Title : Histopathologic Cancer Detection

you must create an algorithm to identify metastatic cancer in small image patches taken from larger digital pathology scans. This dataset was provided by Bas Veeling, with additional input from Babak Ehteshami Bejnordi, Geert Litjens, and Jeroen van der Laak.

1. Prepare the Environment and Load Data

Here, you can import the library, set the path, and import the csv file.

In [1]:

```
# Libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import cv2
import tensorflow as tf
from tensorflow.keras.optimizers import RMSprop
```

In [2]:

```
#Setting up the Path
test_path = "../input/histopathologic-cancer-detection/test/"
train_path = "../input/histopathologic-cancer-detection/train/"
path = "../input/histopathologic-cancer-detection/"
train_files       = os.listdir(train_path)
test_files        = os.listdir(test_path)
```

In [3]:

```
# Load csv file
labels = pd.read_csv(path+"train_labels.csv")
labels
```

Out[3]:

	id	label
0	f38a6374c348f90b587e046aac6079959adf3835	0
1	c18f2d887b7ae4f6742ee445113fa1aef383ed77	1
2	755db6279dae599ebb4d39a9123cce439965282d	0
3	bc3f0c64fb968ff4a8bd33af6971ecae77c75e08	0
4	068aba587a4950175d04c680d38943fd488d6a9d	0
...	...	...
220020	53e9aa9d46e720bf3c6a7528d1fca3ba6e2e49f6	0
220021	d4b854fe38b07fe2831ad73892b3cec877689576	1
220022	3d046cead1a2a5cbe00b2b4847cfb7ba7cf5fe75	0
220023	f129691c13433f66e1e0671ff1fe80944816f5a2	0
220024	a81f84895ddcd522302ddf34be02eb1b3e5af1cb	1

220025 rows × 2 columns

2. EDA

Explore the data to understand how the data is organized.

In [4]:
```
labels['label'].value_counts()
```

Out[4]:
```
0    130908
1     89117
Name: label, dtype: int64
```

Label distinguishes whether it is cancer or not, and 1 is cancer and 0 is not cancer.

According to the label data, there are 89117 data classified as cancer, and 130908 data are not cancer.

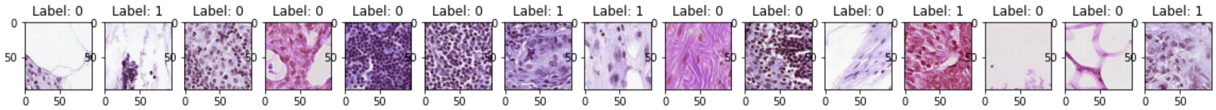
In [5]:
```
cancer_labels = ["No Cancer", "Cancer"]
values = labels.label.value_counts()

chart_donut = go.Figure(data=[go.Pie(labels=cancer_labels, values=values, hole=.5, ma
chart_donut.show()
```

According to the train data, 40.5% of the total images were diagnosed as cancer.

In [6]:
```
number_images = 15
```

```
fig, axs = plt.subplots(1, len(labels[:number_images]), figsize = (20, 2))
for idx, ax in enumerate(axs):
    ax.imshow(cv2.imread(train_path + labels.id[idx] + ".tif"))
    ax.set_title("Label: " + str(labels.label[idx]))
```



Match the image with the label to roughly identify which image is cancer.

In [7]:

```
def img_prep(directory, files, start = 0, end = -1, test=False):
    if end == -1:
        end = len(files)
    X = []
    if test:
        for image in files:
            img = cv2.imread( directory + image)
            img = cv2.resize(img, (96, 96))
            X.append(img)
        print("Image shape: ",X[0].shape)
        X = np.array(X)
        return X
    else:
        for image in files.id[start:end]:
            img = cv2.imread( directory + image + ".tif")
            img = cv2.resize(img, (96, 96))
            X.append(img)
        print("Image shape: ",X[0].shape)
        X = np.array(X)


        y = files.label[start:end]
        return X, y
```

Change the image size to 96x96. To learn from CNN, the image must be of the same size.

