Image Rating Network using Transfer Learning

Created by Ramsey Boyce
Spring 2019

This notebook builds and trains a neural network that assigns aesthetic quality scores to images. It is trained on images from the AVA (Aesthetic Visual Analysis) dataset (http://refbase.cvc.uab.es/files/MMP2012a.pdf), available here (http://academictorrents.com/details/71631f83b11d3d79d8f84efe0a7e12f0ac001460).

It first uses the VGG19 (https://github.com/machrisaa/tensorflow-vgg) network to convert images from the AVA dataset into codes, then runs the codes through a fully-connected network with two hidden layers, to return image ratings. The rating for an image is in the form of a probability distribution over the integers from 1 (lowest quality) to 10 (highest quality).

You will need the AVA dataset on your local computer to run this notebook. Note the hardcoded location ("E:/AVA_dataset") below.

Run AVA images through VGG19 network

In [1]:
```
import os

import numpy as np
import tensorflow as tf

from tensorflow_vgg import vgg19
from tensorflow_vgg import utils
```

In [2]:
```
all_labels = np.genfromtxt("E:/AVA_dataset/AVA.txt", delimiter = " ", dtype=np
.int32)
all_labels = all_labels[:,1:]
all_labels = all_labels[all_labels[:,0].argsort()]
print(all_labels)
print(all_labels.shape)
print(all_labels.shape[0])
```

```
[[    53      1      3 ...      0      0      2]
 [    54      9     13 ...      0      0      2]
 [    66      7      7 ...      0      0      2]
 ...
 [958285      2      3 ...      4     50   1408]
 [958296      1      7 ...      0      0   1408]
 [958297      0      2 ...     10     21   1408]]
(255530, 14)
255530
```

In [ ]:
```
data_dir = 'E:/AVA_dataset/images'
batch_size = 100
codes_list = []
labels = []
batch = []

codes = None

with tf.Session() as sess:

    # First build the VGG network
    vgg = vgg19.Vgg19()
    input_ = tf.placeholder(tf.float32, [None, 224, 224, 3])
    with tf.name_scope("content_vgg"):
        vgg.build(input_)

    for (ii, line) in enumerate(all_labels, 1):
        file = str(line[0]) + '.jpg'

        try:
            # Add images to the current batch
            # utils.load_image crops the input images for us, from the center
            img = utils.load_image(os.path.join(data_dir, file))
            if len(img.shape) == 2:
                # black and white image: duplicate pixel values into each of
 R, G, B planes
                img = np.concatenate((img, img, img))
            batch.append(img.reshape((1, 224, 224, 3)))
            labels.append(line)
        except FileNotFoundError:
            print("File {} not found, skipping".format(file))
        except ValueError:
            print("Got a value error, skipping")
        except TypeError:
            print("Got a type error, skipping")
        except NameError:
            print("Got a name error, skipping")
        except:
            print("Got a general exception, skipping")

        # Running the batch through the network to get the codes
        if ii % batch_size == 0 or ii == all_labels.shape[0]:

            # Image batch to pass to VGG network
            images = np.concatenate(batch)
            print(images.shape)

            # TODO: Get the values from the relu6 layer of the VGG network
            feed_dict = {input_: images}
            codes_batch = sess.run(vgg.relu6, feed_dict=feed_dict)

            # Here I'm building an array of the codes
            if codes is None:
                codes = codes_batch
            else:
                codes = np.concatenate((codes, codes_batch))
```

```
            # Reset to start building the next batch
            batch = []

        if ii % 1000 == 0:
            print('{} images processed'.format(ii))
```

In [ ]:
```
codes.shape, len(labels), len(batch)
```

In [5]:
```
# write codes to file
with open('AVA_codes', 'w') as f:
    codes.tofile(f)

# write labels to file
import csv
with open('AVA_labels', 'w') as f:
    writer = csv.writer(f, delimiter='\n')
    writer.writerow(labels)
```

Checkpoint

In [4]:
```
# read codes and labels from file
import os
import csv

import numpy as np
import tensorflow as tf

print('starting read of AVA_labels...')
labels = []
with open('AVA_labels') as f:
    reader = csv.reader(f, delimiter='\n')
    for line in reader:
        #line = line[0]
        if len(line) > 0:
            labels.append([int(i) for i in line[0][1:-1].split()])
labels = np.array(labels)

print('starting read of AVA_codes...')
with open('AVA_codes') as f:
    codes = np.fromfile(f, dtype=np.float32)
    codes = codes.reshape((labels.shape[0], -1))

print('done')
print('labels shape = {}'.format(labels.shape))
print('codes shape = {}'.format(codes.shape))
```

```
starting read of AVA_labels...
starting read of AVA_codes...
done
labels shape = (255482, 14)
codes shape = (255482, 4096)
```

In [5]:
```
labels_vecs = np.delete(np.delete(labels, 0, 1), np.s_[10:], 1)
labels_vecs = np.array([each / sum(each) for each in labels_vecs])
labels_vecs
```

