In [13]:
```
# QUESTION 1 - IMPLEMENTATION OF LeNeT Model using Non-Linear Activation
```

In [1]:
```
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
import os
from tensorflow.contrib.layers import flatten
from sklearn.utils import shuffle
import numpy as np
from sklearn.model_selection import train_test_split

base_path = os.path.abspath('')
dataset_path = os.path.join(base_path, 'PyCharmProjects/LeNet5_CNN/101_O
batch_size = 1024
n_epochs = 60
batches = 400
keep_prob = tf.placeholder(tf.float32)


def read_dataset(dataset_path):
    images = []
    labels = []
    label = 0
    ##Python3
    for paths in os.walk(dataset_path).__next__()[1]:
        dir = os.path.join(dataset_path, paths)
        walk = os.walk(dir).__next__()  ##Python3
        for image in walk[2]:
            if image.endswith('.jpg'):
                images.append(os.path.join(dir, image))
                labels.append(label)
        label += 1

    imgs = []
    for img in images:
        imgs.append(cv2.imread(img, cv2.COLOR_BGR2RGB))
    return imgs, labels


imgs, labels = read_dataset(dataset_path)
xx = tf.placeholder(tf.float32, (None, 64, 64, 3))
yy = tf.placeholder(tf.int32, None)
one_hot_Y = tf.one_hot(yy, 101)


def LeNet(x, keep_prob):
    # Conv Layer 1
    weight1 = tf.get_variable("weight0", initializer=tf.truncated_normal
```

```
    bias1 = tf.get_variable(name="b_b1", shape=[32],
                            initializer=tf.random_normal_initializer(std
    conv_layer1 = tf.nn.conv2d(x, weight1, strides=[1, 1, 1, 1], padding
    conv_layer1 = tf.nn.relu(conv_layer1)
    pooling1 = tf.nn.max_pool(conv_layer1, ksize=[1, 4, 4, 1], strides=[

    # Conv layer 2
    weight2 = tf.Variable(tf.truncated_normal(shape=[5, 5, 32, 64], stdd
    bias2 = tf.get_variable(name="b_b2", shape=[64],
                            initializer=tf.random_normal_initializer(std
    conv_layer2 = tf.nn.conv2d(pooling1, weight2, strides=[1, 1, 1, 1],
    conv_layer2 = tf.nn.relu(conv_layer2)
    pooling2 = tf.nn.max_pool(conv_layer2, ksize=[1, 4, 4, 1], strides=[

    # Flatten layer
    flatten_layer = flatten(pooling2)

    # FC layer 1
    fc1_weight1 = tf.get_variable("fc1_weight_1_", initializer=tf.trunca
                                  regularizer=tf.contrib.layers.l2_regul
    fc1_bias = tf.get_variable(name="fc_01_bias_", shape=[1024],
                               initializer=tf.random_normal_initializer(
    fc1 = tf.matmul(flatten_layer, fc1_weight1) + fc1_bias
    fc1 = tf.nn.relu(fc1)
    # applied DropOut
    # drop_out_1 = tf.nn.dropout(fc1, keep_prob)

    # FC Layer 2
    fc2_weight2 = tf.get_variable("fc_02_weight2_", initializer=tf.trunc
                                  regularizer=tf.contrib.layers.l2_regul
    fc2_bias = tf.get_variable(name="fc_02_bias_", shape=[84],
                               initializer=tf.random_normal_initializer(
    fc2 = tf.matmul(fc1, fc2_weight2) + fc2_bias
    fc2 = tf.nn.relu(fc2)

    # apply DropOut to hidden layer
    #  drop_out_2 = tf.nn.dropout(fc2, keep_prob)  # DROP-OUT here

    # FC Layer 3
    fc3_weight3 = tf.Variable(tf.truncated_normal(shape=(84, 101), stdde
    fc3_bias = tf.get_variable(name="fc_03_bias_", shape=[101],
                               initializer=tf.random_normal_initializer(
    logits = tf.matmul(fc2, fc3_weight3) + fc3_bias

    return logits



###Training
with tf.variable_scope(tf.get_variable_scope(), reuse=tf.AUTO_REUSE):
```

```
    logits = LeNet(xx, keep_prob)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits
total_loss = tf.reduce_mean(cross_entropy) + tf.losses.get_regularizatio
LR = tf.train.AdamOptimizer(learning_rate=0.001).minimize(total_loss)

##Model Evaluation

pred = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_Y, 1))