In [8]:

```
X_train, y_train = img_prep(train_path, labels, start=10_000, end = 120_000)
```

```
Image shape:  (96, 96, 3)
```

In [9]:

```
test = img_prep(test_path, test_files, test=True)
```

```
Image shape:  (96, 96, 3)
```

3. Build CNN Model

A CNN model is generated using the tensorflow framework. Check the accuracy while adding layers, and also add dropout to prevent overfitting.

Create a Model

> **Model 1**

In [10]:

```
model = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(96, 96
            tf.keras.layers.MaxPooling2D(2, 2),
```

```
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.Dense(1, activation='sigmoid')
            ])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
2022-07-17 18:28:34.866355: I tensorflow/core/common_runtime/process_util.cc:146] Crea
ting new thread pool with default inter op setting: 2. Tune using inter_op_parallelism
_threads for best performance.
```

Train a Model

In [11]:

```
history = model.fit(X_train, y_train, epochs=20, validation_split=.2)
```

```
2022-07-17 18:28:39.678466: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.c
c:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/20
2750/2750 [==============================] - 273s 99ms/step - loss: 1.8302 - accuracy:
0.5931 - val_loss: 0.6268 - val_accuracy: 0.5958
Epoch 2/20
2750/2750 [==============================] - 271s 98ms/step - loss: 0.6408 - accuracy:
0.5947 - val_loss: 0.6467 - val_accuracy: 0.5955
Epoch 3/20
2750/2750 [==============================] - 270s 98ms/step - loss: 0.6682 - accuracy:
0.5953 - val_loss: 0.6749 - val_accuracy: 0.5955
Epoch 4/20
2750/2750 [==============================] - 272s 99ms/step - loss: 0.6749 - accuracy:
0.5956 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 5/20
2750/2750 [==============================] - 275s 100ms/step - loss: 0.6748 - accurac
y: 0.5957 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 6/20
2750/2750 [==============================] - 277s 101ms/step - loss: 0.6778 - accurac
y: 0.5957 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 7/20
2750/2750 [==============================] - 280s 102ms/step - loss: 0.6748 - accurac
y: 0.5958 - val_loss: 0.6751 - val_accuracy: 0.5955
Epoch 8/20
2750/2750 [==============================] - 281s 102ms/step - loss: 0.6748 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 9/20
2750/2750 [==============================] - 281s 102ms/step - loss: 0.6748 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 10/20
2750/2750 [==============================] - 277s 101ms/step - loss: 0.6747 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 11/20
2750/2750 [==============================] - 278s 101ms/step - loss: 0.6747 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 12/20
2750/2750 [==============================] - 280s 102ms/step - loss: 0.6748 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 13/20
2750/2750 [==============================] - 280s 102ms/step - loss: 0.6747 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 14/20
2750/2750 [==============================] - 281s 102ms/step - loss: 0.6748 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 15/20
2750/2750 [==============================] - 282s 102ms/step - loss: 0.6747 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 16/20
2750/2750 [==============================] - 283s 103ms/step - loss: 0.6747 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 17/20
```

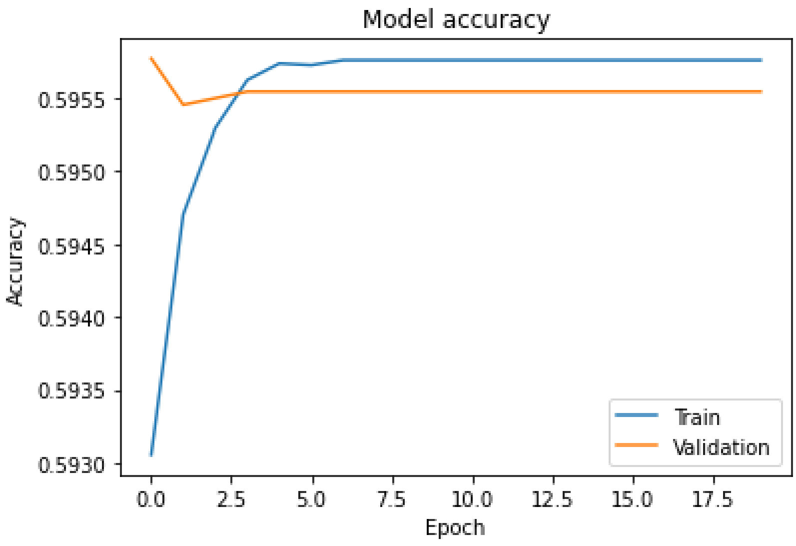
```
2750/2750 [==============================] - 283s 103ms/step - loss: 0.6747 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 18/20
2750/2750 [==============================] - 284s 103ms/step - loss: 0.6747 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 19/20
2750/2750 [==============================] - 283s 103ms/step - loss: 0.6747 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 20/20
2750/2750 [==============================] - 280s 102ms/step - loss: 0.6747 - accurac
y: 0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
```

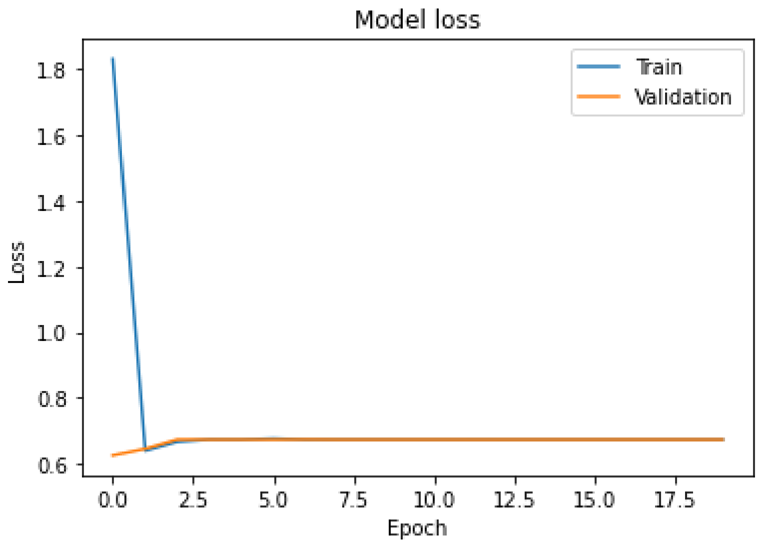
Check accuracy values

In [12]:

```python
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```





Model result

Train data has high accuracy and very little loss. However, if you look at the validation data, you will get different results. This can lead to the conclusion that overfitting has occurred severely.

> **Model 2**

Added one more CNN layer and adjusted the Neuron number.

In [13]:

```
model2 = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(12, (3, 3), activation='relu', input_shape=(96, 96
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.Dense(1, activation='sigmoid')
            ])
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [14]:

```
history2 = model2.fit(X_train, y_train, epochs=20, validation_split=.2)
```

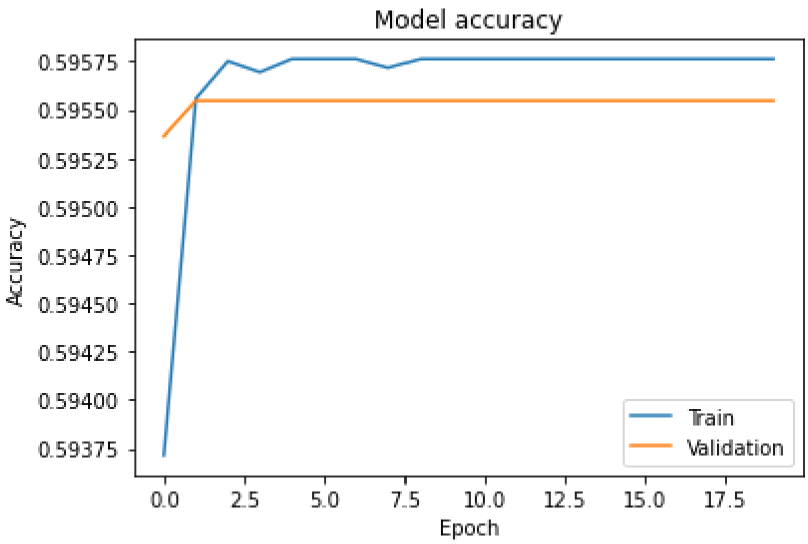
```
Epoch 1/20
2750/2750 [==============================] - 182s 66ms/step - loss: 0.9470 - accuracy:
0.5937 - val_loss: 0.6739 - val_accuracy: 0.5954
Epoch 2/20
2750/2750 [==============================] - 182s 66ms/step - loss: 0.6780 - accuracy:
0.5956 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 3/20
2750/2750 [==============================] - 182s 66ms/step - loss: 0.6757 - accuracy:
0.5957 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 4/20
2750/2750 [==============================] - 181s 66ms/step - loss: 0.6748 - accuracy:
0.5957 - val_loss: 0.6759 - val_accuracy: 0.5955
Epoch 5/20
2750/2750 [==============================] - 182s 66ms/step - loss: 0.6748 - accuracy:
0.5958 - val_loss: 0.6752 - val_accuracy: 0.5955
Epoch 6/20
2750/2750 [==============================] - 182s 66ms/step - loss: 0.6748 - accuracy:
0.5958 - val_loss: 0.6750 - val_accuracy: 0.5955
Epoch 7/20
2750/2750 [==============================] - 183s 66ms/step - loss: 0.6748 - accuracy:
```

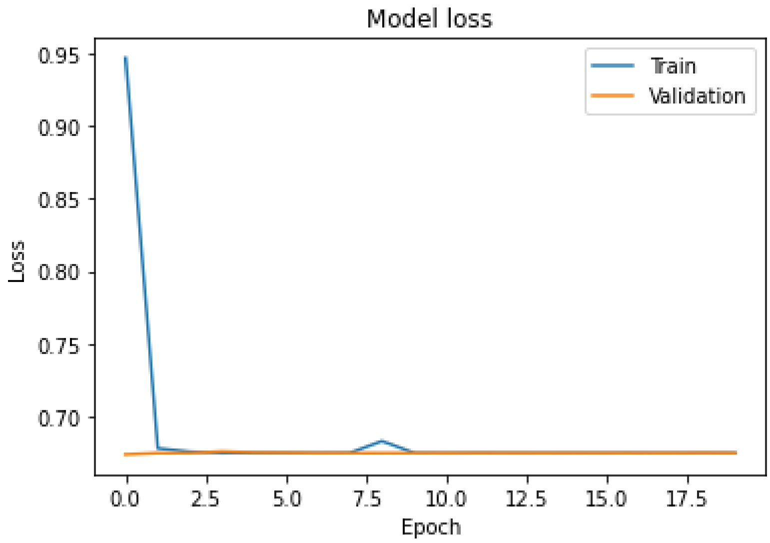
```
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 8/20
2750/2750 [==============================] - 183s 66ms/step - loss: 0.6749 - accuracy:
0.5957 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 9/20
2750/2750 [==============================] - 180s 65ms/step - loss: 0.6830 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 10/20
2750/2750 [==============================] - 183s 67ms/step - loss: 0.6748 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 11/20
2750/2750 [==============================] - 183s 67ms/step - loss: 0.6747 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 12/20
2750/2750 [==============================] - 182s 66ms/step - loss: 0.6748 - accuracy:
0.5958 - val_loss: 0.6749 - val_accuracy: 0.5955
Epoch 13/20
2750/2750 [==============================] - 184s 67ms/step - loss: 0.6748 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 14/20
2750/2750 [==============================] - 182s 66ms/step - loss: 0.6747 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 15/20
2750/2750 [==============================] - 188s 68ms/step - loss: 0.6748 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 16/20
2750/2750 [==============================] - 181s 66ms/step - loss: 0.6747 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 17/20
2750/2750 [==============================] - 186s 67ms/step - loss: 0.6748 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 18/20
2750/2750 [==============================] - 182s 66ms/step - loss: 0.6747 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
Epoch 19/20
2750/2750 [==============================] - 186s 68ms/step - loss: 0.6748 - accuracy:
0.5958 - val_loss: 0.6749 - val_accuracy: 0.5955
Epoch 20/20
2750/2750 [==============================] - 183s 67ms/step - loss: 0.6747 - accuracy:
0.5958 - val_loss: 0.6748 - val_accuracy: 0.5955
```

In [15]:

```
# Plot training & validation accuracy values
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()

# Plot training & validation loss values
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```





Model Result

It seems to have improved compared to Model 1, but overfitting is still severe.

> **Model 3**

Dropout was added, and a Dense layer was added. Optimizer changed from Adam to RMSprop.

In [16]:

```
model3 = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(96, 96
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.Dense(128, activation='relu'),
            tf.keras.layers.Dense(1, activation='sigmoid')
            ])
model3.compile(optimizer=RMSprop(learning_rate=0.0001), loss='binary_crossentropy', m
```

In [17]:

```
history3 = model3.fit(X_train, y_train, epochs=20, validation_split=.2)
```

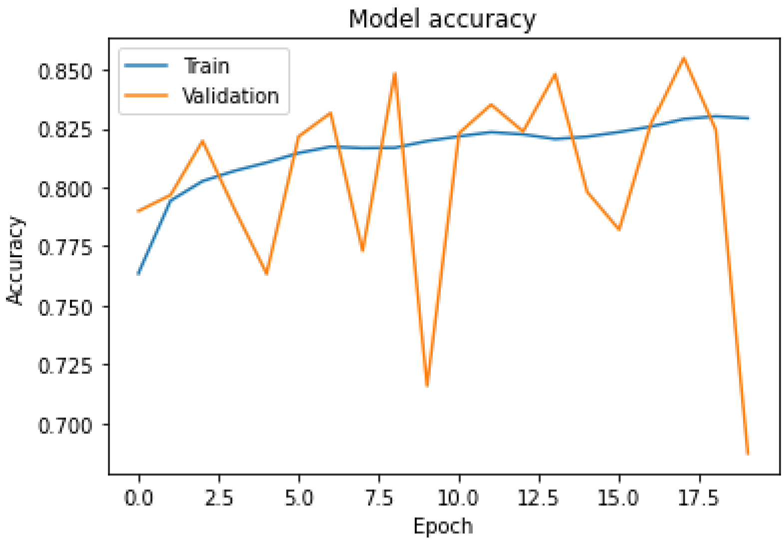
```
Epoch 1/20
```

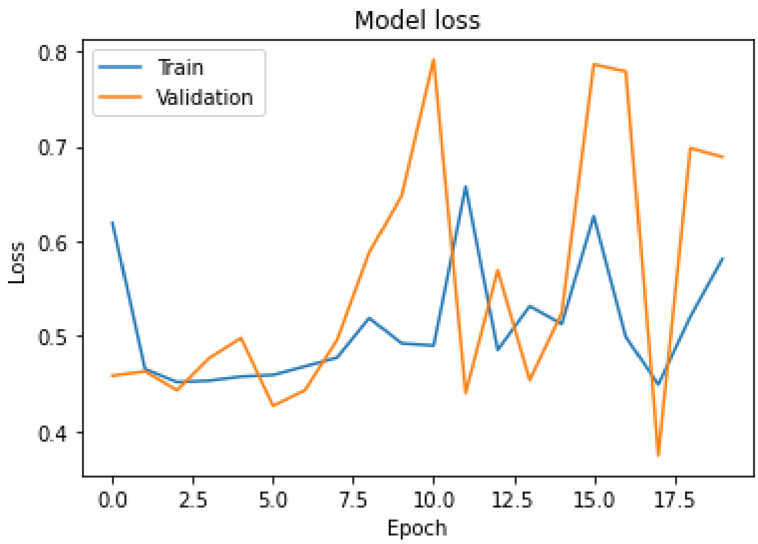
```
2750/2750 [==============================] - 541s 196ms/step - loss: 0.6192 - accurac
y: 0.7635 - val_loss: 0.4581 - val_accuracy: 0.7900
Epoch 2/20
2750/2750 [==============================] - 544s 198ms/step - loss: 0.4651 - accurac
y: 0.7943 - val_loss: 0.4627 - val_accuracy: 0.7967
Epoch 3/20
2750/2750 [==============================] - 546s 198ms/step - loss: 0.4514 - accurac
y: 0.8026 - val_loss: 0.4427 - val_accuracy: 0.8197
Epoch 4/20
2750/2750 [==============================] - 542s 197ms/step - loss: 0.4528 - accurac
y: 0.8070 - val_loss: 0.4765 - val_accuracy: 0.7906
Epoch 5/20
2750/2750 [==============================] - 545s 198ms/step - loss: 0.4572 - accurac
y: 0.8105 - val_loss: 0.4979 - val_accuracy: 0.7634
Epoch 6/20
2750/2750 [==============================] - 545s 198ms/step - loss: 0.4589 - accurac
y: 0.8146 - val_loss: 0.4263 - val_accuracy: 0.8216
Epoch 7/20
2750/2750 [==============================] - 550s 200ms/step - loss: 0.4680 - accurac
y: 0.8173 - val_loss: 0.4429 - val_accuracy: 0.8316
Epoch 8/20
2750/2750 [==============================] - 542s 197ms/step - loss: 0.4772 - accurac
y: 0.8166 - val_loss: 0.4959 - val_accuracy: 0.7730
Epoch 9/20
2750/2750 [==============================] - 549s 200ms/step - loss: 0.5189 - accurac
y: 0.8168 - val_loss: 0.5879 - val_accuracy: 0.8484
Epoch 10/20
2750/2750 [==============================] - 552s 201ms/step - loss: 0.4922 - accurac
y: 0.8196 - val_loss: 0.6478 - val_accuracy: 0.7159
Epoch 11/20
2750/2750 [==============================] - 551s 200ms/step - loss: 0.4899 - accurac
y: 0.8217 - val_loss: 0.7912 - val_accuracy: 0.8230
Epoch 12/20
2750/2750 [==============================] - 551s 200ms/step - loss: 0.6576 - accurac
y: 0.8234 - val_loss: 0.4399 - val_accuracy: 0.8351
Epoch 13/20
2750/2750 [==============================] - 555s 202ms/step - loss: 0.4853 - accurac
y: 0.8225 - val_loss: 0.5694 - val_accuracy: 0.8237
Epoch 14/20
2750/2750 [==============================] - 555s 202ms/step - loss: 0.5315 - accurac
y: 0.8205 - val_loss: 0.4534 - val_accuracy: 0.8481
Epoch 15/20
2750/2750 [==============================] - 555s 202ms/step - loss: 0.5126 - accurac
y: 0.8215 - val_loss: 0.5243 - val_accuracy: 0.7980
Epoch 16/20
2750/2750 [==============================] - 552s 201ms/step - loss: 0.6264 - accurac
y: 0.8235 - val_loss: 0.7860 - val_accuracy: 0.7820
Epoch 17/20
2750/2750 [==============================] - 555s 202ms/step - loss: 0.4991 - accurac
y: 0.8258 - val_loss: 0.7786 - val_accuracy: 0.8274
Epoch 18/20
2750/2750 [==============================] - 549s 200ms/step - loss: 0.4487 - accurac
y: 0.8290 - val_loss: 0.3745 - val_accuracy: 0.8550
Epoch 19/20
2750/2750 [==============================] - 552s 201ms/step - loss: 0.5203 - accurac
y: 0.8302 - val_loss: 0.6982 - val_accuracy: 0.8244
Epoch 20/20
2750/2750 [==============================] - 553s 201ms/step - loss: 0.5812 - accurac
y: 0.8293 - val_loss: 0.6890 - val_accuracy: 0.6871
```

In [18]:

```
# Plot training & validation accuracy values
plt.plot(history3.history['accuracy'])
plt.plot(history3.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```

```
# Plot training & validation loss values
plt.plot(history3.history['loss'])
plt.plot(history3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```





Model result

It is better than Model 2. However, the model performance is very poor. Accuracy does not seem to have converged, and Loss does not converge and comes out very high.

> **Model 4**

In [19]:

```
model4 = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(96, 96
            tf.keras.layers.Dropout(0.25),
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
            tf.keras.layers.Dropout(0.25),
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
            tf.keras.layers.Dropout(0.25),
```

```python
            tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Dropout(0.25),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.Dense(1, activation='sigmoid')
            ])
model4.compile(optimizer=RMSprop(learning_rate=0.0001), loss='binary_crossentropy', m
```

In [20]:

```python
history4 = model4.fit(X_train, y_train, epochs=20, validation_split=.2)
```

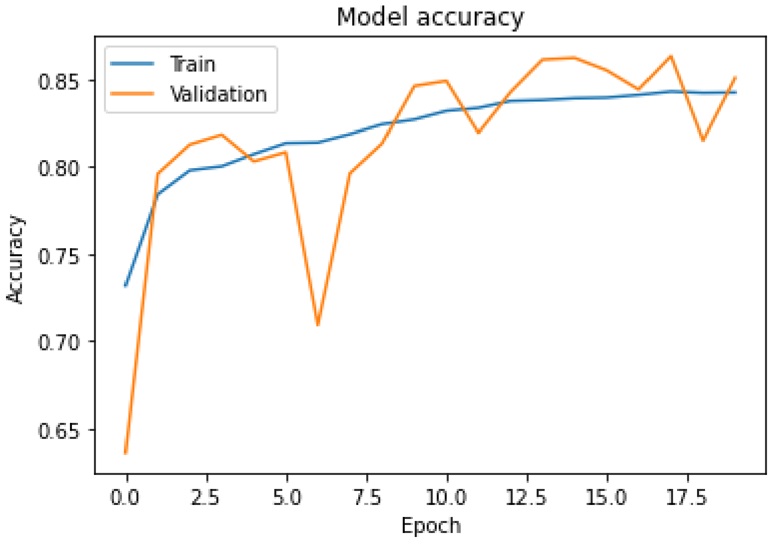
```
Epoch 1/20
2750/2750 [==============================] - 745s 270ms/step - loss: 1.3592 - accurac
y: 0.7319 - val_loss: 0.5981 - val_accuracy: 0.6362
Epoch 2/20
2750/2750 [==============================] - 739s 269ms/step - loss: 0.4803 - accurac
y: 0.7842 - val_loss: 0.5036 - val_accuracy: 0.7960
Epoch 3/20
2750/2750 [==============================] - 742s 270ms/step - loss: 0.4670 - accurac
y: 0.7979 - val_loss: 0.5299 - val_accuracy: 0.8127
Epoch 4/20
2750/2750 [==============================] - 739s 269ms/step - loss: 0.4625 - accurac
y: 0.8002 - val_loss: 0.5144 - val_accuracy: 0.8183
Epoch 5/20
2750/2750 [==============================] - 755s 274ms/step - loss: 0.4458 - accurac
y: 0.8072 - val_loss: 0.5159 - val_accuracy: 0.8030
Epoch 6/20
2750/2750 [==============================] - 744s 270ms/step - loss: 0.4365 - accurac
y: 0.8135 - val_loss: 0.4675 - val_accuracy: 0.8083
Epoch 7/20
2750/2750 [==============================] - 743s 270ms/step - loss: 0.4400 - accurac
y: 0.8138 - val_loss: 0.5590 - val_accuracy: 0.7092
Epoch 8/20
2750/2750 [==============================] - 741s 270ms/step - loss: 0.4288 - accurac
y: 0.8186 - val_loss: 0.4937 - val_accuracy: 0.7962
Epoch 9/20
2750/2750 [==============================] - 750s 273ms/step - loss: 0.4204 - accurac
y: 0.8242 - val_loss: 0.4971 - val_accuracy: 0.8132
Epoch 10/20
2750/2750 [==============================] - 750s 273ms/step - loss: 0.4081 - accurac
y: 0.8270 - val_loss: 0.4643 - val_accuracy: 0.8461
Epoch 11/20
2750/2750 [==============================] - 742s 270ms/step - loss: 0.3941 - accurac
y: 0.8319 - val_loss: 0.4675 - val_accuracy: 0.8490
Epoch 12/20
2750/2750 [==============================] - 744s 270ms/step - loss: 0.3922 - accurac
y: 0.8335 - val_loss: 0.4481 - val_accuracy: 0.8193
Epoch 13/20
2750/2750 [==============================] - 743s 270ms/step - loss: 0.3912 - accurac
y: 0.8375 - val_loss: 0.4312 - val_accuracy: 0.8425
Epoch 14/20
2750/2750 [==============================] - 746s 271ms/step - loss: 0.3866 - accurac
y: 0.8380 - val_loss: 0.4356 - val_accuracy: 0.8612
Epoch 15/20
2750/2750 [==============================] - 754s 274ms/step - loss: 0.3847 - accurac
y: 0.8390 - val_loss: 0.4446 - val_accuracy: 0.8622
Epoch 16/20
2750/2750 [==============================] - 760s 276ms/step - loss: 0.3909 - accurac
y: 0.8394 - val_loss: 0.4552 - val_accuracy: 0.8551
Epoch 17/20
2750/2750 [==============================] - 765s 278ms/step - loss: 0.3809 - accurac
y: 0.8410 - val_loss: 0.4583 - val_accuracy: 0.8441
Epoch 18/20
2750/2750 [==============================] - 761s 277ms/step - loss: 0.3801 - accurac
y: 0.8429 - val_loss: 0.4355 - val_accuracy: 0.8632
Epoch 19/20
2750/2750 [==============================] - 750s 273ms/step - loss: 0.3794 - accurac
y: 0.8421 - val_loss: 0.5040 - val_accuracy: 0.8150
```

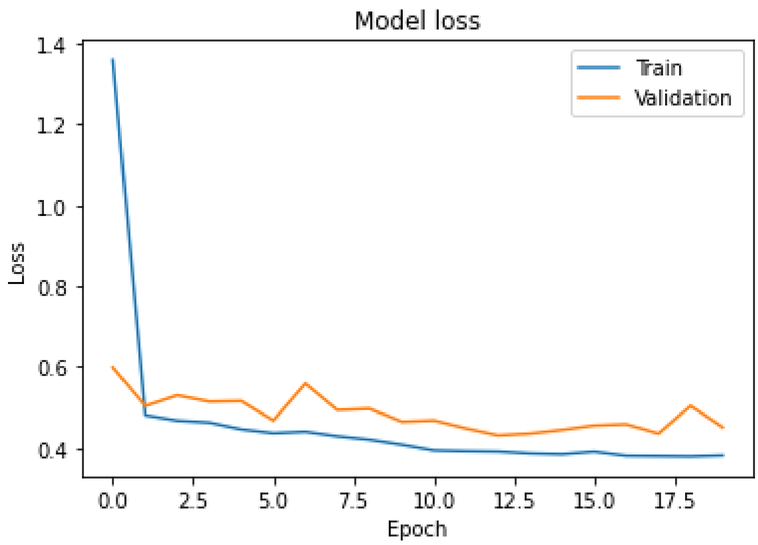
```
Epoch 20/20
2750/2750 [==============================] - 748s 272ms/step - loss: 0.3816 - accurac
y: 0.8424 - val_loss: 0.4508 - val_accuracy: 0.8507
```

In [21]:

```
# Plot training & validation accuracy values
plt.plot(history4.history['accuracy'])
plt.plot(history4.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()

# Plot training & validation loss values
plt.plot(history4.history['loss'])
plt.plot(history4.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```





Model result

It has improved a lot compared to the previous models. Both accuracy and loss are converging, and performance seems to have improved.

Predict Labels

In [22]:
```
pred_test = model4.predict(test)
```

4. Save Results

Save as a Submission.csv file.

In [23]:
```
#Prepare Submission.csv file
lst = []
for item in test_files:
    lst.append(item[:-4])

test_df = pd.DataFrame(lst)
test_df.head()
```

Out[23]:

	0
0	a7ea26360815d8492433b14cd8318607bcf99d9e
1	59d21133c845dff1ebc7a0c7cf40c145ea9e9664
2	5fde41ce8c6048a5c2f38eca12d6528fa312cdbb
3	bd953a3b1db1f7041ee95ff482594c4f46c73ed0
4	523fc2efd7aba53e597ab0f69cc2cbded7a6ce62

In [24]:
```
#Create Submission.csv file
predictions = np.array(pred_test)
test_df["label"] = predictions
test_df.columns = ["id", "label"]
submission = test_df

print(submission.head())
submission.to_csv("submission.csv", index = False, header = True)
```

```
                                         id     label
0  a7ea26360815d8492433b14cd8318607bcf99d9e  0.375566
1  59d21133c845dff1ebc7a0c7cf40c145ea9e9664  0.202148
2  5fde41ce8c6048a5c2f38eca12d6528fa312cdbb  0.244340
3  bd953a3b1db1f7041ee95ff482594c4f46c73ed0  0.470297
4  523fc2efd7aba53e597ab0f69cc2cbded7a6ce62  0.318640
```

In [ ]:
```

```