```
import os
import tempfile
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from termcolor import colored as cl
import itertools
import tensorflow as tf
from tensorflow import keras
import seaborn as sns
import sklearn
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
mpl.rcParams['figure.figsize'] = (12, 10)
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
```

```
!curl -L -O "https://training-images-4995.s3.amazonaws.com/data.zip"
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 69.0M  100 69.0M    0     0  28.7M      0  0:00:02  0:00:02 --:--:-- 28.7M
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

```
!unzip -q "data.zip" -d .
```

```
!head -n 3 processed_creditcards.csv
```

```
,V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,V11,V12,V13,V14,V15,V16,V17,V18,V19,V20,V21,V22,
0,-1.3598071336738,-0.0727811733098497,2.53634673796914,1.37815522427443,-0.3383
1,1.1918571113148602,0.26615071205963,0.16648011335321,0.448154078460911,0.06001
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```
df = pd.read_csv('processed_creditcards.csv')
df.head()
```

	Unnamed: 0	V1	V2	V3	V4	V5	V6	V7	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.
1	1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.
2	2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.
3	3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.
4	4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.

```
total_cases = len(df)
valid_transaction_count = len(df[df.Class == 0])
invalid_transaction_count = len(df[df.Class == 1])
perc_fraudulent = round(fraud_count/nonfraud_count*100, 2)


print("There are", total_cases, "Total Cases")
print("Total", valid_transaction_count, "Not frauduelnt cases")
print("Total", invalid_transaction_count, "actual frauduelnt cases")
print("RESULT = 100*(", invalid_transaction_count, "/",total_cases, ") = ",  100*inva
```

```
There are 284807 Total Cases
Total 284315 Not frauduelnt cases
Total 492 actual frauduelnt cases
RESULT = 100*( 492 / 284807 ) =  0.1727485630620034 PERCENT Fraudulent
```

```
X = df.values
y = df['Class'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15, random_st
```

```
print("X =", X.size)
print("Y =", y.size)
print("X_train=",X_train.size)
print("X_test=",X_test.size)
print("y_train=",y_train.size)
print("y_test=",y_test.size)
```

```
X = 8829017
Y = 284807
X_train= 7504635
X_test= 1324382
```

```
y_train= 242085
y test= 42722
```

```
res = []
tree_max_n = 2
tree_max_acc = 0
tree_max_f1 = 0
tree_knn_yhat = None
for i in range(1,10):
  tree_model = DecisionTreeClassifier(max_depth = i, criterion = 'entropy')
  tree_model.fit(X_train, y_train)
  tree_yhat_loc = tree_model.predict(X_test)
  tree_acc = accuracy_score(y_test, tree_yhat_loc)
  tree_f1 = f1_score(y_test, tree_yhat_loc)
  if tree_acc > tree_max_acc and tree_f1 > tree_max_f1:
    tree_max_acc = tree_acc
    tree_knn_yhat = tree_yhat_loc
    tree_max_f1 = tree_f1
    tree_max_n = i

print("max acc", tree_max_acc)
print("max f1", tree_max_f1)
print("max n", tree_max_n)
# max index 0
```

```
max acc 0.9994557775359011
max f1 0.835978835978836
max n 6
```

```
xgb = XGBClassifier(max_depth = 4)
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)


print(accuracy_score(y_test, xgb_yhat))
print(f1_score(y_test, xgb_yhat))
```

```
0.9994733330992591
0.8421052631578948
```

```
rf = RandomForestClassifier(max_depth = 4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
rf_max_n = 0
rf_max_acc = 0
rf_max_f1 = 0
rf_knn_yhat = None
for i in range(2,10):
  rf = RandomForestClassifier(max_depth = i)
  rf.fit(X_train, y_train)
  rf_yhat_l = rf.predict(X_test)
```

```
  rf_acc = accuracy_score(y_test, knn_yhat_l)
  rf_f1 = f1_score(y_test, knn_yhat_l)
  if rf_acc > rf_max_acc and rf_f1 > rf_max_f1:
    rf_max_acc = rf_acc
    rf_yhat = rf_yhat_l
    rf_max_f1 = rf_f1
    rf_max_n = i
```

```
clg = GaussianNB()
clg.fit(X_train, y_train)
clg_yhat = clg.predict(X_test)
print(accuracy_score(y_test, clg_yhat))
print(f1_score(y_test, clg_yhat))
```

```
0.9783364348161933
0.12233285917496445
```

```
res = []
knn_max_n = 0
knn_max_acc = 0
knn_max_f1 = 0
knn_knn_yhat = None
for i in range(2,10):
  knn = KNeighborsClassifier(n_neighbors = i)
  knn.fit(X_train, y_train)
  knn_yhat_l = knn.predict(X_test)
  res.append(knn_yhat_l)
  knn_acc = accuracy_score(y_test, knn_yhat_l)
  knn_f1 = f1_score(y_test, knn_yhat_l)
  if knn_acc > knn_max_acc and knn_f1 > knn_max_f1:
    knn_max_acc = knn_acc
    knn_yhat = knn_yhat_l
    knn_max_f1 = knn_f1
    knn_max_n = i

print("max acc", knn_max_acc)
print("max f1", knn_max_f1)
print("max n", knn_max_n)
# max index 0
```

```
max acc 0.999403110845827
max f1 0.8089887640449437
max n 2
```

```
for el in res:
  print(accuracy_score(y_test, el))
  print(f1_score(y_test, el))
```

```
print(f1_score(y_test, e1))
```

```
0.999403110845827
0.8089887640449437
0.999385555282469
0.8066298342541437
0.9993504441557529
0.7909604519774012
0.9993328885923949
0.7865168539325842
0.9992977774656788
0.7701149425287357
0.9993153330290369
0.7771428571428572
0.9992451107756047
0.7485380116959064
0.9992451107756047
0.7485380116959064
```

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)
print(accuracy_score(y_test, lr_yhat))
print(f1_score(y_test, lr_yhat))
```

```
0.9992451107756047
0.7542857142857143
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Cc
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressic
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
svm = SVC()
svm.fit(X_train, y_train)
svm_yhat = svm.predict(X_test)
```

```
print(accuracy_score(y_test, svm_yhat))
print(f1_score(y_test, svm_yhat))
```

```
0.998735999438222
0.5
```

```
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues)
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
```

```
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
file = tf.keras.utils
raw_df = pd.read_csv('creditcard 2.csv')
raw_df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053

```
cleaned_df = raw_df.copy()
cleaned_df.pop('Time')
eps = 0.001
cleaned_df['Log Ammount'] = np.log(cleaned_df.pop('Amount')+eps)
```

```
train_df, test_df = train_test_split(cleaned_df, test_size=0.2)
train_df, val_df = train_test_split(train_df, test_size=0.2)

train_labels = np.array(train_df.pop('Class'))
bool_train_labels = train_labels != 0
val_labels = np.array(val_df.pop('Class'))
test_labels = np.array(test_df.pop('Class'))

train_features = np.array(train_df)
val_features = np.array(val_df)
test_features = np.array(test_df)
```

```
scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)

val_features = scaler.transform(val_features)
test_features = scaler.transform(test_features)

train_features = np.clip(train_features, -5, 5)
val_features = np.clip(val_features, -5, 5)
test_features = np.clip(test_features, -5, 5)


print('Training labels shape:', train_labels.shape)
print('Validation labels shape:', val_labels.shape)
print('Test labels shape:', test_labels.shape)

print('Training features shape:', train_features.shape)
print('Validation features shape:', val_features.shape)
print('Test features shape:', test_features.shape)
```

```
Training labels shape: (182276,)
Validation labels shape: (45569,)
Test labels shape: (56962,)
Training features shape: (182276, 29)
Validation features shape: (45569, 29)
Test features shape: (56962, 29)
```

```
METRICS = [
      keras.metrics.TruePositives(name='tp'),
      keras.metrics.FalsePositives(name='fp'),
      keras.metrics.TrueNegatives(name='tn'),
      keras.metrics.FalseNegatives(name='fn'),
      keras.metrics.BinaryAccuracy(name='accuracy'),
      keras.metrics.Precision(name='precision'),
      keras.metrics.Recall(name='recall'),
      keras.metrics.AUC(name='auc'),
      keras.metrics.AUC(name='prc', curve='PR'), # precision-recall curve
]

def make_model(metrics=METRICS, output_bias=None):
  if output_bias is not None:
    output_bias = tf.keras.initializers.Constant(output_bias)
  model = keras.Sequential([
      keras.layers.Dense(
          16, activation='relu',
          input_shape=(train_features.shape[-1],)),
      keras.layers.Dropout(0.5),
      keras.layers.Dense(1, activation='sigmoid',
                         bias_initializer=output_bias),
  ])
```

```
  model.compile(
      optimizer=keras.optimizers.Adam(lr=1e-3),
      loss=keras.losses.BinaryCrossentropy(),
      metrics=metrics)

  return model

EPOCHS = 100
BATCH_SIZE = 2048

early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_prc',
    verbose=1,
    patience=10,
    mode='max',
    restore_best_weights=True)
model = make_model()
model.summary()
model.predict(train_features[:10])


initial_weights = os.path.join(tempfile.mkdtemp(), 'initial_weights')
model.save_weights(initial_weights)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 16)                480
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 17
=================================================================
Total params: 497
Trainable params: 497
Non-trainable params: 0
_________________________________________________________________
```

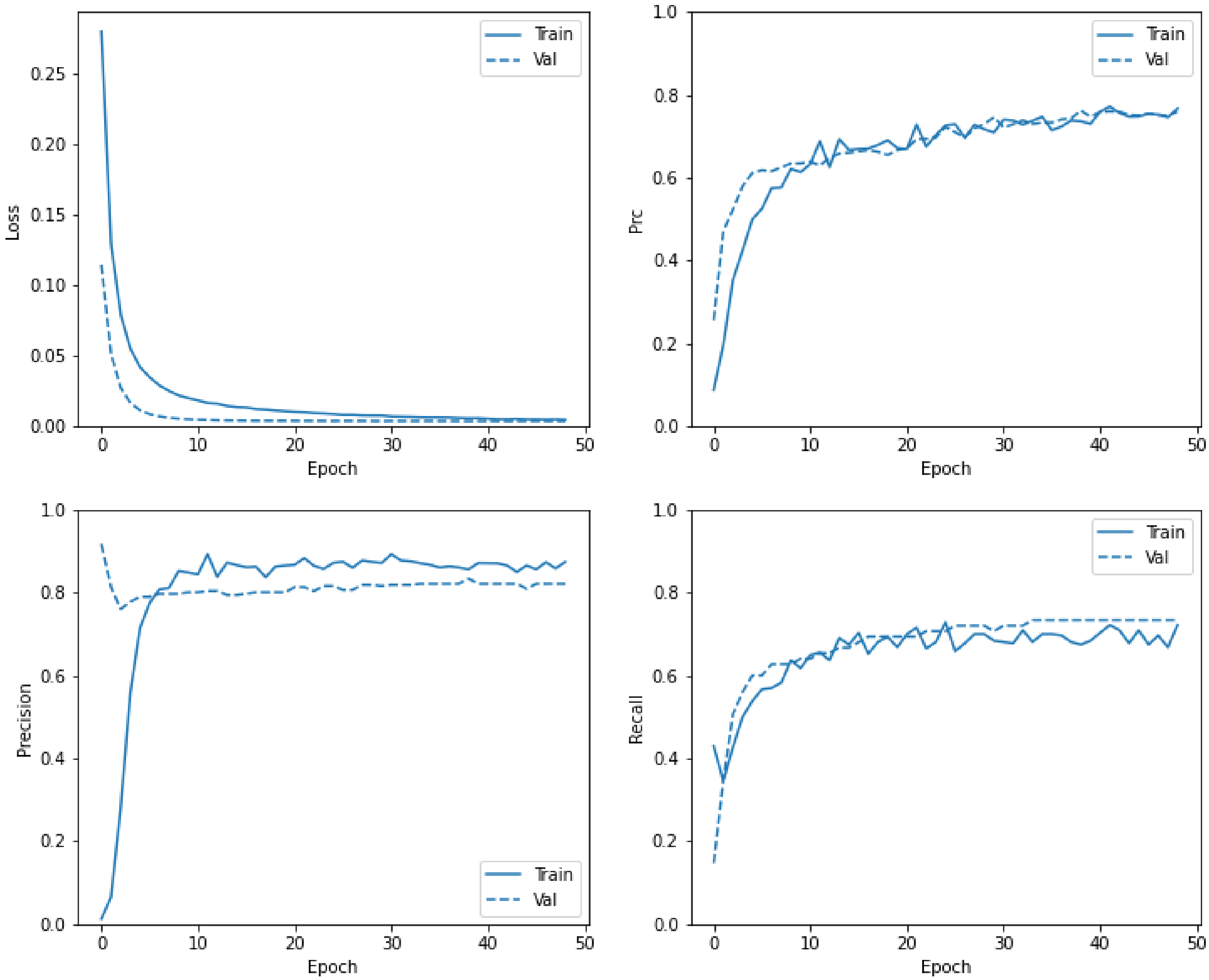
```
model = make_model()
model.load_weights(initial_weights)
baseline_history = model.fit(
    train_features,
    train_labels,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    callbacks=[early_stopping],
    validation_data=(val_features, val_labels))
```

```
Epoch 7/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0308 - tp: 92.68
Epoch 8/100
90/90 [==============================] - 1s 10ms/step - loss: 0.0260 - tp: 92.31
```

```
Epoch 9/100
90/90 [==============================] - 1s 10ms/step - loss: 0.0229 - tp: 102.8
Epoch 10/100
90/90 [==============================] - 1s 10ms/step - loss: 0.0213 - tp: 97.14
Epoch 11/100
90/90 [==============================] - 1s 10ms/step - loss: 0.0191 - tp: 106.2
Epoch 12/100
90/90 [==============================] - 1s 10ms/step - loss: 0.0173 - tp: 106.9
Epoch 13/100
90/90 [==============================] - 1s 10ms/step - loss: 0.0171 - tp: 103.1
Epoch 14/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0145 - tp: 112.5
Epoch 15/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0142 - tp: 107.4
Epoch 16/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0134 - tp: 115.6
Epoch 17/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0127 - tp: 102.3
Epoch 18/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0119 - tp: 105.0
Epoch 19/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0117 - tp: 112.1
Epoch 20/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0101 - tp: 105.7
Epoch 21/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0097 - tp: 119.4
Epoch 22/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0101 - tp: 116.6
Epoch 23/100
90/90 [==============================] - 1s 10ms/step - loss: 0.0100 - tp: 106.1
Epoch 24/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0096 - tp: 103.1
Epoch 25/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0088 - tp: 123.1
Epoch 26/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0078 - tp: 101.5
Epoch 27/100
90/90 [==============================] - 1s 10ms/step - loss: 0.0086 - tp: 109.0
Epoch 28/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0076 - tp: 121.2
Epoch 29/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0078 - tp: 115.5
Epoch 30/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0079 - tp: 112.0
Epoch 31/100

90/90 [==============================] - 1s 10ms/step - loss: 0.0077 - tp: 105.7
Epoch 32/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0073 - tp: 107.7
Epoch 33/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0066 - tp: 112.2
Epoch 34/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0072 - tp: 110.4
Epoch 35/100
90/90 [==============================] - 1s 11ms/step - loss: 0.0067 - tp: 113.9
Epoch 36/100
```

```
def plot_metrics(history):
```

```
    metrics = ['loss', 'prc', 'precision', 'recall']
    for n, metric in enumerate(metrics):
      name = metric.replace("_"," ").capitalize()
      plt.subplot(2,2,n+1)
      plt.plot(history.epoch, history.history[metric], color=colors[0], label='Train')
      plt.plot(history.epoch, history.history['val_'+metric],
               color=colors[0], linestyle="--", label='Val')
      plt.xlabel('Epoch')
      plt.ylabel(name)
      if metric == 'loss':
        plt.ylim([0, plt.ylim()[1]])
      elif metric == 'auc':
        plt.ylim([0.8,1])
      else:
        plt.ylim([0,1])

      plt.legend()
  plot_metrics(baseline_history)
```

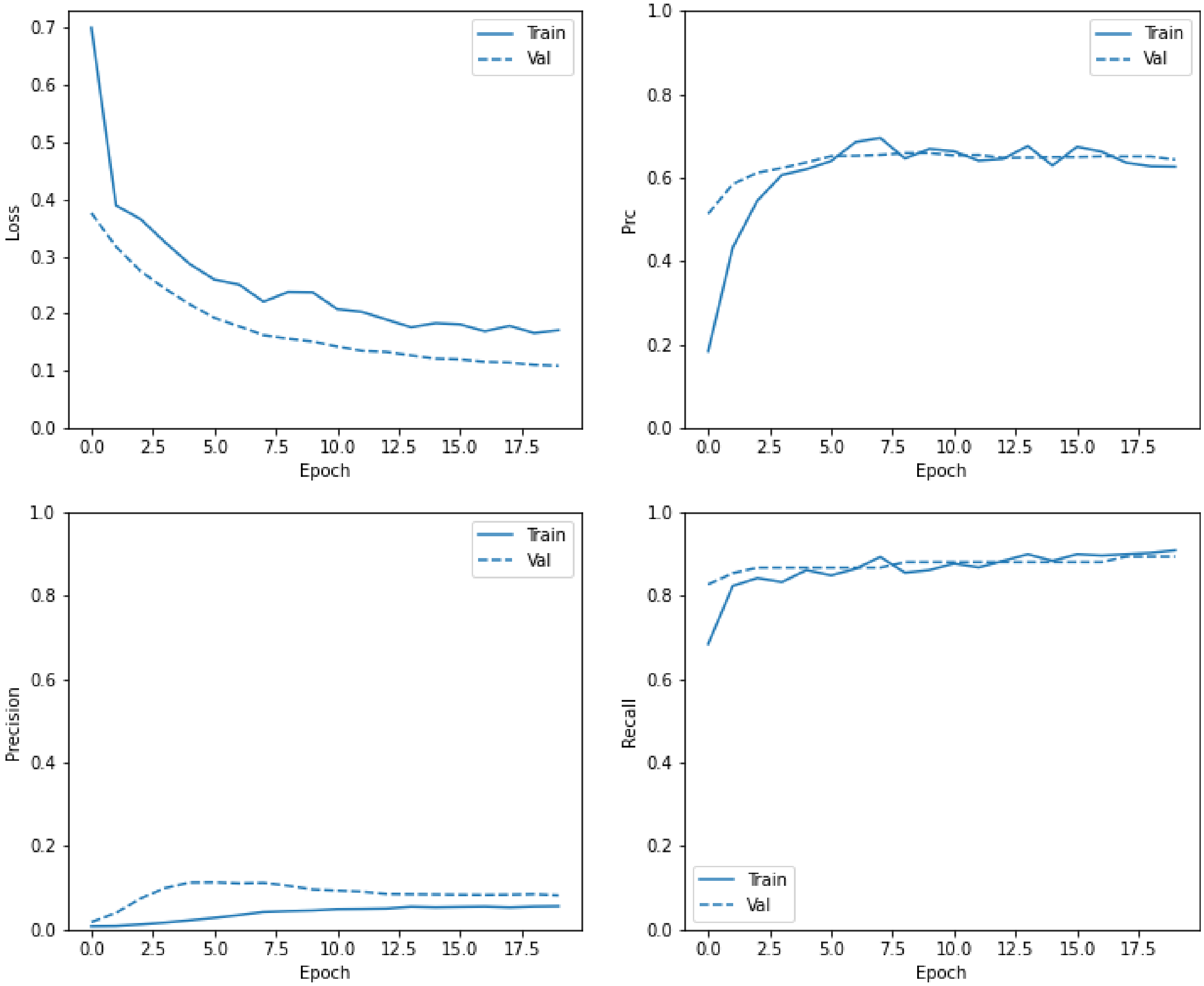


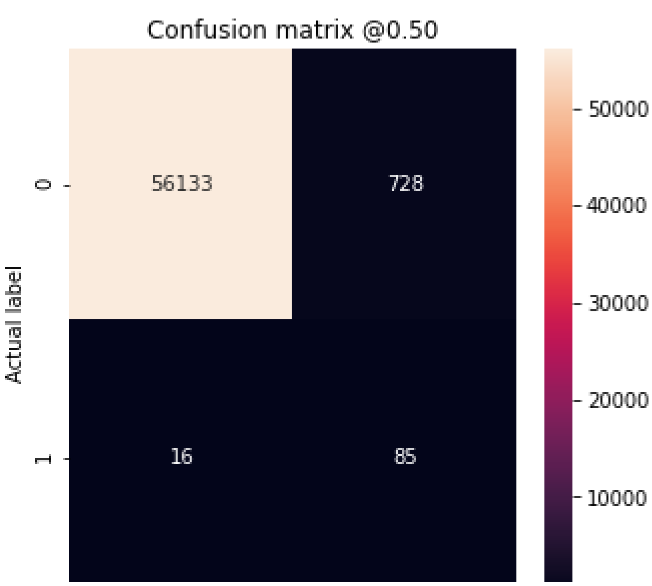
```
train_predictions_baseline = model.predict(train_features, batch_size=BATCH_SIZE)
test_predictions_baseline = model.predict(test_features, batch_size=BATCH_SIZE)
def plot_cm(labels, predictions, p=0.5):
  cm = confusion_matrix(labels, predictions > p)
  plt.figure(figsize=(5,5))
  sns.heatmap(cm, annot=True, fmt="d")
  plt.title('Confusion matrix @{:.2f}'.format(p))
  plt.ylabel('Actual label')
  plt.xlabel('Predicted label')

  print('Legitimate Transactions Detected (True Negatives): ', cm[0][0])
  print('Legitimate Transactions Incorrectly Detected (False Positives): ', cm[0][1])
  print('Fraudulent Transactions Missed (False Negatives): ', cm[1][0])
  print('Fraudulent Transactions Detected (True Positives): ', cm[1][1])
  print('Total Fraudulent Transactions: ', np.sum(cm[1]))

baseline_results = model.evaluate(test_features, test_labels,
                                  batch_size=BATCH_SIZE, verbose=0)
for name, value in zip(model.metrics_names, baseline_results):
  print(name, ': ', value)
print()

plot_cm(test_labels, test_predictions_baseline)
```

```
loss :  0.004249707330018282
tp :  75.0
fp :  13.0
tn :  56848.0
fn :  26.0
accuracy :  0.9993153214454651
```

```
neg, pos = np.bincount(raw_df['Class'])
total = neg + pos
weight_for_0 = (1 / neg)*(total)/2.0
weight_for_1 = (1 / pos)*(total)/2.0

class_weight = {0: weight_for_0, 1: weight_for_1}

print('Weight for class 0: {:.2f}'.format(weight_for_0))
print('Weight for class 1: {:.2f}'.format(weight_for_1))
weighted_model = make_model()
weighted_model.load_weights(initial_weights)

weighted_history = weighted_model.fit(
    train_features,
    train_labels,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    callbacks=[early_stopping],
    validation_data=(val_features, val_labels),
    class_weight=class_weight)

plot_metrics(weighted_history)
train_predictions_weighted = weighted_model.predict(train_features, batch_size=BATCH_
test_predictions_weighted = weighted_model.predict(test_features, batch_size=BATCH_SI
weighted_results = weighted_model.evaluate(test_features, test_labels,
                                           batch_size=BATCH_SIZE, verbose=0)
for name, value in zip(weighted_model.metrics_names, weighted_results):
  print(name, ': ', value)
print()

plot_cm(test_labels, test_predictions_weighted)
```

```
auc :  0.9494603276252747
prc :  0.685690701007843

Legitimate Transactions Detected (True Negatives):  56133
Legitimate Transactions Incorrectly Detected (False Positives):  728
Fraudulent Transactions Missed (False Negatives):  16
Fraudulent Transactions Detected (True Positives):  85
Total Fraudulent Transactions:  101
```






```
print("Accuracy of XG Boost =", accuracy_score(y_test, xgb_yhat), "with F1 Score =",
print("Accuracy of CLG =", accuracy_score(y_test, clg_yhat), "with F1 Score =", f1_sc
print("Accuracy of NKK =", accuracy_score(y_test, knn_yhat), "with F1 Score =", f1_sc
print("Accuracy of Lin. Reg =", accuracy_score(y_test, lr_yhat), "with F1 Score =", f
print("Accuracy of SVM =", accuracy_score(y_test, svm_yhat), "with F1 Score =", f1_sc
print("Accuracy of Decision Tree =", accuracy_score(y_test, tree_knn_yhat), "with F1
print("Accuracy of Random Forest =", accuracy_score(y_test, rf_yhat), "with F1 Score
```

```
Accuracy of XG Boost = 0.9994733330992591 with F1 Score = 0.8421052631578948
Accuracy of CLG = 0.9783364348161933 with F1 Score = 0.12233285917496445
```

```
Accuracy of NKK = 0.999403110845827 with F1 Score = 0.8089887640449437
Accuracy of Lin. Reg = 0.9992451107756047 with F1 Score = 0.7542857142857143
Accuracy of SVM = 0.998735999438222 with F1 Score = 0.5
Accuracy of Decision Tree = 0.9994557775359011 with F1 Score = 0.835978835978836
Accuracy of Random Forest = 0.9990519995786665 with F1 Score = 0.674698795180723
```

```
xgb_matrix = confusion_matrix(y_test, xgb_yhat, labels = [0, 1])
clg_matrix = confusion_matrix(y_test, clg_yhat, labels = [0, 1])
knn_matrix = confusion_matrix(y_test, knn_yhat, labels = [0, 1])
lr_matrix = confusion_matrix(y_test, lr_yhat, labels = [0, 1])
svm_matrix = confusion_matrix(y_test, svm_yhat, labels = [0, 1])
tree_matrix = confusion_matrix(y_test, tree_knn_yhat, labels = [0, 1])
rf_matrix = confusion_matrix(y_test, rf_yhat, labels = [0, 1])

plt.rcParams['figure.figsize'] = (6, 6)

knn_cm_plot = plot_confusion_matrix(xgb_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'XG Boost')

plt.show()
CLG_plot = plot_confusion_matrix(clg_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'CLG')
plt.show()
knn_cm_plot = plot_confusion_matrix(knn_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = '5/K-NN')
plt.show()
lreg_cm_plot = plot_confusion_matrix(lr_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'Linear Reg.')
plt.show()
svm_cm_plot = plot_confusion_matrix(svm_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'SVM')
plt.show()
dt_cm_plot = plot_confusion_matrix(tree_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'Decision Tree')
plt.show()
rf_cm_plot = plot_confusion_matrix(rf_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'Random Forest')

plt.show()
```