accu = tf.reduce_mean(tf.cast(pred, tf.float32))
saver = tf.train.Saver()


def visualize_weights(weights):

    plot_weight = np.moveaxis(weights, -1, 0)
    _, axs = plt.subplots(4, 8, figsize=(12, 12))
    axs = axs.flatten()
    for out, ax in zip(plot_weight, axs):
        v_min = plot_weight.min(axis=(0, 1), keepdims=True)
        v_max = plot_weight.max(axis=(0, 1), keepdims=True)
        out = (plot_weight - v_min) / (v_max - v_min)
        ax.imshow(plot_weight)
    plt.show()


def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0
    sess = tf.get_default_session()
    for offset in range(0, num_examples, batches):
        batch_x, batch_y = X_data[offset:offset + batches], y_data[offse
        accuracy = sess.run(accu, feed_dict={xx: batch_x, yy: batch_y, k
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples


np_read_img = np.array(imgs)
##Normalize the numpy data
np_read_img = (np_read_img - np_read_img.min()) / (np.ptp(np_read_img))
np_labels = np.asarray(labels)

X_train, X_val, y_train, y_val = train_test_split(np_read_img, np_labels
X_train, y_train = shuffle(X_train, y_train)

losses = []
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    total_data = len(X_train)
    print("Training started...")
    for i in range(n_epochs):
        print("EPOCH {} ...".format(i + 1))
```

```
        for offset in range(0, total_data, batches):
            end = offset + batches
            batch_x, batch_y = X_train[offset:end], y_train[offset:end]
            _, loss, acc = sess.run([LR, total_loss, accu], feed_dict={x
            print("Iter " + str(i) + ", Loss= " + \
                  "{:.6f}".format(loss) + ", Training Accuracy= " + \
                  "{:.5f}".format(acc))

        validation_accuracy = evaluate(X_val, y_val)

        print("Validation Accuracy = {:.3f}".format(validation_accuracy)
        saver.save(sess, 'my-Lenet-model')
        print("Model Saved")
```

```
Iter 58, Loss= 1.225723, Training Accuracy= 0.83750
Iter 58, Loss= 1.355819, Training Accuracy= 0.78250
Iter 58, Loss= 1.241440, Training Accuracy= 0.82500
Iter 58, Loss= 0.909884, Training Accuracy= 0.92344
Validation Accuracy = 0.571
Model Saved
EPOCH 60 ...
Iter 59, Loss= 1.346686, Training Accuracy= 0.78500
Iter 59, Loss= 1.316886, Training Accuracy= 0.79500
Iter 59, Loss= 1.401469, Training Accuracy= 0.77500
Iter 59, Loss= 1.370118, Training Accuracy= 0.77500
Iter 59, Loss= 1.276228, Training Accuracy= 0.80750
Iter 59, Loss= 1.309770, Training Accuracy= 0.79250
Iter 59, Loss= 1.266339, Training Accuracy= 0.83000
Iter 59, Loss= 1.200205, Training Accuracy= 0.82750
Iter 59, Loss= 1.217261, Training Accuracy= 0.82500
Iter 59, Loss= 1.363045, Training Accuracy= 0.78500
Iter 59, Loss= 1.380965, Training Accuracy= 0.78500
Iter 59, Loss= 1.349538, Training Accuracy= 0.80750
Iter 59, Loss= 1.245702, Training Accuracy= 0.81500
```

In [ ]:
```
#QUESTION 2.1 - Implementation of LeNet using LINEAR Activation Function
```

In [3]:
```
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
import os
from tensorflow.contrib.layers import flatten
from sklearn.utils import shuffle
import numpy as np
from sklearn.model_selection import train_test_split

base_path = os.path.abspath('')
dataset_path = os.path.join(base_path, 'PyCharmProjects/LeNet5_CNN/101_O
batch_size = 1024
n_epochs = 60
batches = 400
```

```
batches = 400
keep_prob = tf.placeholder(tf.float32)


def read_dataset(dataset_path):
    images = []
    labels = []
    label = 0
    ##Python3
    for paths in os.walk(dataset_path).__next__()[1]:
        dir = os.path.join(dataset_path, paths)
        walk = os.walk(dir).__next__()  ##Python3
        for image in walk[2]:
            if image.endswith('.jpg'):
                images.append(os.path.join(dir, image))
                labels.append(label)
        label += 1

    imgs = []
    for img in images:
        imgs.append(cv2.imread(img, cv2.COLOR_BGR2RGB))
    return imgs, labels


imgs, labels = read_dataset(dataset_path)
xx = tf.placeholder(tf.float32, (None, 64, 64, 3))
yy = tf.placeholder(tf.int32, None)
one_hot_Y = tf.one_hot(yy, 101)


def LeNet(x, keep_prob):
    # Conv Layer 1
    weight1 = tf.Variable(tf.truncated_normal(shape=[5, 5, 3, 32], stdde
    bias1 = tf.get_variable(name="bb1", shape=[32],
                            initializer=tf.random_normal_initializer(std
    conv_layer1 = tf.nn.conv2d(x, weight1, strides=[1, 1, 1, 1], padding
    # conv_layer1 = tf.nn.relu(conv_layer1)
    pooling1 = tf.nn.max_pool(conv_layer1, ksize=[1, 4, 4, 1], strides=[

    # Conv layer 2
    weight2 = tf.Variable(tf.truncated_normal(shape=[5, 5, 32, 64], stdd
    bias2 = tf.get_variable(name="bb2", shape=[64],
                            initializer=tf.random_normal_initializer(std
    conv_layer2 = tf.nn.conv2d(pooling1, weight2, strides=[1, 1, 1, 1],
    # conv_layer2 = tf.nn.relu(conv_layer2)
    pooling2 = tf.nn.max_pool(conv_layer2, ksize=[1, 4, 4, 1], strides=[

    # Flatten layer
    flatten_layer = flatten(pooling2)

    # FC layer 1
```

```
    fc1_weight1 = tf.get_variable("fc1_weight_1_", initializer=tf.trunca
    # regularizer=tf.contrib.layers.l2_regularizer(scale=0.1))
    fc1_bias = tf.get_variable(name="fc_01_bias_", shape=[1024],
                               initializer=tf.random_normal_initializer(
    fc1 = tf.matmul(flatten_layer, fc1_weight1) + fc1_bias
    # fc1 = tf.nn.relu(fc1)
    # applied DropOut
    # drop_out_1 = tf.nn.dropout(fc1, keep_prob)

    # FC Layer 2
    fc2_weight2 = tf.get_variable("fc_02_weight2_", initializer=tf.trunc
    # regularizer=tf.contrib.layers.l2_regularizer(scale=0.1))
    fc2_bias = tf.get_variable(name="fc_02_bias_", shape=[84],
                               initializer=tf.random_normal_initializer(
    fc2 = tf.matmul(fc1, fc2_weight2) + fc2_bias
    # fc2 = tf.nn.relu(fc2)

    # apply DropOut to hidden layer
    #  drop_out_2 = tf.nn.dropout(fc2, keep_prob)  # DROP-OUT here

    # FC Layer 3
    fc3_weight3 = tf.Variable(tf.truncated_normal(shape=(84, 101), stdde
    fc3_bias = tf.get_variable(name="fc_03_bias_", shape=[101],
                               initializer=tf.random_normal_initializer(
    logits = tf.matmul(fc2, fc3_weight3) + fc3_bias

    return logits



###Training
with tf.variable_scope(tf.get_variable_scope(), reuse=tf.AUTO_REUSE):
    logits = LeNet(xx, keep_prob)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits
total_loss = tf.reduce_mean(cross_entropy) + tf.losses.get_regularizatio
LR = tf.train.AdamOptimizer(learning_rate=0.001).minimize(total_loss)

##Model Evaluation

pred = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_Y, 1))
accu = tf.reduce_mean(tf.cast(pred, tf.float32))
saver = tf.train.Saver()



def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0
    sess = tf.get_default_session()
    for offset in range(0, num_examples, batches):
```

```
        batch_x, batch_y = X_data[offset:offset + batches], y_data[offse
        accuracy = sess.run(accu, feed_dict={xx: batch_x, yy: batch_y, k
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples


np_read_img = np.array(imgs)
##Normalize the numpy data
np_read_img = (np_read_img - np_read_img.min()) / (np.ptp(np_read_img))
np_labels = np.asarray(labels)

X_train, X_val, y_train, y_val = train_test_split(np_read_img, np_labels
X_train, y_train = shuffle(X_train, y_train)

losses = []
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    total_data = len(X_train)
    print("Training started...")
    for i in range(n_epochs):
        print("EPOCH {} ...".format(i + 1))
        for offset in range(0, total_data, batches):
            end = offset + batches
            batch_x, batch_y = X_train[offset:end], y_train[offset:end]
            _, loss, acc = sess.run([LR, total_loss, accu], feed_dict={x
            print("Iter " + str(i) + ", Loss= " + \
                  "{:.6f}".format(loss) + ", Training Accuracy= " + \
                  "{:.5f}".format(acc))

        validation_accuracy = evaluate(X_val, y_val)

        print("Validation Accuracy = {:.3f}".format(validation_accuracy)
        print()

    print("Completed")
```

```
Iter 59, Loss= 1.000389, Training Accuracy= 0.90250
Iter 59, Loss= 1.108266, Training Accuracy= 0.88250
Iter 59, Loss= 1.076686, Training Accuracy= 0.88250
Iter 59, Loss= 1.084220, Training Accuracy= 0.91250
Iter 59, Loss= 1.075355, Training Accuracy= 0.89250
Iter 59, Loss= 1.051593, Training Accuracy= 0.90000
Iter 59, Loss= 1.008677, Training Accuracy= 0.90250
Iter 59, Loss= 0.986395, Training Accuracy= 0.92250
Iter 59, Loss= 1.066077, Training Accuracy= 0.88750
Iter 59, Loss= 1.004974, Training Accuracy= 0.89500
Iter 59, Loss= 1.022638, Training Accuracy= 0.91000
Iter 59, Loss= 1.142298, Training Accuracy= 0.86500
Iter 59, Loss= 0.928250, Training Accuracy= 0.95000
Iter 59, Loss= 1.015103, Training Accuracy= 0.93750
```

```
Iter 59, Loss= 0.989270, Training Accuracy= 0.92500
Iter 59, Loss= 0.810233, Training Accuracy= 0.97129
Validation Accuracy = 0.606

Completed
```

In [ ]:
```
#  QUESTION2.1
# What happens if we use a linear activation function in all convolution
# Compare training and validation loss (cross-entropy) after and before

# ANSWER
# I have removed all the relu activation function from all convolutional
# This is now a straight-forward linear activation model. Using the line
# network might result in loosing the classification capability.

# From my observation of the results I achieved, using a linear activati
# provided a minimal decrease in the overall performance (i.e) it me a a
# accuracy than the non linear activation function.

# This however would be due to the addition of regularization and other
# However in this section, we received a a better Training accuracy with
# compared to section 1.

# The validation loss is slightly lower while we are using this linear f
# I could achieve around 60% validation accuracy while running the code

# In order to improve the accuracy I have tried to increase the number o
# performance of the model.
```

In [ ]:
```
##### QUESTION 2.2 - IMPLEMENTATION OF ADDING ADDITIONAL LAYERS TO THE M
```

In [20]:
```
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
import os
from tensorflow.contrib.layers import flatten
from sklearn.utils import shuffle
import numpy as np
from sklearn.model_selection import train_test_split

base_path = os.path.abspath('')
dataset_path = os.path.join(base_path, 'PyCharmProjects/LeNet5_CNN/101_O
batch_size = 1024
n_epochs = 60
batches = 400
keep_prob = tf.placeholder(tf.float32)


def read_dataset(dataset_path):
```

```
    images = []
    labels = []
    label = 0
    ##Python3
    for paths in os.walk(dataset_path).__next__()[1]:
        dir = os.path.join(dataset_path, paths)
        walk = os.walk(dir).__next__()  ##Python3
        for image in walk[2]:
            if image.endswith('.jpg'):
                images.append(os.path.join(dir, image))
                labels.append(label)
        label += 1

    imgs = []
    for img in images:
        imgs.append(cv2.imread(img, cv2.COLOR_BGR2RGB))
    return imgs, labels


imgs, labels = read_dataset(dataset_path)
xx = tf.placeholder(tf.float32, (None, 64, 64, 3))
yy = tf.placeholder(tf.int32, None)
one_hot_Y = tf.one_hot(yy, 101)


def LeNet(x, keep_prob):
    # Conv Layer 1
    weight1 = tf.Variable(tf.truncated_normal(shape=[5, 5, 3, 32], stdde
    bias1 = tf.get_variable(name="bias_1_", shape=[32],
                            initializer=tf.random_normal_initializer(std
    conv_layer1 = tf.nn.conv2d(x, weight1, strides=[1, 1, 1, 1], padding
    # conv_layer1 = tf.nn.relu(conv_layer1)
    pooling1 = tf.nn.max_pool(conv_layer1, ksize=[1, 4, 4, 1], strides=[

    # Conv layer 2
    weight2 = tf.Variable(tf.truncated_normal(shape=[5, 5, 32, 48], stdd
    bias2 = tf.get_variable(name="bias_2", shape=[48],
                            initializer=tf.random_normal_initializer(std
    conv_layer2 = tf.nn.conv2d(pooling1, weight2, strides=[1, 1, 1, 1],
    # conv_layer2 = tf.nn.relu(conv_layer2)
    pooling2 = tf.nn.max_pool(conv_layer2, ksize=[1, 1, 1, 1], strides=[

    # Conv layer 3
    weight3 = tf.Variable(tf.truncated_normal(shape=[5, 5, 48, 64], stdd
    bias3 = tf.get_variable(name="bias_3", shape=[64],
                            initializer=tf.random_normal_initializer(std
    conv_layer3 = tf.nn.conv2d(pooling2, weight3, strides=[1, 1, 1, 1],
    # conv_layer3 = tf.nn.relu(conv_layer3)
    pooling3 = tf.nn.max_pool(conv_layer3, ksize=[1, 3, 3, 1], strides=[

    # Flatten
```

```
    # Flatten layer
    flatten_layer = flatten(pooling3)

    # FC layer 1
    fc1_weight1 = tf.get_variable("fc1_weight_1", initializer=tf.truncat
    # regularizer=tf.contrib.layers.l2_regularizer(scale=0.1))
    fc1_bias = tf.get_variable(name="fc1_bias_01", shape=[1024],
                               initializer=tf.random_normal_initializer(
    fc1 = tf.matmul(flatten_layer, fc1_weight1) + fc1_bias
    # fc1 = tf.nn.relu(fc1)
    # applied DropOut
    # drop_out_1 = tf.nn.dropout(fc1, keep_prob)

    # FC Layer 2
    fc2_weight2 = tf.get_variable("fc2_weight_2", initializer=tf.truncat
    # regularizer=tf.contrib.layers.l2_regularizer(scale=0.1))
    fc2_bias = tf.get_variable(name="fc2_bias_02", shape=[84],
                               initializer=tf.random_normal_initializer(
    fc2 = tf.matmul(fc1, fc2_weight2) + fc2_bias
    # fc2 = tf.nn.relu(fc2)

    # apply DropOut to hidden layer
    #  drop_out_2 = tf.nn.dropout(fc2, keep_prob)  # DROP-OUT here

    # FC Layer 3
    fc3_weight3 = tf.Variable(tf.truncated_normal(shape=(84, 101), stdde
    fc3_bias = tf.get_variable(name="fc3_bias_03", shape=[101],
                               initializer=tf.random_normal_initializer(
    logits = tf.matmul(fc2, fc3_weight3) + fc3_bias

    return logits



###Training
with tf.variable_scope(tf.get_variable_scope(), reuse=tf.AUTO_REUSE):
    logits = LeNet(xx, keep_prob)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits
total_loss = tf.reduce_mean(cross_entropy) + tf.losses.get_regularizatio
LR = tf.train.AdamOptimizer(learning_rate=0.001).minimize(total_loss)

##Model Evaluation

pred = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_Y, 1))
accu = tf.reduce_mean(tf.cast(pred, tf.float32))
saver = tf.train.Saver()



def evaluate(X_data, y_data):
    num_examples = len(X_data)
```

```
    total_accuracy = 0
    sess = tf.get_default_session()
    for offset in range(0, num_examples, batches):
        batch_x, batch_y = X_data[offset:offset + batches], y_data[offse
        accuracy = sess.run(accu, feed_dict={xx: batch_x, yy: batch_y, k
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples


np_read_img = np.array(imgs)
##Normalize the numpy data
np_read_img = (np_read_img - np_read_img.min()) / (np.ptp(np_read_img))
np_labels = np.asarray(labels)

X_train, X_val, y_train, y_val = train_test_split(np_read_img, np_labels
X_train, y_train = shuffle(X_train, y_train)

losses = []
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    total_data = len(X_train)
    print("Training started...")
    for i in range(n_epochs):
        print("EPOCH {} ...".format(i + 1))
        for offset in range(0, total_data, batches):
            end = offset + batches
            batch_x, batch_y = X_train[offset:end], y_train[offset:end]
            _, loss, acc = sess.run([LR, total_loss, accu], feed_dict={x
            print("Iter " + str(i) + ", Loss= " + \
                  "{:.6f}".format(loss) + ", Training Accuracy= " + \
                  "{:.5f}".format(acc))

        validation_accuracy = evaluate(X_val, y_val)

        print("Validation Accuracy = {:.3f}".format(validation_accuracy)
        print()

    print("Completed")
```

```
Iter 59, Loss= 0.009592, Training Accuracy= 1.00000
Iter 59, Loss= 0.007784, Training Accuracy= 1.00000
Iter 59, Loss= 0.008497, Training Accuracy= 1.00000
Iter 59, Loss= 0.017825, Training Accuracy= 0.99500
Iter 59, Loss= 0.007193, Training Accuracy= 1.00000
Iter 59, Loss= 0.014982, Training Accuracy= 0.99750
Iter 59, Loss= 0.016547, Training Accuracy= 0.99750
Iter 59, Loss= 0.009378, Training Accuracy= 1.00000
Iter 59, Loss= 0.009305, Training Accuracy= 1.00000
Iter 59, Loss= 0.009577, Training Accuracy= 1.00000
```

```
Iter 59, Loss= 0.008480, Training Accuracy= 1.00000
Iter 59, Loss= 0.008509, Training Accuracy= 1.00000
Iter 59, Loss= 0.008051, Training Accuracy= 1.00000
Iter 59, Loss= 0.007974, Training Accuracy= 1.00000
Iter 59, Loss= 0.007052, Training Accuracy= 1.00000
Iter 59, Loss= 0.002890, Training Accuracy= 1.00000
Validation Accuracy = 0.578

Completed
```

In [ ]:
```
#Question 2.2: Can we compensate for the effect of removing the non-line
#linear layers?

# Answer: In this section the number of hidden convolutional layers are
# This increases the depth of the model and gives enough strength to the
# From the observation of the results, the validation loss is much lesse
# However the Validation accuracy seems to be lesser now compared to the
# than the previous.

# Thus adding the number of layers proved to improved at the initial sta
# as the above. So we cannot assure that adding the new layer would comp
# activation function.

# So in my case, adding the layers instead of removing the non-linear ac
# we avoid overfitting and resolve other issues.

# In order to get a much better performance in accuracy, we can try reso
# and also try to fine tune the hyper parameters to achieve better impro
```

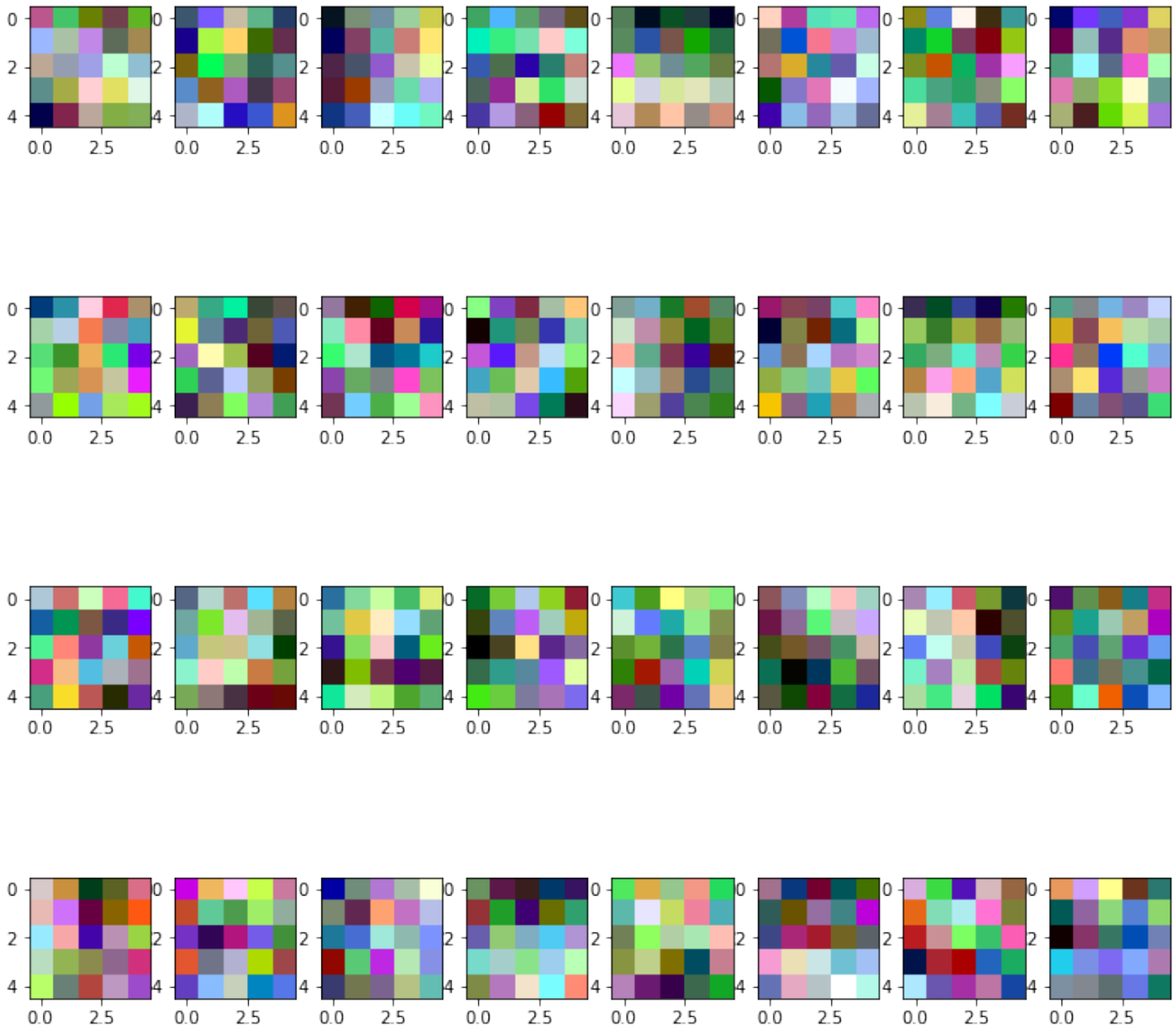
In [ ]:
```
##Assignment_2_q_3. Visualizing the weights
```

In [4]:
```
def visualize_weights(weights):
    plot_weight = np.moveaxis(weights, -1, 0)
    _, axs = plt.subplots(4, 8, figsize=(12, 12))
    axs = axs.flatten()
    for out, ax in zip(plot_weight, axs):
        v_min = out.min(axis=(0, 1), keepdims=True)
        v_max = out.max(axis=(0, 1), keepdims=True)
        out = (out - v_min) / (v_max - v_min)
        ax.imshow(out)
    print("visualized")
    plt.show()

sess = tf.Session()
new_saver = tf.train.import_meta_graph('my-Lenet-model.meta')
new_saver.restore(sess, tf.train.latest_checkpoint('./'))
all_vars = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES)
var_name = 'weight0'
graph = tf.get_default_graph()
```

```
w2 = graph.get_tensor_by_name(var_name + ":0")
w2_saved = sess.run(w2)  # print out tensor
visualize_weights(w2_saved)
print("Completed")
```

```
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-packages/ten
sorflow/python/training/saver.py:1266: checkpoint_exists (from tensorf
low.python.training.checkpoint_management) is deprecated and will be r
emoved in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from ./my-Lenet-model
visualized
```



```
Completed
```

In [ ]: