

```
In [1]: from utils import *
import numpy as np
import pandas as pd
import tensorflow as tf
```

```
In [2]: tf.enable_eager_execution()
```

```
In [3]: ### Loads in Labels
ytest = np.load("channels/channel_5_ytest.npy")
```

```
In [4]: ### Specify channels to load in, in the input list (see utils.py for code)
chan_dict = load_channels([3,4,5,7,6])###Here
```

```
In [18]: chan_list = list(chan_dict.keys())
print("-The Channels you have loaded are: ",)
print("\t",list(chan_dict.keys()))
```

```
-The Channels you have loaded are:
      ['channel5', 'channel3', 'channel4', 'channel7', 'channel6']
```

```
In [6]: ### specify models to load in
model_dict = load_models([3,4,5,6])
```

WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python/ops/resource_variable_ops.py:642: 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.

WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python/keras/layers/core.py:143: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.

Instructions for updating:

Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.

```
In [7]: model_list = list(model_dict.keys())
print("The models you have loaded are: ")
print("\t", model_list)
```

```
The models you have loaded are:
      ['channel5', 'channel3', 'channel4', 'channel6']
```

```
In [8]: ### Inference for each model
from tensorflow.contrib.metrics import f1_score
loss = []
acc = []
f1 = []
for index in range(len(model_list)):

    print("Inference for ", model_list[index])
    model = model_dict[model_list[index]]
    opt = tf.keras.optimizers.Adam()
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])

    score = model.evaluate(chan_dict[model_list[index]], ytest)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])
    loss.append(score[0])
    acc.append(score[1])
    print()
    pred = model.predict(chan_dict[model_list[index]])
    f1.append(f1_score(ytest, pred)[0].numpy())
```

Inference for channel5

236/236 [=====] - 2s 7ms/sample - loss: 0.2640 - acc: 0.9703

Test loss: 0.26396609564958995

Test accuracy: 0.970339

WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python/ops/metrics_impl.py:526: to_float (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/contrib/metrics/python/metrics/classification.py:162: div (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Deprecated in favor of operator or tf.math.divide.

Inference for channel3

236/236 [=====] - 2s 7ms/sample - loss: 2.4484 - acc: 0.9449

Test loss: 2.44839819407059

Test accuracy: 0.94491524

Inference for channel4

236/236 [=====] - 1s 6ms/sample - loss: 2.2138 - acc: 0.8983

Test loss: 2.213776362144341

Test accuracy: 0.89830506

Inference for channel6

236/236 [=====] - 2s 7ms/sample - loss: 2.0069 - acc: 0.9576

Test loss: 2.006942728818473

Test accuracy: 0.9576271

```

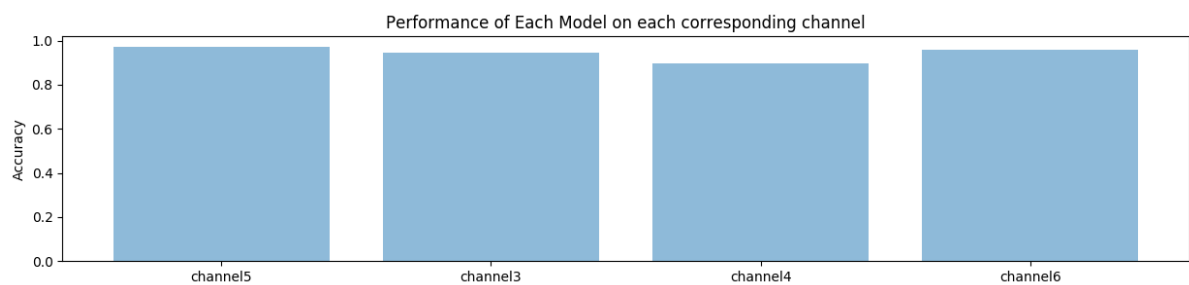
In [9]: ### visualization
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = tuple(model_list)
y_pos = np.arange(len(objects))
performance = acc

plt.figure(figsize=(15,3))
plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('Accuracy')
plt.title('Performance of Each Model on each corresponding channel')

plt.show()

