Tensorflow

I used the Bank Authentication Data Set (https://archive.ics.uci.edu/ml/datasets/banknote+authentication) from the UCI repository.

The data consists of 5 columns:

- variance of Wavelet Transformed image (continuous)
- skewness of Wavelet Transformed image (continuous)
- curtosis of Wavelet Transformed image (continuous)
- entropy of image (continuous)
- class (integer)

Data

In [4]:

```
import pandas as pd
```

In [11]:

```
data = pd.read_csv('bank_note_data.csv')
```

In [6]:

```
data.head()
```

Out[6]:

	Image.Var	Image.Skew	Image.Curt	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

EDA

In [7]:

```
import seaborn as sns
%matplotlib inline
```

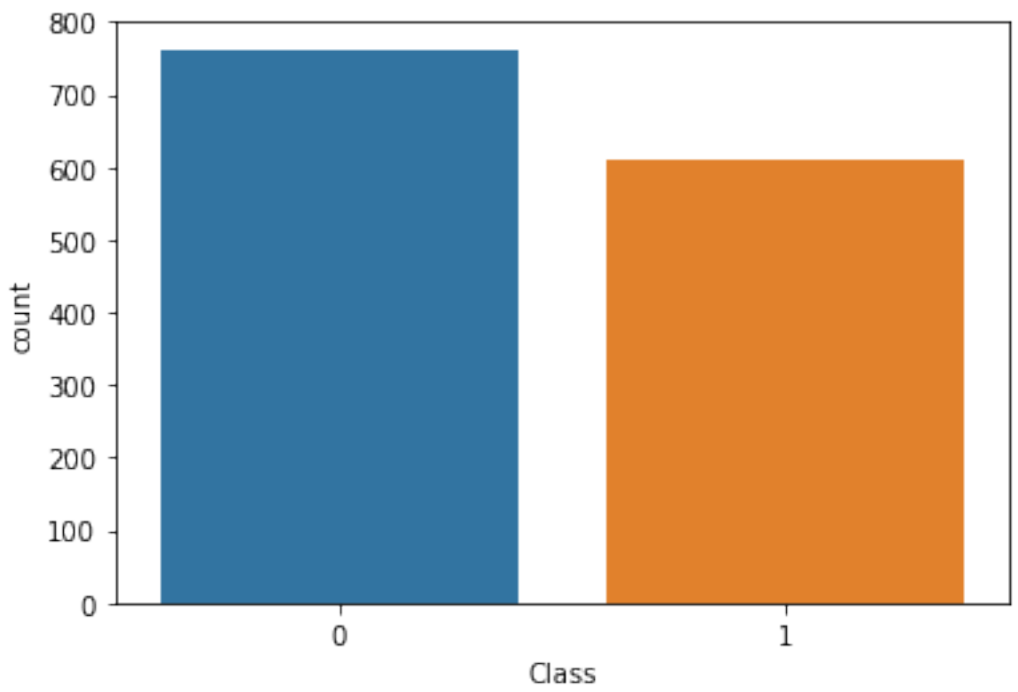
Countplot of the Classes (Authentic 1 vs Fake 0)

In [10]:

```
sns.countplot(x='Class',data=data)
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1232ac90>
```

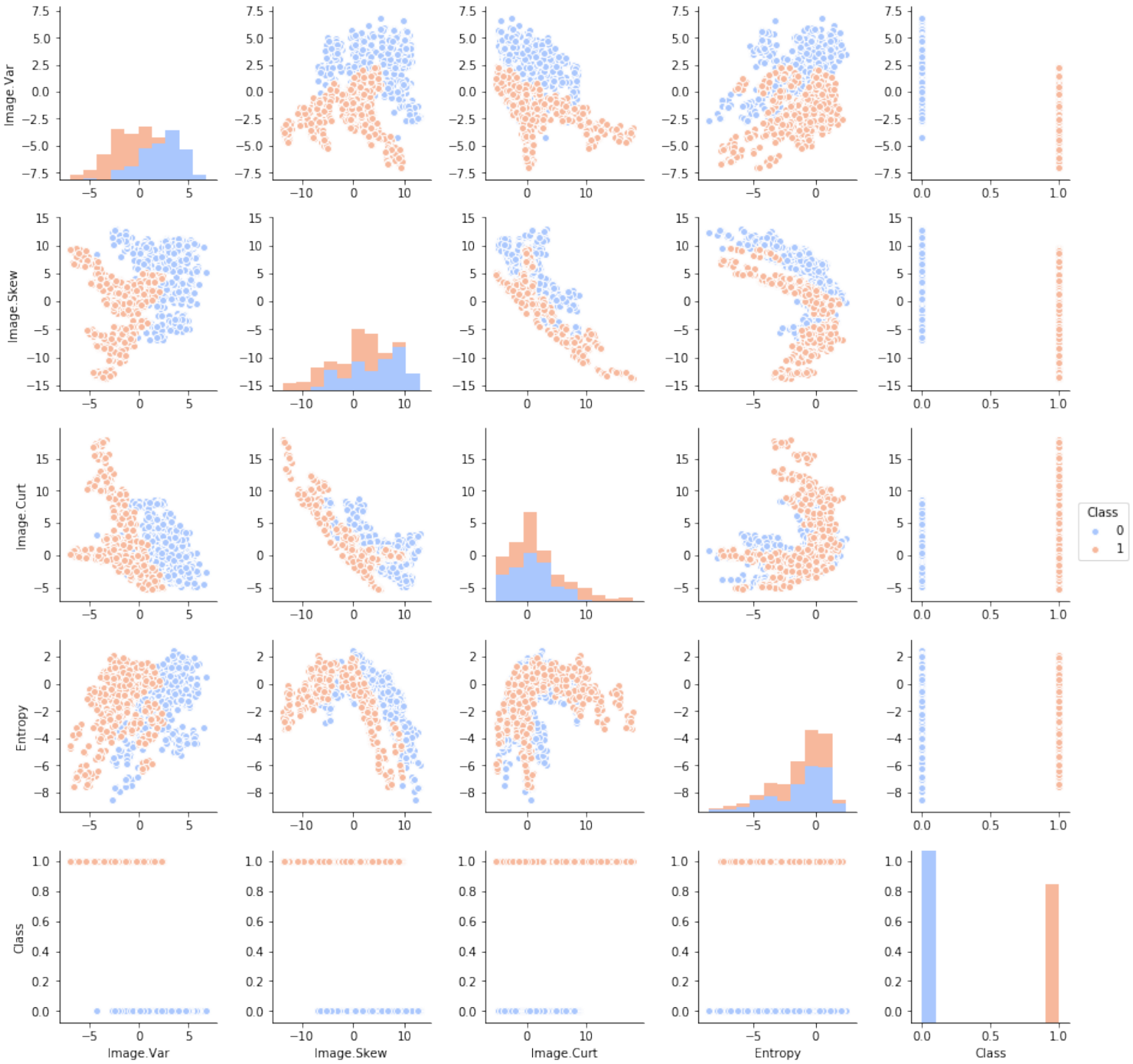


In [9]:

```
sns.pairplot(data,hue='Class',palette='coolwarm')
```

Out[9]:

<seaborn.axisgrid.PairGrid at 0x1a114af750>



Data Preparation

Standardize the data

Standard Scaling

In [12]:

```
from sklearn.preprocessing import StandardScaler
```

In [13]:

```
scaler = StandardScaler()
```

Fit scaler to the features.

In [14]:

```
scaler.fit(data.drop('Class',axis=1))
```

Out[14]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

Transforming the features to a scaled version.

In [15]:

```
scaled_features = scaler.fit_transform(data.drop('Class',axis=1))
```

In [16]:

```
df_feat = pd.DataFrame(scaled_features,columns=data.columns[:-1])
df_feat.head()
```

Out[16]:

	Image.Var	Image.Skew	Image.Curt	Entropy
0	1.121806	1.149455	-0.975970	0.354561
1	1.447066	1.064453	-0.895036	-0.128767
2	1.207810	-0.777352	0.122218	0.618073
3	1.063742	1.295478	-1.255397	-1.144029
4	-0.036772	-1.087038	0.736730	0.096587

Train Test Split

In [79]:

```
X = df_feat
```

In [80]:

```
y = data['Class']
```

Use the .as_matrix() method on X and Y and reset them equal to this result. Inorder for TensorFlow to accept the data in Numpy array form instead of a pandas series.

In [81]:

```
X = X.as_matrix()
y = y.as_matrix()
```

In [45]:

```
from sklearn.cross_validation import train_test_split
```

In [46]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Contrib.learn

In [19]:

```
import tensorflow.contrib.learn as learn
```

In [ ]:

```
classifier = learn.DNNClassifier(hidden_units=[10, 20, 10], n_classes=2)
```

In [94]:

```
classifier.fit(X_train, y_train, steps=200, batch_size=20)
```

```
/Users/marci/anaconda/lib/python3.5/site-packages/tensorflow/python/
ops/array_ops.py:1197: VisibleDeprecationWarning: converting an arra
y with ndim > 0 to an index will result in an error in the future
  result_shape.insert(dim, 1)
```

Out[94]:

```
DNNClassifier()
```

Model Evaluation

In [95]:

```
note_predictions = classifier.predict(X_test)
```

In [96]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [97]:

```
print(confusion_matrix(y_test,note_predictions))
```

```
[[237   0]
 [  1 174]]
```

In [98]:

```
print(classification_report(y_test,note_predictions))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       237
          1       1.00      0.99      1.00       175

avg / total       1.00      1.00      1.00       412
```

Comparison

Using SciKit Learn to create a Random Forest Classifier and comparing the confusion matrix and classification report to the DNN model

In [99]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [100]:

```
rfc = RandomForestClassifier(n_estimators=200)
```

In [101]:

```
rfc.fit(X_train,y_train)
```

Out[101]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion=
'gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None
,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=1
,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [102]:

```
rfc_preds = rfc.predict(X_test)
```

In [103]:

```
print(classification_report(y_test,rfc_preds))
```

```
             precision    recall  f1-score   support

          0       1.00      0.98      0.99       237
          1       0.98      0.99      0.99       175

avg / total       0.99      0.99      0.99       412
```

In [104]:

```
print(confusion_matrix(y_test,rfc_preds))
```

```
[[233   4]
 [  1 174]]
```