Out[5]:
```
array([[0.01162791, 0.03488372, 0.03488372, ..., 0.23255814, 0.09302326,
        0.1744186 ],
       [0.10465116, 0.15116279, 0.1627907 , ..., 0.02325581, 0.08139535,
        0.03488372],
       [0.08235294, 0.08235294, 0.09411765, ..., 0.07058824, 0.02352941,
        0.        ],
       ...,
       [0.01680672, 0.02521008, 0.1512605 , ..., 0.00840336, 0.        ,
        0.        ],
       [0.00826446, 0.05785124, 0.05785124, ..., 0.05785124, 0.00826446,
        0.02479339],
       [0.        , 0.01639344, 0.01639344, ..., 0.04918033, 0.02459016,
        0.02459016]])
```

In [6]:
```
from sklearn.model_selection import ShuffleSplit

ss = ShuffleSplit(n_splits=1, test_size=0.2, random_state=0)

for train_idx, test_idx in ss.split(codes, labels_vecs):
    train_x, train_y = codes[train_idx], labels_vecs[train_idx]
    half = len(test_idx) // 2
    val_x, val_y = codes[test_idx[:half]], labels_vecs[test_idx[:half]]
    test_x, test_y = codes[test_idx[half:]], labels_vecs[test_idx[half:]]
```

In [7]:
```
print("Train shapes (x, y):", train_x.shape, train_y.shape)
print("Validation shapes (x, y):", val_x.shape, val_y.shape)
print("Test shapes (x, y):", test_x.shape, test_y.shape)
```

```
Train shapes (x, y): (204385, 4096) (204385, 10)
Validation shapes (x, y): (25548, 4096) (25548, 10)
Test shapes (x, y): (25549, 4096) (25549, 10)
```

Build network to rate images

In [8]:
```
import math

num_inputs = 4096 # train_x.shape[1]
num_outputs = 10 # labels_vecs.shape[1]
num_hidden = 1000
num_hidden2 = 1000
num_hidden3 = 1000

inputs_ = tf.placeholder(tf.float32, shape=[None, num_inputs])
labels_ = tf.placeholder(tf.float32, shape=[None, num_outputs])

# four layers in our network: three hidden layers with num_hidden units each,
 and an output layer
# with num_outputs (= 10) units corresponding to the rating

layer1_W = tf.Variable(tf.truncated_normal([num_inputs, num_hidden], stddev=ma
th.sqrt(2.0/num_inputs)))
layer1_bias = tf.Variable(tf.zeros([num_hidden]))
layer1 = tf.nn.leaky_relu(tf.add(tf.matmul(inputs_, layer1_W), layer1_bias), a
lpha=0.4)

layer2_W = tf.Variable(tf.truncated_normal([num_hidden, num_hidden2], stddev=m
ath.sqrt(2.0/num_hidden)))
layer2_bias = tf.Variable(tf.zeros([num_hidden2]))
layer2 = tf.nn.leaky_relu(tf.add(tf.matmul(layer1, layer2_W), layer2_bias), al
pha=0.4)

layer3_W = tf.Variable(tf.truncated_normal([num_hidden2, num_outputs], stddev=
math.sqrt(2.0/num_hidden2)))
layer3_bias = tf.Variable(tf.zeros([num_outputs]))
layer3 = tf.add(tf.matmul(layer2, layer3_W), layer3_bias)

logits = tf.identity(layer3, name='logits')

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits
, labels=labels_))

optimizer = tf.train.AdamOptimizer().minimize(cost)


# Operations for validation/test accuracy

predicted = tf.nn.softmax(logits)
scores = tf.transpose(tf.constant([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]], dtype=tf.
float32))
mean_score_predicted = tf.squeeze(tf.matmul(predicted, scores), axis=[1])
mean_score_actual = tf.squeeze(tf.matmul(labels_, scores), axis=[1])

accuracy = 1 - tf.reduce_mean(tf.abs(mean_score_predicted - mean_score_actual)
/ 10)
error = tf.abs(mean_score_predicted - mean_score_actual)


# correct_pred = tf.equal(tf.argmax(predicted, 1), tf.argmax(labels_, 1))
# accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
# accuracy = 1 - cost
```

In [9]:
```
def get_batches(x, y, n_batches=100):
    """ Return a generator that yields batches from arrays x and y. """
    batch_size = len(x)//n_batches

    for ii in range(0, n_batches*batch_size, batch_size):
        # If we're not on the last batch, grab data with size batch_size
        if ii != (n_batches-1)*batch_size:
            X, Y = x[ii: ii+batch_size], y[ii: ii+batch_size]
        # On the last batch, grab the rest of the data
        else:
            X, Y = x[ii:], y[ii:]
        # I love generators
        yield X, Y
```

Train network

In [10]:
```
epochs = 100

accuracy_sum = 0
acc_one_sum = 0
acc_half_sum = 0
total_sum = 0

saver = tf.train.Saver()
with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())

    val_cost, val_acc = sess.run((cost, accuracy), feed_dict={inputs_:val_x, l
abels_:val_y})
    print("Starting, val. cost = {:f}, val. accuracy = {:f}".format(val_cost,
val_acc))

    for epoch in range(epochs):
        for x, y in get_batches(train_x, train_y):
            feed = {inputs_:x, labels_:y}
            sess.run(optimizer, feed_dict=feed)

        val_cost, val_acc, val_error = sess.run((cost, accuracy, error), feed_
dict={inputs_:val_x, labels_:val_y})

        accuracy_sum += val_acc
        acc_one_sum += len([x for x in val_error if x < 1.0])
        acc_half_sum += len([x for x in val_error if x < 0.5])
        total_sum += len(val_error)

        print("After epoch {}/{}, val. cost = {:f}, val. accuracy = {:f}".form
at(epoch, epochs, val_cost, val_acc))

    saver.save(sess, "checkpoints/AAR_network.ckpt")

accuracy_sum /= epochs
acc_one_sum /= total_sum
acc_half_sum /= total_sum
print(f'accuracy avg = {accuracy_sum}, accuracy (within 1.0) = {acc_one_sum},
 accuracy (within 0.5) = {acc_half_sum}')
```

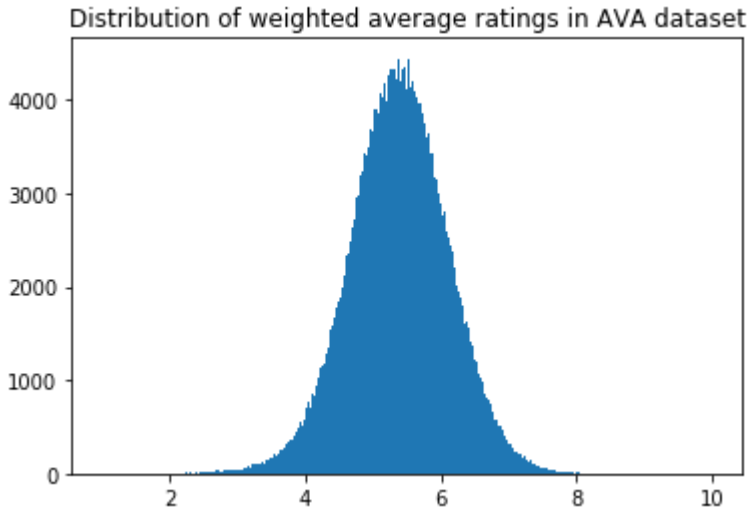
```
Starting, val. cost = 5.751604, val. accuracy = 0.868239
After epoch 0/100, val. cost = 1.989330, val. accuracy = 0.944911
After epoch 1/100, val. cost = 1.931620, val. accuracy = 0.947299
After epoch 2/100, val. cost = 1.893146, val. accuracy = 0.948880
After epoch 3/100, val. cost = 1.882555, val. accuracy = 0.949214
After epoch 4/100, val. cost = 1.878212, val. accuracy = 0.949280
After epoch 5/100, val. cost = 1.865093, val. accuracy = 0.949784
After epoch 6/100, val. cost = 1.862868, val. accuracy = 0.949975
After epoch 7/100, val. cost = 1.872177, val. accuracy = 0.949765
After epoch 8/100, val. cost = 1.861570, val. accuracy = 0.949908
After epoch 9/100, val. cost = 1.860302, val. accuracy = 0.950058
After epoch 10/100, val. cost = 1.859392, val. accuracy = 0.950066
After epoch 11/100, val. cost = 1.859988, val. accuracy = 0.950072
After epoch 12/100, val. cost = 1.860679, val. accuracy = 0.950219
After epoch 13/100, val. cost = 1.860784, val. accuracy = 0.950052
After epoch 14/100, val. cost = 1.856457, val. accuracy = 0.950324
After epoch 15/100, val. cost = 1.855849, val. accuracy = 0.950442
After epoch 16/100, val. cost = 1.857207, val. accuracy = 0.950349
After epoch 17/100, val. cost = 1.857451, val. accuracy = 0.950440
After epoch 18/100, val. cost = 233.749680, val. accuracy = 0.922279
After epoch 19/100, val. cost = 10.804334, val. accuracy = 0.922474
After epoch 20/100, val. cost = 2.024744, val. accuracy = 0.941513
After epoch 21/100, val. cost = 1.991022, val. accuracy = 0.948082
After epoch 22/100, val. cost = 1.892341, val. accuracy = 0.948472
After epoch 23/100, val. cost = 1.881980, val. accuracy = 0.949003
After epoch 24/100, val. cost = 1.878152, val. accuracy = 0.948684
After epoch 25/100, val. cost = 3.764726, val. accuracy = 0.892659
After epoch 26/100, val. cost = 1.882202, val. accuracy = 0.949106
After epoch 27/100, val. cost = 1.869978, val. accuracy = 0.949478
After epoch 28/100, val. cost = 1.865192, val. accuracy = 0.949661
After epoch 29/100, val. cost = 1.862290, val. accuracy = 0.949832
After epoch 30/100, val. cost = 1.860487, val. accuracy = 0.949955
After epoch 31/100, val. cost = 1.859625, val. accuracy = 0.950004
After epoch 32/100, val. cost = 1.859726, val. accuracy = 0.950003
After epoch 33/100, val. cost = 1.862378, val. accuracy = 0.949890
After epoch 34/100, val. cost = 1.860624, val. accuracy = 0.949964
After epoch 35/100, val. cost = 1.857924, val. accuracy = 0.950169
After epoch 36/100, val. cost = 1.856217, val. accuracy = 0.950286
After epoch 37/100, val. cost = 67.750542, val. accuracy = 0.922279
After epoch 38/100, val. cost = 10.782505, val. accuracy = 0.884030
After epoch 39/100, val. cost = 2.386578, val. accuracy = 0.923412
After epoch 40/100, val. cost = 1.902082, val. accuracy = 0.947931
After epoch 41/100, val. cost = 1.879560, val. accuracy = 0.948964
After epoch 42/100, val. cost = 1.870183, val. accuracy = 0.949312
After epoch 43/100, val. cost = 1.865569, val. accuracy = 0.949470
After epoch 44/100, val. cost = 1.862972, val. accuracy = 0.949552
After epoch 45/100, val. cost = 1.861059, val. accuracy = 0.949647
After epoch 46/100, val. cost = 1.858534, val. accuracy = 0.949862
After epoch 47/100, val. cost = 1.857017, val. accuracy = 0.949998
After epoch 48/100, val. cost = 1.856058, val. accuracy = 0.950060
After epoch 49/100, val. cost = 1.855190, val. accuracy = 0.950120
After epoch 50/100, val. cost = 1.855699, val. accuracy = 0.950006
After epoch 51/100, val. cost = 1.918502, val. accuracy = 0.949666
After epoch 52/100, val. cost = 18.450184, val. accuracy = 0.847766
After epoch 53/100, val. cost = 5.032366, val. accuracy = 0.945344
After epoch 54/100, val. cost = 1.877560, val. accuracy = 0.949354
After epoch 55/100, val. cost = 1.865135, val. accuracy = 0.949729
```

```
After epoch 56/100, val. cost = 1.860224, val. accuracy = 0.949898
After epoch 57/100, val. cost = 1.857386, val. accuracy = 0.950016
After epoch 58/100, val. cost = 1.855612, val. accuracy = 0.950095
After epoch 59/100, val. cost = 1.854449, val. accuracy = 0.950158
After epoch 60/100, val. cost = 1.853615, val. accuracy = 0.950207
After epoch 61/100, val. cost = 1.853209, val. accuracy = 0.950238
After epoch 62/100, val. cost = 1.853108, val. accuracy = 0.950252
After epoch 63/100, val. cost = 1.852957, val. accuracy = 0.950281
After epoch 64/100, val. cost = 1.852324, val. accuracy = 0.950329
After epoch 65/100, val. cost = 1.851226, val. accuracy = 0.950383
After epoch 66/100, val. cost = 1.850773, val. accuracy = 0.950390
After epoch 67/100, val. cost = 1.851091, val. accuracy = 0.950363
After epoch 68/100, val. cost = 1.851457, val. accuracy = 0.950337
After epoch 69/100, val. cost = 1.851060, val. accuracy = 0.950358
After epoch 70/100, val. cost = 1.849963, val. accuracy = 0.950464
After epoch 71/100, val. cost = 1.850691, val. accuracy = 0.950478
After epoch 72/100, val. cost = 1.851123, val. accuracy = 0.950479
After epoch 73/100, val. cost = 1.852121, val. accuracy = 0.950470
After epoch 74/100, val. cost = 1.865996, val. accuracy = 0.950403
After epoch 75/100, val. cost = 1.854179, val. accuracy = 0.950463
After epoch 76/100, val. cost = 1.852403, val. accuracy = 0.950367
After epoch 77/100, val. cost = 1.851036, val. accuracy = 0.950443
After epoch 78/100, val. cost = 1.849285, val. accuracy = 0.950595
After epoch 79/100, val. cost = 1.848696, val. accuracy = 0.950652
After epoch 80/100, val. cost = 1.848843, val. accuracy = 0.950631
After epoch 81/100, val. cost = 1.848938, val. accuracy = 0.950632
After epoch 82/100, val. cost = 1.848395, val. accuracy = 0.950703
After epoch 83/100, val. cost = 1.848701, val. accuracy = 0.950732
After epoch 84/100, val. cost = 1.849202, val. accuracy = 0.950715
After epoch 85/100, val. cost = 1.849040, val. accuracy = 0.950783
After epoch 86/100, val. cost = 1.872133, val. accuracy = 0.949550
After epoch 87/100, val. cost = 1.849209, val. accuracy = 0.950923
After epoch 88/100, val. cost = 1.847659, val. accuracy = 0.950902
After epoch 89/100, val. cost = 1.847909, val. accuracy = 0.950863
After epoch 90/100, val. cost = 1.848039, val. accuracy = 0.950874
After epoch 91/100, val. cost = 1.847763, val. accuracy = 0.951099
After epoch 92/100, val. cost = 1.849267, val. accuracy = 0.950847
After epoch 93/100, val. cost = 1.857027, val. accuracy = 0.950899
After epoch 94/100, val. cost = 1.871040, val. accuracy = 0.951092
After epoch 95/100, val. cost = 1.851732, val. accuracy = 0.949591
After epoch 96/100, val. cost = 1.849334, val. accuracy = 0.950516
After epoch 97/100, val. cost = 1.849402, val. accuracy = 0.950991
After epoch 98/100, val. cost = 1.850866, val. accuracy = 0.949915
After epoch 99/100, val. cost = 1.851694, val. accuracy = 0.949653
accuracy avg = 0.9464908510446548, accuracy (within 1.0) = 0.862466337873806
2, accuracy (within 0.5) = 0.5650892437764209
```

Statistics of training dataset and network predictions

In [26]:
```
x = np.array([1,2,3,4,5,6,7,8,9,10])
meanscores = np.fromiter((np.dot(vec, x) for vec in labels_vecs), float)
valscores = np.fromiter((np.dot(vec, x) for vec in val_y), float)
```

In [27]:
```
%matplotlib inline

import matplotlib
import matplotlib.pyplot as plt

plt.hist(meanscores, range=[1.0, 10.0], bins='auto')
plt.title("Distribution of weighted average ratings in AVA dataset")
plt.show()
```



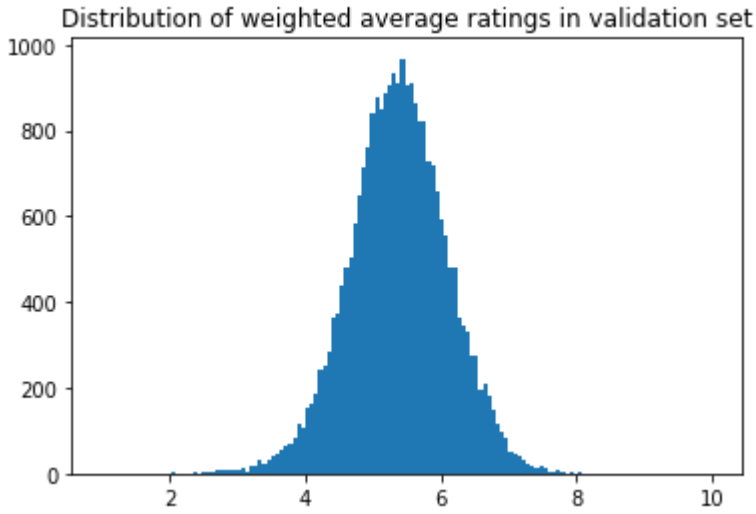
In [28]:
```
np.average(meanscores)
```

Out[28]: 5.3832862847076335

In [29]:
```
np.std(meanscores)
```

Out[29]: 0.7313037217017233

In [30]:
```
plt.hist(valscores, range=[1.0, 10.0], bins='auto')
plt.title("Distribution of weighted average ratings in validation set")
plt.show()
```



In [31]:
```
np.average(valscores)
```

Out[31]: 5.3746428766336605

In [32]:
```
np.std(valscores)
```

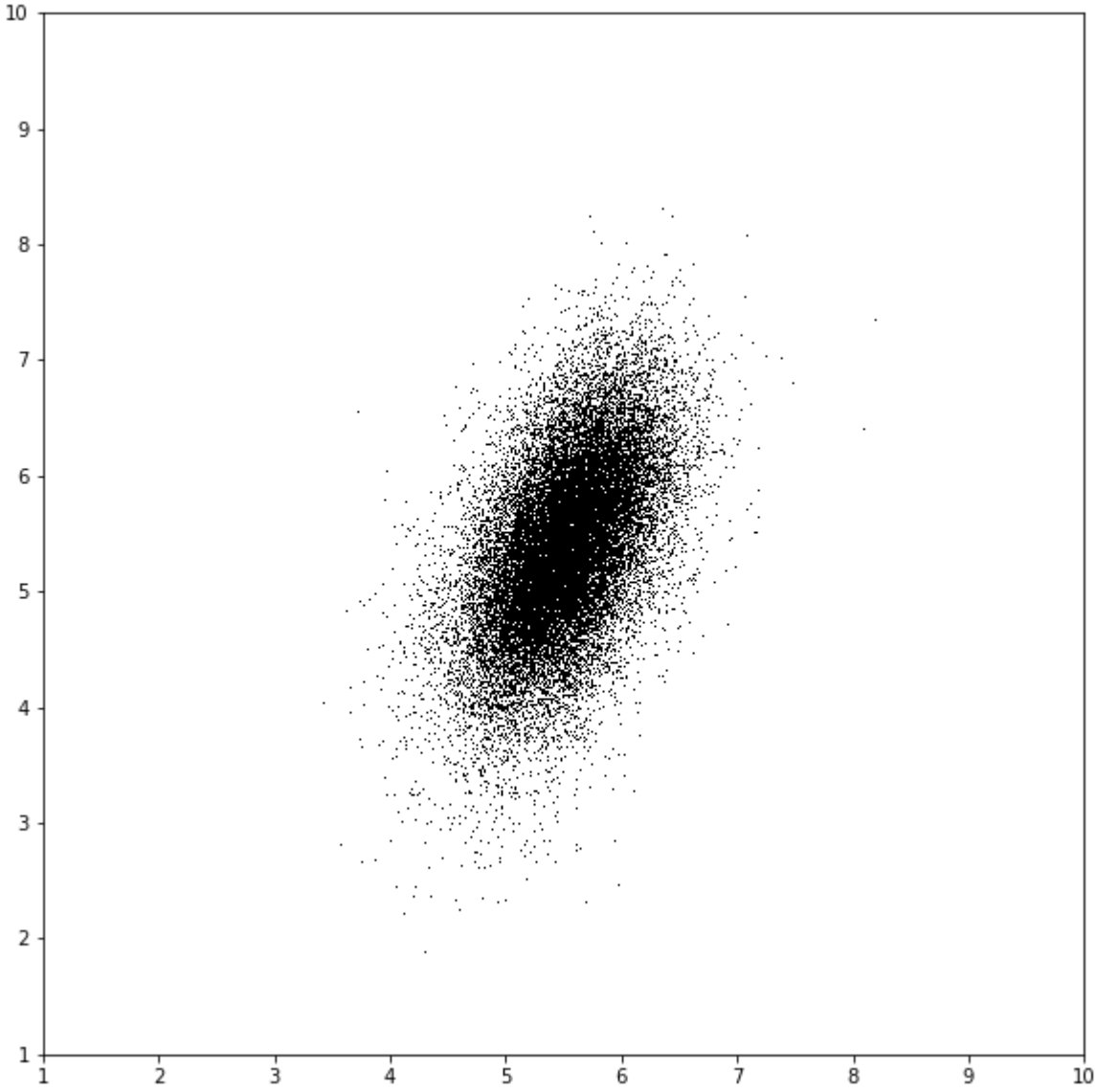
Out[32]: 0.7240856440424394

Correlation between predicted and actual scores, for test images

In [33]:
```
saver = tf.train.Saver()

with tf.Session() as sess:
    saver.restore(sess, "checkpoints/AAR_network.ckpt")

    feed_dict={inputs_:test_x, labels_:test_y}

    score_pred, score_act = sess.run((mean_score_predicted, mean_score_actual
), feed_dict=feed_dict)
```

```
INFO:tensorflow:Restoring parameters from checkpoints/AAR_network.ckpt
```

In [34]:
```
plt.figure(figsize=(10,10))
plt.plot(score_pred, score_act, ',', color='black')
plt.xlim(1, 10)
plt.ylim(1, 10)
plt.gca().set_aspect('equal', adjustable='box')
plt.draw()
```



In [35]:
```
np.corrcoef(score_pred, score_act)
```

Out[35]:
```
array([[1.        , 0.51732752],
       [0.51732752, 1.        ]])
```

Relative importance of each of the 4096 VGG-derived codes on rating

Estimate importance of each VGG code by training a simple one-layer network from 4096 inputs to 10 outputs (ratings).

After training, the weights matrix contains information on whether each VGG code boosts or lowers rating, and by how much.

In [36]:
```
# build network

import math

num_inputs = 4096 # train_x.shape[1]
num_outputs = 10 # labels_vecs.shape[1]

imp_inputs_ = tf.placeholder(tf.float32, shape=[None, num_inputs])
imp_labels_ = tf.placeholder(tf.float32, shape=[None, num_outputs])

# one layer in our network

imp_layer1_W = tf.Variable(tf.truncated_normal([num_inputs, num_outputs], stdd
ev=math.sqrt(2.0/num_inputs)))
imp_layer1_bias = tf.Variable(tf.zeros([num_outputs]))
imp_layer1 = tf.add(tf.matmul(imp_inputs_, imp_layer1_W), imp_layer1_bias)

imp_logits = tf.identity(imp_layer1, name='logits')

imp_cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=im
p_logits, labels=imp_labels_))

imp_optimizer = tf.train.AdamOptimizer().minimize(imp_cost)


# Operations for validation/test accuracy

imp_predicted = tf.nn.softmax(imp_logits)
imp_scores = tf.transpose(tf.constant([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]], dtype
=tf.float32))
imp_mean_score_predicted = tf.squeeze(tf.matmul(imp_predicted, imp_scores), ax
is=[1])
imp_mean_score_actual = tf.squeeze(tf.matmul(imp_labels_, imp_scores), axis=[1
])

imp_accuracy = 1 - tf.reduce_mean(tf.abs(imp_mean_score_predicted - imp_mean_s
core_actual) / 10)
imp_error = tf.abs(imp_mean_score_predicted - imp_mean_score_actual)
```

In [39]:
```
epochs = 5

accuracy_sum = 0
acc_one_sum = 0
acc_half_sum = 0
total_sum = 0

saver = tf.train.Saver()
with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())

    val_cost, val_acc = sess.run((imp_cost, imp_accuracy), feed_dict={imp_inpu
ts_:val_x, imp_labels_:val_y})
    print("Starting, val. cost = {:f}, val. accuracy = {:f}".format(val_cost,
val_acc))

    for epoch in range(epochs):
        for x, y in get_batches(train_x, train_y):
            feed = {inputs_:x, labels_:y}
            sess.run(optimizer, feed_dict=feed)

        val_cost, val_acc, val_error = sess.run((imp_cost, imp_accuracy, imp_e
rror), feed_dict={imp_inputs_:val_x, imp_labels_:val_y})

        accuracy_sum += val_acc
        acc_one_sum += len([x for x in val_error if x < 1.0])
        acc_half_sum += len([x for x in val_error if x < 0.5])
        total_sum += len(val_error)

        print("After epoch {}/{}, val. cost = {:f}, val. accuracy = {:f}".form
at(epoch, epochs, val_cost, val_acc))

    saver.save(sess, "checkpoints/AAR_network_importance.ckpt")

accuracy_sum /= epochs
acc_one_sum /= total_sum
acc_half_sum /= total_sum
print(f'accuracy avg = {accuracy_sum}, accuracy (within 1.0) = {acc_one_sum},
 accuracy (within 0.5) = {acc_half_sum}')
```

```
Starting, val. cost = 9.416849, val. accuracy = 0.766799
After epoch 0/5, val. cost = 9.416849, val. accuracy = 0.766799
After epoch 1/5, val. cost = 9.416849, val. accuracy = 0.766799
After epoch 2/5, val. cost = 9.416849, val. accuracy = 0.766799
After epoch 3/5, val. cost = 9.416849, val. accuracy = 0.766799
After epoch 4/5, val. cost = 9.416849, val. accuracy = 0.766799
accuracy avg = 0.7667993307113647, accuracy (within 1.0) = 0.2452246751213402
2, accuracy (within 0.5) = 0.12337560670111164
```

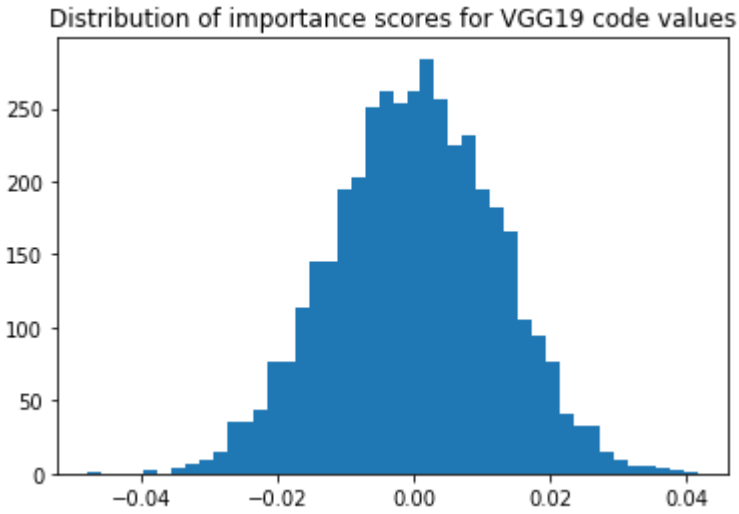
In [45]:
```
importance_factors = tf.transpose(tf.constant([[-0.2, -0.2, -0.2, -0.2, -0.2,
0.2, 0.2, 0.2, 0.2, 0.2]], dtype=tf.float32))
importance = tf.squeeze(tf.matmul(imp_layer1_W, importance_factors), axis=[1])

with tf.Session() as sess:
    saver.restore(sess, "checkpoints/AAR_network_importance.ckpt")

    feed_dict={inputs_:val_x, labels_:val_y}

    val_importance = sess.run((importance), feed_dict=feed_dict)

    imps = sorted([(imp, index) for (index, imp) in enumerate(val_importance
)],
                  key=lambda x: x[0])
    worst_features = imps[:5]
    best_features = imps[-5:]
    worst_features, best_features
```

```
INFO:tensorflow:Restoring parameters from checkpoints/AAR_network_importance.
ckpt
```

In [46]:
```
val_importance.shape
```

Out[46]:
```
(4096,)
```

In [47]:
```
%matplotlib inline

import matplotlib
import matplotlib.pyplot as plt

plt.hist(val_importance, bins='auto')
plt.title("Distribution of importance scores for VGG19 code values")
plt.show()
```



In [48]:
```
best_features
```

Out[48]:
```
[(0.036169462, 1802),
 (0.037255943, 178),
 (0.03815395, 794),
 (0.03872242, 2292),
 (0.041737203, 1277)]
```

Show examples of AVA images with a high value of a given VGG code number

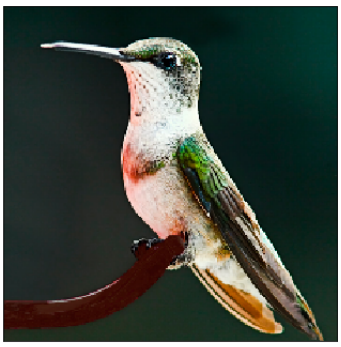
In [19]:
```
%matplotlib inline

import matplotlib
import matplotlib.pyplot as plt
from tensorflow_vgg import utils
import random

data_dir = 'E:/AVA_dataset/images'

def show_examples(code_index=None, cols=3, rows=3):
    grid_size = cols * rows
    if code_index is not None:
        image_code_value = [(row[code_index], str(labels[idx][0])+'.jpg') for
(idx, row) in enumerate(codes)]
        top_images = sorted(image_code_value, key=lambda x: x[0])[-grid_size:]
        images = [utils.load_image(os.path.join(data_dir, filename)) for (_, f
ilename) in top_images]
    else:
        indices = [random.randint(0, labels.shape[0]) for _ in range(grid_size
)]
        filenames = [str(labels[idx][0])+'.jpg' for idx in indices]
        images = [utils.load_image(os.path.join(data_dir, filename)) for filen
ame in filenames]

    fig, axarr = plt.subplots(rows, cols)
    fig.set_size_inches(20, 20 * rows / cols)
    #frame.axes.get_xaxis().set_visible(False)
    #frame.axes.get_yaxis().set_visible(False)
    for (idx, image) in enumerate(images):
        axarr[idx // cols, idx % cols].imshow(image)
        axarr[idx // cols, idx % cols].set_xticks([])
        axarr[idx // cols, idx % cols].set_yticks([])
```

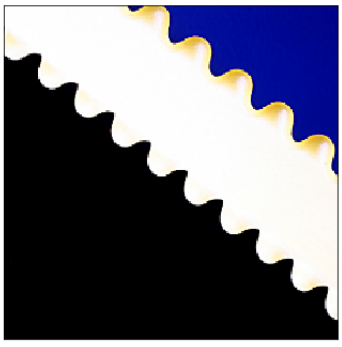
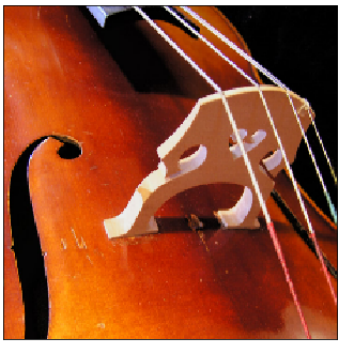
In [60]:
```
show_examples(code_index=3728, cols=3, rows=5)
```









Function to run arbitrary image through rating network (incomplete)

In [24]:
```
import numpy as np
import tensorflow as tf

from tensorflow_vgg import vgg19
from tensorflow_vgg import utils

def image_to_vgg_code(filepath):
    codes = None

    with tf.Session() as sess:

        # First build the VGG network
        vgg = vgg19.Vgg19()
        input_ = tf.placeholder(tf.float32, [None, 224, 224, 3])
        with tf.name_scope("content_vgg"):
            vgg.build(input_)

        try:
            # utils.load_image crops the input images for us, from the center
            img = utils.load_image(filepath)
            print(f'original image shape: {img.shape}')
            if len(img.shape) == 2:
                # black and white image: duplicate pixel values into each of
 R, G, B planes
                img = np.concatenate((img, img, img))
            image = img.reshape((1, 224, 224, 3))
            print(f'reshaped image shape: {image.shape}')

            # Run the image through the VGG network to get the codes
            # Get the values from the relu6 layer of the VGG network
            feed_dict = {input_: image}
            codes = sess.run(vgg.relu6, feed_dict=feed_dict)
        except FileNotFoundError:
            print(f"File {filepath} not found")
        except ValueError:
            print("Got a value error")
        except TypeError:
            print("Got a type error")
        except NameError:
            print("Got a name error")
        except:
            print("Got a general exception")

    return codes.reshape((-1))

def codes_to_rating(codes):
    num_inputs = 4096
    num_outputs = 10

    inputs_, labels_, logits = build_network(num_inputs, num_outputs)
    predicted = tf.nn.softmax(logits)
    prediction = None

    saver = tf.train.Saver()
    labels = [0] * 10
```

```
    with tf.Session() as sess:
        saver.restore(sess, "checkpoints/AAR_network.ckpt")

        feed_dict={inputs_:codes, labels_:labels}

        prediction = sess.run((predicted), feed_dict=feed_dict)

    return prediction

def rate_image(filepath):
    codes = image_to_vgg_code(filepath)
    prediction = codes_to_rating(codes)

    print(prediction)
```

In [16]:
```
codes = image_to_vgg_code(r'C:\Users\Ramsey\Desktop\133.jpg')
```

```
C:\Users\Ramsey\Documents\AAR Project\tensorflow_vgg\vgg19.npy
npy file loaded
build model started
build model finished: 2s
original image shape: (224, 224, 3)
reshaped image shape: (1, 224, 224, 3)
```