ECE 763 Project 2

Submitted by: Binoy Thomas
200260021

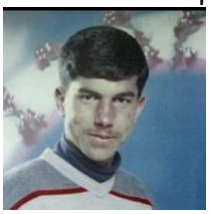
Objective: Face image classification using a simple neural network to get familiar with steps of training neural networks.

Data Preparation

Data was prepared using the FDDB dataset
Images in the dataset were not perfect and hence it was necessary to create a robust data set for face images.
The autocrop.py function available on python was used to crop and create a dataset which contained only the face images.

autocrop -i temp12 -o temp11 -w 100 -H 100 --facePercent 50 --padUp 10 --padDown 10 --padLeft 10 --padRight 10

This script will create a cropped image from the temp12 folder to the temp11 folder
Before autocrop



After autocrop

A dataset of 10000 training faces and nonfaces and 1000 testing faces and non faces were created.

Data preprocessing

Data preprocessing is done in order to get the training data to be normalized.
For data processing, we first made the data zero centered and then divided it by the standard deviation in order to normalize it.

```
imageVector = imageVector - np.mean(imageVector, axis=0)
imageVector = imageVector / np.std(imageVector, axis=0)
```

Convolutional Neural Network

The Convolutional Neural Network was made using MNIST tutorial available on tensorflow. It was a simple 3 layer perceptron.I t consisted of 2 convolutional layers, 2 pooling layers and one dense layer.

The convolution layers were 32 and 64 5*5 filters and the dense layer contains 2 neurons for the 2 classes(face and non face). The label was set to 0 for a non face image and 1 for a face image.

Tests

For a learning rate of 0.001

```
  if mode == tf.estimator.ModeKeys.TRAIN:
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
    train_op = optimizer.minimize(
        loss=loss,
        global_step=tf.train.get_global_step())
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)
```

Initially the loss is 0.680

```
INFO:tensorflow:probabilities = [[0.49944501 0.50055499]
 [0.48924981 0.51075019]
 [0.51393548 0.48606452]
 [0.47821335 0.52178665]
 [0.48110102 0.51889898]
 [0.48804115 0.51195885]
 [0.51395482 0.48604518]
 [0.49961822 0.50038178]
 [0.51297782 0.48702218]
 [0.51333247 0.48666753]]
INFO:tensorflow:loss = 0.6809408068656921, step = 1
```

The results after training for 20000 steps were as follows

```
INFO:tensorflow:loss = 0.0053718676790595055, step = 39901 (4.850 sec)
INFO:tensorflow:probabilities = [[0.00000007 0.99999993]
 [0.99921195 0.00078805]
 [0.99395258 0.00604742]
 [0.99765511 0.00234489]
 [0.00006156 0.99993844]
 [0.04669811 0.95330189]
 [0.9970468  0.0029532 ]
 [0.00000412 0.99999588]
 [0.88534339 0.11465661]
 [0.99966349 0.00033651]] (2.272 sec)
INFO:tensorflow:Saving checkpoints for 40000 into /tmp/mnist_convnet_model3/model.ckpt.
INFO:tensorflow:Loss for final step: 0.012965328991413116.

<tensorflow_estimator.python.estimator.estimator.Estimator at 0x1a41bad128>
```

The loss obtained was 0.0129

The testing results were as follows

```
eval_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": eval_data},
    y=eval_labels,
    num_epochs=10,
    shuffle=False)

eval_results = mnist_classifier.evaluate(input_fn=eval_input_fn)
print(eval_results)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2019-04-08T17:33:20Z
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /tmp/mnist_convnet_model3/model.ckpt-40000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2019-04-08-17:33:25
INFO:tensorflow:Saving dict for global step 40000: accuracy = 0.98, global_step = 40000, loss = 0.06970101
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 40000: /tmp/mnist_convnet_model3/model.ckpt-40000
{'accuracy': 0.98, 'loss': 0.06970101, 'global_step': 40000}
```

The accuracy of the evaluated testing images was 98%

For a learning rate of 1e-6

```
  if mode == tf.estimator.ModeKeys.TRAIN:
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.000001)
    train_op = optimizer.minimize(
        loss=loss,
        global_step=tf.train.get_global_step())
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)
```

As expected the model was the loss barely changes and since the learning rate is too low

```
INFO:tensorflow:loss = 0.6517248153686523, step = 19901 (4.313 sec)
INFO:tensorflow:probabilities = [[0.48602667 0.51397333]
 [0.49787739 0.50212261]
 [0.44305666 0.55694334]
 [0.5151084  0.4848916 ]
 [0.44926104 0.55073896]
 [0.47870152 0.52129848]
 [0.46475527 0.53524473]
 [0.4942631  0.5057369 ]
 [0.48480104 0.51519896]
 [0.49515195 0.50484805]] (1.946 sec)
INFO:tensorflow:Saving checkpoints for 20000 into /tmp/mnist_convnet_model4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.6656574010848999.
```

Hence we can say that the ideal range of learning rate is between 1e-3 and 1e-5

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2019-04-08T16:37:50Z
INFO:tensorflow:Graph was finalized.
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorflow/python/training/saver.py:1266: checkpoint_e
xists (from tensorflow.python.training.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from /tmp/mnist_convnet_model4/model.ckpt-20000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2019-04-08-16:37:54
INFO:tensorflow:Saving dict for global step 20000: accuracy = 0.52, global_step = 20000, loss = 0.6878175
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 20000: /tmp/mnist_convnet_model4/model.ckpt-20000
{'accuracy': 0.52, 'loss': 0.6878175, 'global_step': 20000}
```

The accuracy was as low as 52%

For a learning rate of 1

The loss was expected to explode and it was noticed that for all the batches the value was the same and the system failed to train

```
INFO:tensorflow:probabilities = [[0.51032418 0.48967582]
 [0.51032418 0.48967582]
 [0.51032418 0.48967582]
 [0.51032418 0.48967582]
 [0.51032418 0.48967582]
 [0.51032418 0.48967582]
 [0.51032418 0.48967582]
 [0.51032418 0.48967582]
 [0.51032418 0.48967582]] (1.986 sec)
INFO:tensorflow:Saving checkpoints for 20000 into /tmp/mnist_convnet_model5/model.ckpt.
INFO:tensorflow:Loss for final step: 0.6874350309371948.

<tensorflow_estimator.python.estimator.estimator.Estimator at 0x1a409a00b8>
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2019-04-08T18:28:37Z
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /tmp/mnist_convnet_model5/model.ckpt-20000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2019-04-08-18:28:42
INFO:tensorflow:Saving dict for global step 20000: accuracy = 0.5, global_step = 20000, loss = 0.70250714
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 20000: /tmp/mnist_convnet_model5/model.ckpt-20000
{'accuracy': 0.5, 'loss': 0.70250714, 'global_step': 20000}
```

Code:

```
#!/usr/bin/env python
# coding: utf-8

# In[115]:


from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf
import numpy as np
import cv2
import numpy as np
from scipy.stats import multivariate_normal
import gc
from decimal import Decimal
import matplotlib.pyplot as plt
tf.logging.set_verbosity(tf.logging.INFO)

def shuffle_in_unison(a, b):
    assert len(a) == len(b)
    shuffled_a = np.empty(a.shape, dtype=a.dtype)
    shuffled_b = np.empty(b.shape, dtype=b.dtype)
    permutation = np.random.permutation(len(a))
    for old_index, new_index in enumerate(permutation):
        shuffled_a[new_index] = a[old_index]
        shuffled_b[new_index] = b[old_index]
    return shuffled_a, shuffled_b


def cnn_model_fn(features, labels, mode):
  """Model function for CNN."""
  # Input Layer
  input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])

  # Convolutional Layer #1
  conv1 = tf.layers.conv2d(
      inputs=input_layer,
      filters=32,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)

  # Pooling Layer #1
  pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
```

```
# Convolutional Layer #2 and Pooling Layer #2
conv2 = tf.layers.conv2d(
    inputs=pool1,
    filters=64,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

# Dense Layer
pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 64])
dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.relu)
dropout = tf.layers.dropout(
    inputs=dense, rate=0.4, training=mode == tf.estimator.ModeKeys.TRAIN)

# Logits Layer
    logits = tf.layers.dense(inputs=dropout, units=2)

predictions = {
    # Generate predictions (for PREDICT and EVAL mode)
    "classes": tf.argmax(input=logits, axis=1),
    # Add `softmax_tensor` to the graph. It is used for PREDICT and by the
    # `logging_hook`.
    "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
}

if mode == tf.estimator.ModeKeys.PREDICT:
  return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

# Calculate Loss (for both TRAIN and EVAL modes)
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)

# Configure the Training Op (for TRAIN mode)
if mode == tf.estimator.ModeKeys.TRAIN:
  optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
  train_op = optimizer.minimize(
      loss=loss,
      global_step=tf.train.get_global_step())
  return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

# Add evaluation metrics (for EVAL mode)
eval_metric_ops = {
    "accuracy": tf.metrics.accuracy(
        labels=labels, predictions=predictions["classes"])
```

```
 }
 return tf.estimator.EstimatorSpec(
     mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)


l=7500
imageVector = np.empty((15000,28,28))
for i in range(l):
        img = cv2.imread("/Users/binoythomas/Desktop/face_images1/"+str(i+1)+str(1)+".jpg")
        im_resized = cv2.resize(img, (28, 28), interpolation=cv2.INTER_LINEAR)

        image = cv2.cvtColor(im_resized, cv2.COLOR_BGR2GRAY)
        imageVector[i][:][:] = image
        #plt.imshow(image,cmap='gray')
        #plt.show()


for i in range(l):
        img = cv2.imread("/Users/binoythomas/Desktop/nonfaces/"+str(i+1)+str(1)+".jpg")
        im_resized = cv2.resize(img, (28, 28), interpolation=cv2.INTER_LINEAR)
        image = cv2.cvtColor(im_resized, cv2.COLOR_RGB2GRAY)
        imageVector[7500+i][:][:] = image
        #plt.imshow(image)
        #plt.show()

print(np.shape(imageVector))

imageVector = imageVector - np.mean(imageVector, axis=0)

imageVector = imageVector / np.std(imageVector, axis=0)

imageLabels = np.empty((15000))
for i in range(l):
    imageLabels[i]=1


# In[124]:


for i in range(l):
    imageLabels[7500+i]=0


# In[126]:
```

```
# In[127]:


a = imageVector
b = imageLabels
ab=shuffle_in_unison(a, b)

# In[128]:


train_data=ab[0]
train_labels=ab[1]

# In[130]:


train_data = train_data/np.float32(255)
train_labels = train_labels.astype(np.int32)

# In[131]:


I=250
testVector = np.empty((500,28,28))
for i in range(I):
        img = cv2.imread("/Users/binoythomas/Desktop/face_images1/"+str(i+7500)+str(1)+".jpg")
        im_resized = cv2.resize(img, (28, 28), interpolation=cv2.INTER_LINEAR)
        image = cv2.cvtColor(im_resized, cv2.COLOR_RGB2GRAY)
        testVector[i][:][:] = image


for i in range(I):
        img = cv2.imread("/Users/binoythomas/Desktop/non face images/"+str(i+7500)+str(1)+".jpg")
        im_resized = cv2.resize(img, (28, 28), interpolation=cv2.INTER_LINEAR)
        image = cv2.cvtColor(im_resized, cv2.COLOR_RGB2GRAY)
        testVector[250+i][:][:] = image

testLabels = np.empty((500))
for i in range(250):
   testLabels[i]=1
for i in range(250):
   testLabels[250+i]=0

eval_data = testVector/np.float32(255)
eval_labels = testLabels.astype(np.int32)
```

```
# In[132]:


mnist_classifier = tf.estimator.Estimator(
    model_fn=cnn_model_fn, model_dir="/tmp/mnist_convnet_model3")


# In[133]:


# Set up logging for predictions
tensors_to_log = {"probabilities": "softmax_tensor"}

logging_hook = tf.train.LoggingTensorHook(
    tensors=tensors_to_log, every_n_iter=50)


# In[134]:


train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": train_data},
    y=train_labels,
    batch_size=10,
    num_epochs=None,
    shuffle=True)


# In[135]:


mnist_classifier.train(
    input_fn=train_input_fn,
    steps=20000,
    hooks=[logging_hook])


# In[ ]:


eval_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": eval_data},
    y=eval_labels,
    num_epochs=10,
    shuffle=False)

eval_results = mnist_classifier.evaluate(input_fn=eval_input_fn)
print(eval_results)
```