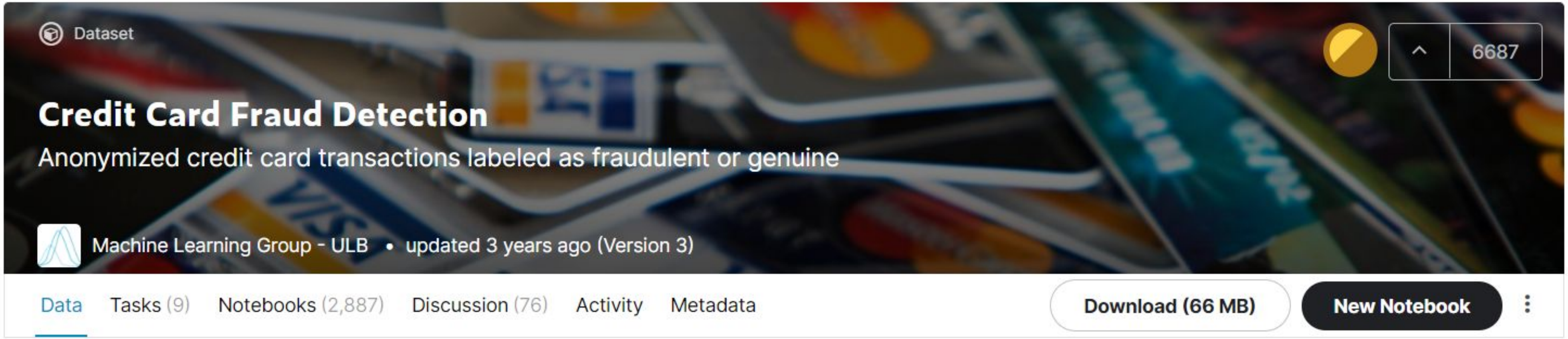
Credit Card Fraud Detection

Filipe do Vale Melo
CBPF

Introduction



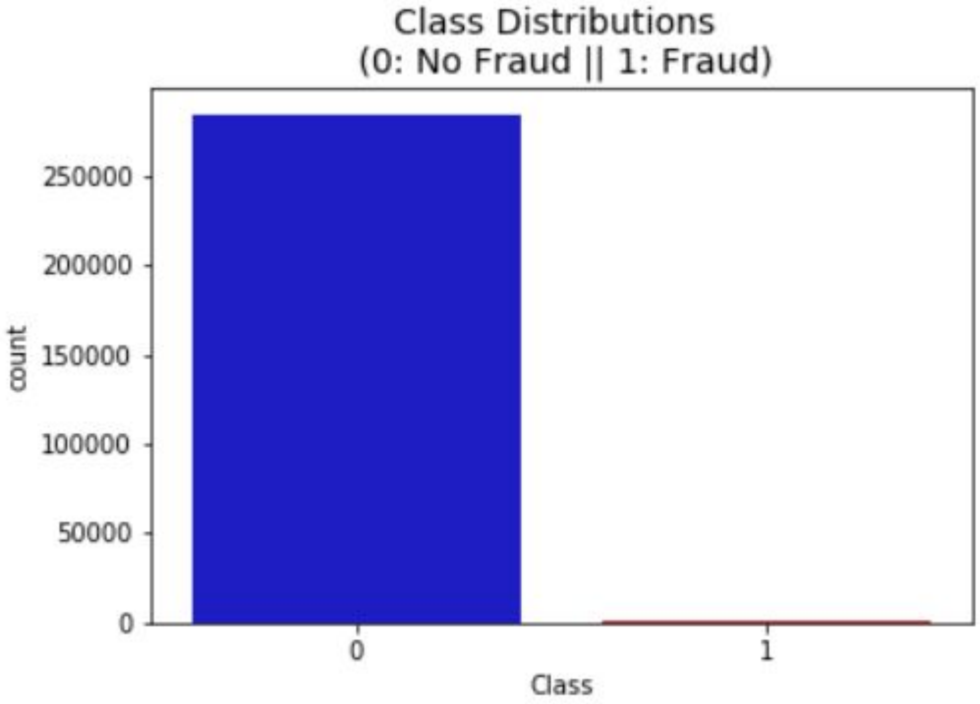
- The datasets contains transactions made by credit cards in September 2013 by european cardholders.
- The goal is to identify fraudulent transactions.
- The principal feature of this dataset is the imbalance between the classes: less than 0.2% of the dataset consists of fraudulent transactions.

Dataset

```
raw_df[['Time','Amount','V1','V2','V3','V26','V27','V28','Class']]
```

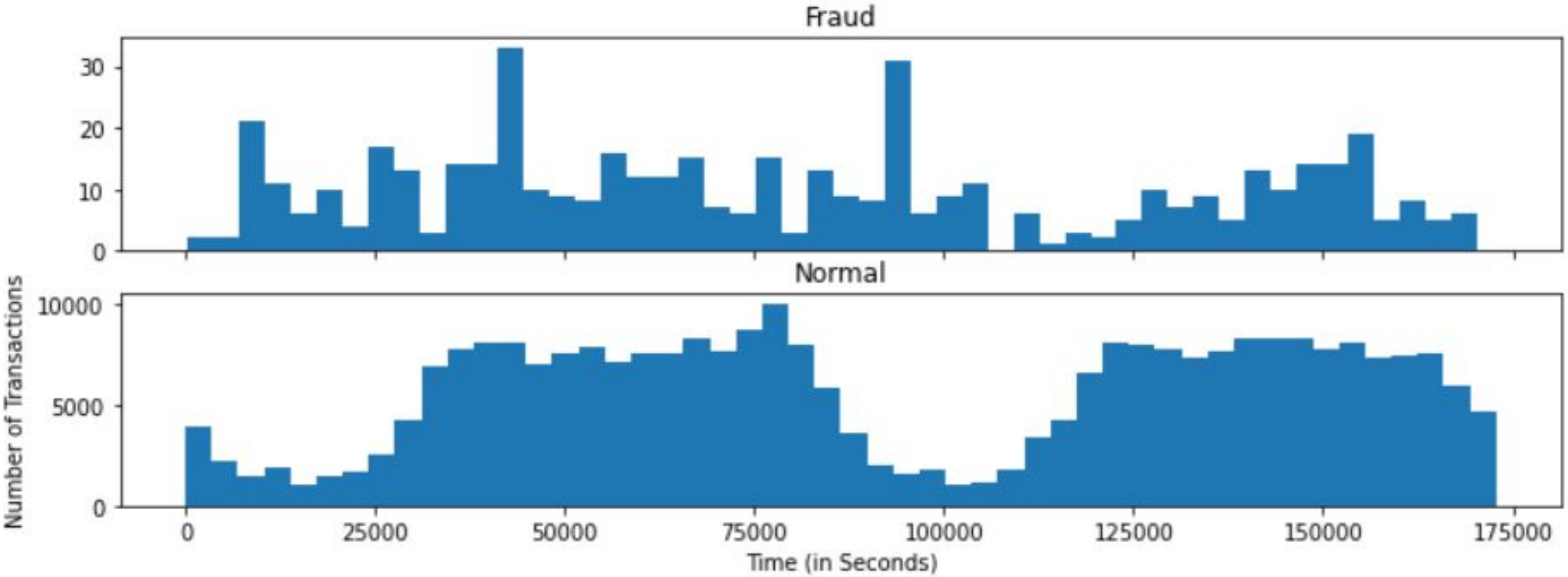
	Time	Amount	V1	V2	V3	V26	V27	V28	Class
0	0.0	149.62	-1.359807	-0.072781	2.536347	-0.189115	0.133558	-0.021053	0
1	0.0	2.69	1.191857	0.266151	0.166480	0.125895	-0.008983	0.014724	0
2	1.0	378.66	-1.358354	-1.340163	1.773209	-0.139097	-0.055353	-0.059752	0
3	1.0	123.50	-0.966272	-0.185226	1.792993	-0.221929	0.062723	0.061458	0
4	2.0	69.99	-1.158233	0.877737	1.548718	0.502292	0.219422	0.215153	0

Dataset

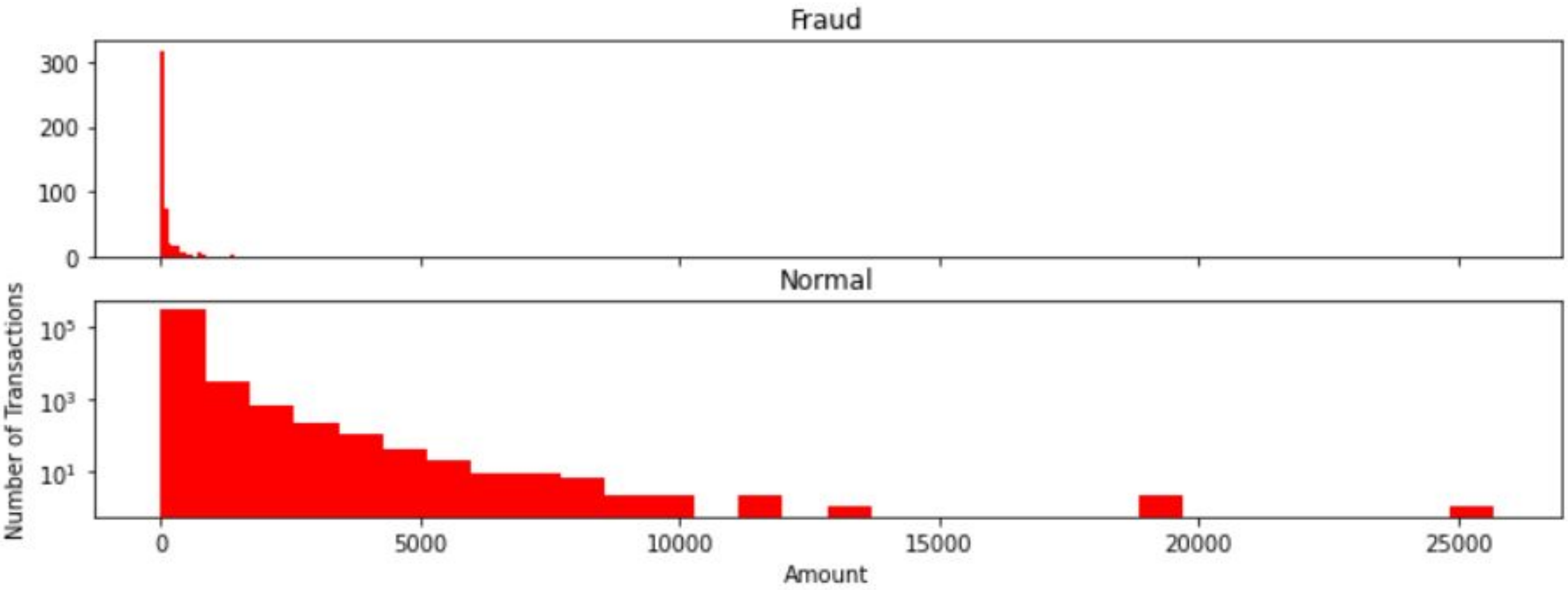



```
Total: 284807
Negative: 284315 (99.83% of total)
Positive: 492 (0.17% of total)
```

Exploratory Data Analysis (EDA)



Exploratory Data Analysis (EDA)



Standard Scaling

	Time	V1	V2	V3	V4
count	2.278450e+05	2.278450e+05	2.278450e+05	2.278450e+05	2.278450e+05
mean	5.842771e-17	-1.651849e-18	4.946046e-17	-2.680381e-17	2.466908e-17
std	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00

Solution proposals

- We are going to test three approaches to this problem:
    1. An approach that, during the training, does not take into account the fact that the dataset is imbalanced (baseline model).
    2. An approach that applies weights to the fraud samples.
    3. An approach trained with a dataset undersampled with the non-fraud samples in the dataset.
- We will compare the three approaches, plotting the confusion matrices (and measure precision and recall) and ROC/AUC to see which one performs better.

Metrics

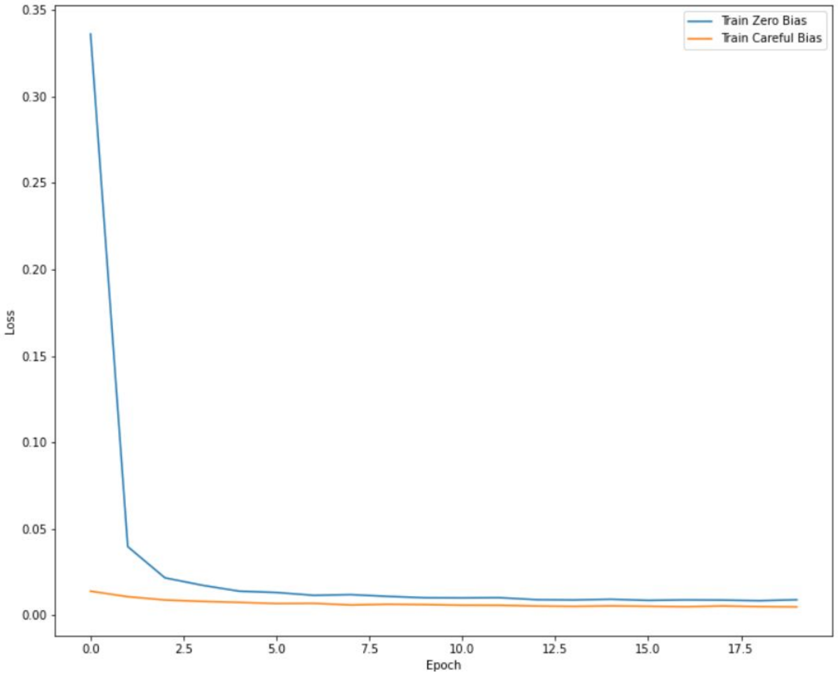
```
METRICS = [
      keras.metrics.TruePositives(name='tp'),
      keras.metrics.FalsePositives(name='fp'),
      keras.metrics.TrueNegatives(name='tn'),
      keras.metrics.FalseNegatives(name='fn'),
      keras.metrics.BinaryAccuracy(name='accuracy'),
      keras.metrics.Precision(name='precision'),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

Neural Network Architecture

```
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)
```

Bias Initialization

- In the the approaches with the unbalanced datasets, a careful bias initialization was used.



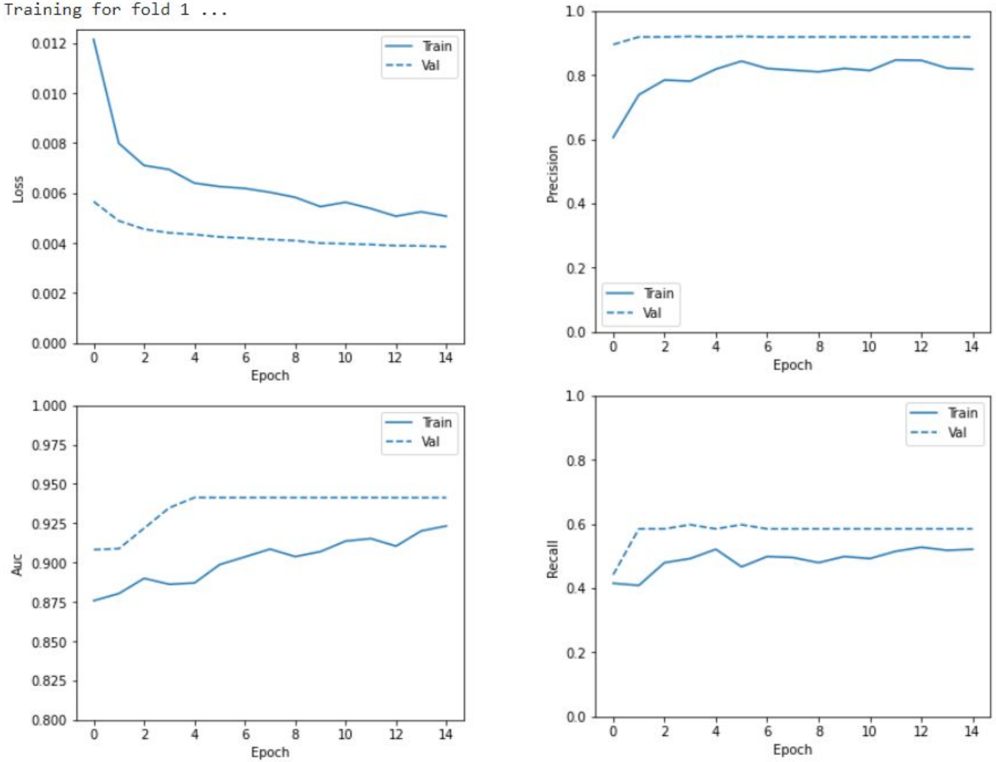
Baseline Model: Train-Test Split

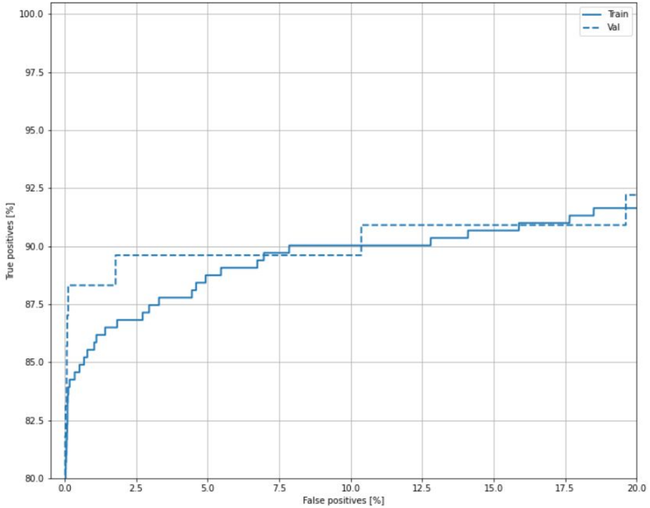
```python
# Here we use a utility from sklearn to help us split and shuffle the dataset.
train_df, test_df = train_test_split(raw_df, test_size=0.2)

train_df_features = train_df.copy()
test_df_features = test_df.copy()

# Convert the DataFrame into np arrays of labels and features.
train_df_labels = train_df_features.pop('Class')
test_df_labels = test_df_features.pop('Class')

# Count the negatives and positives
neg_train, pos_train = np.bincount(train_df_labels)
total_train = neg_train + pos_train

neg_test, pos_test = np.bincount(test_df_labels)
total_test = neg_test + pos_test

print('Train:\n  Total: {}\n  Negative: {} ({:.2f}% of total)\n  Positive: {} ({:.2f}% of total)\n'.format(
    total_train, neg_train, 100 * neg_train / total_train, pos_train, 100 * pos_train / total_train))

print('Test:\n  Total: {}\n  Negative: {} ({:.2f}% of total)\n  Positive: {} ({:.2f}% of total)\n'.format(
    total_test, neg_test, 100 * neg_test / total_test, pos_test, 100 * pos_test / total_test))
```

```
Train:
  Total: 227845
  Negative: 227457 (99.83% of total)
  Positive: 388 (0.17% of total)

Test:
  Total: 56962
  Negative: 56858 (99.82% of total)
  Positive: 104 (0.18% of total)
```

Baseline model: Cross Validation

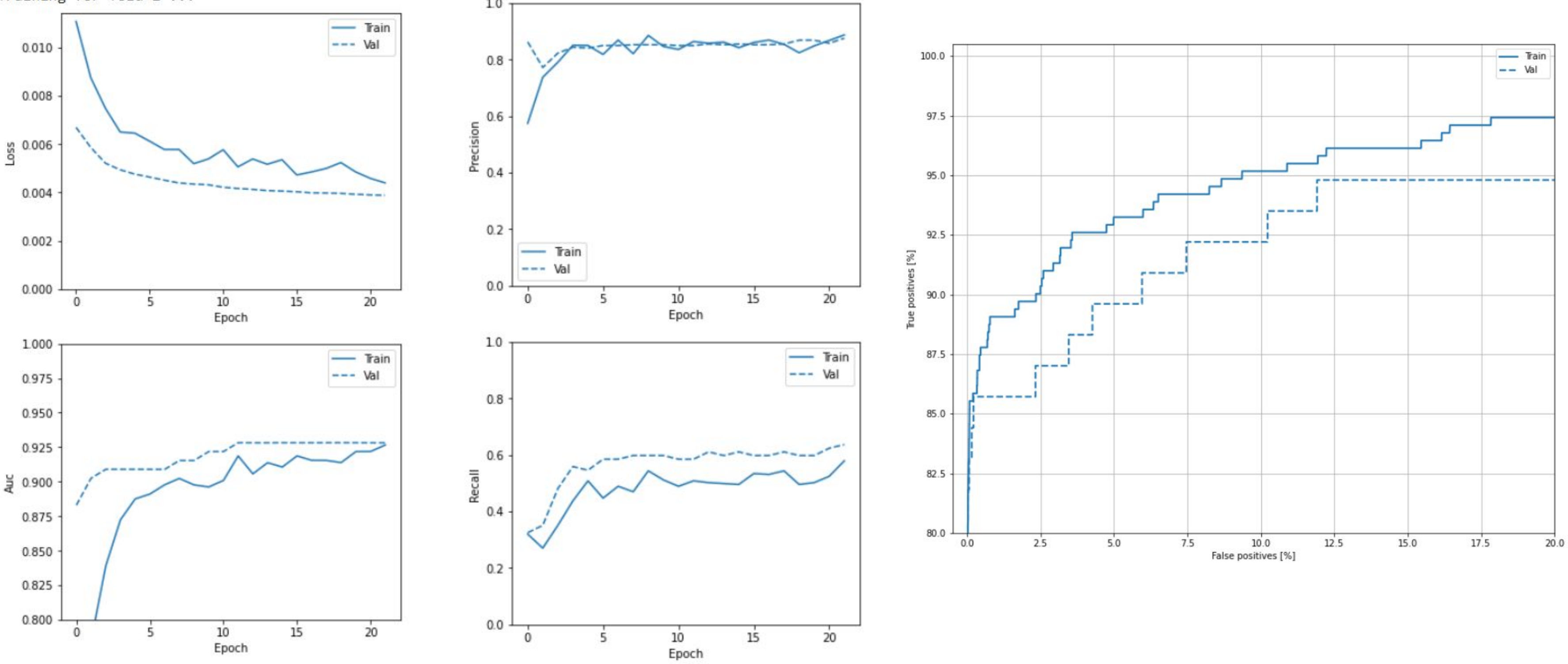
Training for fold 1 ...





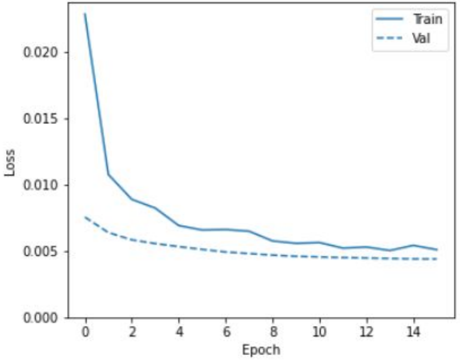
Baseline model: Cross Validation

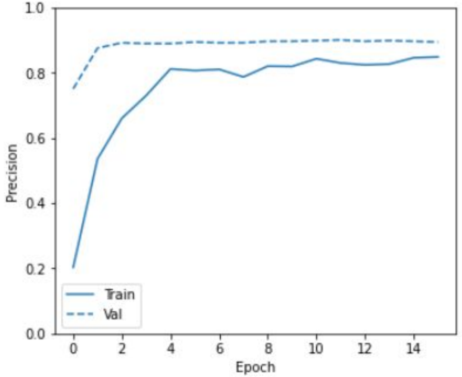
Training for fold 2 ...

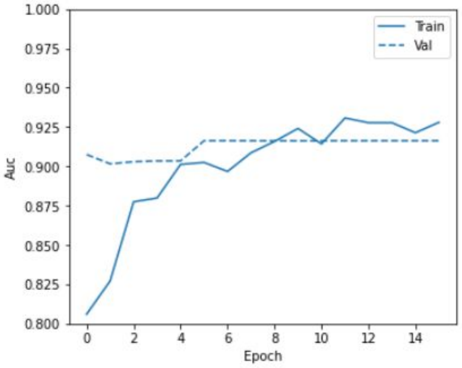


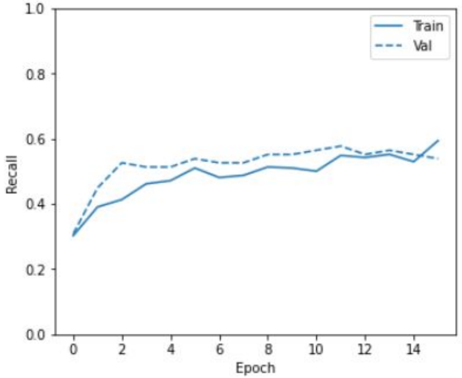
Baseline model: Cross Validation

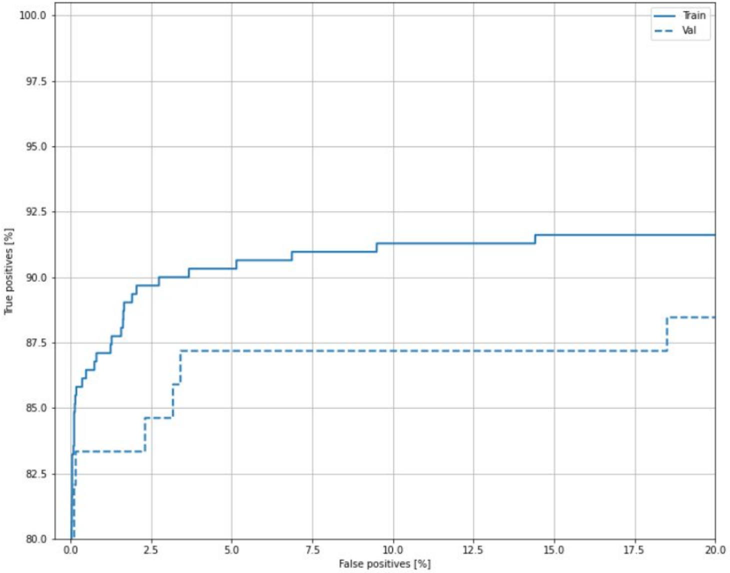
Training for fold 3 ...





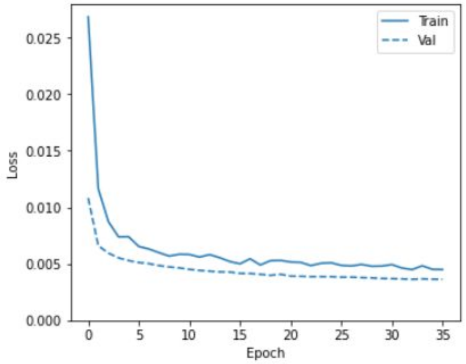


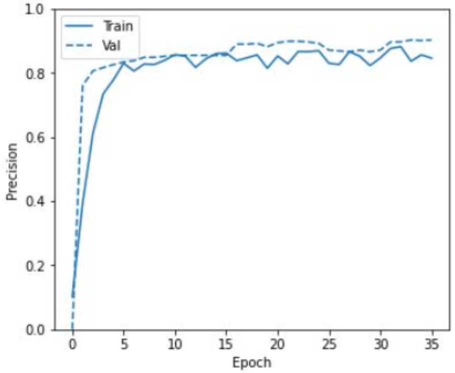


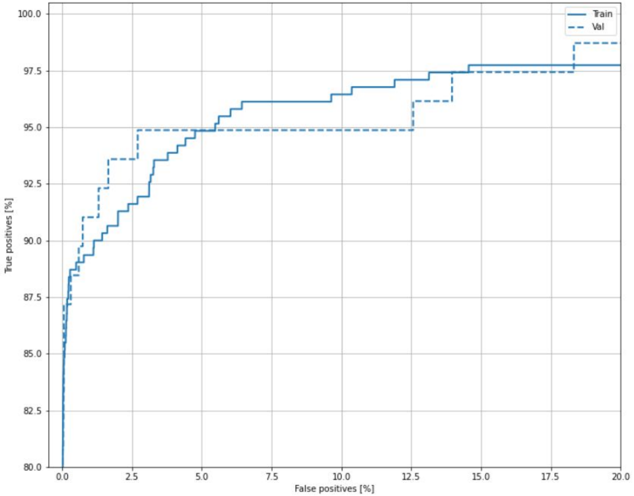


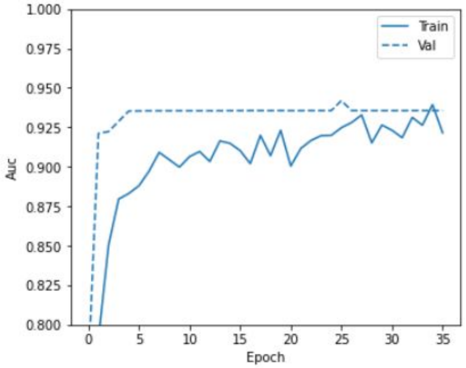
Baseline model: Cross Validation

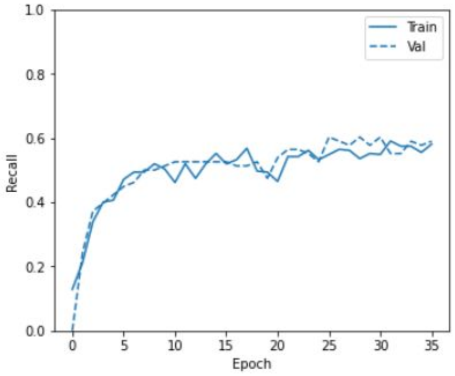
Training for fold 4 ...





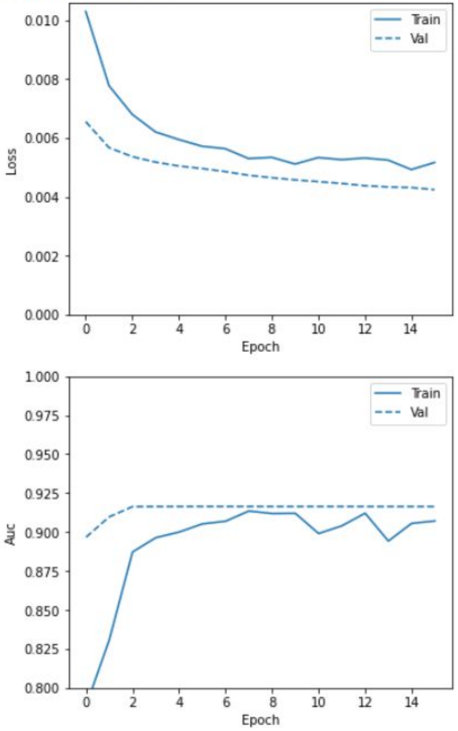


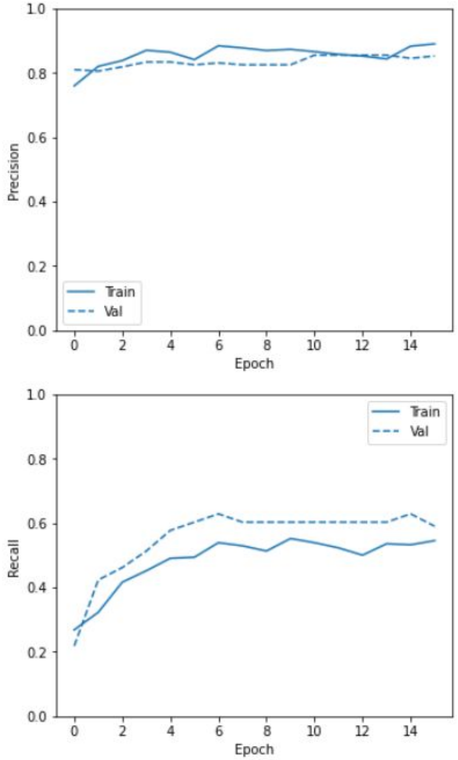


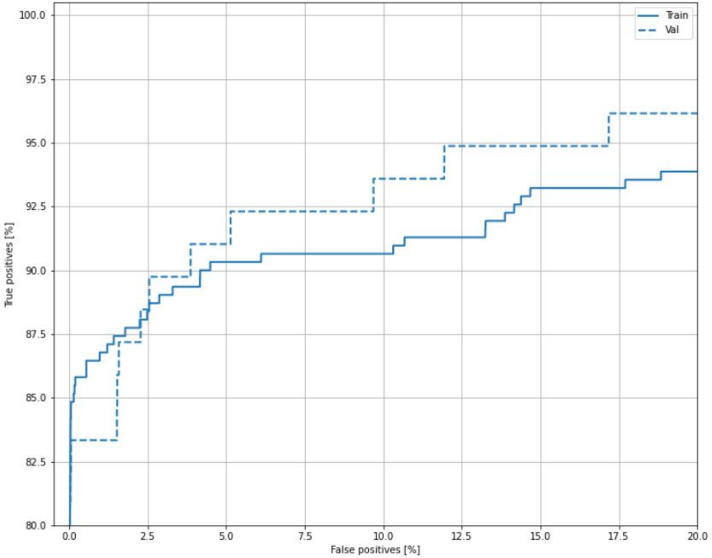


Baseline model: Cross Validation

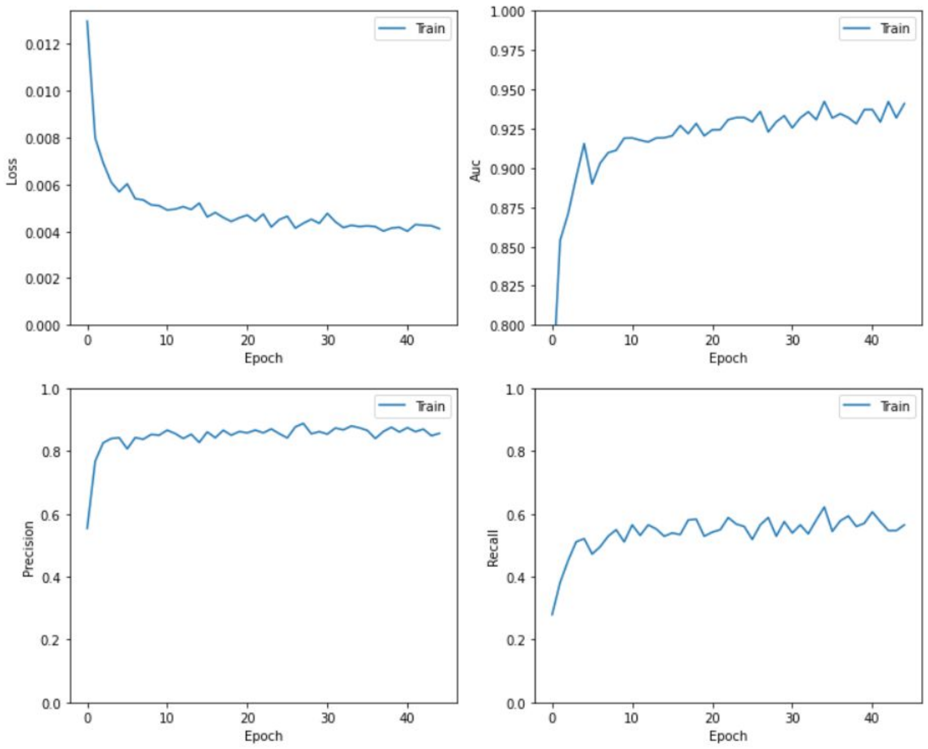
Training for fold 5 ...







Baseline model: Training



Training:

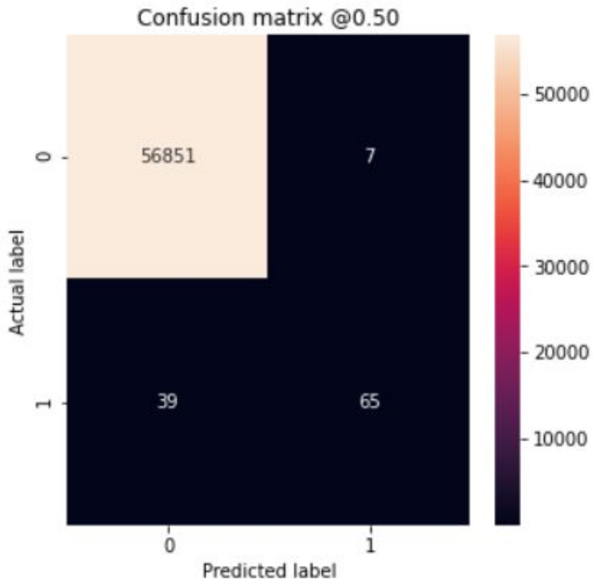
Accuracy : 0.999
Precision : 0.888
Recall : 0.592
AUC : 0.942

Cross-Validation:

Accuracy: 0.999 +/- 4.25e-05
Precision: 0.871 +/- 0.0328
Recall: 0.582 +/- 0.0234
AUC: 0.929 +/- 0.0113

Baseline model: Test

```
Legitimate Transactions Detected (True Negatives):  56851
Legitimate Transactions Incorrectly Detected (False Positives):  7
Fraudulent Transactions Missed (False Negatives):  39
Fraudulent Transactions Detected (True Positives):  65
Total Fraudulent Transactions:  104
```



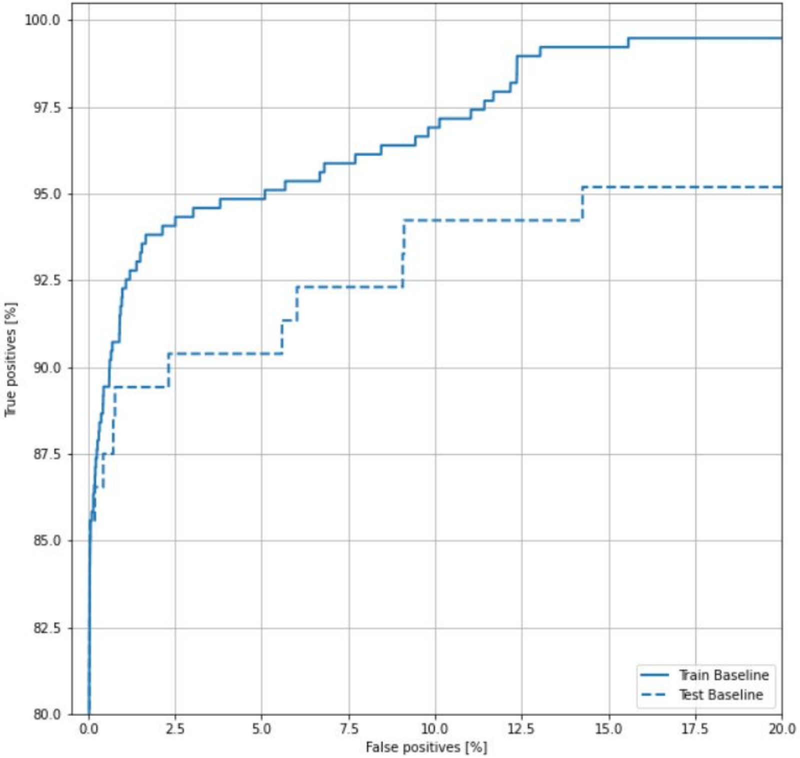
Training:

Accuracy : 0.999
Precision : 0.888
Recall : 0.592
AUC : 0.942

Test:

Accuracy: 0.999
Precision: 0.903
Recall: 0.625
AUC: 0.932

Baseline model: ROC Curve



Weighted model

```
# Scaling by total/2 helps keep the loss to a similar magnitude.
# The sum of the weights of all examples stays the same.
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))
```

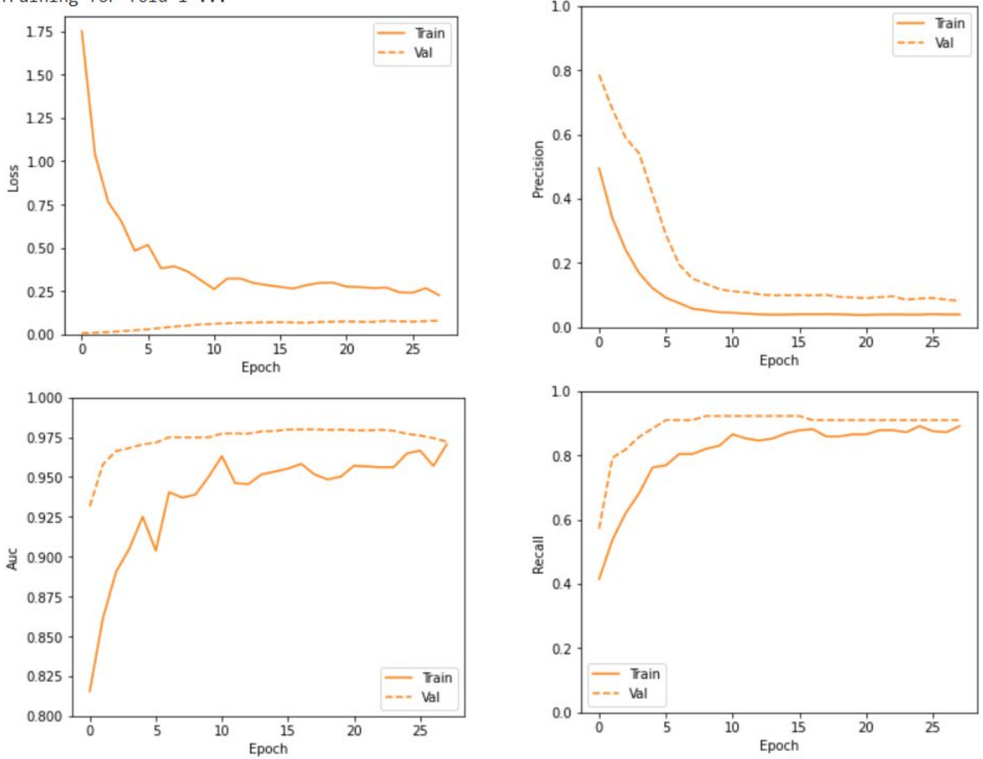
```
Weight for class 0: 0.50
Weight for class 1: 289.44
```

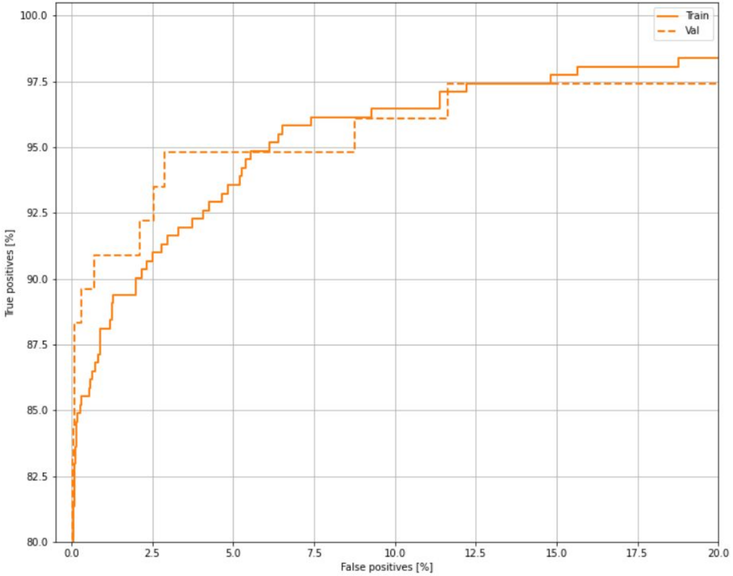
Weighted model

```
weighted_history = model_weighted.fit(
    train_features,
    train_labels,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    callbacks = callbacks_list,
    class_weight = class_weight, # Here we define the class weights
    verbose=1)
```

Weighted model: Cross Validation

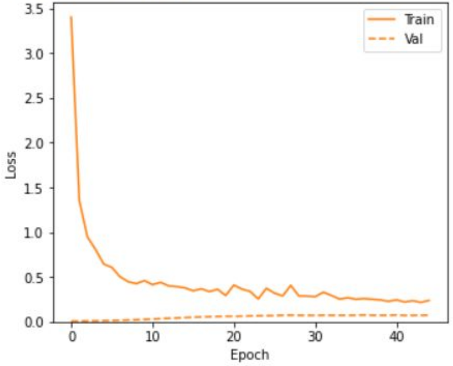
Training for fold 1 ...

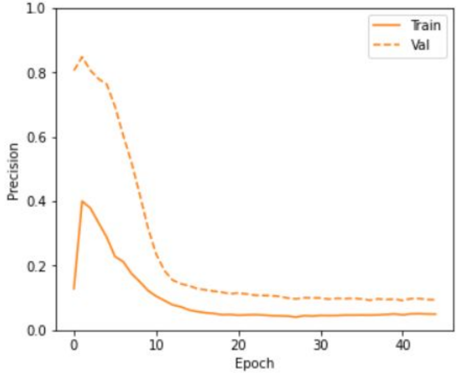


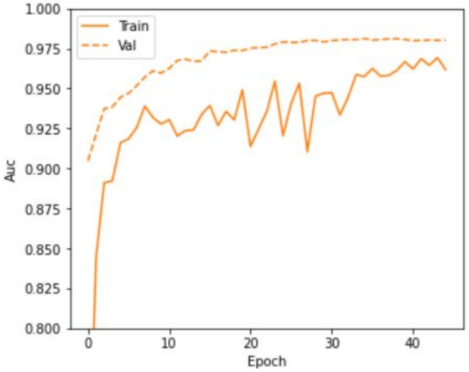


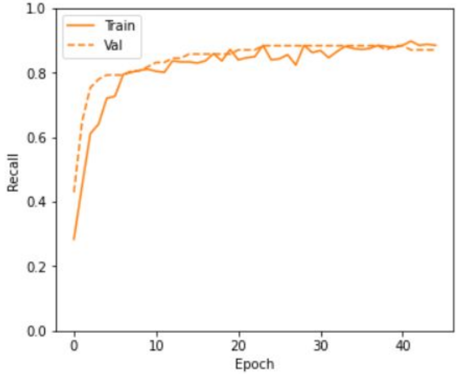
Weighted model: Cross Validation

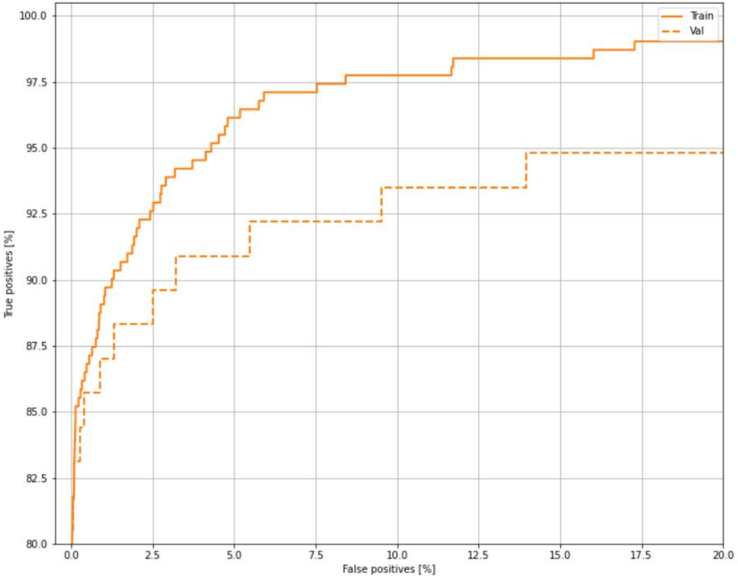
```
Training for fold 2 ...
```





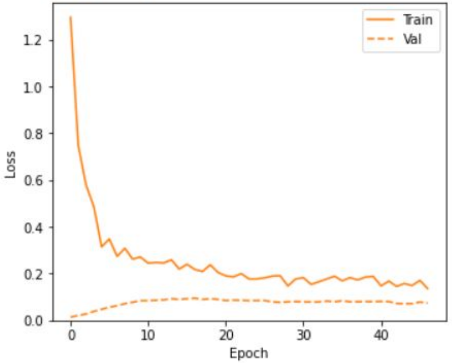


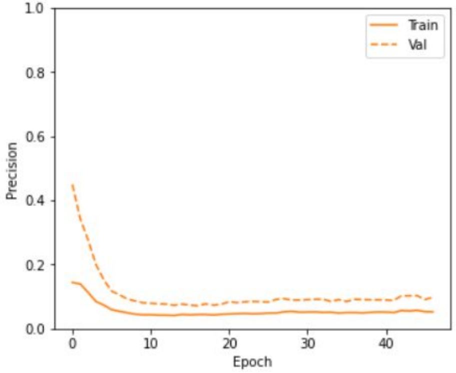


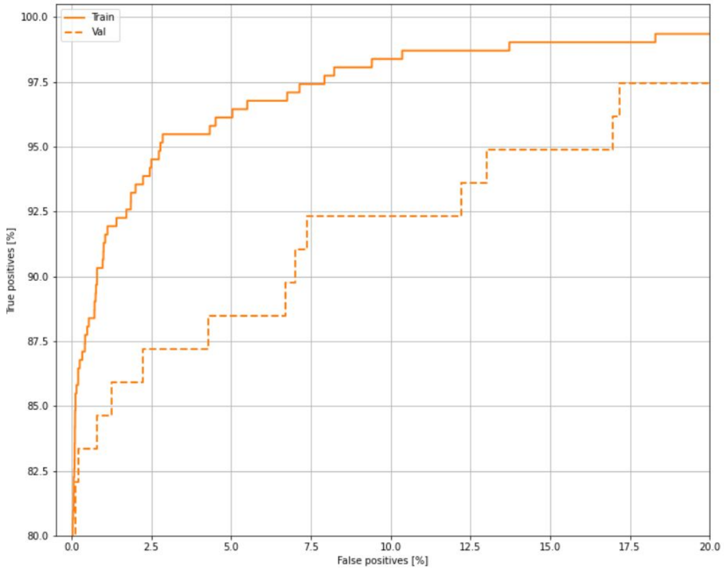


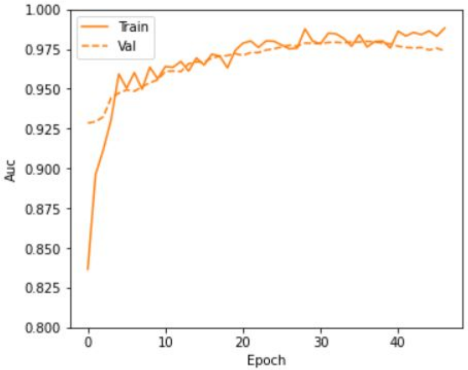
Weighted model: Cross Validation

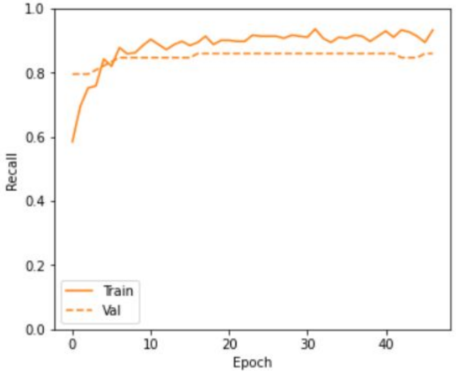
```
Training for fold 3 ...
```





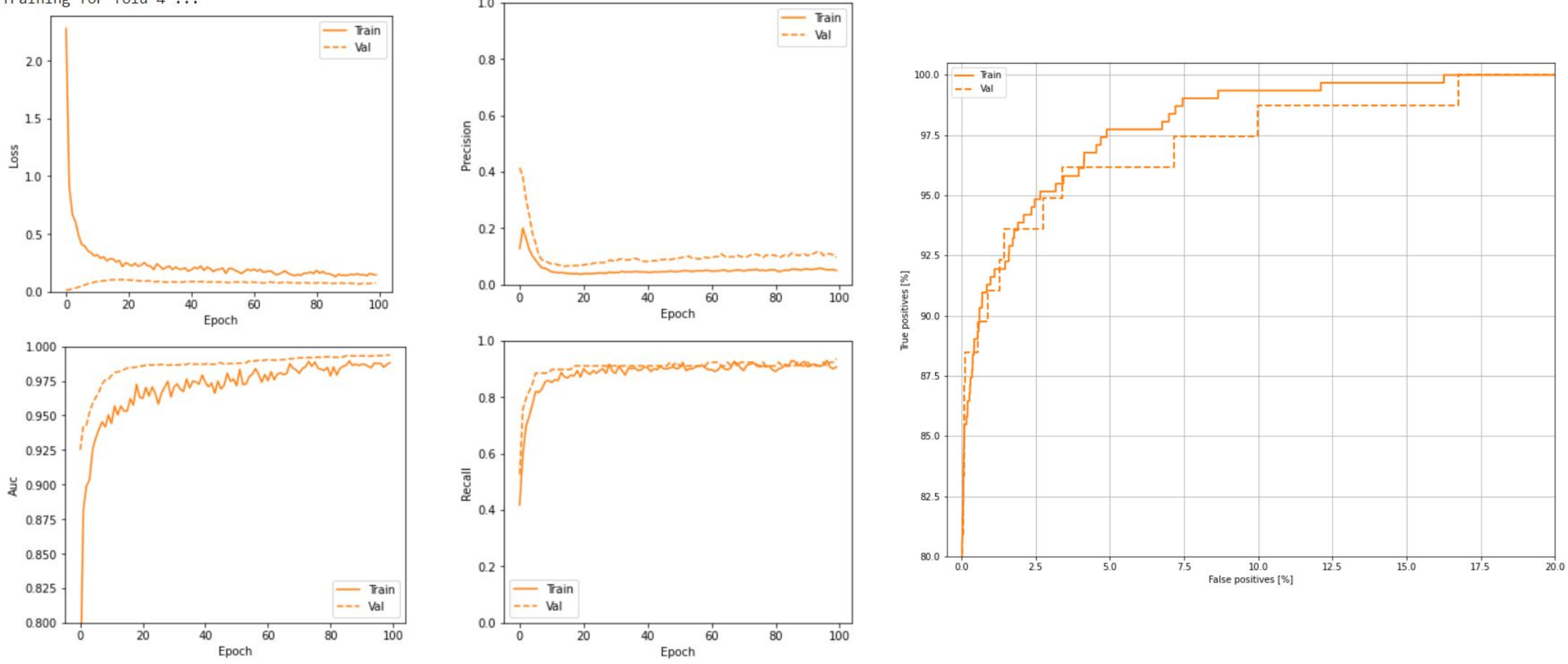






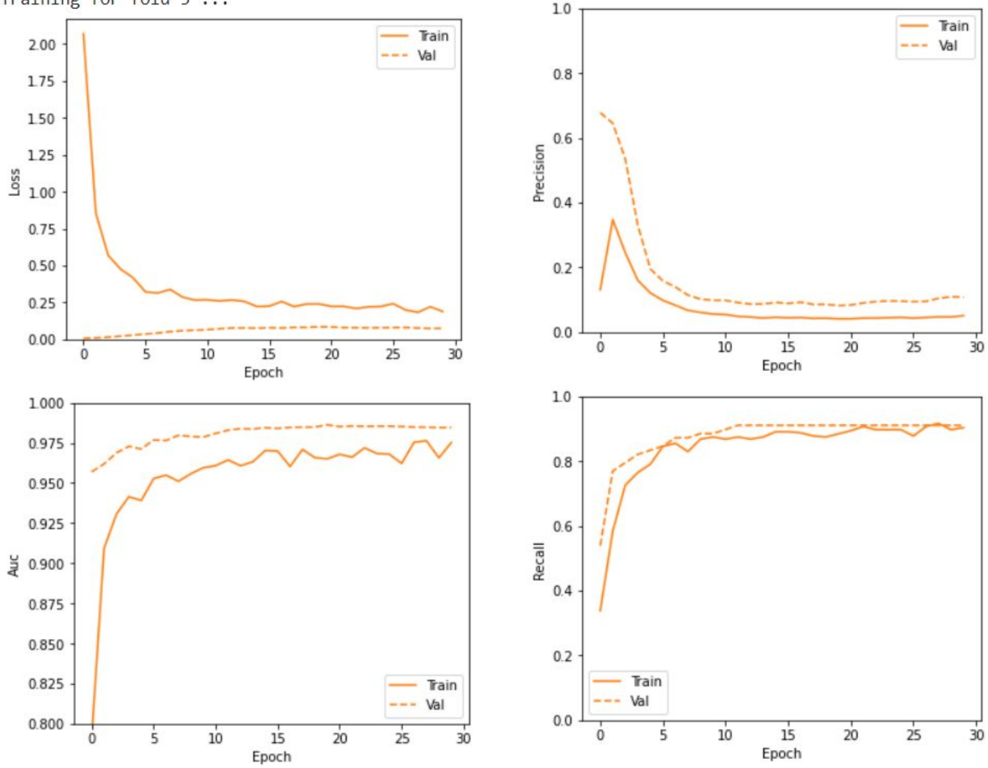
Weighted model: Cross Validation

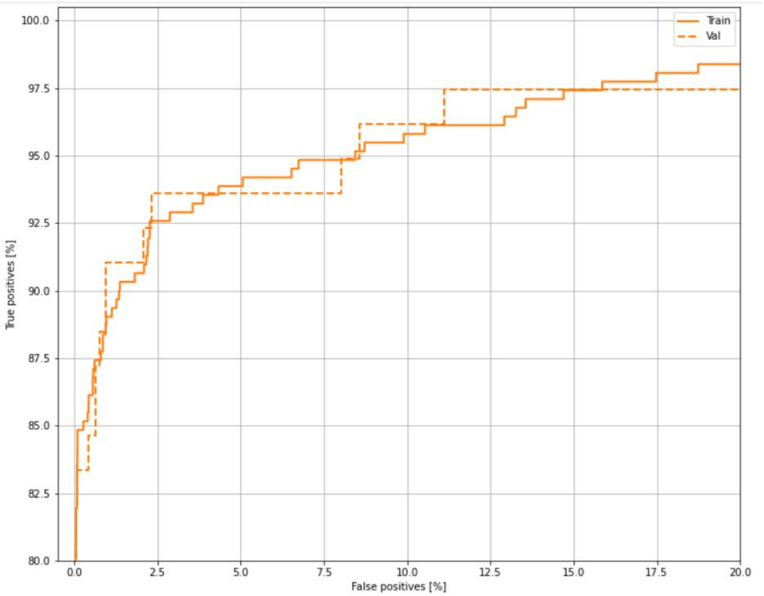
```
Training for fold 4 ...
```



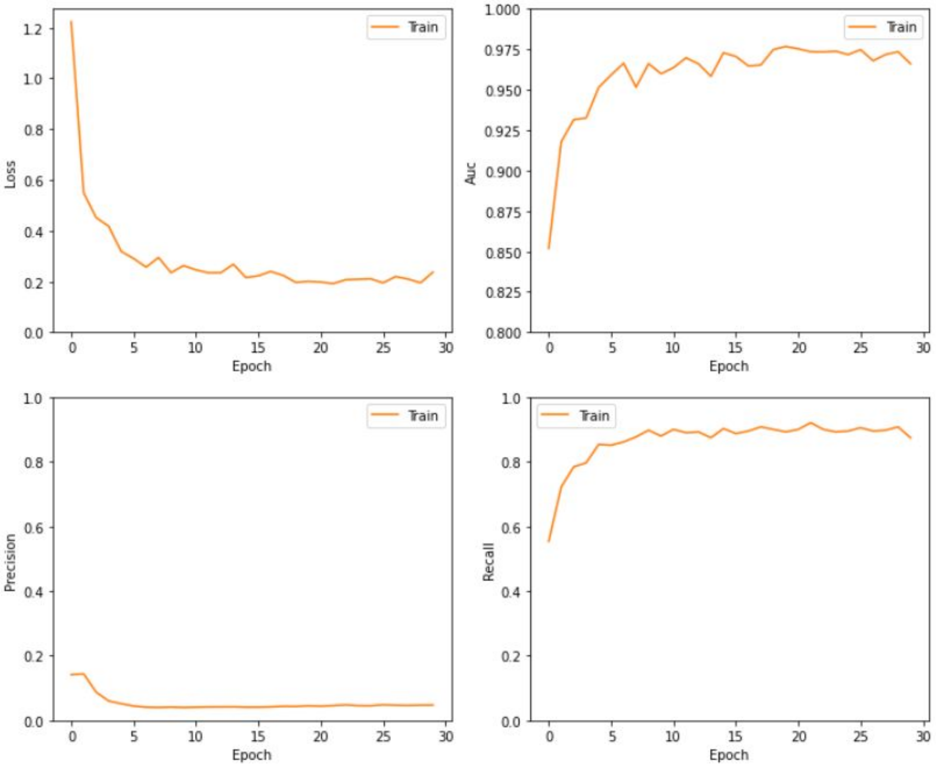
Weighted model: Cross Validation

