Lab Course - Distributed Data Analytics
Exercise 7

Function name : unpickle
Parameter : file name from which data is to be retrieved

This function takes the file names as input and extracts the image data and corresponding labels from the same.

```
def unpickle(file):
    with open(file, 'rb') as fo:
        dict = cPickle.load(fo,encoding="latin1")
        data = dict["data"]
        labels = dict['labels']
    return data,labels
```

```
data1,labels1 = unpickle("gdrive/My Drive/DDA/CIFAR/Dataset/cifar-10-batches-py/data_batch_1")
data2,labels2 = unpickle("gdrive/My Drive/DDA/CIFAR/Dataset/cifar-10-batches-py/data_batch_2")
data3,labels3 = unpickle("gdrive/My Drive/DDA/CIFAR/Dataset/cifar-10-batches-py/data_batch_3")
data4,labels4 = unpickle("gdrive/My Drive/DDA/CIFAR/Dataset/cifar-10-batches-py/data_batch_4")
data5,labels5 = unpickle("gdrive/My Drive/DDA/CIFAR/Dataset/cifar-10-batches-py/data_batch_5")
```

Function name : get_labels
Parameter : target classes

This function one-hot codes the target classes to binary values of 10 values as the total number of classes is 10.

```
def get_labels(y):
    labels = np.zeros((len(y), 10))
    for idx, val in enumerate(y):
        labels[idx][val] = 1
    return labels
```

```
train= np.concatenate([data1, data2, data3, data4, data5], axis=0)
train= np.dstack((train[:, :1024], train[:, 1024:2048], train[:, 2048:])) / 1.0
train= (train - 128) / 255.0
train= train.reshape(-1, 32, 32, 3)

label = np.concatenate([labels1, labels2, labels3, labels4, labels5], axis=0)
label = get_labels(label)
```

The input data and the target values are splitted for train and test as shown in the snippet of code below.

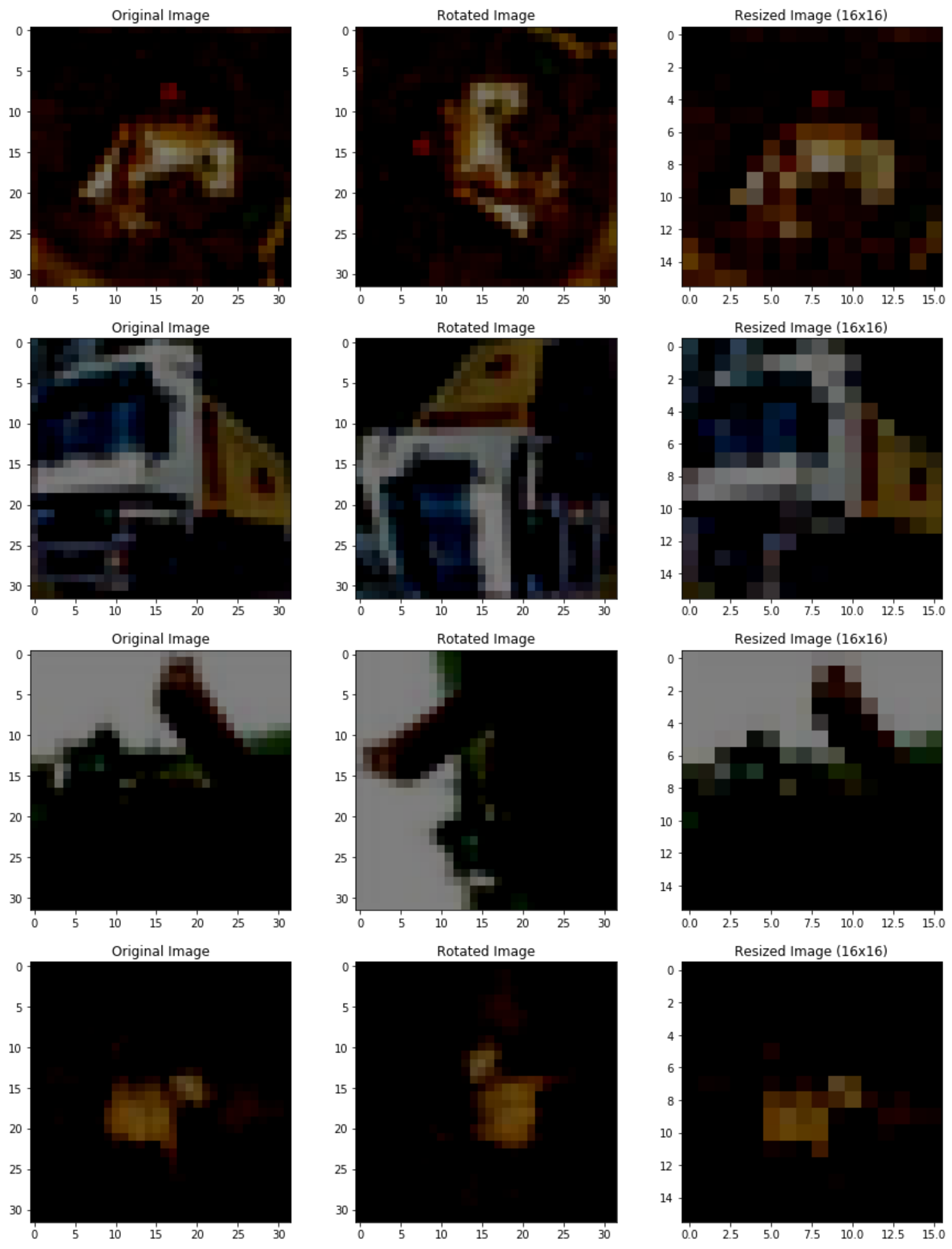
```
x_train, x_test, y_train, y_test = train_test_split(train, label, test_size=0.08, random_state=42)
```

The following part of the report shows the augmentation techniques performed with the tensorflow for resizing the image and rotating the same.

```
X = tf.placeholder(tf.float32, shape = (32,32,3))
rotated = tf.image.rot90(X)
resize = tf.image.resize_images(X,(16,16),tf.image.ResizeMethod.NEAREST_NEIGHBOR)
```

We create placeholder for the input image and also the operations for rotating and resizing the images are set and are tensors are created for computation.

```
with tf.Session() as sess:
    for image in X_tr[:4]:
        rot_img,res_img = sess.run([rotated,resize],feed_dict={X:image})
        rotated_images.append(rot_img)
        resized_images.append(res_img)
```



Once the variables are initialised, a session is created and the augmentation on these images are performed and used in the data for training the model.

Function name : conv2d
Parameter : input,weights,bias

This function performs a 2-dimensional convolution on the input image with the initialised weights and biases. A stride of length 1 is set and the padding is made same. Once the convolution is performed, bias is added to it and sigmoid activation function is used. The activation function is tf.nn.relu for exercise1.

```
def conv2d(x,weight,bias):
    x = tf.nn.conv2d(x,weight,strides=[1,1,1,1],padding="SAME")
    x = tf.nn.bias_add(x,bias)
    x = tf.nn.selu(x)
    return x
```

Function name : maxpool2d
Parameter : input

This function performs a down sampling on the convoluted layer. The pooling is based on average of the values with a kernel of size 2 and stride of 2. The padding used here is same.

```
def maxpool2d(x):
    x = tf.layers.max_pooling2d(x,pool_size=2,strides=2,padding="SAME")
    return x
```

Function name : batch_normalize
Parameter : input

This function performs a normalization of the batch of given inputs with tensorflow layers and returns the same.

```
def batch_normalize(x):
    x = tf.layers.batch_normalization(x)
    return x
```

Function name : flatten
Parameter : input

This function flattens the feature maps to a single vector which is then fed into the fully connected layers.

```
def flatten(x):
    flat = tf.contrib.layers.flatten(x)
    return flat
```

Function name : dropout
Parameter : input, probability

This function acts as regularization for the neural network which drops out the neuron that doesn't full the input probability and keeps only the ones that has probability more than the specified ones.

```
def dropout(x,keep_prob):
    fc = tf.nn.dropout(x,keep_prob)
    return fc
```

Function name : fc_layer
Parameter : inputs, number of outputs

This function is the output layer and has the number of neurons equal to the number of classes. This provides the feature map by using relu activation function for exercise 1 and selu for exercise 2. The output of this layer is sent to output layer for predicting the activity.

```
def fc_layer(x,num_outputs):
    fc = tf.contrib.layers.fully_connected(inputs=x,
                                           num_outputs=num_outputs,
                                           activation_fn=tf.nn.selu)
    return fc
```

Function name : out_layer
Parameter : inputs, number of outputs

This function is the output layer and has the number of neurons equal to the number of classes. This provides the feature map by multiplying the weights with input feature map and adding the bias to it. The output of this layer is sent to softmax for predicting the activity.

```
def out_layer(x,num_outputs):
    x = tf.contrib.layers.fully_connected(inputs=x, num_outputs=num_outputs, activation_fn=None)
    return x
```

Exercise 1:

The following are the parameters set, variables, weights and bias initialised for the networks to train the CIFAR-CNN model.

```
x = tf.placeholder(tf.float32, shape = (None,32,32,3))
y = tf.placeholder(tf.float32,(None,n_class))

with tf.variable_scope("Weights", reuse=tf.AUTO_REUSE):
    w_conv1 = tf.get_variable('w_c1',shape=(3,3,3,64),
                              initializer=tf.contrib.layers.xavier_initializer(uniform=False))
    tf.summary.histogram('w1',w_conv1)

with tf.variable_scope("Bias", reuse=tf.AUTO_REUSE):
    b_conv1 = tf.get_variable('B_c1',shape=(64),
                             initializer=tf.contrib.layers.xavier_initializer(uniform=False))
```

Function name : CIFAR_CNN
Parameter : inputs

The function is used to perform the training of the model by calling the required convolutions, pooling and fully connected layers as specified in the architecture.

```
def CIFAR_CNN(x):

    conv1 = conv2d(x,w_conv1,b_conv1)
    conv1_pool = maxpool2d(conv1)

    flat = flatten(conv1_pool)

    fc1 = fc_layer(flat,64)
    out = out_layer(fc1,10)

    return out
```

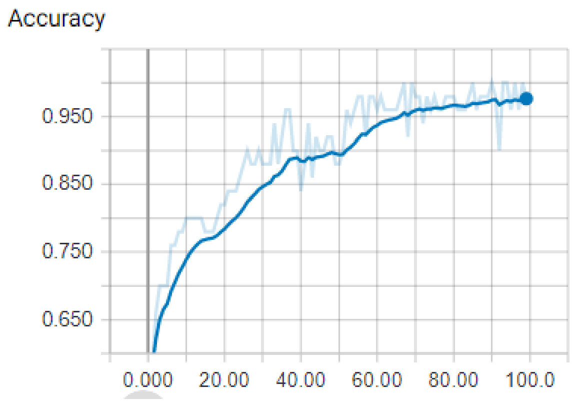
The following snippet of code shows the declaration of the optimizers and other operations. The activation function used in the output layer is softmax. The loss function used here is softmax cross entropy loss and the optimizer used is Adam Optimizer with learning rate 0.001 and other default parameters as specified in the tensorflow framework. Also, the scalar and histogram summaries are recorded to visualise with tensorboard.

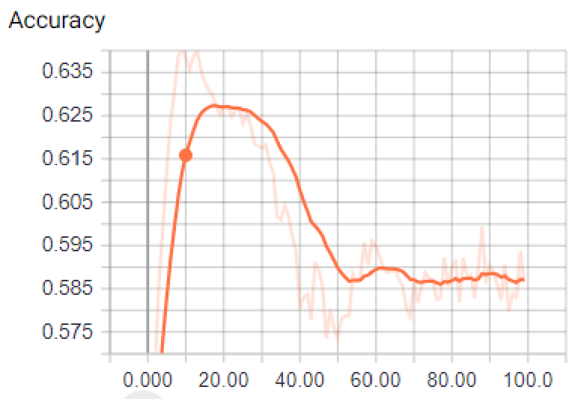
```
model = CIFAR_CNN(x)

predictions = tf.nn.softmax_cross_entropy_with_logits(logits=model,labels=y)
tf.summary.histogram("predictions", predictions)

loss = tf.reduce_mean(predictions)
tf.summary.scalar("Loss_Scalar",loss)
tf.summary.histogram("loss",loss)

with tf.variable_scope("Adam", reuse=tf.AUTO_REUSE):
    optimizer= tf.train.AdamOptimizer().minimize(loss)

correct_pred = tf.equal(tf.argmax(model, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
tf.summary.scalar("Accuracy",accuracy)
tf.summary.histogram("Accuracy_histogram",accuracy)
```

Once the variables are initialised and optimizers are defined, the variables are globally initialised and the session for the tensorflow is started. For each epoch, a batch of size 50 is made on the input train data. Also, in this part the training and testing summary and results are recorded for tensorboard visualisations.

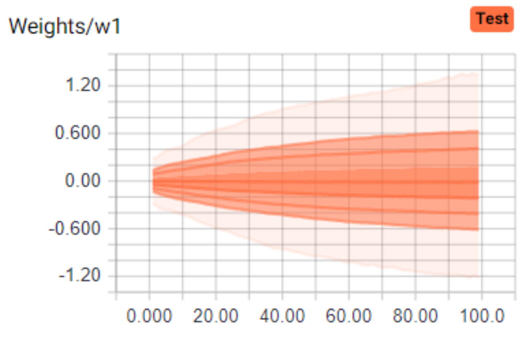
```
init = tf.global_variables_initializer()
epochs = 100
batch_size = 50
train_acc,test_acc,train_loss,test_loss=[],[],[],[]
with tf.Session() as sess:
    train_writer = tf.summary.FileWriter("./log5/1/Train",sess.graph)
    test_writer = tf.summary.FileWriter("./log5/1/Test",sess.graph)
    merge = tf.summary.merge_all()
    sess.run(init)
    counter = 0
    for ep in range(epochs):
        counter += 1
        for batch in range(len(x_train)//batch_size):
            batch_x = x_train[batch*batch_size:min((batch+1)*batch_size,len(x_train))]
            batch_y = y_train[batch*batch_size:min((batch+1)*batch_size,len(y_train))]
            assert not np.any(np.isnan(batch_x))
            assert not np.any(np.isnan(batch_y))
            feed = {x : batch_x, y : batch_y}
            summary, cost, _ , acc = sess.run([merge, loss, optimizer, accuracy], feed_dict = feed)
        print("\nEpoch: "+str(ep)+"\tTraining loss: "+str(cost)+"\tTraining Accuracy: "+str(acc))
        train_acc.append(acc)
        train_loss.append(cost)
        train_writer.add_summary(summary,counter)
        feed = {x : x_test, y : y_test}
        su, cost, acc = sess.run([merge, loss, accuracy], feed_dict = feed)
        test_acc.append(acc)
        test_loss.append(cost)
        test_writer.add_summary(su,counter)
        print("\t\tTesting loss: "+str(cost)+"\tTesting Accuracy: "+str(acc))
```

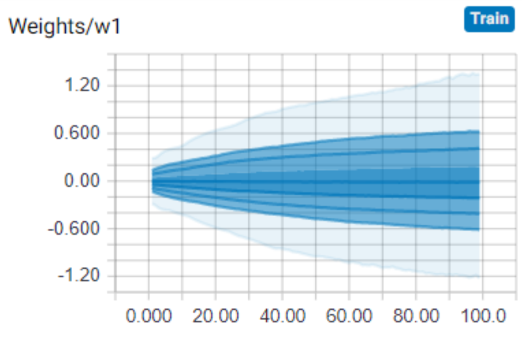
The maximum training accuracy achieved was 99% and the testing accuracy was noted to be 63%. The accuracy of training and testing are recorded and plotted respectively as shown below.



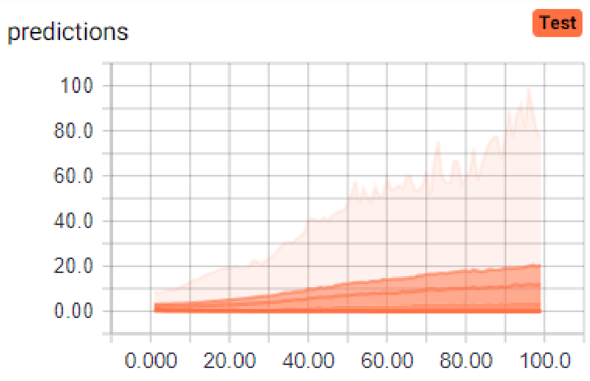


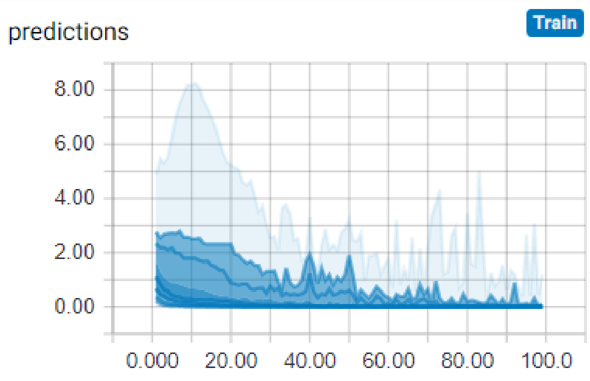
Weight Distributions:



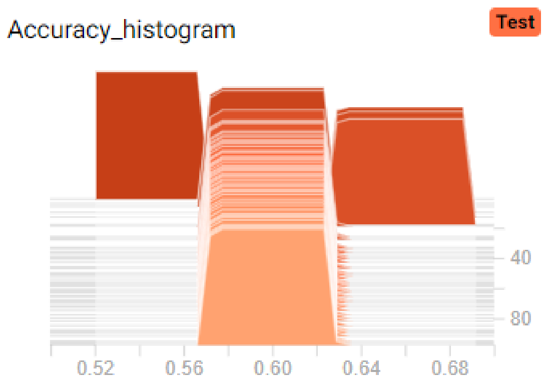


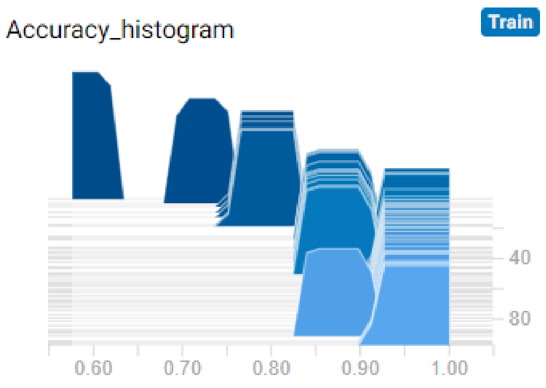
Prediction Distributions:



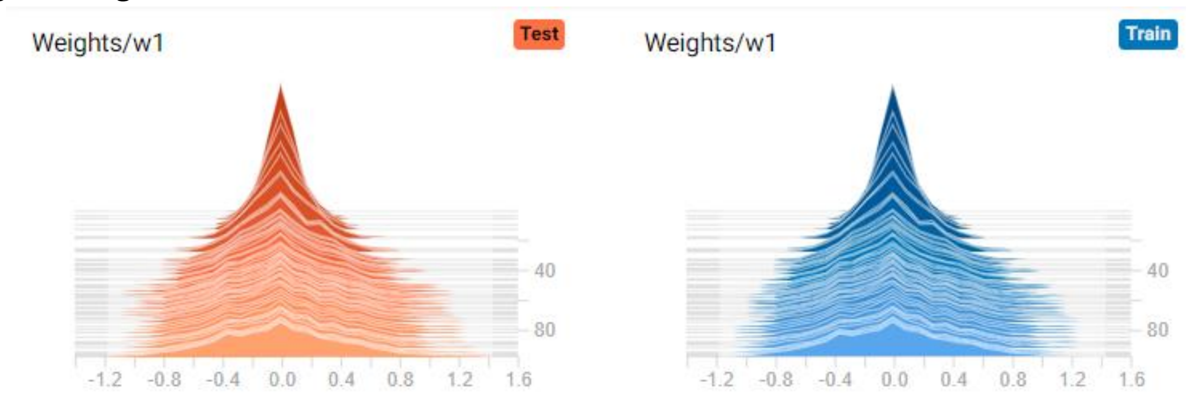


Accuracy Histogram:

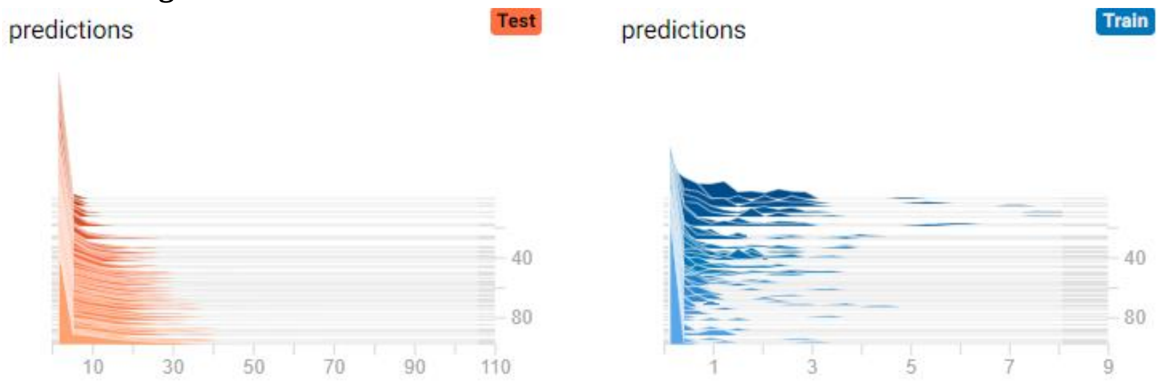




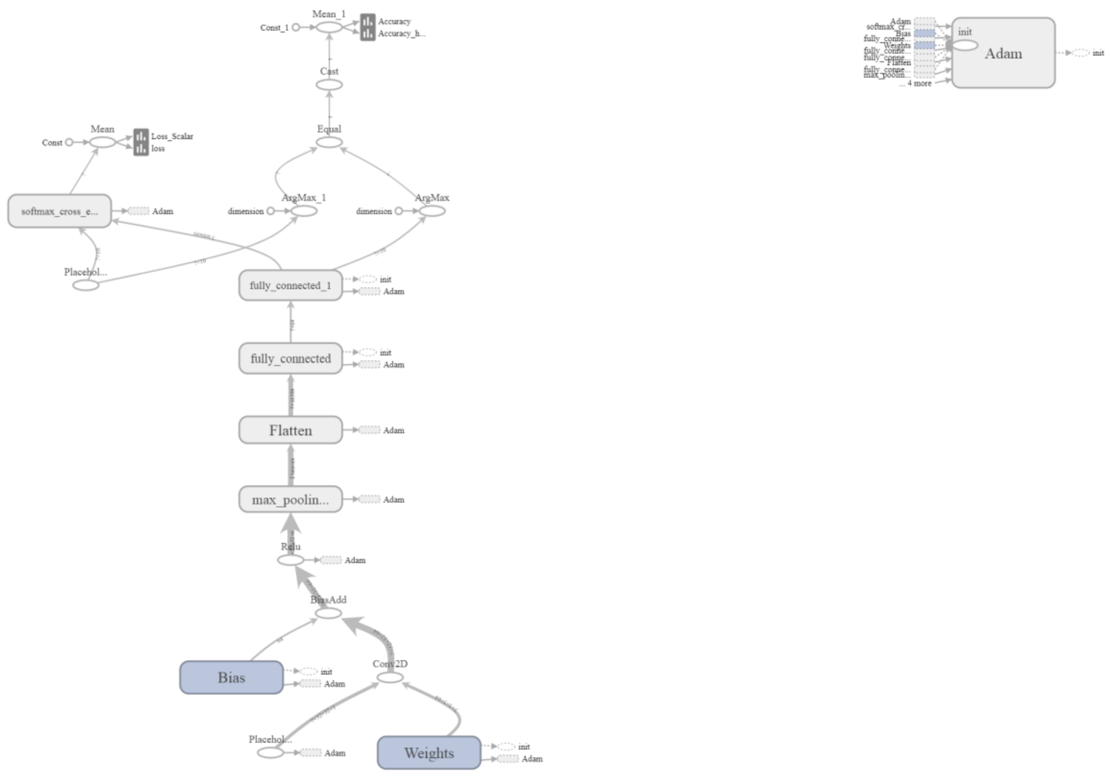
Weight Histogram:



Prediction Histogram:

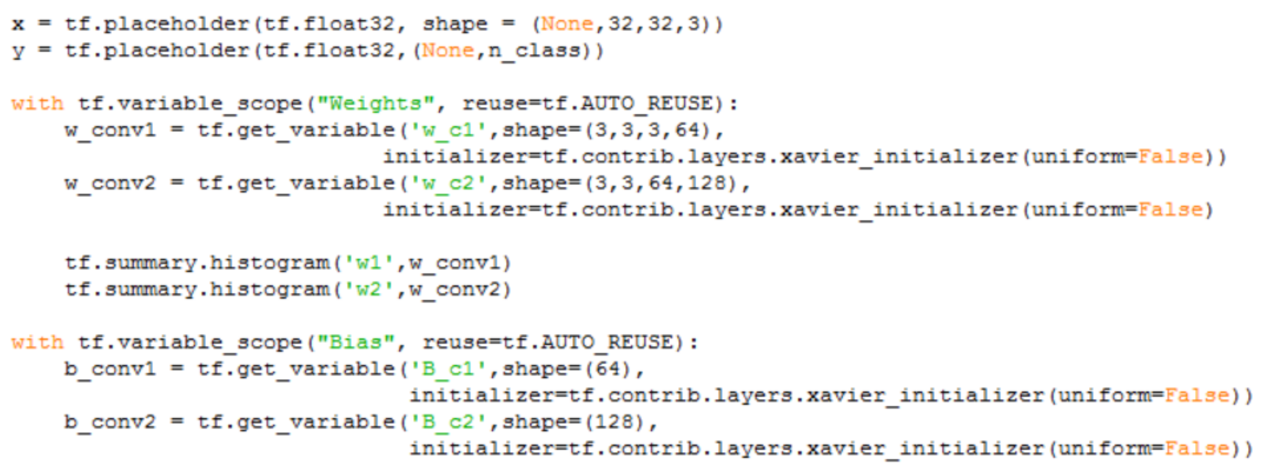


Computational Graph:



Exercise 2 with Batch Normalization, relu and dropout:

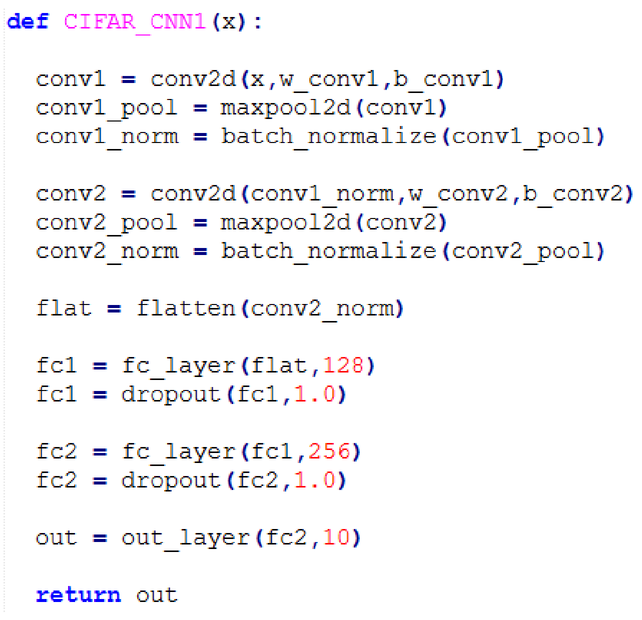
The following are the parameters set, variables, weights and bias initialised for the networks to train the CIFAR-CNN1 model with ReLU activation function.



```
x = tf.placeholder(tf.float32, shape = (None,32,32,3))
y = tf.placeholder(tf.float32,(None,n_class))

with tf.variable_scope("Weights", reuse=tf.AUTO_REUSE):
    w_conv1 = tf.get_variable('w_c1',shape=(3,3,3,64),
                            initializer=tf.contrib.layers.xavier_initializer(uniform=False))
    w_conv2 = tf.get_variable('w_c2',shape=(3,3,64,128),
                            initializer=tf.contrib.layers.xavier_initializer(uniform=False)

    tf.summary.histogram('w1',w_conv1)
    tf.summary.histogram('w2',w_conv2)

with tf.variable_scope("Bias", reuse=tf.AUTO_REUSE):
    b_conv1 = tf.get_variable('B_c1',shape=(64),
                              initializer=tf.contrib.layers.xavier_initializer(uniform=False))
    b_conv2 = tf.get_variable('B_c2',shape=(128),
                              initializer=tf.contrib.layers.xavier_initializer(uniform=False))
```

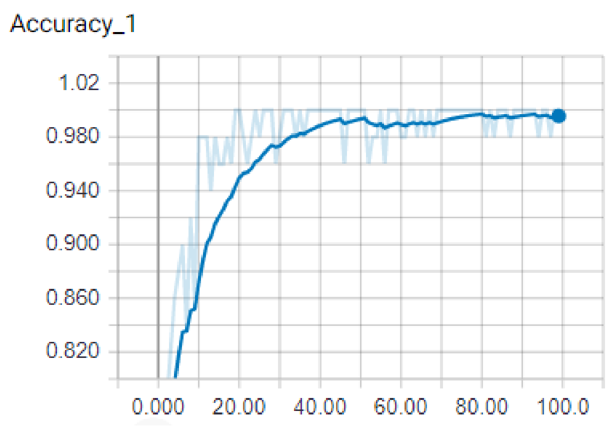
Function name : CIFAR_CNN1
Parameter : inputs

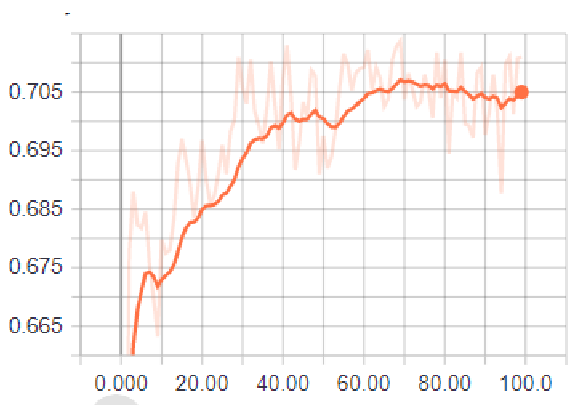
The function is used to perform the training of the model by calling the required convolutions, pooling and fully connected layers with relu activations, batch normalization and dropout regularization.



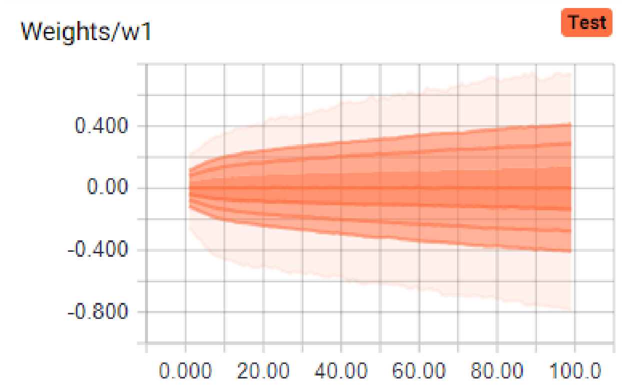
```
def CIFAR_CNN1(x):

  conv1 = conv2d(x,w_conv1,b_conv1)
  conv1_pool = maxpool2d(conv1)
  conv1_norm = batch_normalize(conv1_pool)

  conv2 = conv2d(conv1_norm,w_conv2,b_conv2)
  conv2_pool = maxpool2d(conv2)
  conv2_norm = batch_normalize(conv2_pool)

  flat = flatten(conv2_norm)

  fc1 = fc_layer(flat,128)
  fc1 = dropout(fc1,1.0)

  fc2 = fc_layer(fc1,256)
  fc2 = dropout(fc2,1.0)

  out = out_layer(fc2,10)

  return out
```

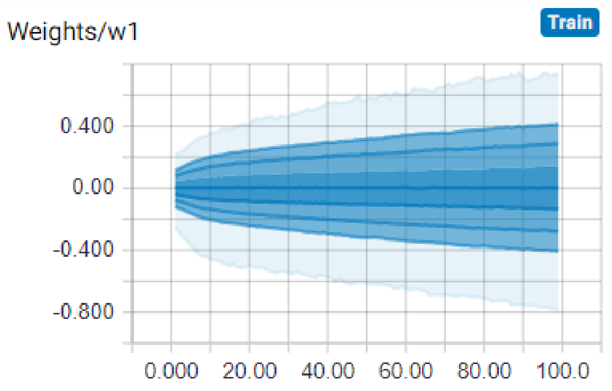
The maximum training accuracy achieved was 99% and the testing accuracy was noted to be 71%. The accuracy of training and testing are recorded and plotted respectively as shown below.



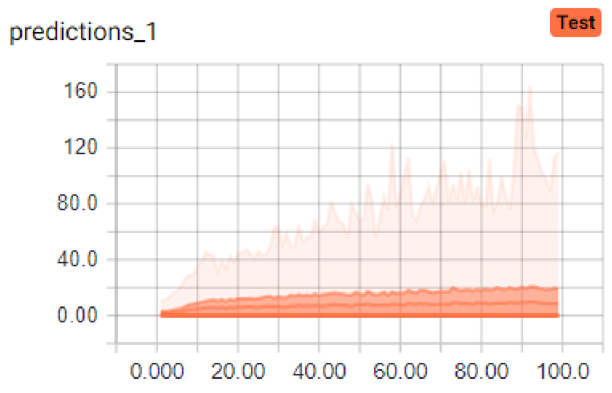


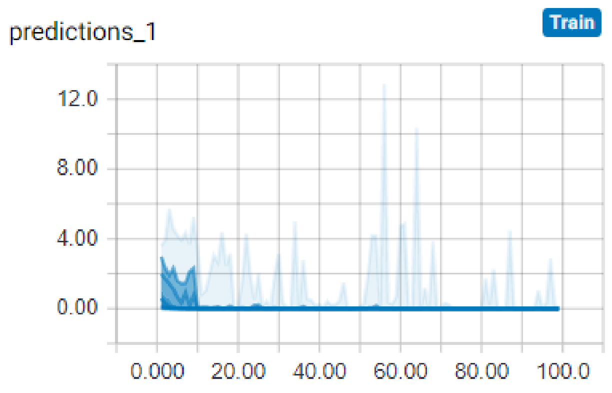
Weight Distribution:



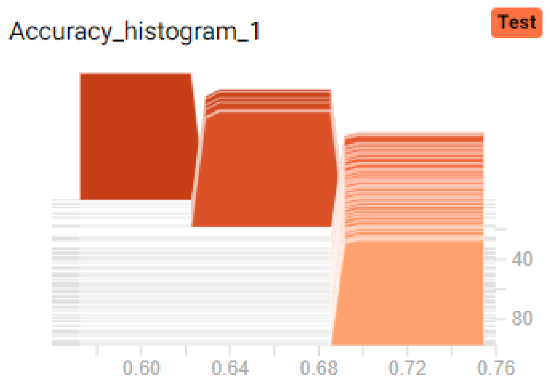


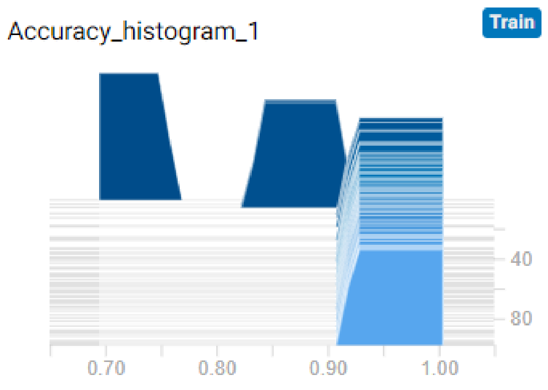
Prediction Distribution:



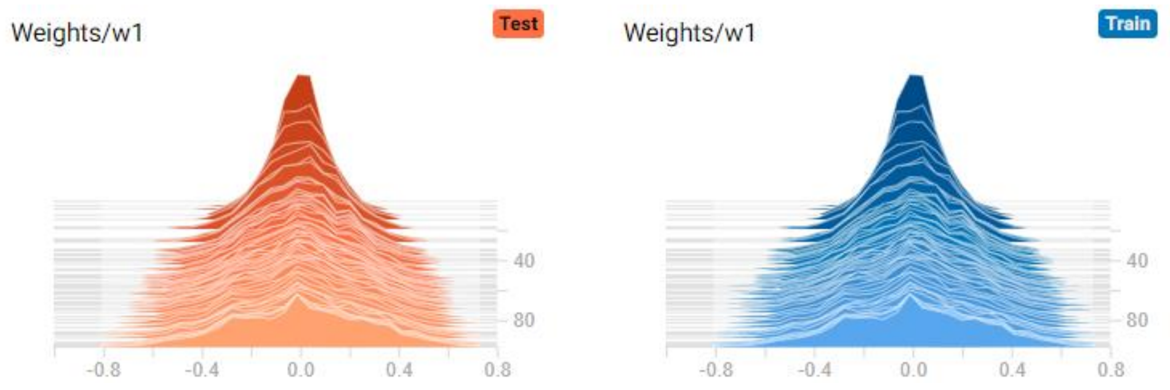


Accuracy Histogram:

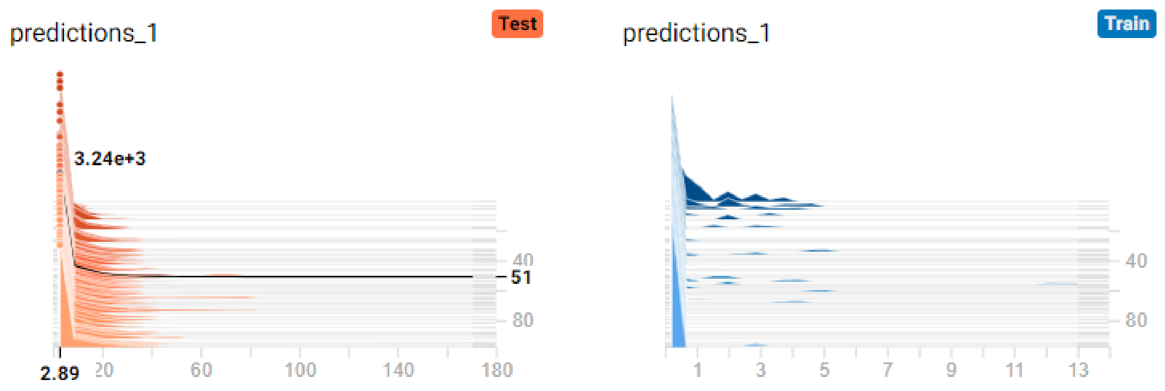




Weight Histogram:



Prediction Histogram:



Exercise 2 with Batch Normalization, selu and dropout:

The following are the parameters set, variables, weights and bias initialised for the networks to train the CIFAR-CNN1 model with selu activation function.

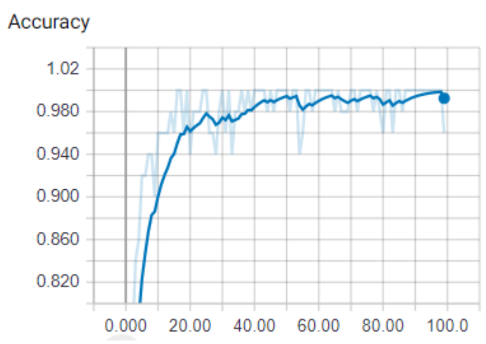
```
x = tf.placeholder(tf.float32, shape = (None,32,32,3))
y = tf.placeholder(tf.float32,(None,n_class))

with tf.variable_scope("Weights", reuse=tf.AUTO_REUSE):
    w_conv1 = tf.get_variable('w_c1',shape=(3,3,3,64),
                        initializer=tf.contrib.layers.xavier_initializer(uniform=False))
    w_conv2 = tf.get_variable('w_c2',shape=(3,3,64,128),
                        initializer=tf.contrib.layers.xavier_initializer(uniform=False)

    tf.summary.histogram('w1',w_conv1)
    tf.summary.histogram('w2',w_conv2)

with tf.variable_scope("Bias", reuse=tf.AUTO_REUSE):
    b_conv1 = tf.get_variable('B_c1',shape=(64),
                          initializer=tf.contrib.layers.xavier_initializer(uniform=False))
    b_conv2 = tf.get_variable('B_c2',shape=(128),
                          initializer=tf.contrib.layers.xavier_initializer(uniform=False))
```

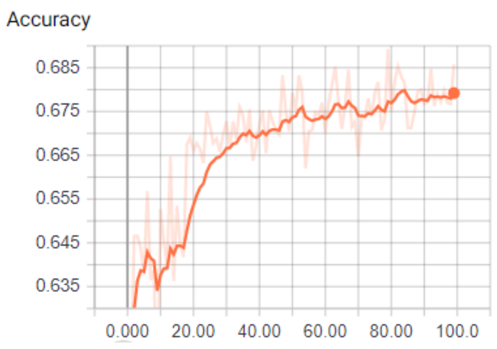
Function name : CIFAR_CNN1
Parameter : inputs

The function is used to perform the training of the model by calling the required convolutions, pooling and fully connected layers with selu activations, batch normalization and dropout regularization.

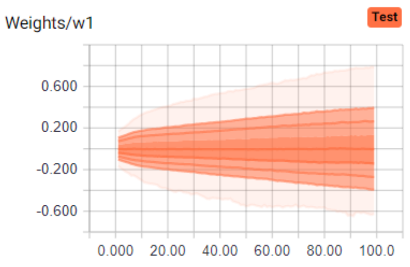
```
def CIFAR_CNN1(x):

  conv1 = conv2d(x,w_conv1,b_conv1)
  conv1_pool = maxpool2d(conv1)
  conv1_norm = batch_normalize(conv1_pool)

  conv2 = conv2d(conv1_norm,w_conv2,b_conv2)
  conv2_pool = maxpool2d(conv2)
  conv2_norm = batch_normalize(conv2_pool)

  flat = flatten(conv2_norm)

  fc1 = fc_layer(flat,128)
  fc1 = dropout(fc1,1.0)

  fc2 = fc_layer(fc1,256)
  fc2 = dropout(fc2,1.0)

  out = out_layer(fc2,10)

  return out
```

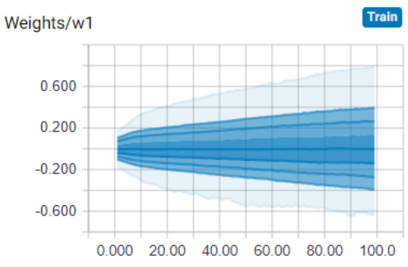
The maximum training accuracy achieved was 99% and the testing accuracy was noted to be 68%. The accuracy of training and testing are recorded and plotted respectively as shown below.

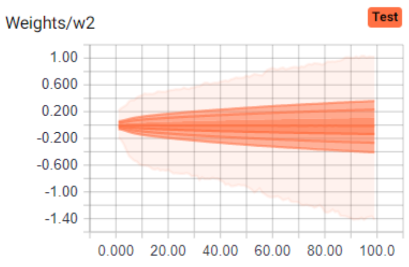




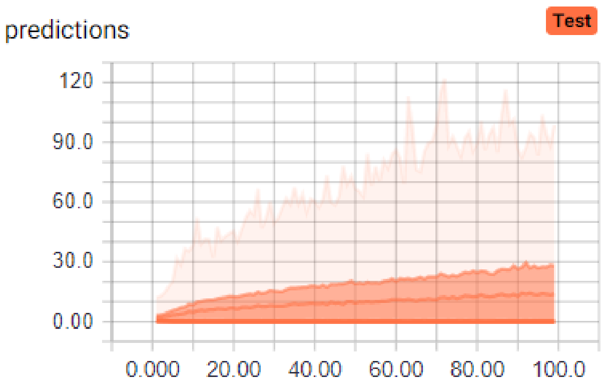
Weight Distribution:

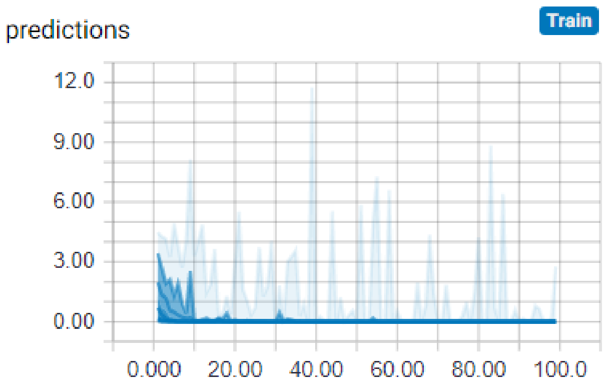




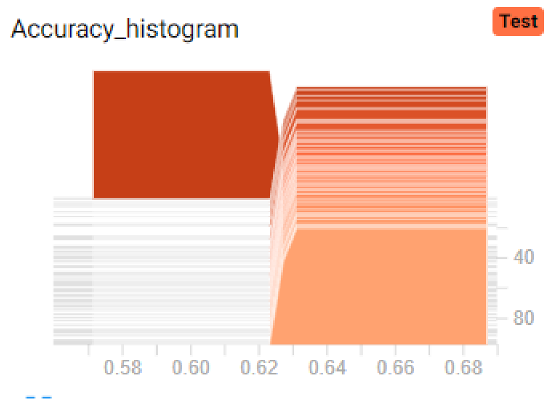


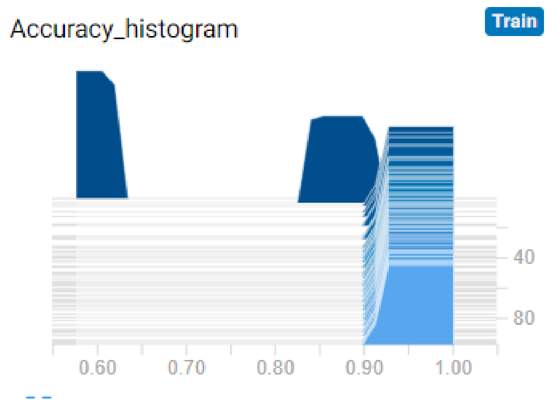
Prediction Distribution:



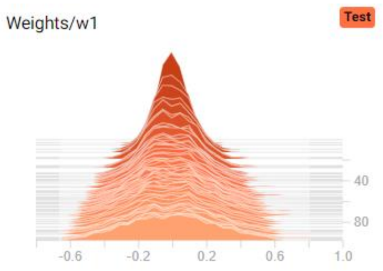


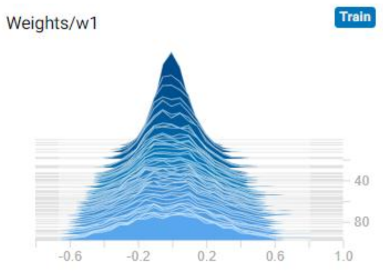
Accuracy Histogram:

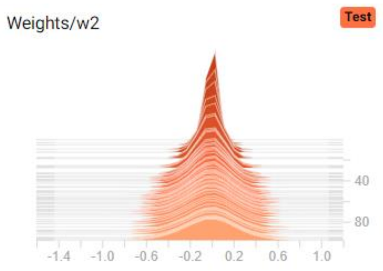




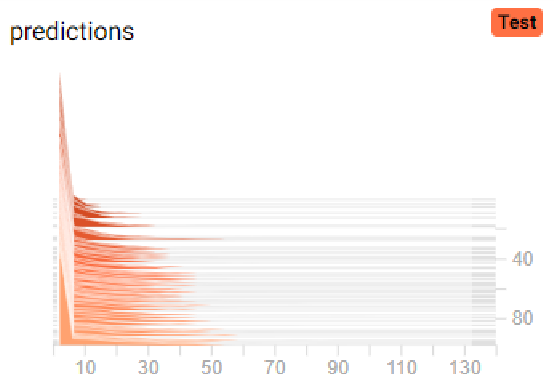
Weight Histogram:

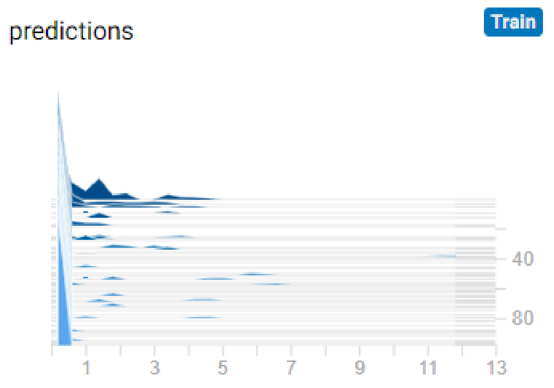






Prediction Histogram:





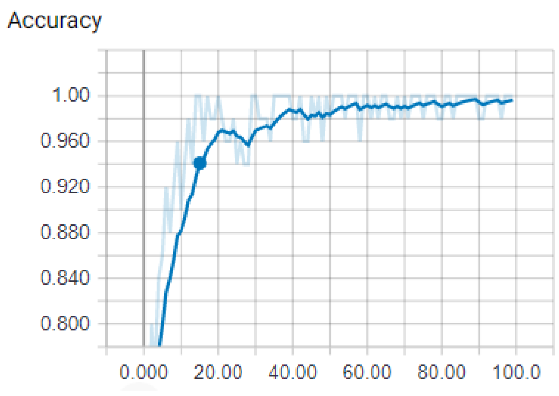
Exercise 2 with Batch Normalization and selu:

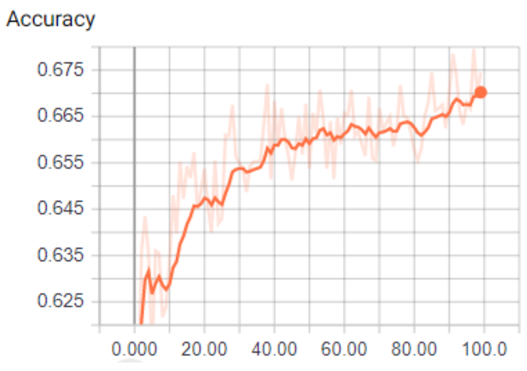
Function name : CIFAR_CNN2
Parameter : inputs

The function is used to perform the training of the model by calling the required convolutions, pooling and fully connected layers with selu activations and batch normalization.

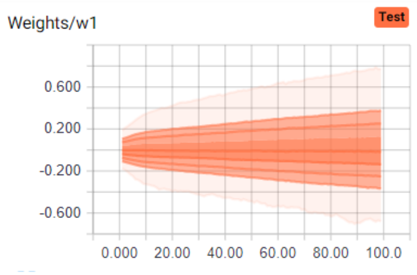
```
def CIFAR_CNN2(x):

  conv1 = conv2d(x,w_conv1,b_conv1)
  conv1_pool = maxpool2d(conv1)
  conv1_norm = batch_normalize(conv1_pool)

  conv2 = conv2d(conv1_norm,w_conv2,b_conv2)
  conv2_pool = maxpool2d(conv2)
  conv2_norm = batch_normalize(conv2_pool)

  flat = flatten(conv2_norm)

  fc1 = fc_layer(flat,128)

  fc2 = fc_layer(fc1,256)

  out = out_layer(fc2,10)

  return out
```

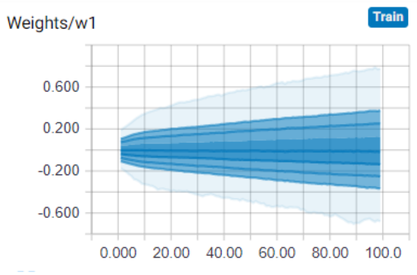
The maximum training accuracy achieved was 99% and the testing accuracy was noted to be 67%. The accuracy of training and testing are recorded and plotted respectively as shown below.

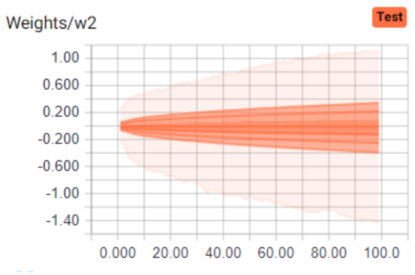




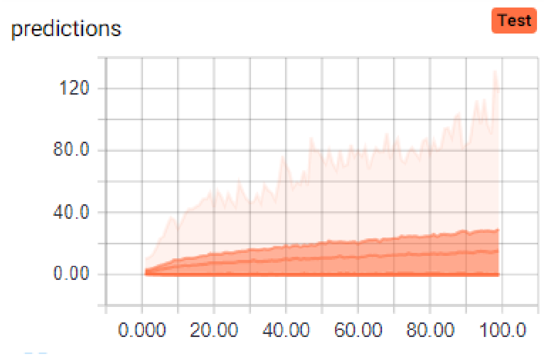
Weight Distribution:

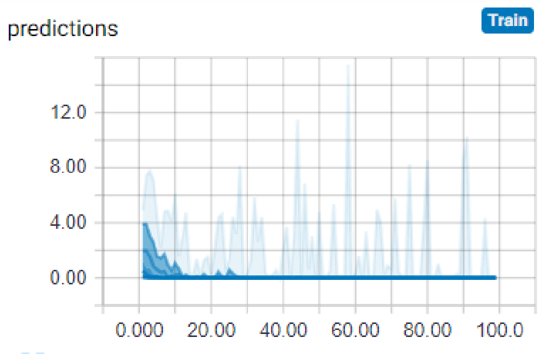




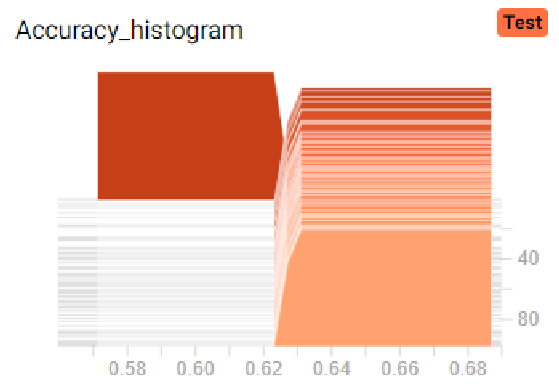


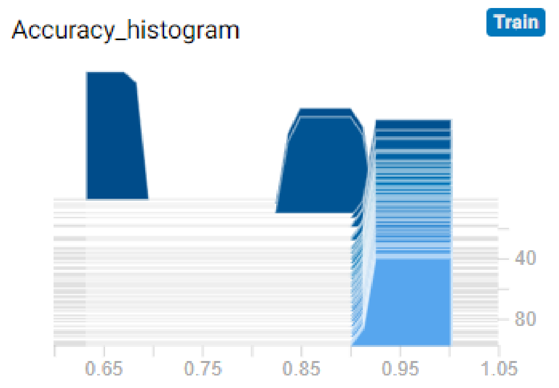
Prediction Distribution:



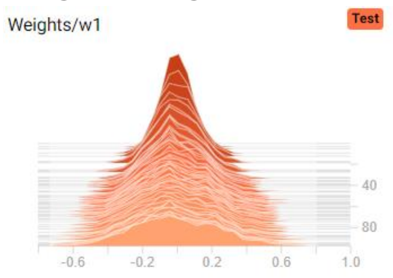


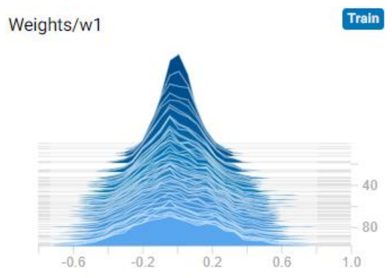
Accuracy Histogram:

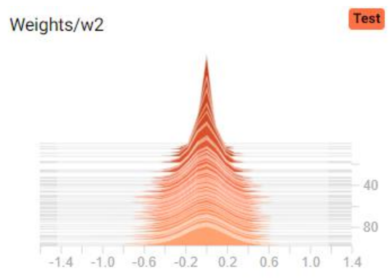




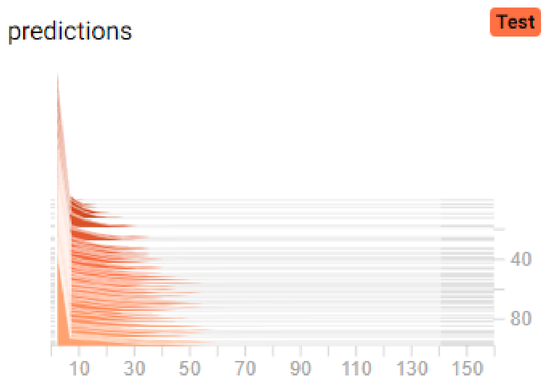
Weight Histogram:

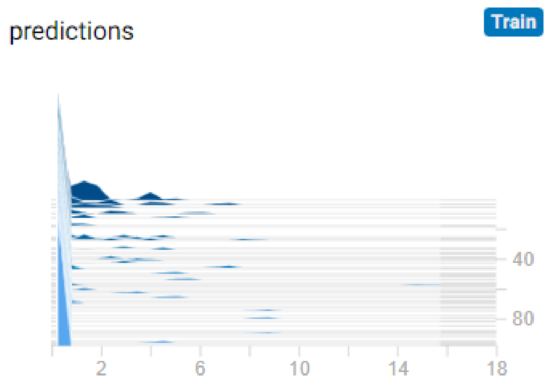






Prediction Histogram:


predictions
Test


predictions
Train