691999912261963
Epoch  2, CIFAR-10 Batch 5:  Cost = 1.7407573461532593 - Validation Accuracy = 0.3846000134944916
Epoch  3, CIFAR-10 Batch 1:  Cost = 1.7929636240005493 - Validation Accuracy = 0.40459999442100525
Epoch  3, CIFAR-10 Batch 2:  Cost = 1.6771470308303833 - Validation Accuracy = 0.4334000051021576
Epoch  3, CIFAR-10 Batch 3:  Cost = 1.5652251243591309 - Validation Accuracy = 0.4496000111103058
Epoch  3, CIFAR-10 Batch 4:  Cost = 1.4940121173858643 - Validation Accuracy = 0.46160000562667847
Epoch  3, CIFAR-10 Batch 5:  Cost = 1.3686408996582031 - Validation Accuracy = 0.4729999899864197
Epoch  4, CIFAR-10 Batch 1:  Cost = 1.4570103883743286 - Validation Accuracy = 0.48399999737739563
Epoch  4, CIFAR-10 Batch 2:  Cost = 1.3770177364349365 - Validation Accuracy = 0.49720001220703125
Epoch  4, CIFAR-10 Batch 3:  Cost = 1.3127710819244385 - Validation Accuracy = 0.5041999816894531
Epoch  4, CIFAR-10 Batch 4:  Cost = 1.2679089307785034 - Validation Accuracy = 0.5175999999046326
Epoch  4, CIFAR-10 Batch 5:  Cost = 1.1604394912719727 - Validation Accuracy = 0.5216000080108643
Epoch  5, CIFAR-10 Batch 1:  Cost = 1.2559391260147095 - Validation Accuracy = 0.5242000222206116
Epoch  5, CIFAR-10 Batch 2:  Cost = 1.2053776979446411 - Validation Accuracy = 0.5347999930381775
Epoch  5, CIFAR-10 Batch 3:  Cost = 1.15341317653656 - Validation Accuracy = 0.5479999780654907
Epoch  5, CIFAR-10 Batch 4:  Cost = 1.1412394046783447 - Validation Accuracy = 0.5410000085830688
Epoch  5, CIFAR-10 Batch 5:  Cost = 1.038729190826416 - Validation Accuracy = 0.5550000071525574
```

I have run each training batch for 5 epoch, as I don't have GPU and my CPU takes more time for running. The model which I have saved for training will now be used back for predicting the results for any random 4 images from the test dataset. It produces random images with its top-n (here it's 3) probability of that image belonging to those classes and testing accuracy is plotted. My average training accuracy at the end of epoch 5 is 0.55 (i.e. 55%)

```
try:
    if batch_size:
        pass
except NameError:
    batch_size = 64
#sample of images to be displayed is 4
n_samples = 4
#prediction of each image is its top 3 prediction.
top_n_predictions = 3

def test_model():

    test_features, test_labels = pickle.load(open('preprocess_test.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_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 test_feature_batch, test_label_batch in batch_features_labels(test_features, test_labels, batch_size):
            test_batch_acc_total += sess.run(
                loaded_acc,
                feed_dict={loaded_x: test_feature_batch, loaded_y: test_label_batch})
            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})
        display_image_predictions(random_test_features, random_test_labels, random_test_predictions)

test_model()
```

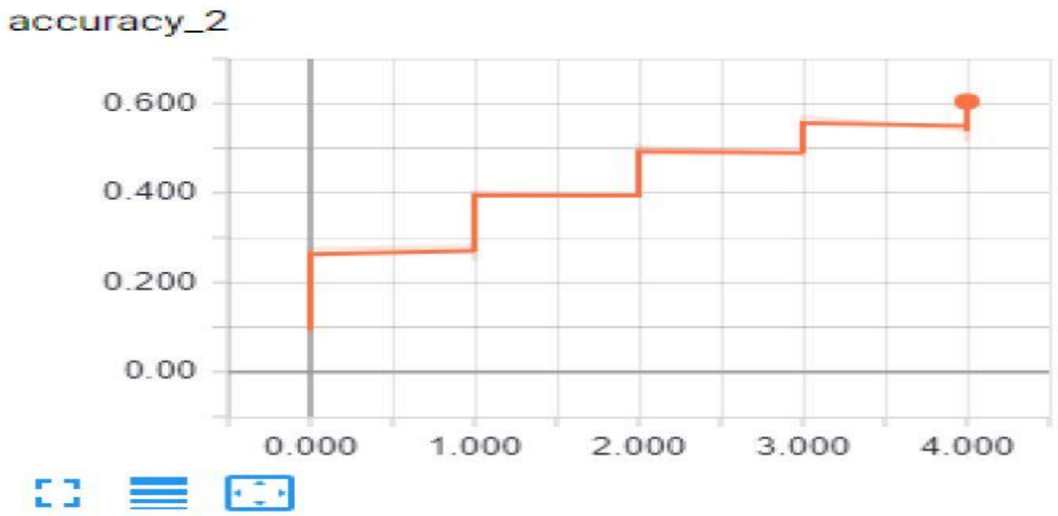
```
INFO:tensorflow:Restoring parameters from D:\Uni Hildesheim\4th semester\DDA Lab\lab 7\cifar-10-python\cifar-10-batches-py\imag
e_classification
Testing Accuracy: 0.5525171399116516
```

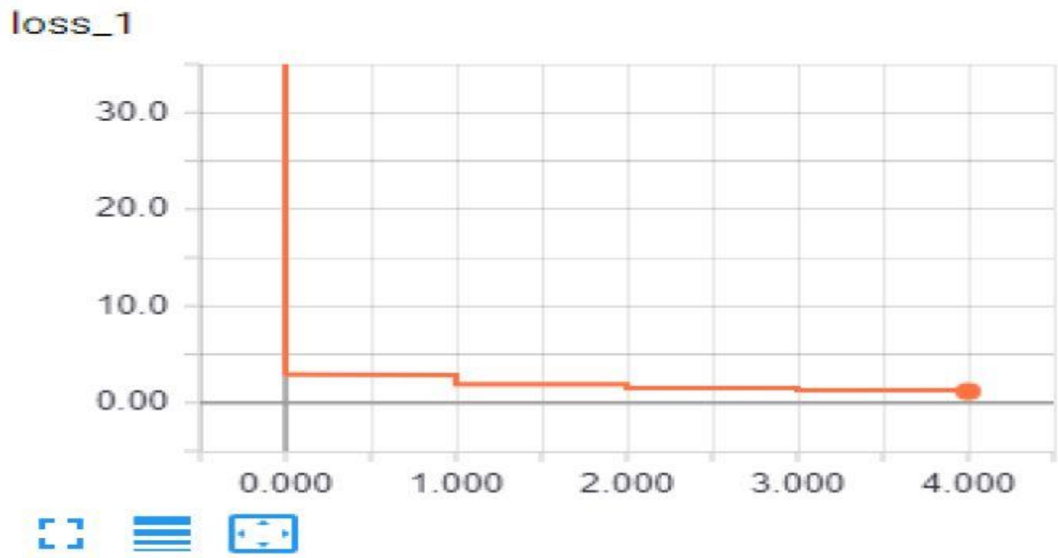


My testing accuracy was 0.552 or (55.2%) for cifar-10 dataset using elu as activation function without any batch normalization. The above are the prediction of random 4 images and top 3 prediction with respect to each images was predicted correctly or not. For this example image 1 and image 4 were predicted properly with SoftMax probability > 0.5, for 2nd and 3rd image it was misclassified (expected bird but displayed as cat and expected cat but displayed as bird as higher probability)

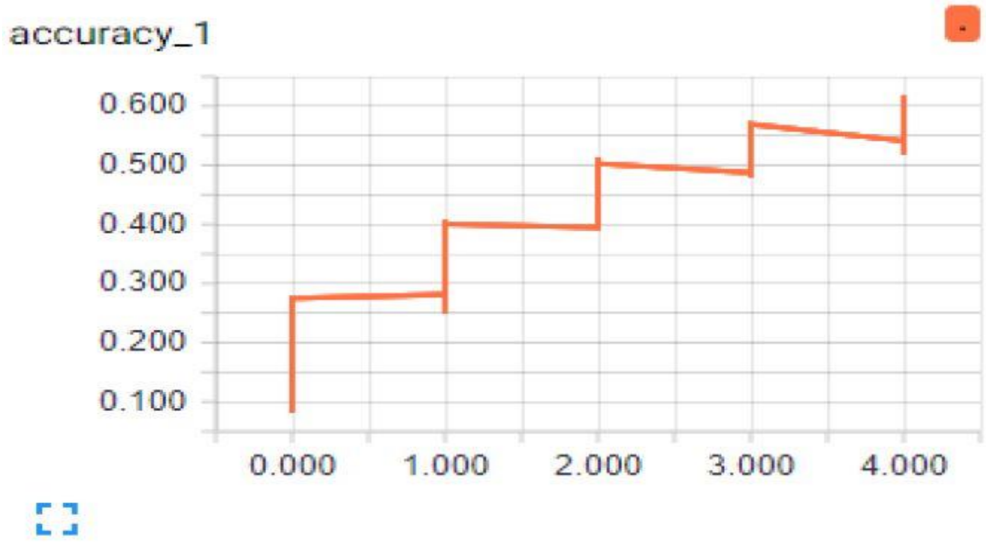
Tensorboard Visualization

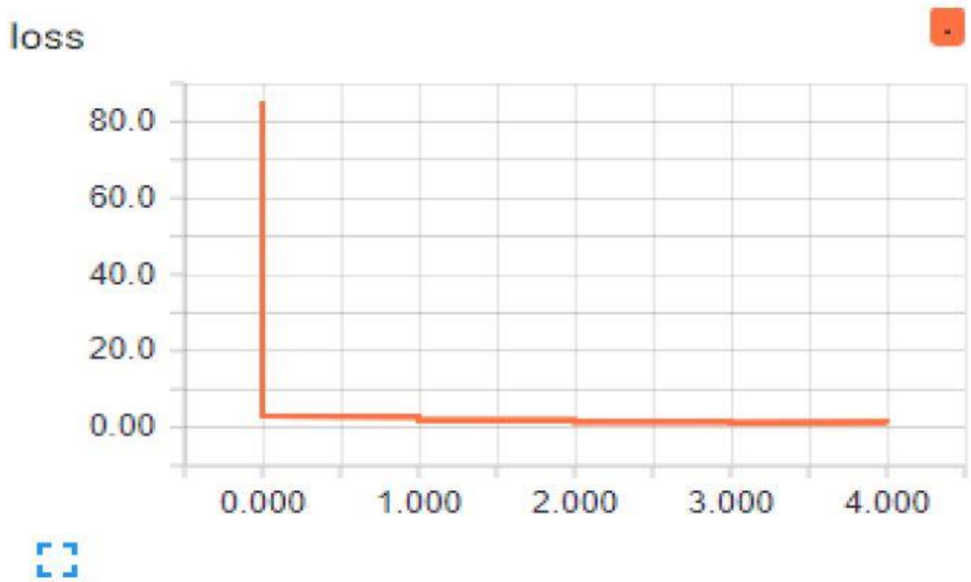
Scalar



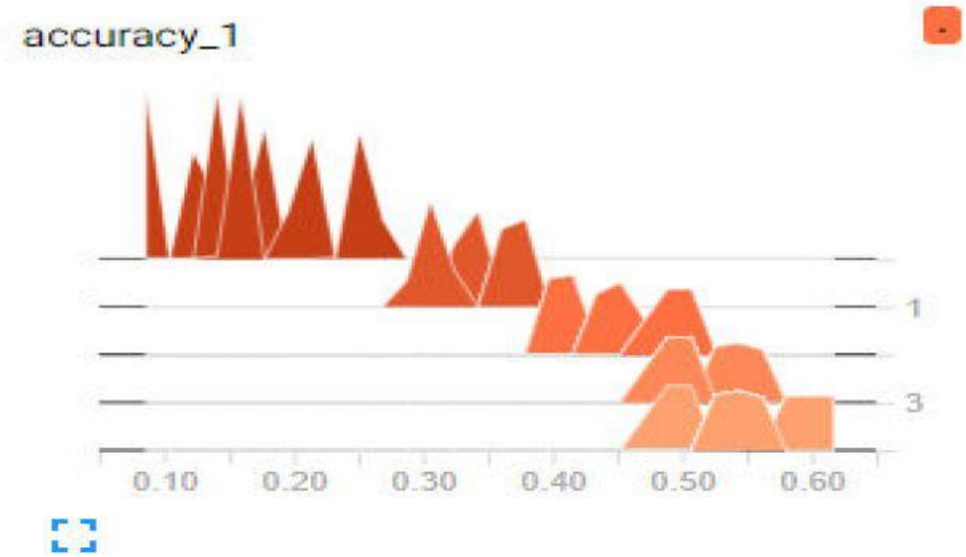


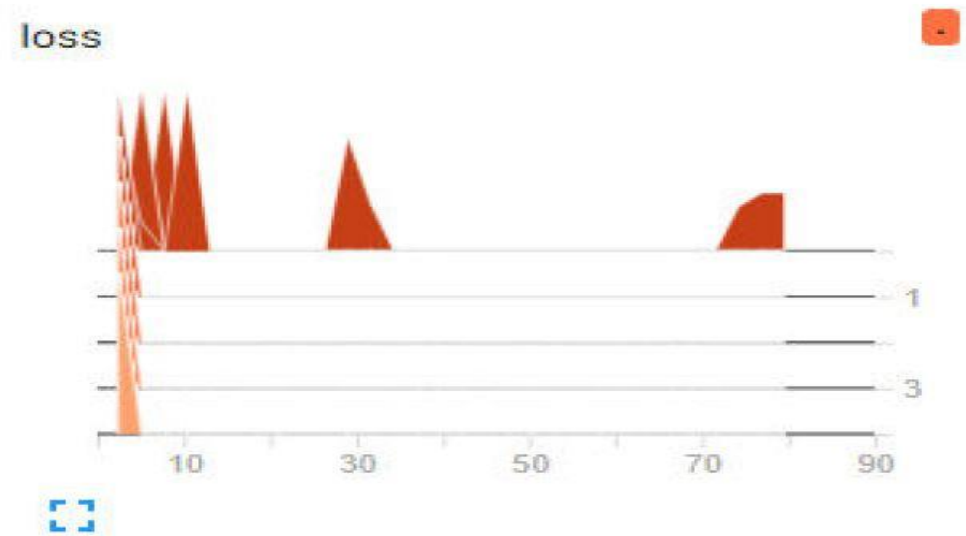
Distributions



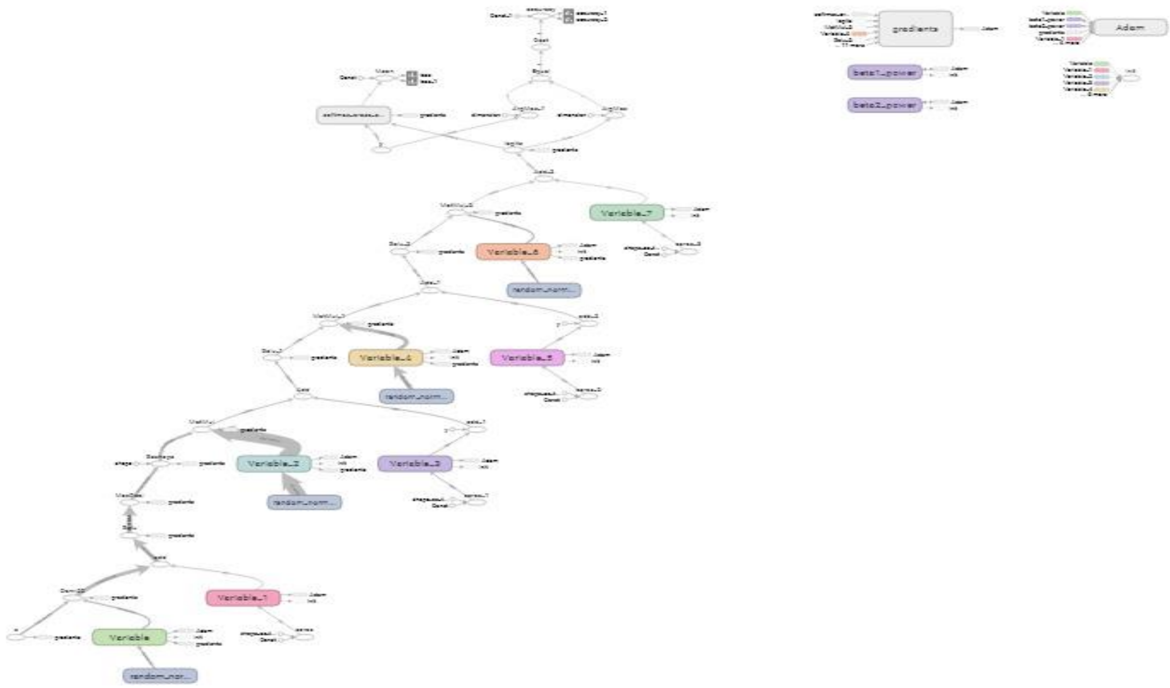


Histograms





Graph


gradients
Adam
beta1_power
beta2_power
Variable_7
Variable_6
Variable_5
Variable_4
Variable_3
Variable_2
Variable_1
Variable

Exercise 2: Convolutional Neural Networks with Selu activation function

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




# This function returns the number of labels in the cifar10 dataset
def _load_label_names():

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

# This function takes the batches from the cifar10 folder and convert each batch size 3072 to the transformation of depth,length,
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:
        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

#This function is used to display the image statistics based on the batch_id and sample_id
#If the batch _id and sample_id is 4 and 1 respectively, it prints the first mage from training batch 4 along with its statistics
```

```python
#This function is used to display the image statistics based on the batch_id and sample_id
#If the batch _id and sample_id is 4 and 1 respectively, it prints the first mage from training batch 4 along with its statistics
def display_stats(cifar10_dataset_folder_path, batch_id, sample_id):

    batch_ids = list(range(1, 6))

    if batch_id not in batch_ids:
        print('Batch Id out of Range. Possible Batch Ids: {}'.format(batch_ids))
        return None

    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, 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]
    label_names = _load_label_names()

    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)

#This function is used to do preproccesing stuff like normalizing the features and one hot encoding the labels and the result is
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'))
```

```python
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)
        validation_count = int(len(features) * 0.1)

        # Prprocess and save a batch of training data
        _preprocess_and_save(
            normalize,
            one_hot_encode,
            features[:-validation_count],
            labels[:-validation_count],
            'preprocess_batch_' + str(batch_i) + '.p')

        # Use a portion of training batch for validation
        valid_features.extend(features[-validation_count:])
        valid_labels.extend(labels[-validation_count:])

    # Preprocess and Save all validation data
    _preprocess_and_save(
        normalize,
        one_hot_encode,
        np.array(valid_features),
        np.array(valid_labels),
        'preprocess_validation.p')

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

    # load 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_test.p')
```

```python
def batch_features_labels(features, labels, batch_size):

    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):

    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)

#Display the prediction for the random test data based on the model saved after training it.
def display_image_predictions(features, labels, 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=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, predi
        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])

#This is the folder path where i downloaded and extracted my cifar10 dataset
cifar10_dataset_folder_path = 'D:\\Uni Hildesheim\\4th semester\\DDA Lab\\lab 7\\cifar-10-python\\cifar-10-batches-py'
```

```python
def normalize(x):

    maximum = np.max(x)
    minimum = np.min(x)
    return (x - minimum) / (maximum - minimum)

def one_hot_encode(x):

    nx = np.max(x) + 1
    return np.eye(nx)[x]

preprocess_and_save_data(cifar10_dataset_folder_path, normalize, one_hot_encode)
valid_features, valid_labels = pickle.load(open('preprocess_validation.p', mode='rb'))

#Defining the feature tensor
def neural_net_image_input(image_shape):

    return tf.placeholder(
        tf.float32,
        [None, image_shape[0], image_shape[1], 3],
        name='x'
    )

#Defining the label tensor
def neural_net_label_input(n_classes):

    return tf.placeholder(
        tf.float32,
        [None, n_classes],
        name='y')




tf.reset_default_graph()
```

```python
#Defining the convolutional neural network with number of output, kernel size, strides, and pool side and padding
def conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, pool_strides):

    # Tensorflow has the crappiest API I've ever seen
    input_depth = x_tensor.get_shape().as_list()[-1]
    W = tf.Variable(tf.random_normal(
        [conv_ksize[0], conv_ksize[1], input_depth, conv_num_outputs],
        stddev=0.1
    ))
    b = tf.Variable(tf.zeros(conv_num_outputs))
    conv = tf.nn.conv2d(x_tensor, W, [1, conv_strides[0], conv_strides[1], 1], 'SAME') + b
    conv = tf.nn.selu(conv)
    return tf.nn.max_pool(
        conv,
        [1, pool_ksize[0], pool_ksize[1], 1],
        [1, pool_strides[0], pool_strides[1], 1],
        'SAME')

def flatten(x_tensor):

    shape = x_tensor.get_shape().as_list()
    return tf.reshape(x_tensor, [-1, np.prod(shape[1:])])

#Defined a Fully connected layer with selu activation function
def fully_conn(x_tensor, num_outputs):

    shape = x_tensor.get_shape().as_list()
    W = tf.Variable(tf.random_normal([shape[-1], num_outputs], stddev=0.1))
    b = tf.Variable(tf.zeros(num_outputs)) + 0.11
    return tf.nn.selu(tf.add(tf.matmul(x_tensor, W), b))

#defining the function for output of each network
def output(x_tensor, num_outputs):

    shape = x_tensor.get_shape().as_list()
    W = tf.Variable(tf.random_normal([shape[-1], num_outputs]))
    b = tf.Variable(tf.zeros(num_outputs))
    return tf.add(tf.matmul(x_tensor, W), b)
```

```python
def conv_net(x):

    #    conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, pool_strides)
    tmp = conv2d_maxpool(x, 64, [3, 3], [1, 1], [3, 3], [2, 2])

    #Batch normalization
    tf.nn.batch_normalization(tmp,mean=tf.zeros([64]),variance=tf.ones([64]),variance_epsilon=0.01,scale=None,offset=None)
    #Another maxpooling function

    tmp = tf.nn.max_pool(tmp,[1, 3, 3, 1],[1, 2, 2, 1],'SAME')

    tmp = flatten(tmp)
    #one fully connected layer fc1
    tmp = fully_conn(tmp, 384)

    #second fully connected layer fc2
    tmp = fully_conn(tmp, 192)

    return output(tmp, 10)

tf.reset_default_graph()

# Inputs
x = neural_net_image_input((32, 32, 3))
y = neural_net_label_input(10)

# Model
logits = conv_net(x)

# 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().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')
tf.summary.histogram("loss",cost)
tf.summary.scalar("loss", cost)
# Create a summary to monitor accuracy tensor
tf.summary.histogram("accuracy", accuracy)
tf.summary.scalar("accuracy", accuracy)
# Merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()
```

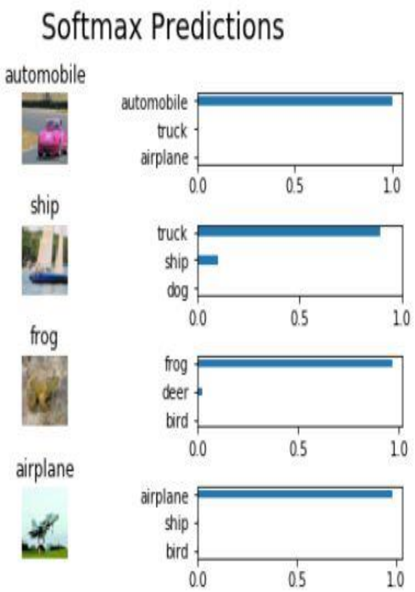
```python
def conv_net(x):

    #    conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, pool_strides)
    tmp = conv2d_maxpool(x, 64, [3, 3], [1, 1], [3, 3], [2, 2])

    #Batch normalization
    tf.nn.batch_normalization(tmp,mean=tf.zeros([64]),variance=tf.ones([64]),variance_epsilon=0.01,scale=None,offset=None)
    #Another maxpooling function

    tmp = tf.nn.max_pool(tmp,[1, 3, 3, 1],[1, 2, 2, 1],'SAME')

    tmp = flatten(tmp)
    #one fully connected layer fc1
    tmp = fully_conn(tmp, 384)

    #second fully connected layer fc2
    tmp = fully_conn(tmp, 192)

    return output(tmp, 10)

tf.reset_default_graph()

# Inputs
x = neural_net_image_input((32, 32, 3))
y = neural_net_label_input(10)

# Model
logits = conv_net(x)

# 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().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')
tf.summary.histogram("loss",cost)
tf.summary.scalar("loss", cost)
# Create a summary to monitor accuracy tensor
tf.summary.histogram("accuracy", accuracy)
tf.summary.scalar("accuracy", accuracy)
# Merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()
```

```
WARNING:tensorflow:From <ipython-input-1-06b5b4e2ae54>:278: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_op
s) is deprecated and will be removed in 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}.

Training...
Epoch  1, CIFAR-10 Batch 1:  Cost = 80.84798431396484 - Validation Accuracy = 0.17839999496936798
Epoch  1, CIFAR-10 Batch 2:  Cost = 20.6914119720459 - Validation Accuracy = 0.18320000171661377
Epoch  1, CIFAR-10 Batch 3:  Cost = 8.855873107910156 - Validation Accuracy = 0.25099998712539673
Epoch  1, CIFAR-10 Batch 4:  Cost = 8.763263702392578 - Validation Accuracy = 0.2669999897480011
Epoch  1, CIFAR-10 Batch 5:  Cost = 5.199768543243408 - Validation Accuracy = 0.3296000063419342
Epoch  2, CIFAR-10 Batch 1:  Cost = 4.5552825927734375 - Validation Accuracy = 0.33559998869895935
Epoch  2, CIFAR-10 Batch 2:  Cost = 3.9713339805603027 - Validation Accuracy = 0.35519999265670776
Epoch  2, CIFAR-10 Batch 3:  Cost = 3.5299386978149414 - Validation Accuracy = 0.36160001158714294
Epoch  2, CIFAR-10 Batch 4:  Cost = 3.0858263969421387 - Validation Accuracy = 0.3792000114917755
Epoch  2, CIFAR-10 Batch 5:  Cost = 2.8657493591308594 - Validation Accuracy = 0.38999998569488525
Epoch  3, CIFAR-10 Batch 1:  Cost = 2.725909948348999 - Validation Accuracy = 0.3858000040054321З
Epoch  3, CIFAR-10 Batch 2:  Cost = 2.603868007659912 - Validation Accuracy = 0.3950000107288360б
Epoch  3, CIFAR-10 Batch 3:  Cost = 2.544926166534424 - Validation Accuracy = 0.39340001344680786
Epoch  3, CIFAR-10 Batch 4:  Cost = 2.435222864151001 - Validation Accuracy = 0.4156000018119812
Epoch  3, CIFAR-10 Batch 5:  Cost = 2.32495379447937 - Validation Accuracy = 0.41339999437332153
Epoch  4, CIFAR-10 Batch 1:  Cost = 2.1152091026306152 - Validation Accuracy = 0.4334000051021576
Epoch  4, CIFAR-10 Batch 2:  Cost = 2.1303391456604004 - Validation Accuracy = 0.42820000648498535
Epoch  4, CIFAR-10 Batch 3:  Cost = 2.05873703956604 - Validation Accuracy = 0.4431999921798706
Epoch  4, CIFAR-10 Batch 4:  Cost = 1.951916217803955 - Validation Accuracy = 0.448199987411499
Epoch  4, CIFAR-10 Batch 5:  Cost = 2.0254485607147217 - Validation Accuracy = 0.453000009059906
Epoch  5, CIFAR-10 Batch 1:  Cost = 1.832728624343872 - Validation Accuracy = 0.46860000491142273
Epoch  5, CIFAR-10 Batch 2:  Cost = 1.959446668624878 - Validation Accuracy = 0.4431999921798706
Epoch  5, CIFAR-10 Batch 3:  Cost = 1.8937727212905884 - Validation Accuracy = 0.46480000019073486
Epoch  5, CIFAR-10 Batch 4:  Cost = 1.6802111864089966 - Validation Accuracy = 0.47839999198913574
Epoch  5, CIFAR-10 Batch 5:  Cost = 1.8028337955474854 - Validation Accuracy = 0.47679999470710754
```

I have run each training batch for 5 epoch, as I don’t have GPU and my CPU takes more time for running. The model which I have saved for training will now be used back for predicting the results for any random 4 images from the test dataset. It produces random images with its top-n (here it's 3) probability of that image belonging to those classes and testing accuracy is plotted. My average training accuracy at the end of epoch 5 is 0.47 (i.e. 47%)

```
try:
    if batch_size:
        pass
except NameError:
    batch_size = 64


n_samples = 4
top_n_predictions = 3

def test_model():

    test_features, test_labels = pickle.load(open('preprocess_test.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_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 test_feature_batch, test_label_batch in batch_features_labels(test_features, test_labels, batch_size):
            test_batch_acc_total += sess.run(
                loaded_acc,
                feed_dict={loaded_x: test_feature_batch, loaded_y: test_label_batch})
            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})
        display_image_predictions(random_test_features, random_test_labels, random_test_predictions)

test_model()
```

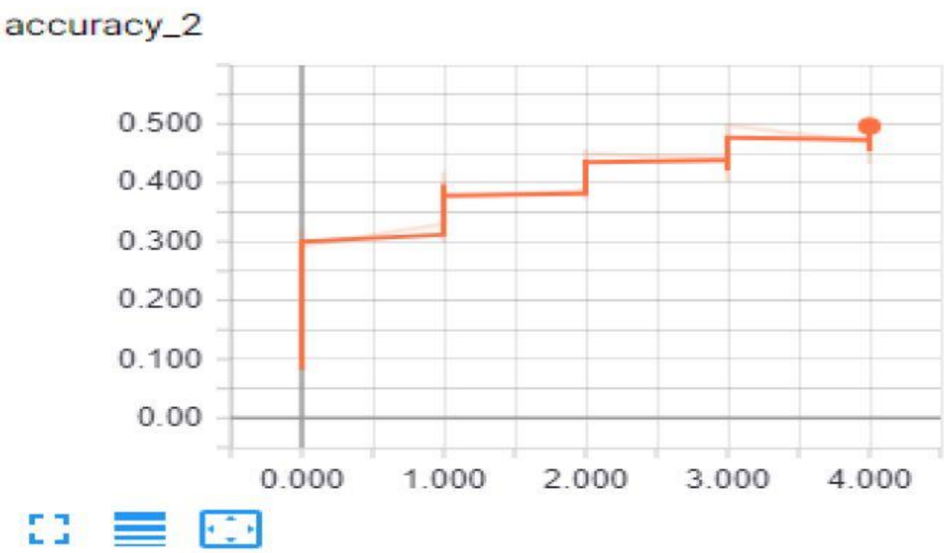
```
INFO:tensorflow:Restoring parameters from D:\Uni Hildesheim\4th semester\DDA Lab\lab 7\cifar-10-python\cifar-10-batches-py\imag
e_classification
Testing Accuracy: 0.489899554848671
```



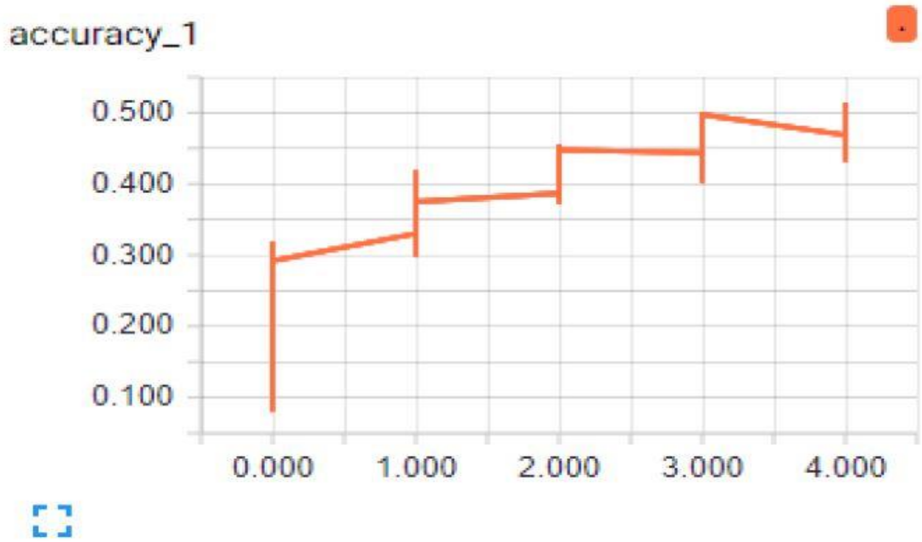
My testing accuracy was 0.48 or (48%) for cifar-10 dataset using selu as activation function without any batch normalization. The above are the prediction of random 4 images and top 3 prediction with respect to each images was predicted correctly or not. For this example image 1, image 3 and image 4 were predicted properly with SoftMax probability > 0.5, for 2nd image it was misclassified (expected ship but displayed as truck)

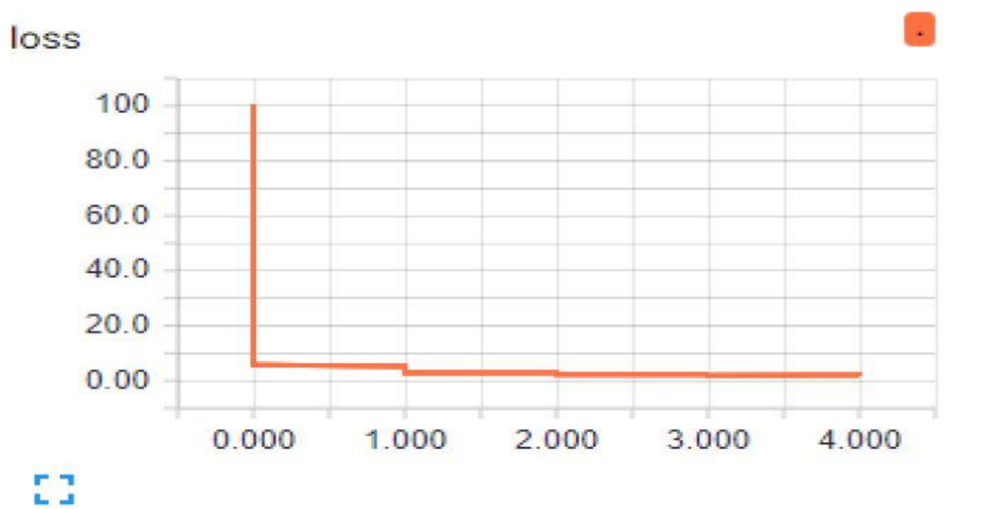
Tensorboard Visualization

Scalar

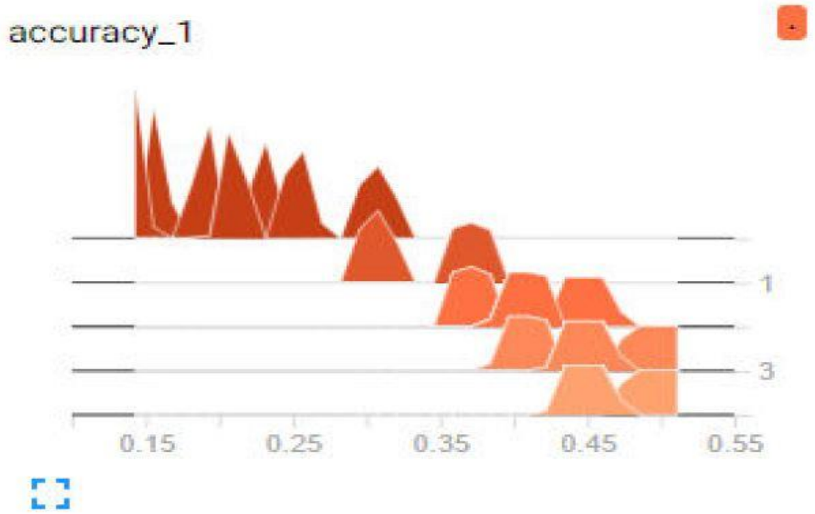


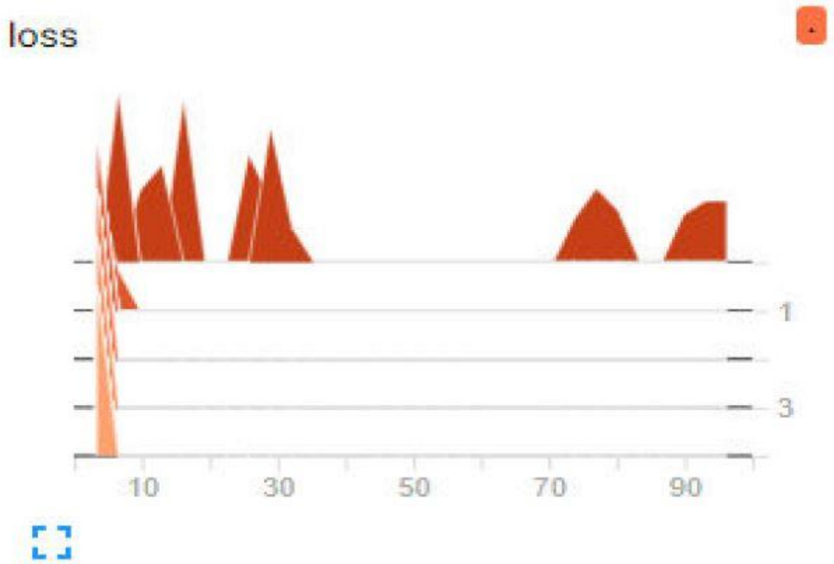
Distribution





Histogram





Graph