```
Training for fold 5 ...
```





Weighted model: Training



Training:

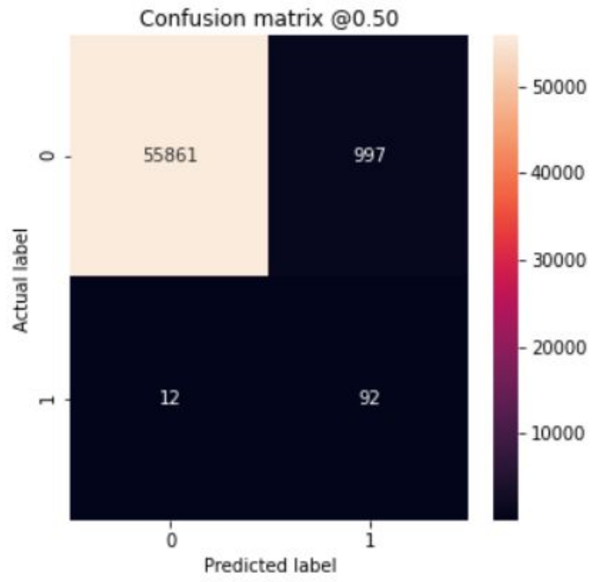
Accuracy : 0.982
Precision : 0.0807
Recall : 0.9123
AUC : 0.987

Cross-Validation:

Accuracy: 0.985 +/- 0.00138
Precision: 0.0938 +/- 0.00670
Recall: 0.899 +/- 0.0262
AUC: 0.984 +/- 0.00535

Weighted model: Test

```
Legitimate Transactions Detected (True Negatives):  55861
Legitimate Transactions Incorrectly Detected (False Positives):  997
Fraudulent Transactions Missed (False Negatives):  12
Fraudulent Transactions Detected (True Positives):  92
Total Fraudulent Transactions:  104
```



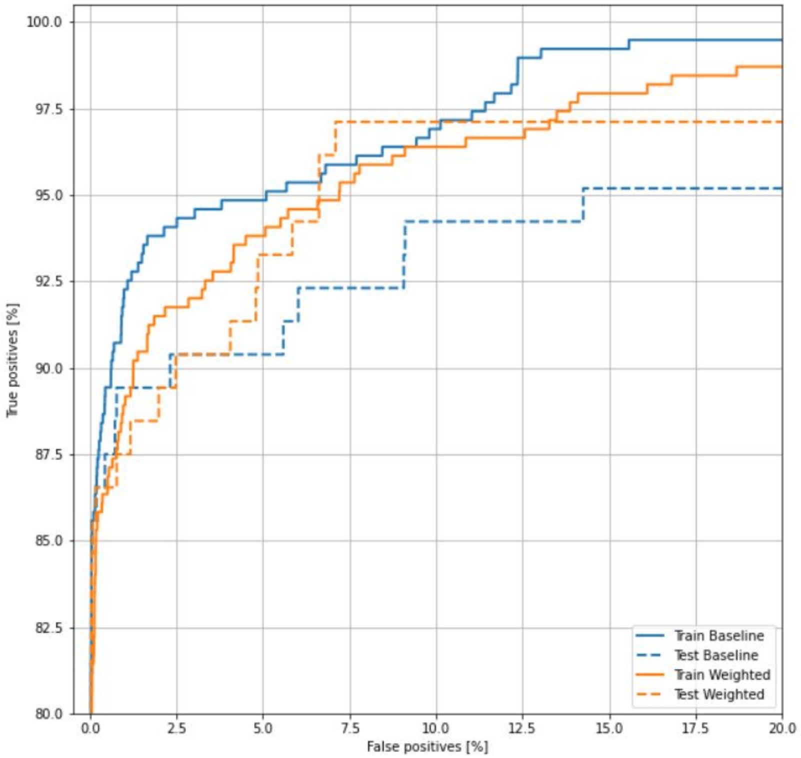
Training:

Accuracy : 0.982
Precision : 0.0807
Recall : 0.9123
AUC : 0.987

Test:

Accuracy: 0.982
Precision: 0.0845
Recall: 0.885
AUC: 0.983

Weighted model: ROC Curve

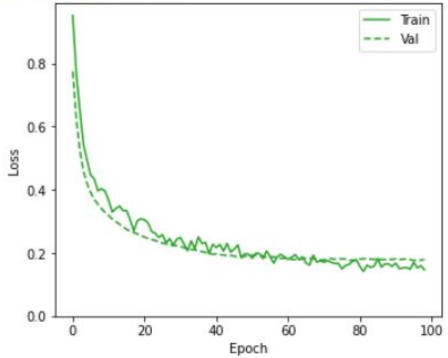


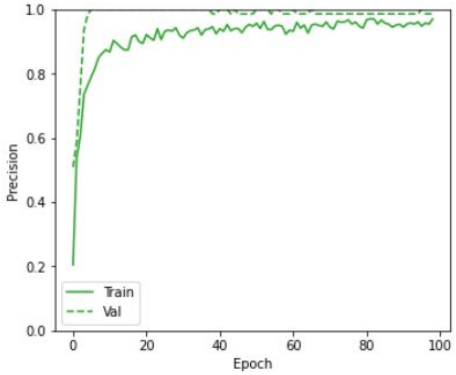
Undersampling

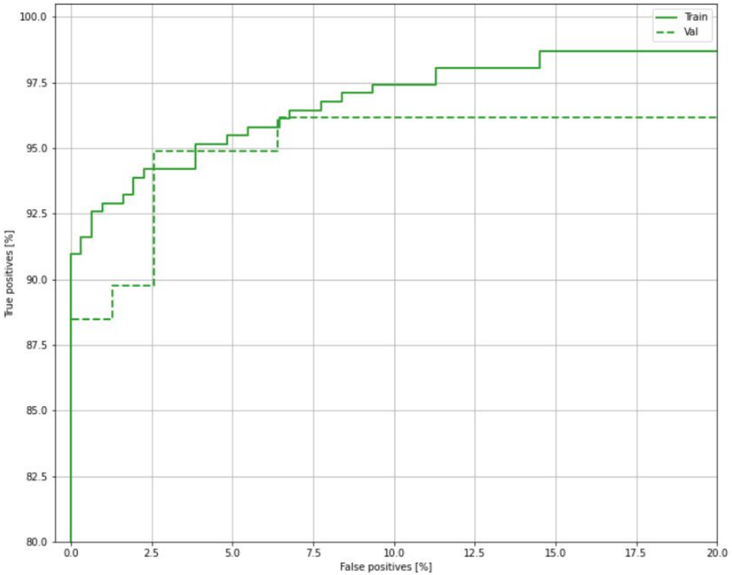
```
Train (Balanced):
  Total: 776
  Negative: 388 (50.00% of total)
  Positive: 388 (50.00% of total)

Test (Balanced):
  Total: 208
  Negative: 104 (50.00% of total)
  Positive: 104 (50.00% of total)