```



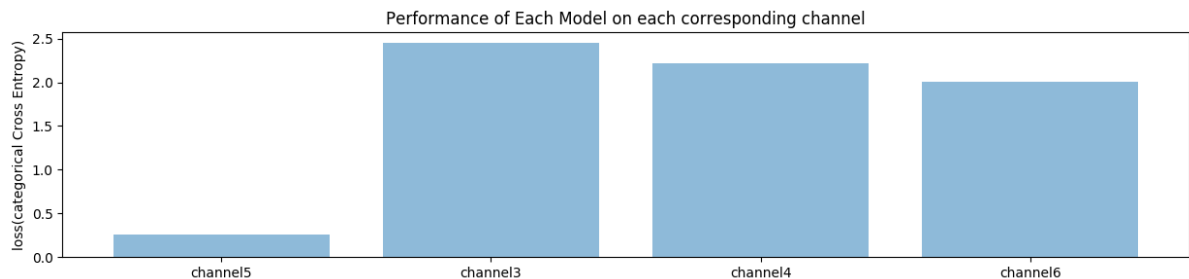
```

In [10]: ### visualization
objects = tuple(model_list)
y_pos = np.arange(len(objects))
performance = loss

plt.figure(figsize=(15,3))
plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('loss(categorical Cross Entropy)')
plt.title('Performance of Each Model on each corresponding channel')

plt.show()

```



The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

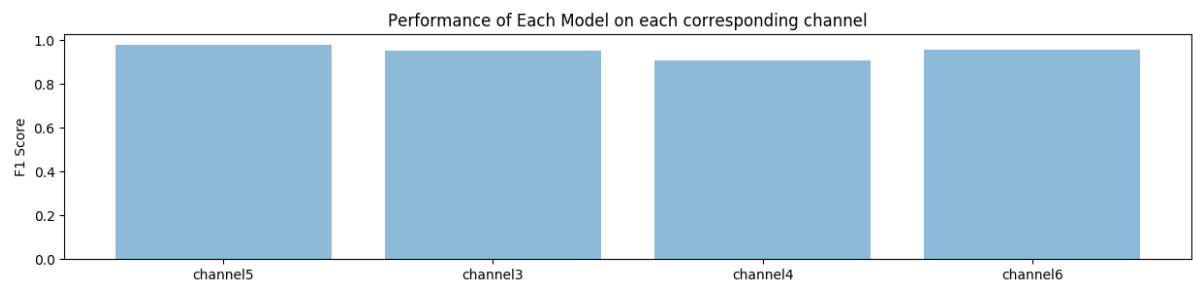
$$-F1 = 2 (\textit{precision recall}) / (\textit{precision} + \textit{recall})$$

As quoted from sklearn's documentation

```
In [11]: ### visualization
objects = tuple(model_list)
y_pos = np.arange(len(objects))
performance = f1

plt.figure(figsize=(15,3))
plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('F1 Score')
plt.title('Performance of Each Model on each corresponding channel')

plt.show()
```



```
In [12]: for index in range(len(model_list)):

    print("Confusion Matrix for ", model_list[index])
    model = model_dict[model_list[index]]
    opt = tf.keras.optimizers.Adam()
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])
    pred = model.predict(chan_dict[model_list[index]])
    print("The Confusion Matrix")
    confuse = tf.math.confusion_matrix(np.argmax(ytest,axis = 1), np.argmax(pred, axis = 1), num_classes = 10)
    print(confuse)
    print()
    print()
```

Confusion Matrix for channel5

The Confusion Matrix

WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python/ops/confusion_matrix.py:193: to_int64 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python/ops/confusion_matrix.py:194: 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.

```
tf.Tensor(
[[26  0  0  0  0  0  0  0  0  0]
 [ 0 25  0  0  0  0  0  0  0  0]
 [ 0  0 23  1  0  0  0  0  0  0]
 [ 0  0  0 23  0  0  0  0  0  0]
 [ 0  0  1  0 24  0  0  0  0  0]
 [ 0  0  0  0  0 19  3  0  0  0]
 [ 0  0  0  0  0  0 24  0  0  0]
 [ 0  0  0  1  0  0  0 27  0  0]
 [ 0  0  0  0  0  0  0  0 20  0]
 [ 0  0  0  0  0  0  0  1  0 18]], shape=(10, 10), dtype=int32)
```

Confusion Matrix for channel3

The Confusion Matrix

```
tf.Tensor(
[[24  0  1  1  0  0  0  0  0  0]
 [ 0 25  0  0  0  0  0  0  0  0]
 [ 0  0 23  0  0  0  0  1  0  0]
 [ 0  0  0 21  0  2  0  0  0  0]
 [ 0  0  1  0 24  0  0  0  0  0]
 [ 0  0  1  0  0 18  3  0  0  0]
 [ 0  0  0  0  0  0 24  0  0  0]
 [ 0  0  0  1  0  0  0 27  0  0]
 [ 0  0  0  0  0  0  0  0 20  0]
 [ 0  0  0  0  0  0  0  2  0 17]], shape=(10, 10), dtype=int32)
```

Confusion Matrix for channel4

The Confusion Matrix

```
tf.Tensor(
[[20  0  0  0  2  4  0  0  0  0]
 [ 0 22  0  0  3  0  0  0  0  0]
 [ 0  0 23  1  0  0  0  0  0  0]
 [ 0  0  0 22  0  1  0  0  0  0]
 [ 0  0  0  0 24  1  0  0  0  0]
 [ 0  0  0  1  0 18  3  0  0  0]
 [ 0  0  0  0  0  2 22  0  0  0]
 [ 0  0  0  1  0  0  0 27  0  0]
 [ 0  0  0  0  0  1  0  0 19  0]
 [ 0  0  0  0  1  0  0  3  0 15]], shape=(10, 10), dtype=int32)
```

Confusion Matrix for channel6

The Confusion Matrix

```
tf.Tensor(
[[26  0  0  0  0  0  0  0  0  0]
 [ 0 25  0  0  0  0  0  0  0  0]
 [ 0  0 24  0  0  0  0  0  0  0]
 [ 0  0  0 21  0  2  0  0  0  0]
 [ 0  0  0  0 25  0  0  0  0  0]
 [ 0  0  1  0  0 18  3  0  0  0]
 [ 0  0  0  0  0  0 24  0  0  0]
 [ 0  0  0  2  1  0  0 25  0  0]
 [ 0  0  0  0  0  0  1  0 19  0]
 [ 0  0  0  0  0  0  0  0  0 19]], shape=(10, 10), dtype=int32)
```

```
In [19]: ### ensemble prediction
import scipy
from sklearn.metrics import accuracy_score
ensem_pred = [] ### initialize list for prediction fro each model
for index in range(len(model_list)):
    model = model_dict[model_list[index]]
    opt = tf.keras.optimizers.Adam()
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])
    ### each model predicts based on each channel
    for i in chan_list:
        prede = model.predict(chan_dict[i])
        ensem_pred.append(np.argmax(prede, axis = 1))
###The mode of each prediction is obtained
ensem_pred1 = np.array(ensem_pred)
ensem_pred1 = scipy.stats.mode(ensem_pred)[0]
```

```
In [21]: ensemble_accuracy = accuracy_score(np.argmax(ytest, axis = 1),
                                             scipy.stats.mode(ensem_pred1)[0].T)
print("The Ensemble Accuracy is ", ensemble_accuracy)
```

The Ensemble Accuracy is 0.9491525423728814

```
In [ ]: print("The ensemble Confusion Matrix")
confuse = tf.math.confusion_matrix(np.argmax(ytest,axis = 1),
                                   ensem_pred.T,
                                   num_classes = 10
                                   )
print(confuse)
```

In []: