DEEP_FOOL

May 12, 2020

[ ]:

1 DEEP FOOL

2 MNIST DATA SET

[1]:
```
import numpy as np
from keras.datasets import mnist
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers,models
import keras
from keras.callbacks import ModelCheckpoint
```

```
Using TensorFlow backend.
C:\Users\AALAP RANA\Anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:526: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\AALAP RANA\Anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:527: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\AALAP RANA\Anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:528: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\AALAP RANA\Anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:529: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\AALAP RANA\Anaconda3\lib\site-
```

```
packages\tensorflow\python\framework\dtypes.py:530: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\AALAP RANA\Anaconda3\lib\site-
packages\tensorflow\python\framework\dtypes.py:535: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

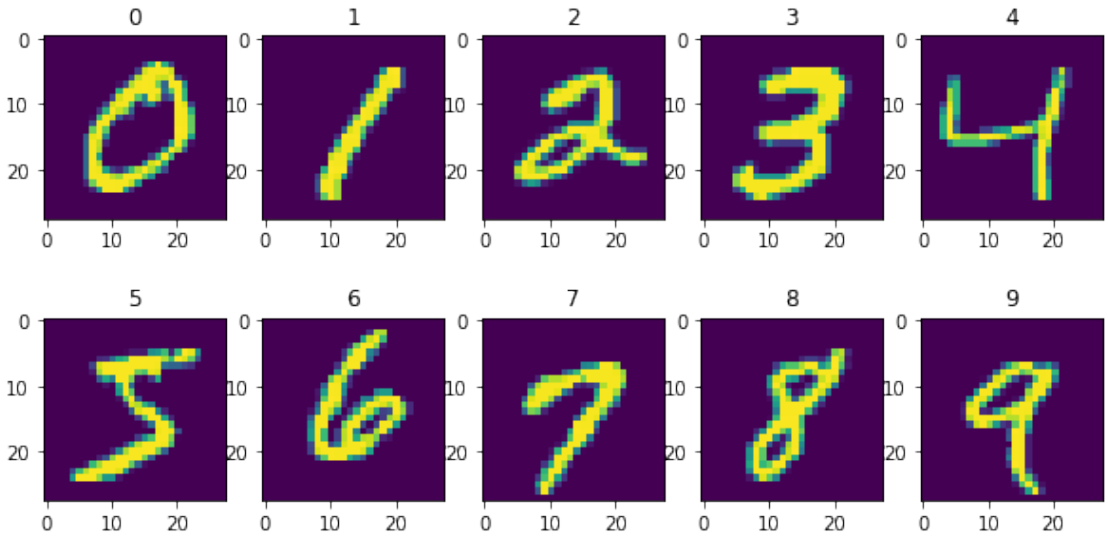
[2]:
```
session = tf.Session()
keras.backend.set_session(session)
```

[3]:
```
(mn_x_train, mn_y_train), (mn_x_test, mn_y_test) = mnist.load_data()
```

[4]:
```
print ("Training Examples: %d" % len(mn_x_train))
print ("Test Examples: %d" % len(mn_x_test))
```

```
Training Examples: 60000
Test Examples: 10000
```

[5]:
```
n_classes = 10
inds=np.array([mn_y_train==i for i in range(n_classes)])
f,ax=plt.subplots(2,5,figsize=(10,5))
ax=ax.flatten()
for i in range(n_classes):
    ax[i].imshow(mn_x_train[np.argmax(inds[i])].reshape(28,28))
    ax[i].set_title(str(i))
plt.show()
```



3 MODEL

[6]:
```
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
print(network.summary())
```

```
WARNING:tensorflow:From C:\Users\AALAP RANA\Anaconda3\lib\site-
packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
None
```

4 IMAGE PRE-PROCESSING AND NORMALIZATION

[7]:
```
train_images_1d = mn_x_train.reshape((60000, 28 * 28))
train_images_1d = train_images_1d.astype('float32') / 255
test_images_1d = mn_x_test.reshape((10000, 28 * 28))
test_images_1d = test_images_1d.astype('float32') / 255
```

5 ONE HOT SHORT LABEL

[8]:
```
from keras.utils import to_categorical
train_labels = to_categorical(mn_y_train)
test_labels = to_categorical(mn_y_test)
```

6 MODEL TRAINING

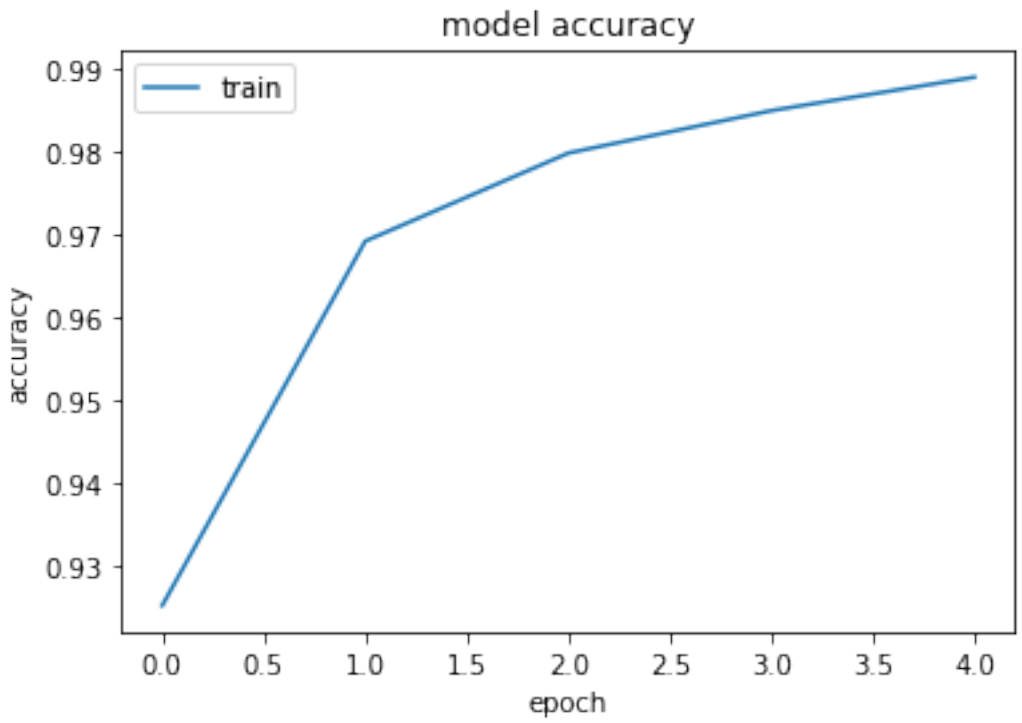
[9]:
```
h=network.fit(train_images_1d,
              train_labels,
              epochs=5,
              batch_size=128,
              shuffle=True,
             callbacks=[ModelCheckpoint('tutorial_MNIST.
 ↪h5',save_best_only=True)])
```

```
WARNING:tensorflow:From C:\Users\AALAP RANA\Anaconda3\lib\site-
packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/5
60000/60000 [==============================] - 2s 40us/sample - loss: 0.2560 -
acc: 0.9252
Epoch 2/5
 3968/60000 [>...] - ETA: 2s - loss: 0.1269 - acc:
0.9597

C:\Users\AALAP RANA\Anaconda3\lib\site-
packages\keras\callbacks\callbacks.py:707: RuntimeWarning: Can save best model
only with val_loss available, skipping.
  'skipping.' % (self.monitor), RuntimeWarning)

60000/60000 [==============================] - 2s 40us/sample - loss: 0.1023 -
acc: 0.9691
Epoch 3/5
60000/60000 [==============================] - 2s 37us/sample - loss: 0.0676 -
acc: 0.9797
Epoch 4/5
60000/60000 [==============================] - 2s 37us/sample - loss: 0.0498 -
acc: 0.9848
Epoch 5/5
60000/60000 [==============================] - 2s 37us/sample - loss: 0.0372 -
acc: 0.9889
```

[10]:
```
plt.plot(h.history['acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



[11]:
```
score, acc = network.evaluate(test_images_1d,
                              test_labels,
                              batch_size=128)

print ("Test Accuracy: %.5f" % acc)
```

```
10000/10000 [==============================] - 0s 16us/sample - loss: 0.0651 -
acc: 0.9810
Test Accuracy: 0.98100
```

7 CLEVERHANS LIB- DEEP FOOL METHOD

[12]:
```
from cleverhans.utils_keras import KerasModelWrapper
wrap = KerasModelWrapper(network)
```

[13]:
```
x = tf.placeholder(tf.float32, shape=(None, 784))
y = tf.placeholder(tf.float32, shape=(None, 10))
```

[14]:
```
from cleverhans.attacks import DeepFool
df = DeepFool(wrap, sess=session)
df_rate = 0.09
df_params = {'overshoot': df_rate,'max_iter':50,'clip_min': 0.,'clip_max': 1.
 ↪,'nb_candidate':10}
```

```
adv_x = df.generate(x, **df_params)
adv_x = tf.stop_gradient(adv_x)
adv_prob = network(adv_x)
```

```
C:\Users\AALAP RANA\cleverhans\attacks_tf.py:27: UserWarning: attacks_tf is
deprecated and will be removed on 2019-07-18 or after. Code should import
functions from their new locations directly.
  warnings.warn("attacks_tf is deprecated and will be removed on 2019-07-18"
```

```
WARNING:tensorflow:From C:\Users\AALAP RANA\cleverhans\attacks\deep_fool.py:81:
py_func (from tensorflow.python.ops.script_ops) is deprecated and will be
removed in a future version.
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, use
    tf.py_function, which takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
```

[15]:
```
fetches = [adv_prob]
fetches.append(adv_x)
outputs = session.run(fetches=fetches, feed_dict={x:test_images_1d})
adv_prob = outputs[0]
adv_examples = outputs[1]
```

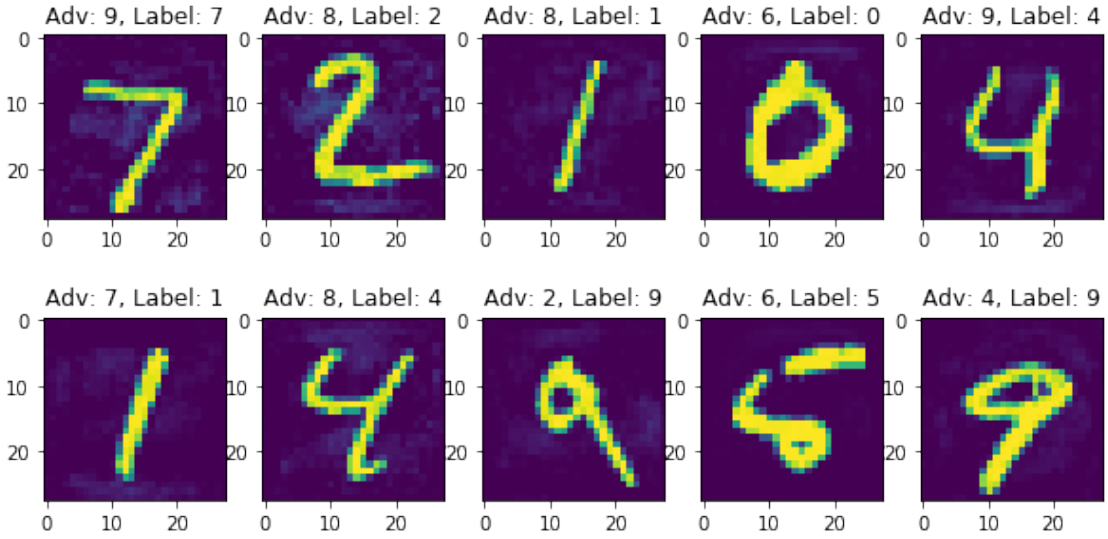
```
[INFO 2020-04-23 22:09:33,729 cleverhans] Attack result at iteration 5 is [7 2 1
... 4 5 6]
[INFO 2020-04-23 22:09:42,402 cleverhans] Attack result at iteration 10 is [7 2
1 ... 9 5 6]
[INFO 2020-04-23 22:09:49,512 cleverhans] Attack result at iteration 15 is [7 2
8 ... 9 8 6]
[INFO 2020-04-23 22:09:55,668 cleverhans] Attack result at iteration 20 is [7 8
8 ... 9 8 6]
[INFO 2020-04-23 22:10:01,259 cleverhans] Attack result at iteration 25 is [7 8
8 ... 9 8 6]
[INFO 2020-04-23 22:10:06,409 cleverhans] Attack result at iteration 30 is [9 8
8 ... 9 8 6]
[INFO 2020-04-23 22:10:11,217 cleverhans] Attack result at iteration 35 is [9 8
8 ... 9 8 6]
[INFO 2020-04-23 22:10:15,939 cleverhans] Attack result at iteration 40 is [9 8
8 ... 9 8 4]
[INFO 2020-04-23 22:10:20,655 cleverhans] Attack result at iteration 45 is [9 8
8 ... 9 8 4]
[INFO 2020-04-23 22:10:25,075 cleverhans] Attack result at iteration 50 is [9 8
8 ... 9 8 4]
[INFO 2020-04-23 22:10:25,091 cleverhans] 9868 out of 10000 become adversarial
```

```
examples at iteration 50
```

[16]:
```
adv_predicted = adv_prob.argmax(1)
adv_accuracy = np.mean(adv_predicted == mn_y_test)
print("Adversarial accuracy: %.5f" % adv_accuracy)
```

```
Adversarial accuracy: 0.01400
```

[17]:
```
n_classes = 10
f,ax=plt.subplots(2,5,figsize=(10,5))
ax=ax.flatten()
for i in range(n_classes):
    ax[i].imshow(adv_examples[i].reshape(28,28))
    ax[i].set_title("Adv: %d, Label: %d" % (adv_predicted[i], mn_y_test[i]))
plt.show()
```



[ ]: