In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
df = pd.read_csv('/content/drive/MyDrive/CI/spambase/spambaseFIXED.csv',
                 sep=";")
df.describe
```

Podstawowe informacje

Dane poddane analizie pochodzą ze kilku korporacji oraz osób niezwiązanych z projektem. Zbiór został stworzony przez 4 naukowców na potrzeby implementacji systemu filtrującego pocztę mailową.

Głównym celem tworzenia zbioru danych było przewidywanie i klasyfikacja wiadomości e-mail jako spam.

Charakterystyka danych

Liczba instancji - 4601

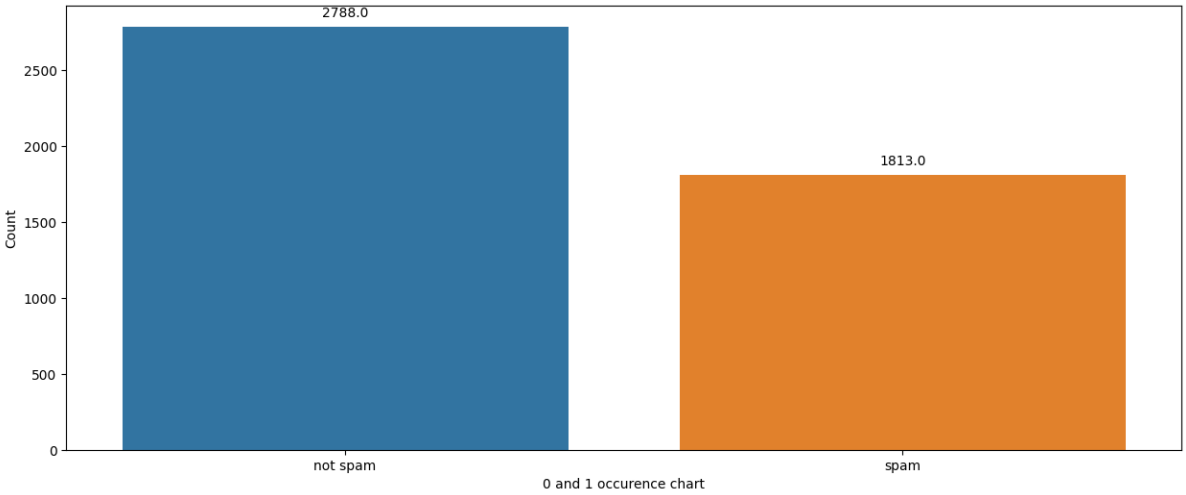
Liczba atrybutów - 57 - w tym

binarne

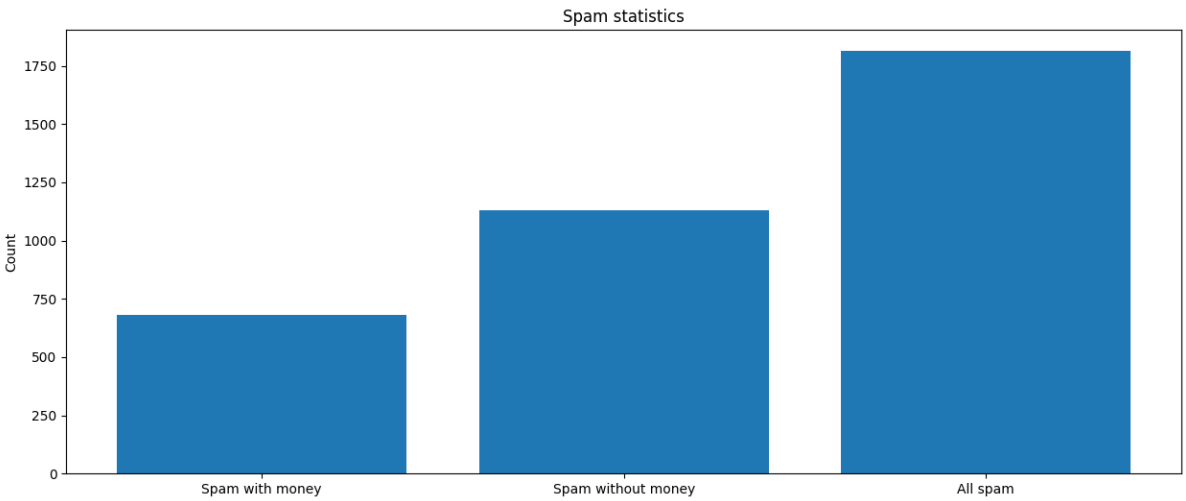
liczbowe

Brakujące wartości - 1582

In [3]:
```
import seaborn as sns

plt.figure(figsize=(15, 6))
ax = sns.countplot(data=df, x="Column58")

for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2.,
                p.get_height()),
                ha='center', va='center', xytext=(0, 10),
                textcoords='offset points')

plt.xlabel("0 and 1 occurence chart")

ax.set_xticklabels(['not spam', 'spam'])
plt.ylabel('Count')

plt.show()
```



In [ ]:
```
money_spam_count = ((df['Column24'] != 0) & (df['Column58'] == 1)).sum()
no_money_spam_count = ((df['Column24'] == 0) & (df['Column58'] == 1)).sum()
all_spam = (df['Column58'] == 1).sum()

categories = ['Spam with money',"Spam without money", 'All spam']
values = [money_spam_count, no_money_spam_count, all_spam]
plt.figure(figsize=(15, 6))
plt.bar(categories, values)
plt.ylabel('Count')
plt.title('Spam statistics')
plt.show()
```



Przygotowanie danych

In [4]:
```
# Sprawdzamy kompletność danych
print("Liczba brakujących wartości : ", df.isnull().sum().sum())
```

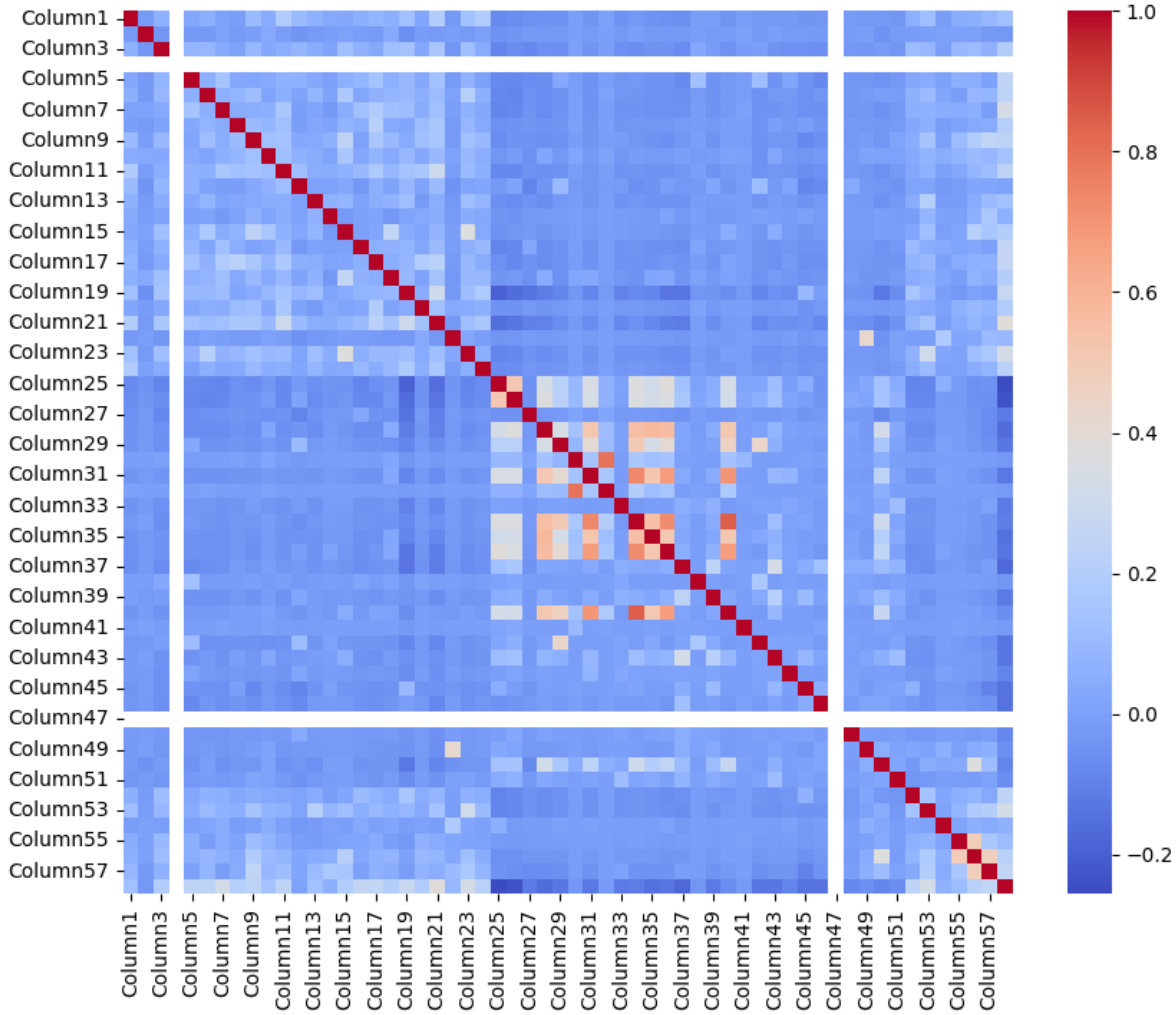
```
Liczba brakujących wartości :  1582
```

In [5]:
```
# Usuwamy wiersze zawierające brakujące wartości
df.fillna(0, inplace=True)
print("Liczba brakujących wartości : ", df.isnull().sum().sum())
```

```
Liczba brakujących wartości :  0
```

In [ ]:
```
# Tworzymy mapę korelacji
import seaborn as sns
import matplotlib.pyplot as plt

corr_matrix = df.corr()
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm')
plt.show()

# znajdywanie 10 najlepszych korelacji
corr_last_column = corr_matrix[df.columns[-1]]
top_corr = corr_last_column.sort_values(ascending=False).head(11)
print(top_corr)

top_corr = corr_last_column.sort_values(ascending=True).head(11)
print(top_corr)
```



```
Column58    1.000000
Column21    0.383234
Column23    0.334787
Column7     0.332117
Column53    0.323629
Column19    0.273651
Column16    0.263215
Column17    0.263204
Column57    0.249164
Column5     0.241920
Column52    0.241888
Name: Column58, dtype: float64
Column25   -0.256723
Column26   -0.232968
Column37   -0.178045
Column28   -0.158800
Column35   -0.149225
Column46   -0.146138
Column45   -0.140408
Column42   -0.136615
Column36   -0.136134
Column43   -0.135664
Column29   -0.133523
Name: Column58, dtype: float64
```

In [6]:
```
# Osobna kolumna dla wartości do przewidywania
is_spam_array = df['Column58']

# Usuwamy wartość do przewidywania
df = df.drop('Column58', axis=1)

# Usuwamy kolumny z samymi zerami
df = df.drop('Column4', axis=1)
df = df.drop('Column47', axis=1)
```

Podział danych

Ogólnie przyjętą praktyką jest stosowanie proporcji w zakresie 60-80% danych treningowych, 10-20% danych walidacyjnych i 10-20% danych testowych.

W naszym projekcie najlepsze rezultaty otrzymaliśmy przy wyborze:

- 70% - dane treningowe
- 15% - dane walidujące
- 15% - dane testowe

In [7]:
```
# Podział zbioru danych na dane treningowe, walidacyjne i testowe w proporcji
# 70%-15%-15%
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

data_train, data_rest, is_spam_train, is_spam_rest = train_test_split(
    df, is_spam_array, test_size=0.3, random_state=43)

data_test, data_validate, is_spam_test, is_spam_validate = train_test_split(
    data_rest, is_spam_rest, test_size=0.5, random_state=43)
```

Pierwsza próba stworzenia sieci

Przy pierwszej próbie tworzenia modelu sieci zdecydowaliśmy się na model 3 - warstwowy z jedną warstwą ukrytą o 16 neuronach. W warstwie ukrytej, jak i wejściowej funkcja aktywacji została ustawiona na "ReLU" która stosowana jest do uczenia się nieliniowych zależności między parametrami.

In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense

# Inicjalizacja modelu
model = Sequential()

# Dodanie warstw
model.add(Dense(64, input_dim=data_train.shape[1], activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Kompilacja modelu
model.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['accuracy'])
epochs = 10
```

Trening sieci

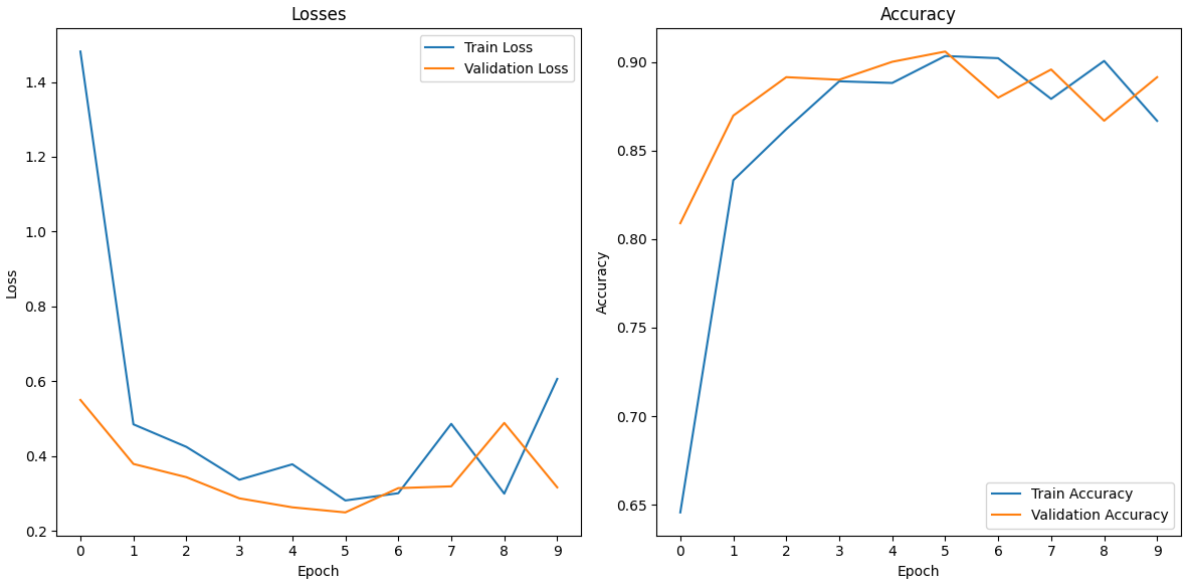
Do uczenia naszej sieci wykorzystujemy algorytm optymalizatora "Adam" - łączy on cechy Stochastic Gradient Descent z adaptacyjnym skalowaniem współczynników uczenia się. Model uczy się na przestrzeni 10 epok, w przypadku większej ilości nie było widocznej poprawy dokładności sieci. Argument batch_size wskazuje co ile próbek danych wagi dopasowania są aktualizowane.

In [ ]:
```
fitting_progress = model.fit(data_train, is_spam_train, epochs=epochs,
                             batch_size=32,
                             validation_data=(data_validate, is_spam_validate))
```

```
Epoch 1/10
101/101 [==============================] - 2s 6ms/step - loss: 2.4576 - accuracy:
0.6112 - val_loss: 0.5637 - val_accuracy: 0.7800
Epoch 2/10
101/101 [==============================] - 0s 4ms/step - loss: 0.6861 - accuracy:
0.7891 - val_loss: 0.6075 - val_accuracy: 0.6990
Epoch 3/10
101/101 [==============================] - 0s 2ms/step - loss: 0.5996 - accuracy:
0.8311 - val_loss: 0.3668 - val_accuracy: 0.8726
Epoch 4/10
101/101 [==============================] - 0s 3ms/step - loss: 0.4801 - accuracy:
0.8519 - val_loss: 0.3422 - val_accuracy: 0.8813
Epoch 5/10
101/101 [==============================] - 0s 3ms/step - loss: 0.4187 - accuracy:
0.8727 - val_loss: 0.3162 - val_accuracy: 0.8929
Epoch 6/10
101/101 [==============================] - 0s 2ms/step - loss: 0.4212 - accuracy:
0.8854 - val_loss: 0.2887 - val_accuracy: 0.8900
Epoch 7/10
101/101 [==============================] - 0s 3ms/step - loss: 0.3748 - accuracy:
0.8904 - val_loss: 0.3045 - val_accuracy: 0.8886
Epoch 8/10
101/101 [==============================] - 0s 2ms/step - loss: 0.3083 - accuracy:
0.9009 - val_loss: 0.3601 - val_accuracy: 0.8698
Epoch 9/10
101/101 [==============================] - 0s 3ms/step - loss: 0.2975 - accuracy:
0.9022 - val_loss: 0.5739 - val_accuracy: 0.8437
Epoch 10/10
101/101 [==============================] - 0s 3ms/step - loss: 0.4285 - accuracy:
0.8829 - val_loss: 0.5775 - val_accuracy: 0.7974
```

In [ ]:

```python
import matplotlib.pyplot as plt

# Wykres straty
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(fitting_progress.history['loss'], label='Train Loss')
plt.plot(fitting_progress.history['val_loss'], label='Validation Loss')
plt.title('Losses')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.xticks(range(0, epochs))
plt.legend()

# Wykres dokładności
plt.subplot(1, 2, 2)
plt.plot(fitting_progress.history['accuracy'], label='Train Accuracy')
plt.plot(fitting_progress.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.xticks(range(0, epochs))
plt.legend()

plt.tight_layout()
plt.show()
```

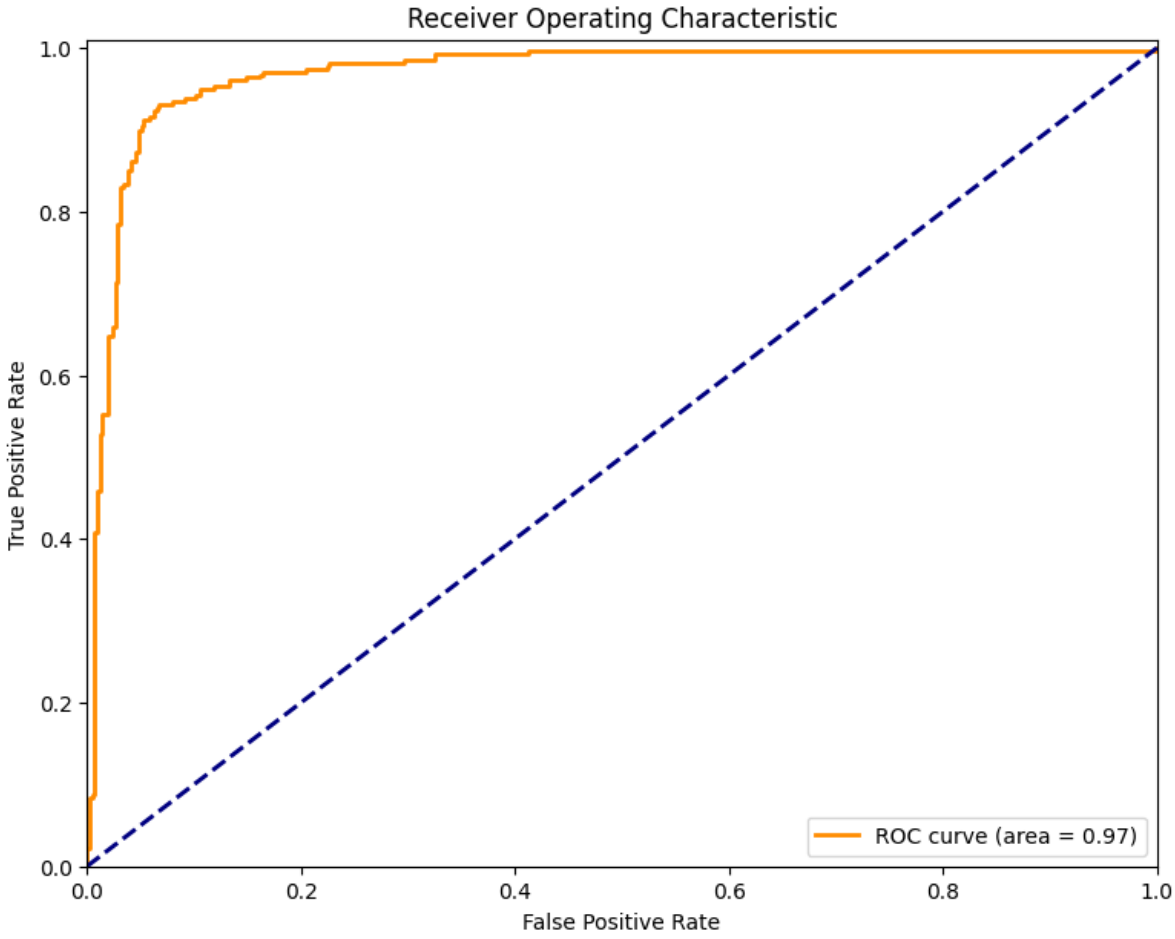


Jak widać na wykresach powyżej, sieć już w pierwszej wersji dobrze radzi sobie z powierzonym zadaniem, osiągając dokładność na poziomie 0.8829

In [ ]:
```
# Przewidywanie prawdopodobieństw
is_spam_prob = model.predict(data_test)
```

```
22/22 [==============================] - 0s 1ms/step
```

In [ ]:
```
from sklearn.metrics import roc_curve, auc

# Obliczanie krzywej ROC
fpr, tpr, thresholds = roc_curve(is_spam_test, is_spam_prob)

# Obliczanie AUC (Area Under Curve)
roc_auc = auc(fpr, tpr)

# Rysowanie krzywej ROC
plt.figure(figsize=(9, 7))
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.01])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Druga próba stworzenia sieci - dostrajanie parametrów

W drugim modelu sieci zdecydowaliśmy się na wariant 3 warstwowy z jedną warstwą ukrytą mającą 50 neuronów. Warstwa wejściowa została zaś rozszerzona do 1000 neuronów. Wybór takich wartości podyktowany był testami, w których najlepsze wyniki osiągaliśmy przy takiej dystrybucji neuronów. Liczba epok została zwiększona do 20, by dać sieci czas na lepszą "naukę" zależności.

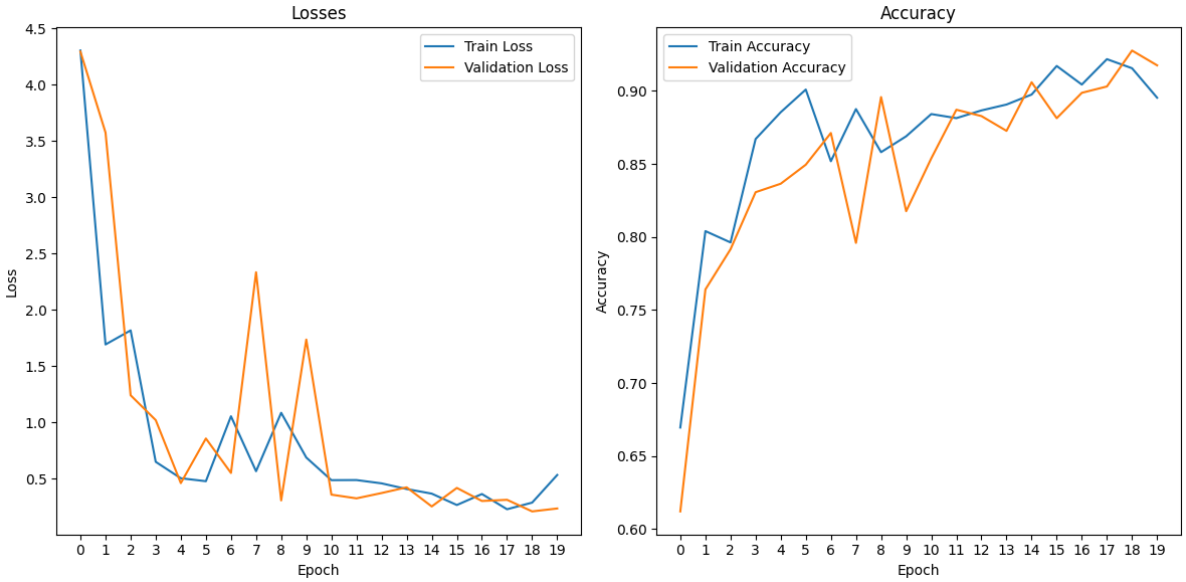
In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense

# Inicjalizacja modelu
model = Sequential()

# Dodanie warstw
model.add(Dense(1000, input_dim=data_train.shape[1], activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Kompilacja modelu
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Trenowanie modelu
epochs = 20
fitting_progress = model.fit(data_train, is_spam_train, epochs=epochs,
                             batch_size=32,
                             validation_data=(data_validate, is_spam_validate))
```

```
Epoch 1/20
101/101 [==============================] - 1s 4ms/step - loss: 4.3015 - accuracy:
0.6696 - val_loss: 4.2917 - val_accuracy: 0.6122
Epoch 2/20
101/101 [==============================] - 0s 3ms/step - loss: 1.6925 - accuracy:
0.8040 - val_loss: 3.5733 - val_accuracy: 0.7641
Epoch 3/20
101/101 [==============================] - 0s 3ms/step - loss: 1.8175 - accuracy:
0.7963 - val_loss: 1.2416 - val_accuracy: 0.7916
Epoch 4/20
101/101 [==============================] - 0s 3ms/step - loss: 0.6508 - accuracy:
0.8671 - val_loss: 1.0212 - val_accuracy: 0.8307
Epoch 5/20
101/101 [==============================] - 0s 3ms/step - loss: 0.5051 - accuracy:
0.8854 - val_loss: 0.4619 - val_accuracy: 0.8365
Epoch 6/20
101/101 [==============================] - 0s 3ms/step - loss: 0.4788 - accuracy:
0.9009 - val_loss: 0.8579 - val_accuracy: 0.8495
Epoch 7/20
101/101 [==============================] - 0s 3ms/step - loss: 1.0555 - accuracy:
0.8519 - val_loss: 0.5519 - val_accuracy: 0.8712
Epoch 8/20
101/101 [==============================] - 0s 3ms/step - loss: 0.5675 - accuracy:
0.8876 - val_loss: 2.3336 - val_accuracy: 0.7959
Epoch 9/20
101/101 [==============================] - 0s 3ms/step - loss: 1.0862 - accuracy:
0.8581 - val_loss: 0.3084 - val_accuracy: 0.8958
Epoch 10/20
101/101 [==============================] - 0s 3ms/step - loss: 0.6889 - accuracy:
0.8689 - val_loss: 1.7362 - val_accuracy: 0.8177
Epoch 11/20
101/101 [==============================] - 0s 3ms/step - loss: 0.4882 - accuracy:
0.8842 - val_loss: 0.3596 - val_accuracy: 0.8538
Epoch 12/20
101/101 [==============================] - 0s 3ms/step - loss: 0.4891 - accuracy:
0.8814 - val_loss: 0.3266 - val_accuracy: 0.8871
Epoch 13/20
101/101 [==============================] - 0s 3ms/step - loss: 0.4598 - accuracy:
0.8866 - val_loss: 0.3737 - val_accuracy: 0.8828
Epoch 14/20
101/101 [==============================] - 0s 3ms/step - loss: 0.4095 - accuracy:
0.8907 - val_loss: 0.4245 - val_accuracy: 0.8726
Epoch 15/20
101/101 [==============================] - 0s 3ms/step - loss: 0.3685 - accuracy:
0.8975 - val_loss: 0.2547 - val_accuracy: 0.9059
Epoch 16/20
101/101 [==============================] - 0s 3ms/step - loss: 0.2675 - accuracy:
0.9171 - val_loss: 0.4192 - val_accuracy: 0.8813
Epoch 17/20
101/101 [==============================] - 0s 3ms/step - loss: 0.3651 - accuracy:
0.9043 - val_loss: 0.3033 - val_accuracy: 0.8987
Epoch 18/20
101/101 [==============================] - 0s 3ms/step - loss: 0.2299 - accuracy:
0.9217 - val_loss: 0.3138 - val_accuracy: 0.9030
Epoch 19/20
101/101 [==============================] - 0s 3ms/step - loss: 0.2888 - accuracy:
0.9155 - val_loss: 0.2108 - val_accuracy: 0.9276
Epoch 20/20
101/101 [==============================] - 0s 3ms/step - loss: 0.5342 - accuracy:
0.8953 - val_loss: 0.2368 - val_accuracy: 0.9175
```

In [ ]:

```
import matplotlib.pyplot as plt

# Wykres straty
```

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(fitting_progress.history['loss'], label='Train Loss')
plt.plot(fitting_progress.history['val_loss'], label='Validation Loss')
plt.title('Losses')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.xticks(range(0, epochs))
plt.legend()

# Wykres dokładności
plt.subplot(1, 2, 2)
plt.plot(fitting_progress.history['accuracy'], label='Train Accuracy')
plt.plot(fitting_progress.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.xticks(range(0, epochs))
plt.legend()

plt.tight_layout()
plt.show()
```



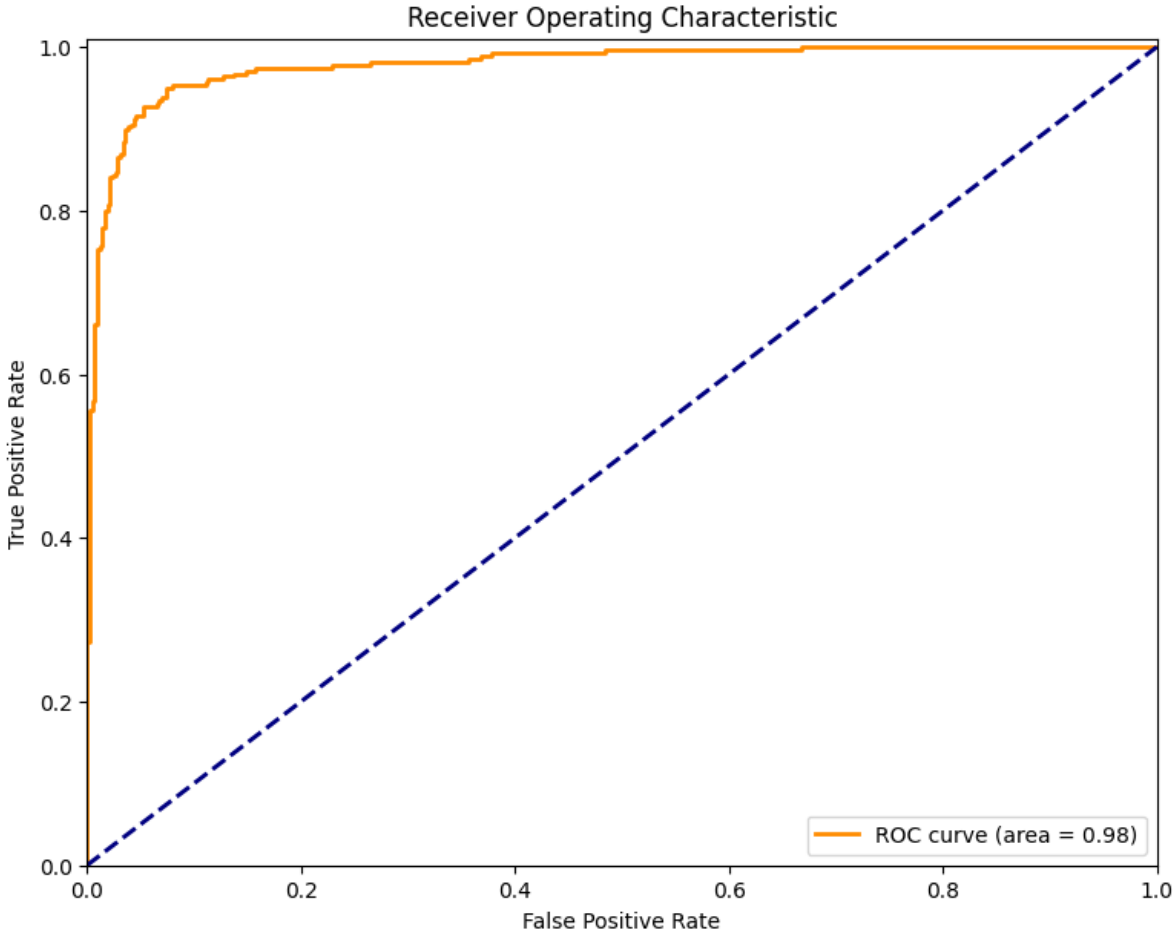
Tak jak przy pierwszej wersji sieci, druga sieć osiągneła wysoką dokładność. Zmiany w strukturze modelu miały niewielki wpływ na poprawę wyników, co wynika z silnej korelacji między atrybutami w naszym zbiorze danych

In [ ]:
```
# Przewidywanie prawdopodobieństw
is_spam_prob = model.predict(data_test)
```

```
22/22 [==============================] - 0s 1ms/step
```

In [ ]:
```
from sklearn.metrics import roc_curve, auc

# Obliczanie krzywej ROC
fpr, tpr, thresholds = roc_curve(is_spam_test, is_spam_prob)

# Obliczanie AUC (Area Under Curve)
roc_auc = auc(fpr, tpr)

# Rysowanie krzywej ROC
plt.figure(figsize=(9, 7))
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label='ROC curve (area = %0.2f)' % roc_auc)
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.01])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Jak widać, nasza sieć świetnie kalsyfikuje maile ze spamem jako spam. Dodanie znacznie większej liczby neuronów w warstwach oraz podwojenie liczby iteracji nieznacznie zwiększyło dokładność naszej sieci, która i tak już przy pierwszej próbie odznaczała się bardzo małym błędem.

Wnioski

Nasza sieć spełniła oczekiwania. Bardzo dobrze rozpoznaje maile jako spam na podstawie danych ze zbioru. Ostateczne pole pod krzywą ROC wyniosło 0,98, co jest świetnym wynikiem.