Test (Imbalanced):
  Total: 56962
  Negative: 56858 (99.82% of total)
  Positive: 104 (0.18% of total)
```

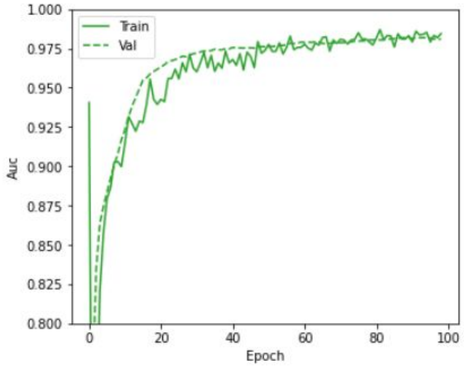
Undersampling: Cross Validation

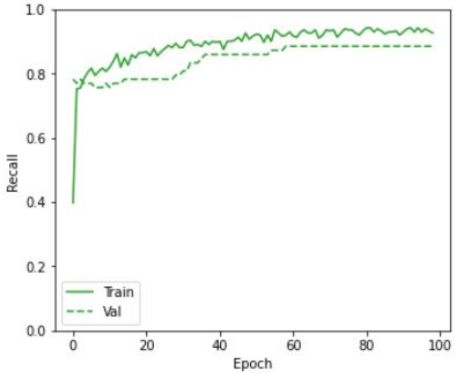
```
Training for fold 1 ...
```





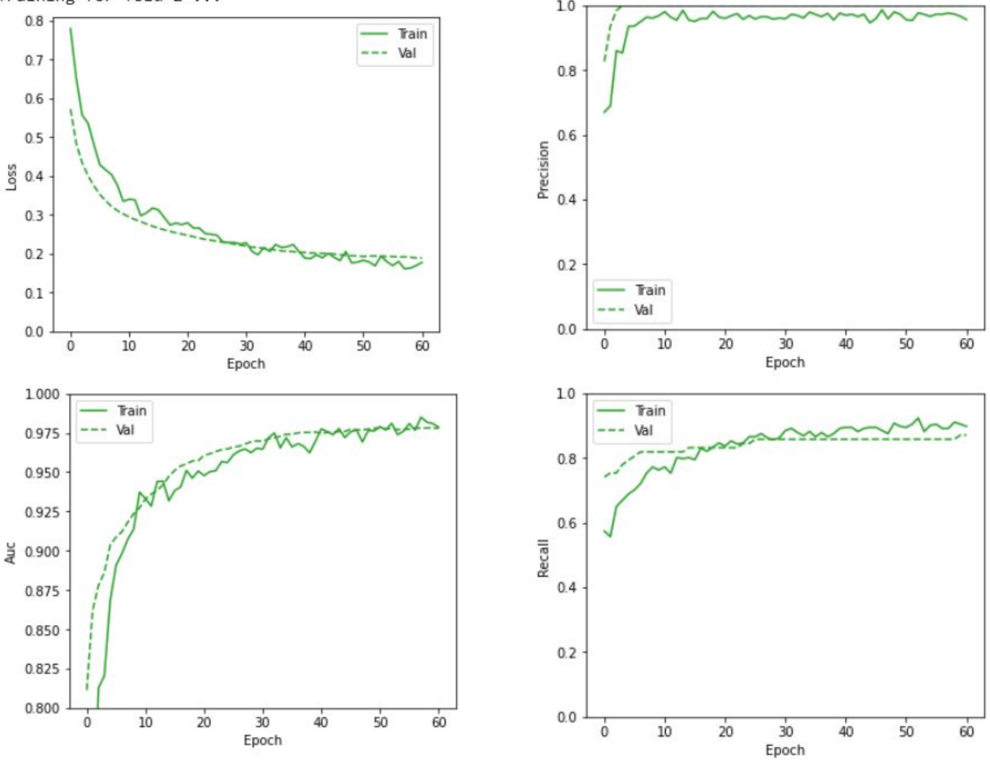


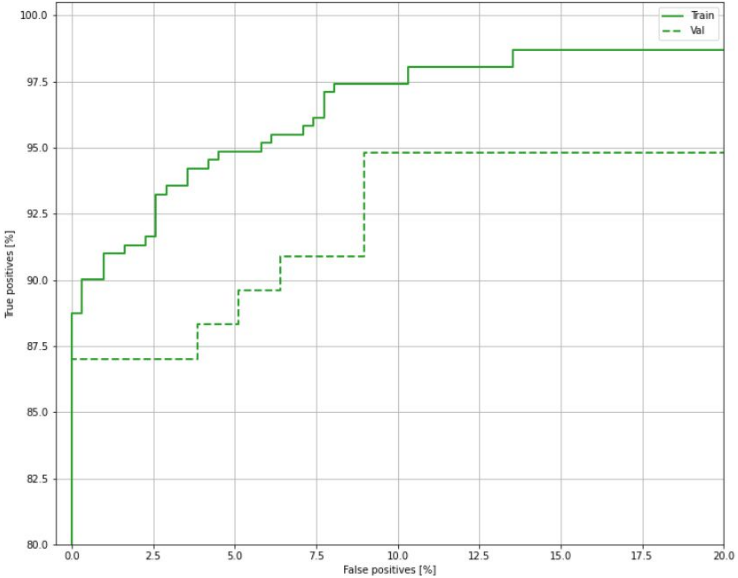




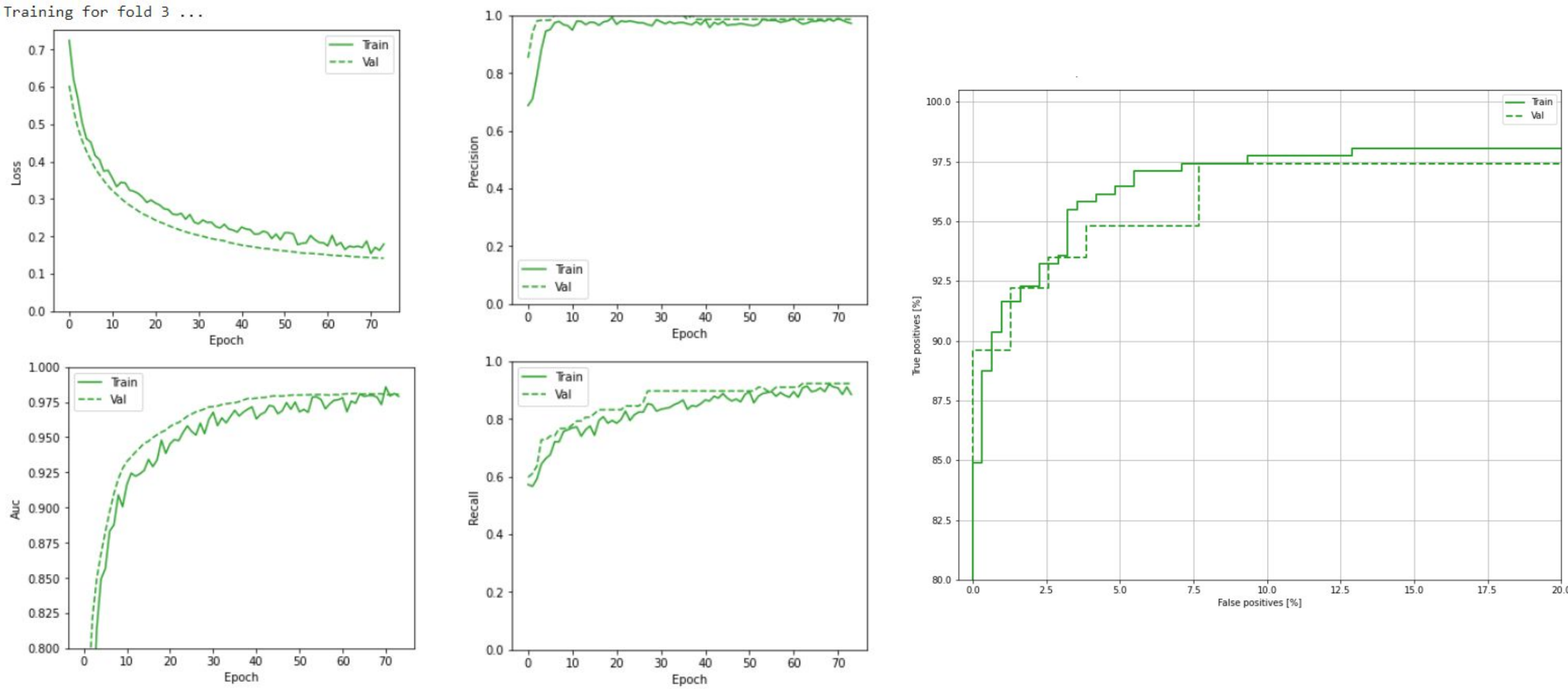
Undersampling: Cross Validation

Training for fold 2 ...

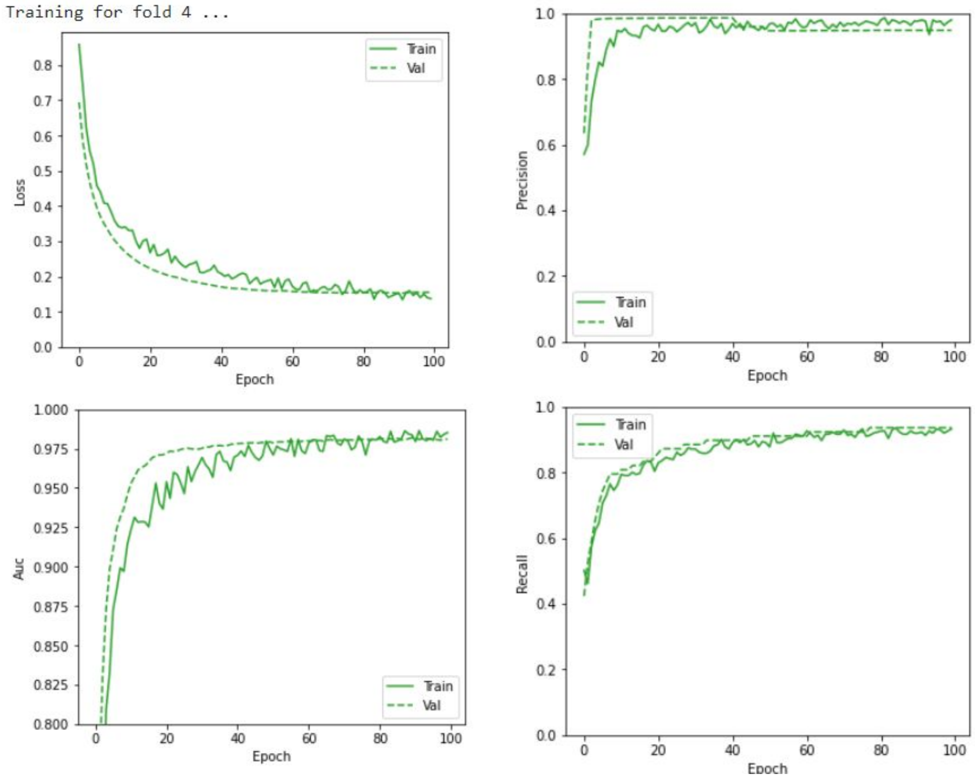


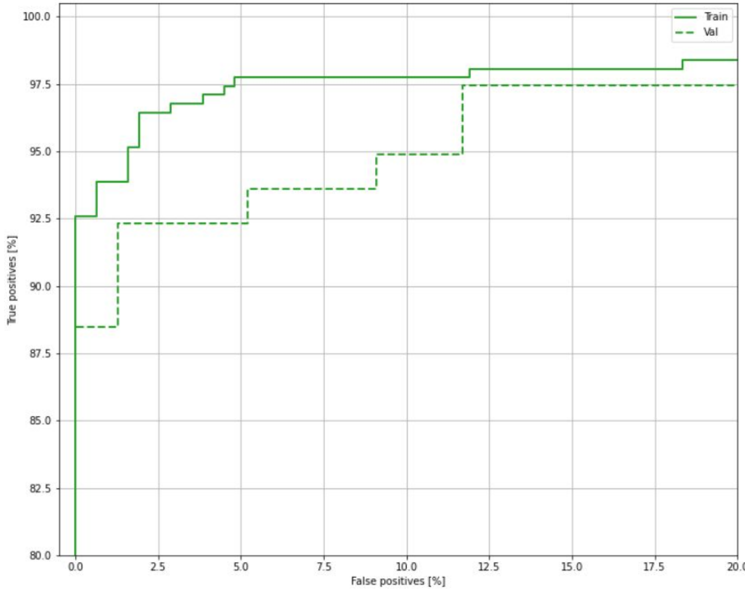


Undersampling: Cross Validation



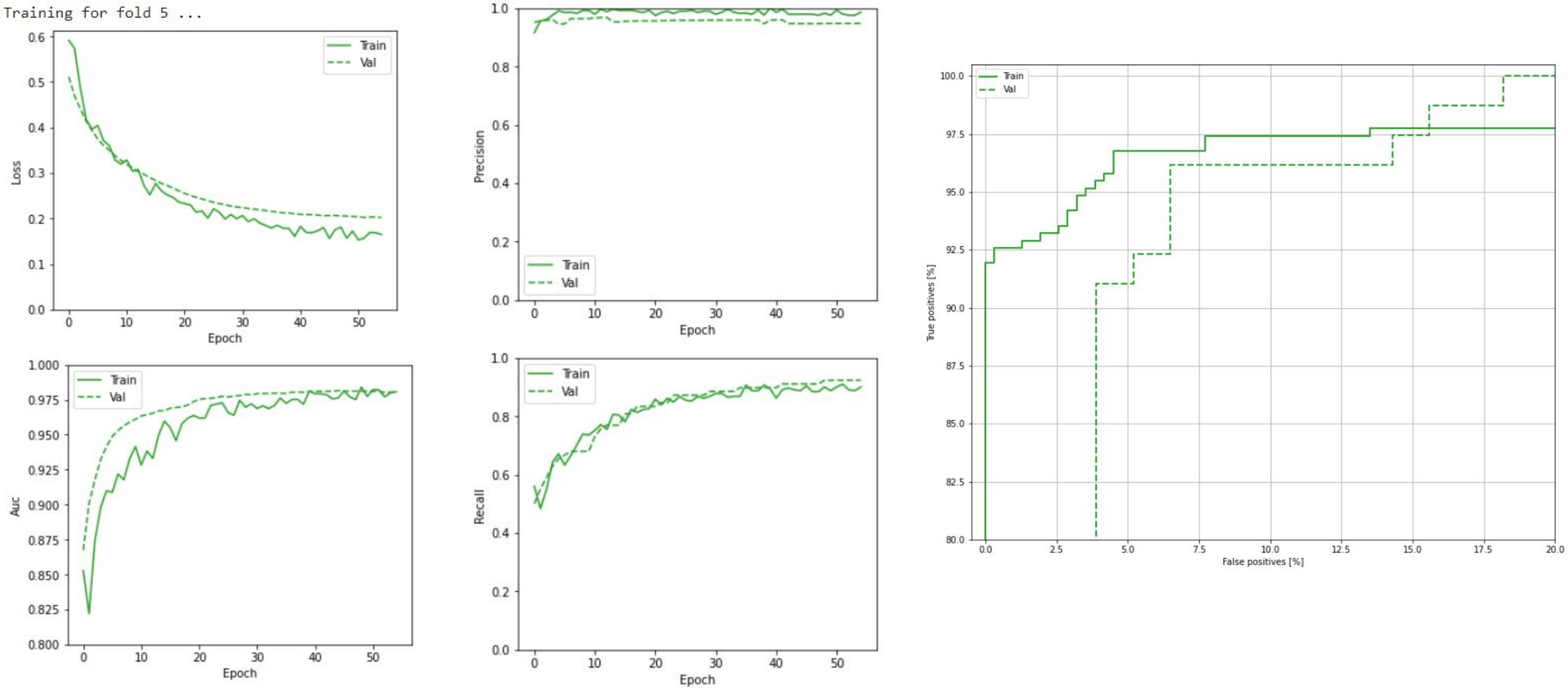
Undersampling: Cross Validation



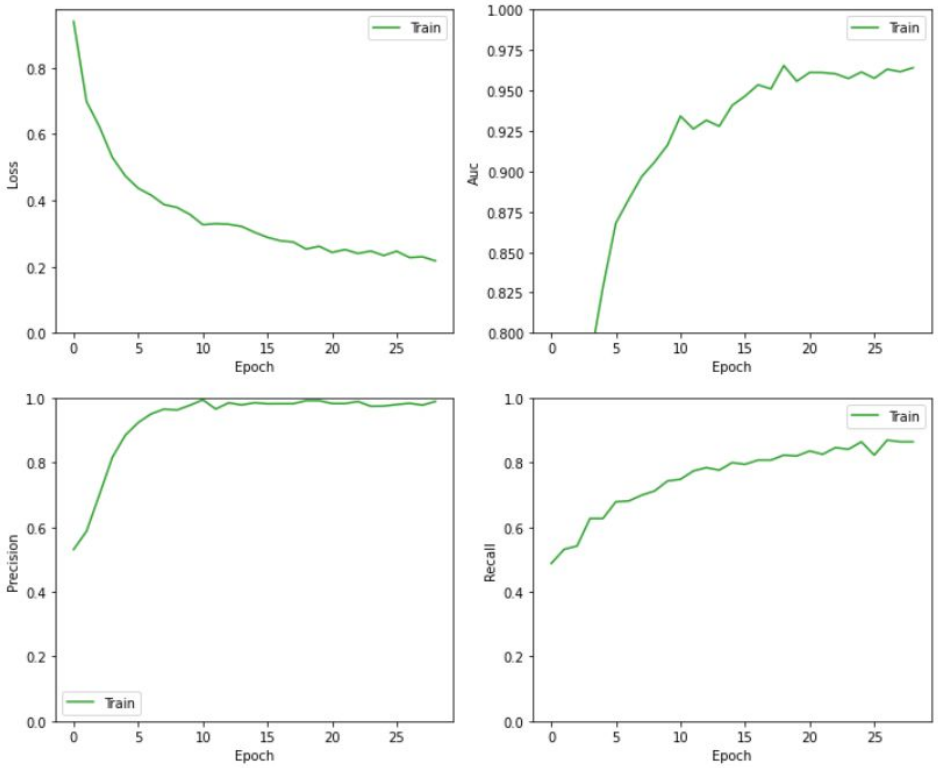


Undersampling: Cross Validation

```
Training for fold 5 ...
```



Undersampling: Training



Training:

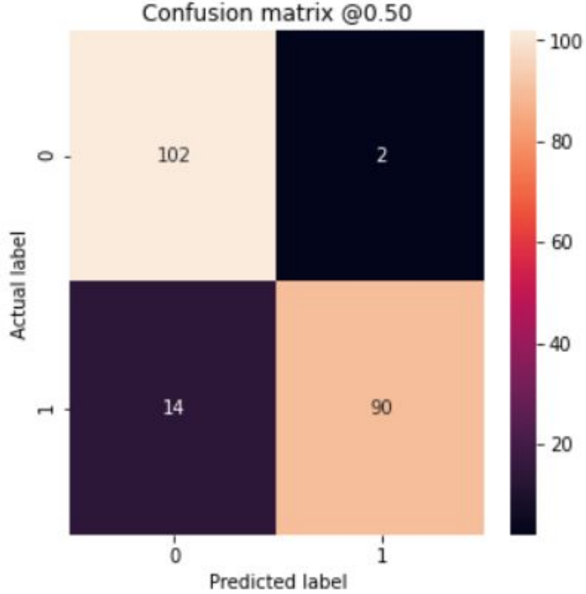
Accuracy: 0.917
Precision: 0.991
Recall: 0.843
AUC: 0.978

Cross-Validation:

Accuracy: 0.938 +/- 0.00963
Precision: 0.973 +/- 0.0218
Recall: 0.902 +/- 0.0280
AUC: 0.981 +/- 0.00132

Undersampling: Test (Balanced)

```
Legitimate Transactions Detected (True Negatives):  102
Legitimate Transactions Incorrectly Detected (False Positives):  2
Fraudulent Transactions Missed (False Negatives):  14
Fraudulent Transactions Detected (True Positives):  90
Total Fraudulent Transactions:  104
```



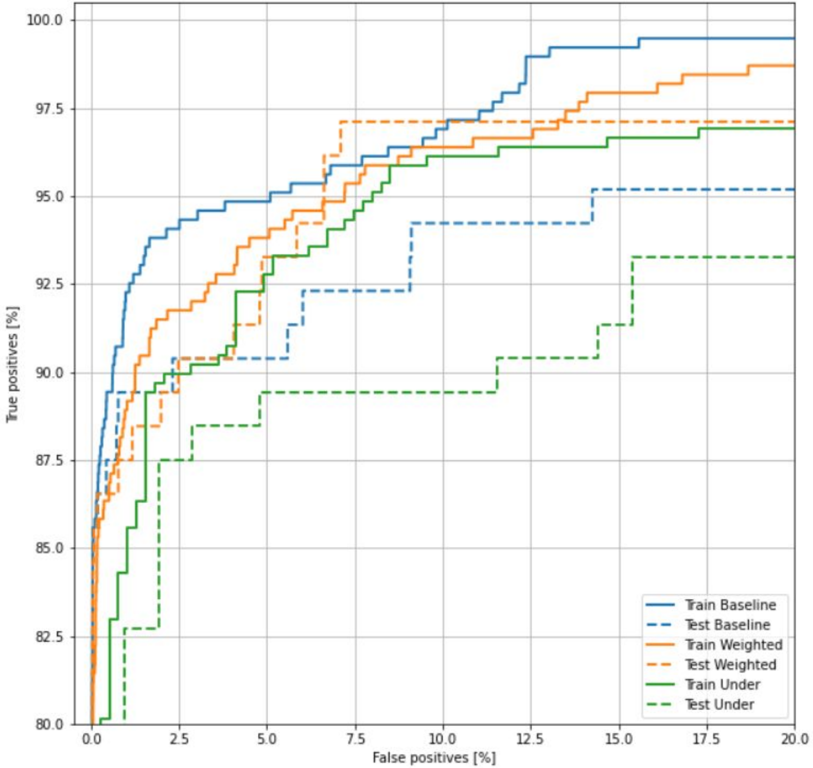
Training:

Accuracy: 0.917
Precision: 0.991
Recall: 0.843
AUC: 0.978

Test:

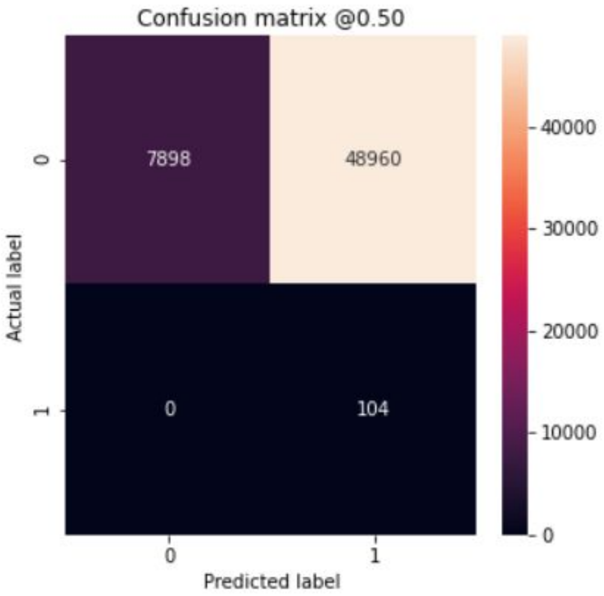
Accuracy: 0.923
Precision: 0.978
Recall: 0.865
AUC: 0.960

Undersampling: ROC Curve (Balanced)



Undersampling: Test (Imbalanced)

```
Legitimate Transactions Detected (True Negatives):  7898
Legitimate Transactions Incorrectly Detected (False Positives):  48960
Fraudulent Transactions Missed (False Negatives):  0
Fraudulent Transactions Detected (True Positives):  104
Total Fraudulent Transactions:  104
```



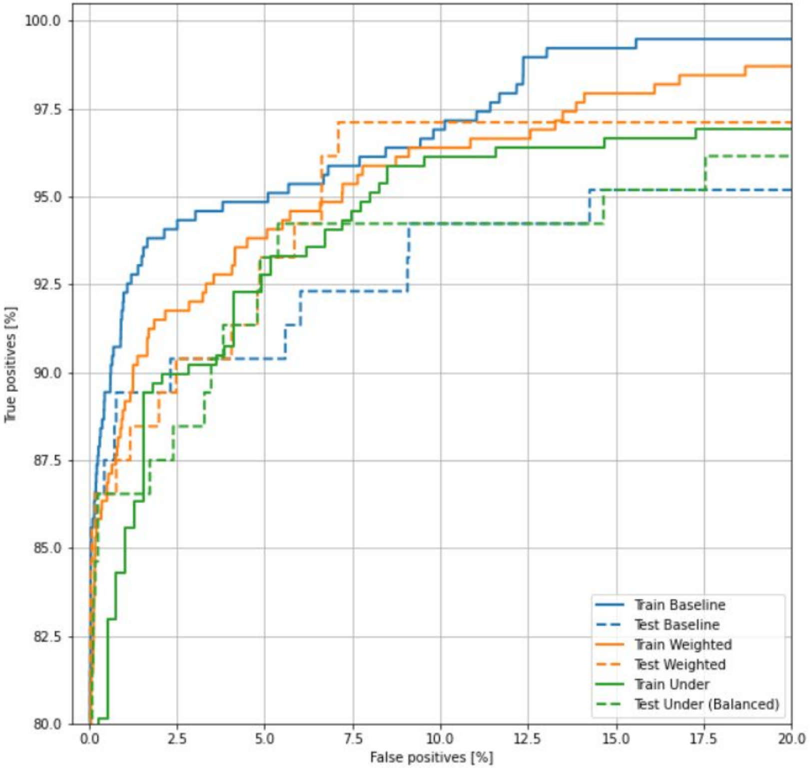
Training:

Accuracy: 0.917
Precision: 0.991
Recall: 0.843
AUC: 0.978

Test:

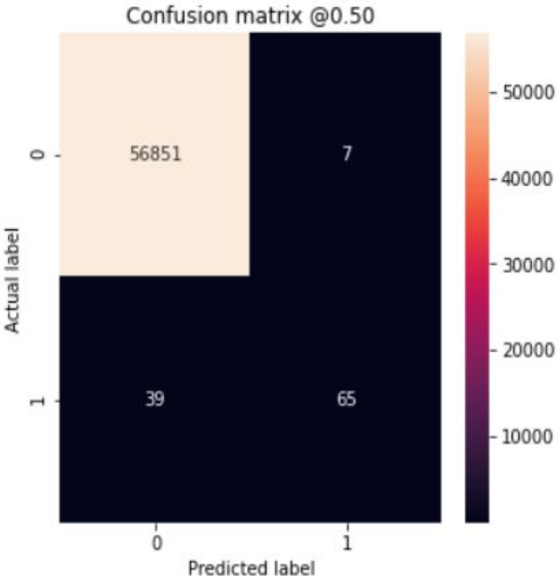
Accuracy: 0.140
Precision: 0.00212
Recall: 1.0
AUC: 0.970

Undersampling: ROC Curve (Imbalanced)

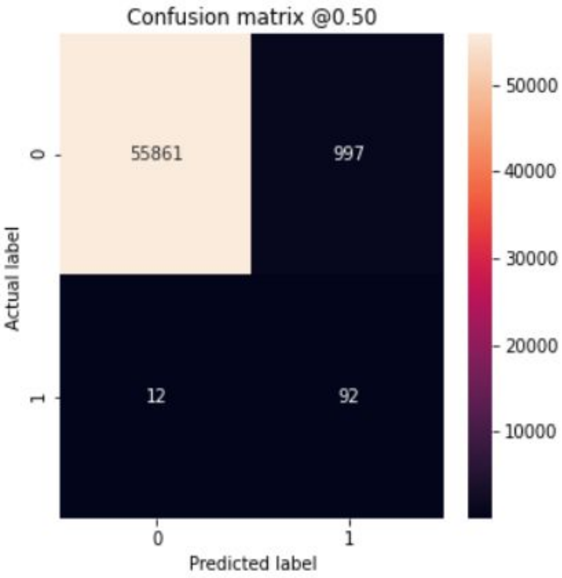


Conclusion

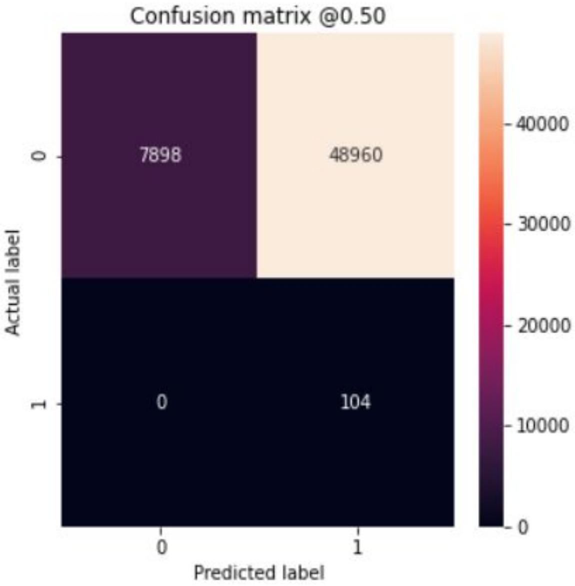
Test Baseline



Test Weighted



Test Undersampling



Conclusion

- The baseline model has a good precision, but it fails in classifying most of the positive samples, which is a bad thing given the context of the problem. This approach has high precision, but average recall.

- The weighted model classifies many legitimate transactions as fraudulent, but it is capable of catching most of the true positives. So here we have high recall, with low precision.

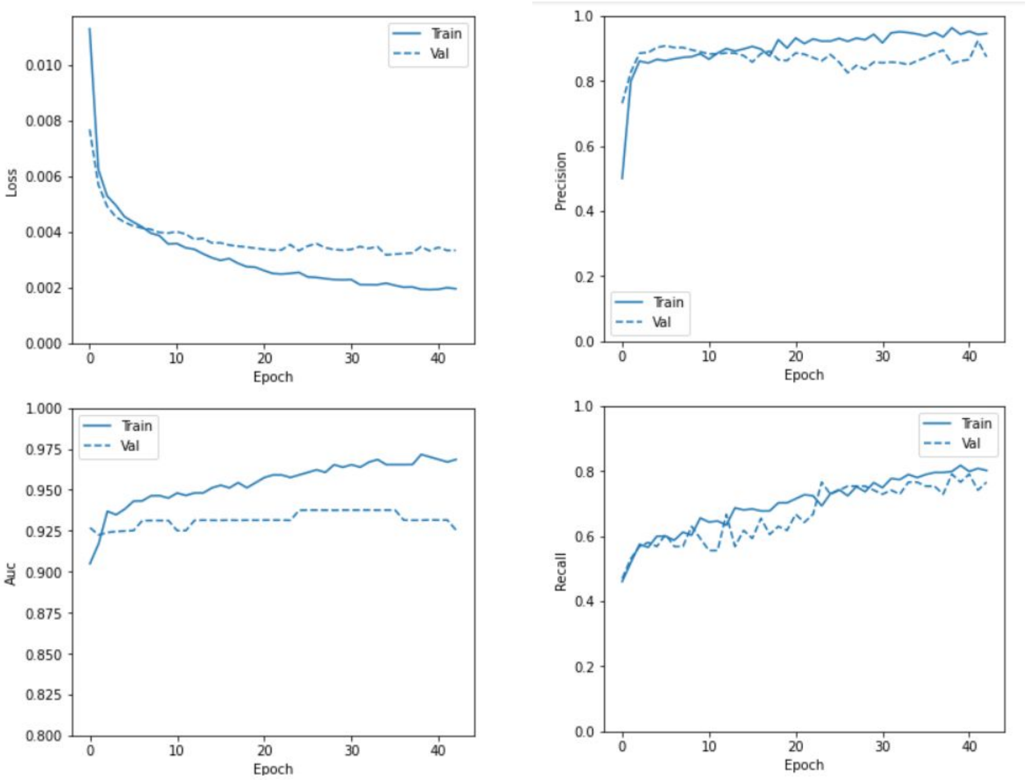
- The undersample approach has high precision and recall when applied to a balanced dataset, but classifies a very large number of legitimate transactions as fraudulent when applied to the imbalanced dataset.

Final verdict: The weighted model is probably the best approach to this problem.

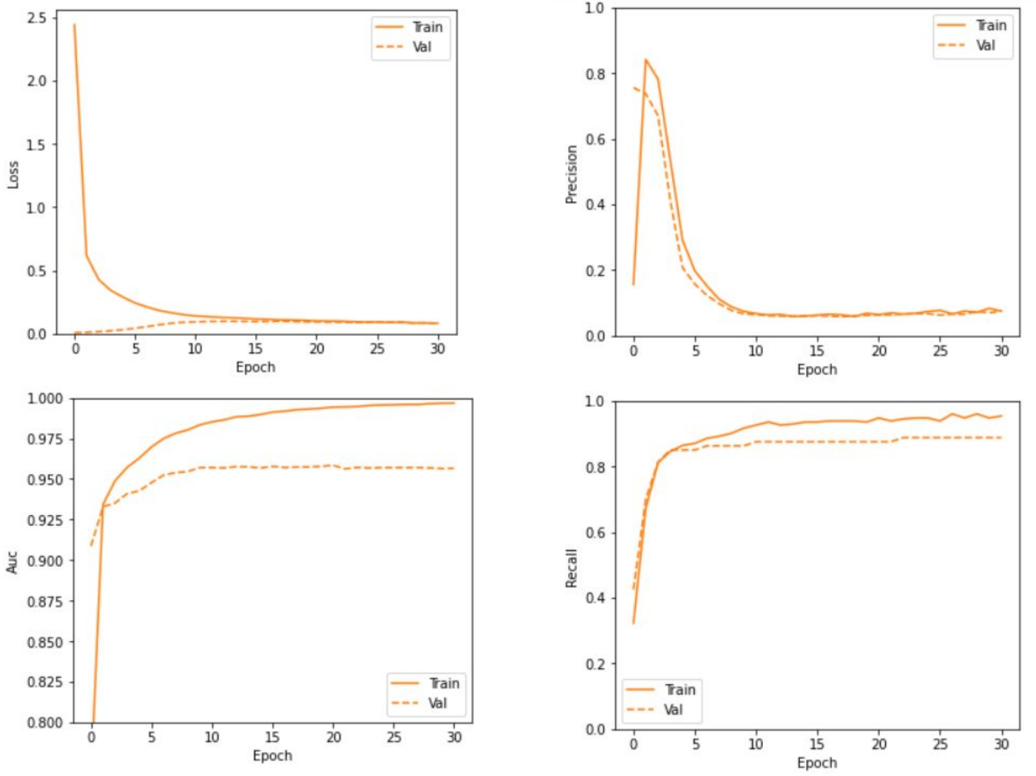
Batch Normalization: Not a good idea...

```
model = keras.Sequential([
  keras.layers.Dense(
      16, activation='relu',
      input_shape=(train_features.shape[-1],)),
  keras.layers.BatchNormalization(),
  keras.layers.Dense(1, activation='sigmoid',
                         bias_initializer=output_bias),
])
```

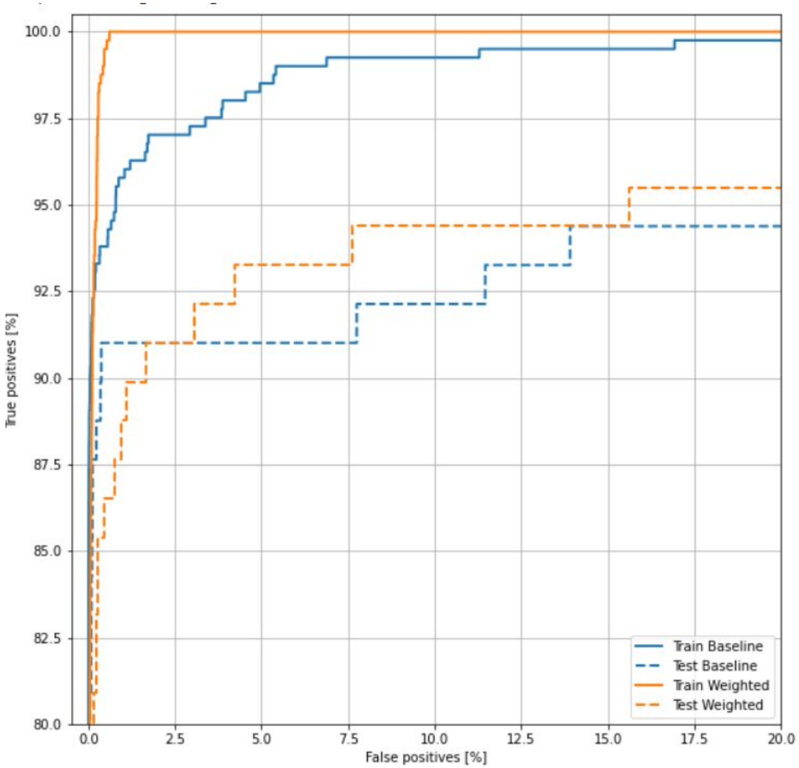
Batch Normalization



Batch Normalization

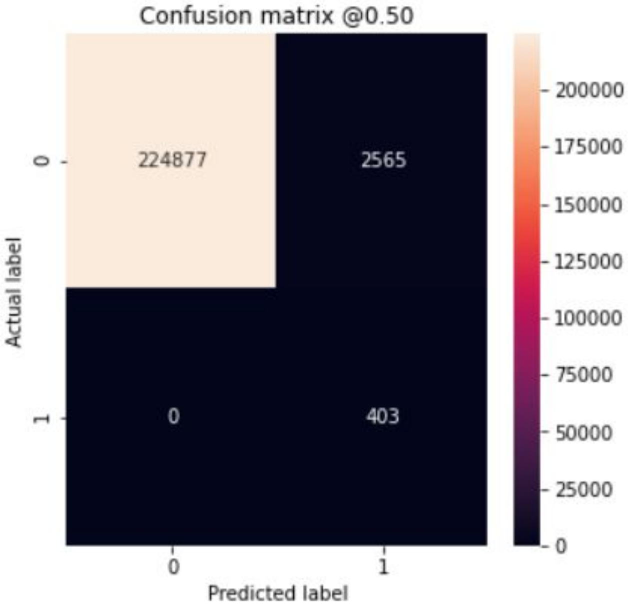


Batch Normalization

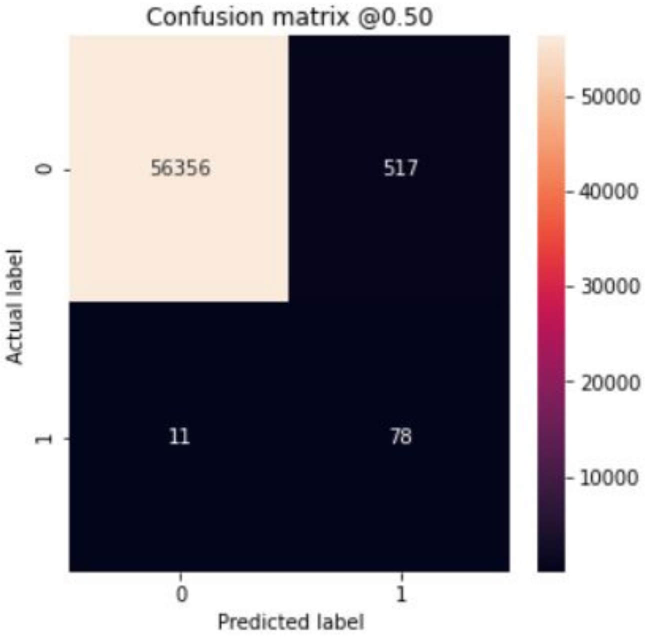


Batch Normalization

Train (Weighted)



Test (Weighted)

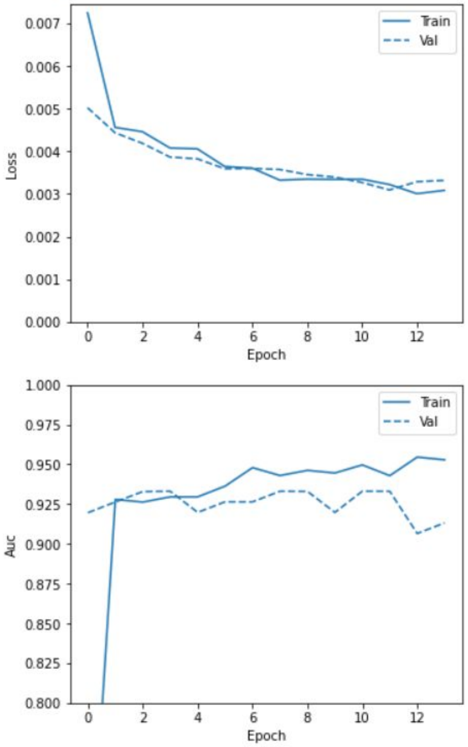


Deeper Network

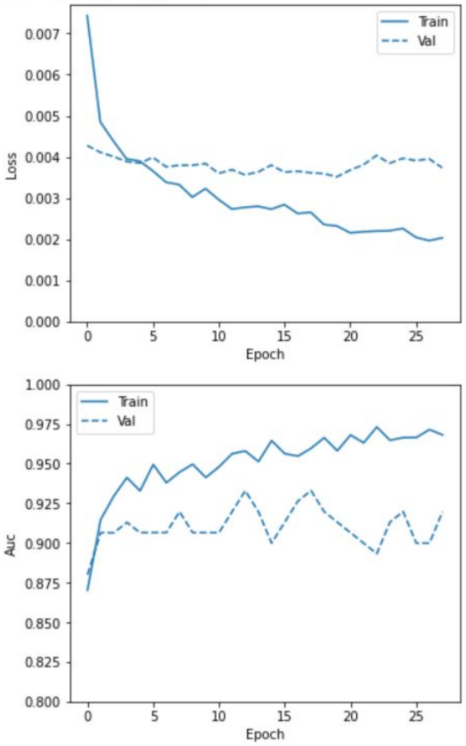
```
model = keras.Sequential([
    keras.layers.Dense(
        256, activation='relu',
        input_shape=(train_features.shape[-1],)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation='sigmoid',
                       bias_initializer=output_bias),
])
```

Deeper Network

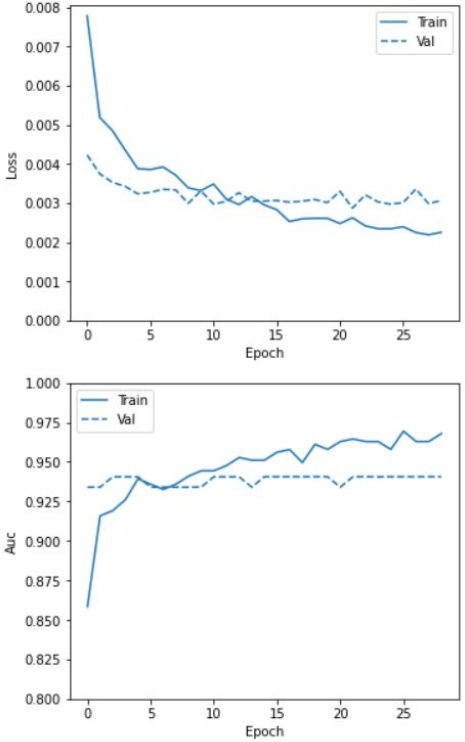
Training for fold 1 ...



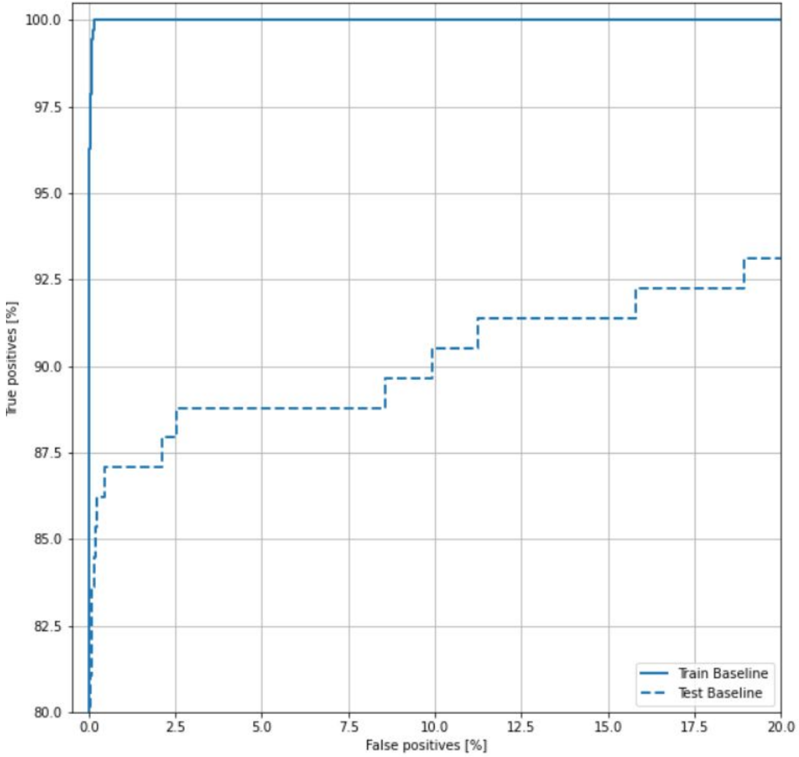
Training for fold 2 ...



Training for fold 5 ...



Deeper Network



“Excellent!”