Niloufar Baba Ahmadi 610398103 HW3

Mount Google Drive to access files

```
from google.colab import drive
drive.mount('/content/drive')
```

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

Imports

```
import cv2
import os
import numpy as np
import shutil
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
import pickle
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
```

Data Augmentation and Dataset Separation

This piece of code performs image data augmentation and dataset separation into training and test sets.

1. Image Processing:
    - A list, `image_paths`, is initialized to store the paths of the processed images.
    - For each file in the dataset directory:
        - If the file ends with the extension ".tif":
            - The image is loaded using OpenCV and stored in `img`.
            - If the image has more than two dimensions, it is converted to grayscale.
            - The image is horizontally flipped, and the flipped image is saved with a filename prefix of "aug_horiz_".
            - The flipped image's path is appended to `image_paths`.
            - Histogram equalization is applied to the original image, and the result is saved with a filename prefix of "aug_eqhist_".
            - The equalized image's path is appended to `image_paths`.

- The image is rotated 90 degrees clockwise, and the rotated image is saved with a filename prefix of "aug_rot90_".
- The rotated image's path is appended to `image_paths`.
- The image is translated using a predefined translation matrix, and the translated image is saved with a filename prefix of "aug_trans_".
- The translated image's path is appended to `image_paths`.
- The image is sheared using a predefined shearing matrix, and the sheared image is saved with a filename prefix of "aug_shear_".
- The sheared image's path is appended to `image_paths`.

2. Counting:
   - The total number of color images is counted by iterating through `image_paths` and checking if each image has more than two dimensions.
   - The count of color images is printed.
3. Dataset Separation:
   - The `train_test_split` function is used to split `image_paths` into training and test sets with a test size of 0.2 and a random state of 42.

```
# Path to dataset
dataset_path = '/content/drive/MyDrive/image
processing/dataset/dataset'

# A new directory to store augmented images
augmented_path = '/content/drive/MyDrive/image processing/augmented
images'
os.makedirs(augmented_path, exist_ok=True)

# Lists to store image paths
image_paths = []

# Iterating through the images in the dataset
for filename in os.listdir(dataset_path):
    if filename.endswith('.tif'):
        img_path = os.path.join(dataset_path, filename)
        image_paths.append(img_path)

        # Loading the image
        img = cv2.imread(img_path, cv2.IMREAD_UNCHANGED)

        # Converting the image to grayscale
        if img.ndim > 2:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Flip the image horizontally
        flipped_img = cv2.flip(img, 1)
        augmented_filename = 'aug_horiz_' + filename
```

```
        augmented_filepath = os.path.join(augmented_path,
augmented_filename)
        cv2.imwrite(augmented_filepath, flipped_img)
        image_paths.append(augmented_filepath)

        # Apply histogram equalization
        equalized_img = cv2.equalizeHist(img)
        augmented_filename = 'aug_eqhist_' + filename
        augmented_filepath = os.path.join(augmented_path,
augmented_filename)
        cv2.imwrite(augmented_filepath, equalized_img)
        image_paths.append(augmented_filepath)

        # Rotate the image by 90 degrees
        rotated_img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
        augmented_filename = 'aug_rot90_' + filename
        augmented_filepath = os.path.join(augmented_path,
augmented_filename)
        cv2.imwrite(augmented_filepath, rotated_img)
        image_paths.append(augmented_filepath)

        # Translate the image
        M = np.float32([[1, 0, 50], [0, 1, 50]])  # Translation matrix
        translated_img = cv2.warpAffine(img, M, (img.shape[1],
img.shape[0]))
        augmented_filename = 'aug_trans_' + filename
        augmented_filepath = os.path.join(augmented_path,
augmented_filename)
        cv2.imwrite(augmented_filepath, translated_img)
        image_paths.append(augmented_filepath)

        # Shear the image
        M = np.float32([[1, 0.2, 0], [0.2, 1, 0]])  # Shearing matrix
        sheared_img = cv2.warpAffine(img, M, (img.shape[1],
img.shape[0]))
        augmented_filename = 'aug_shear_' + filename
        augmented_filepath = os.path.join(augmented_path,
augmented_filename)
        cv2.imwrite(augmented_filepath, sheared_img)
        image_paths.append(augmented_filepath)

print("Data augmentation is complete!")

# Counting the total number of color images
total_color_images = sum(1 for path in image_paths if
cv2.imread(path).ndim > 2)
print("Total color images after data augmentation:",
total_color_images)
```

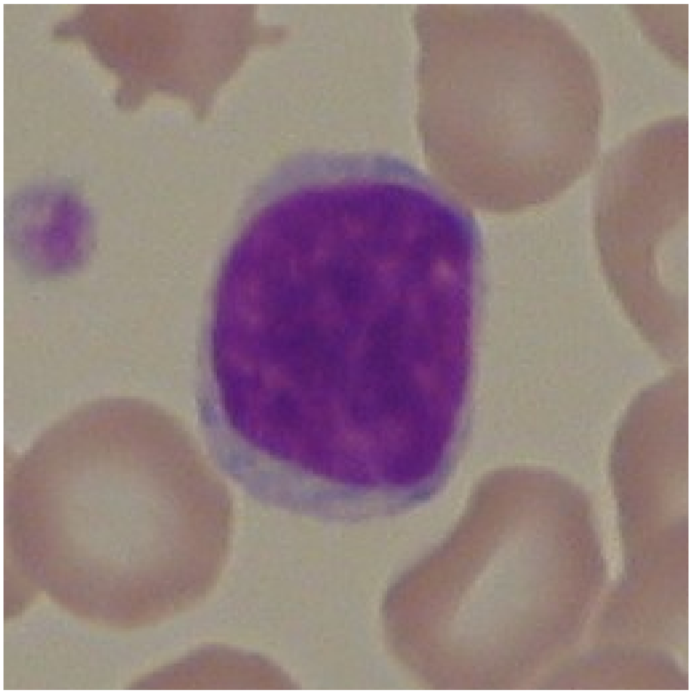
```
# Split the dataset into training and test sets
train_paths, test_paths = train_test_split(image_paths, test_size=0.2,
random_state=42)

# Creating directories for the training and test sets
train_path = '/content/drive/MyDrive/image processing/train set'
test_path = '/content/drive/MyDrive/image processing/test set'
os.makedirs(train_path, exist_ok=True)
os.makedirs(test_path, exist_ok=True)

# Copy the training images to the train dataset folder
for path in train_paths:
    filename = os.path.basename(path)
    destination_path = os.path.join(train_path, filename)
    shutil.copy2(path, destination_path)

# Copy the test images to the test dataset folder
for path in test_paths:
    filename = os.path.basename(path)
    destination_path = os.path.join(test_path, filename)
    shutil.copy2(path, destination_path)

print("Data separation is complete!")
```

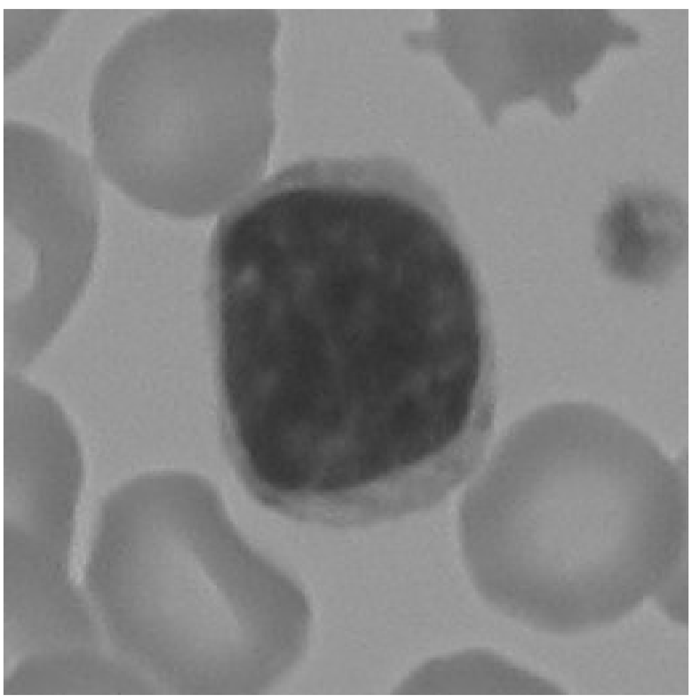
```
Mounted at /content/drive
Data augmentation is complete!
Total color images after data augmentation: 1560
Data separation is complete!
```

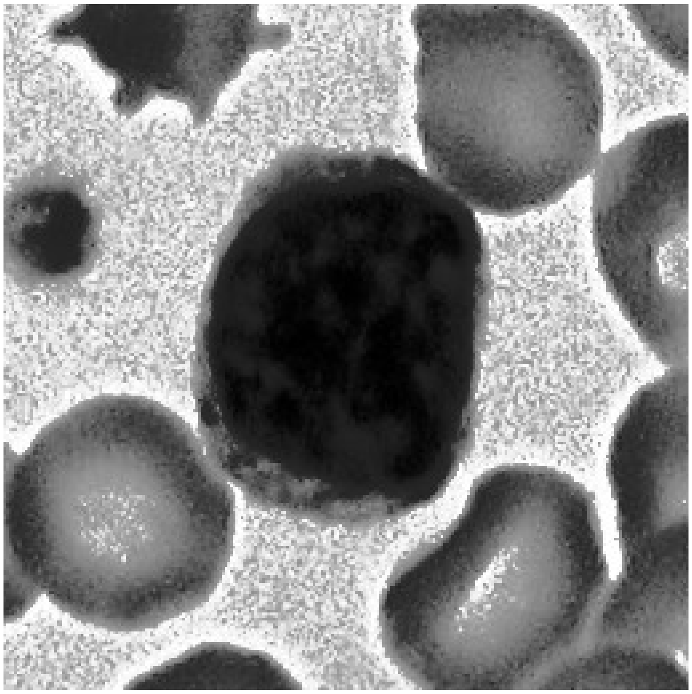
The original image and the augmented ones

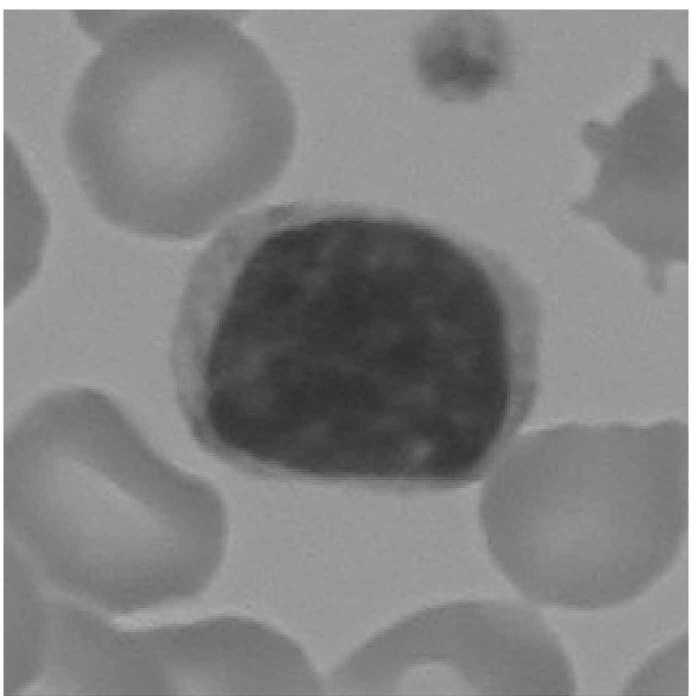
```
from google.colab.patches import cv2_imshow

original_image_path = '/content/drive/MyDrive/image
processing/dataset/dataset/Im259_0.tif'
original_image = cv2.imread(original_image_path, cv2.IMREAD_UNCHANGED)
augmented_images_path = '/content/drive/MyDrive/image
processing/augmented images'

# Iterating through the augmented images
for filename in os.listdir(augmented_images_path):
    if filename.startswith('aug_') and
filename.endswith('Im259_0.tif'):
        augmented_image_path = os.path.join(augmented_images_path,
filename)

        # Load and display each augmented image
        augmented_image = cv2.imread(augmented_image_path,
cv2.IMREAD_UNCHANGED)
        if augmented_image is None:
            print(f"Failed to load the augmented image from
```

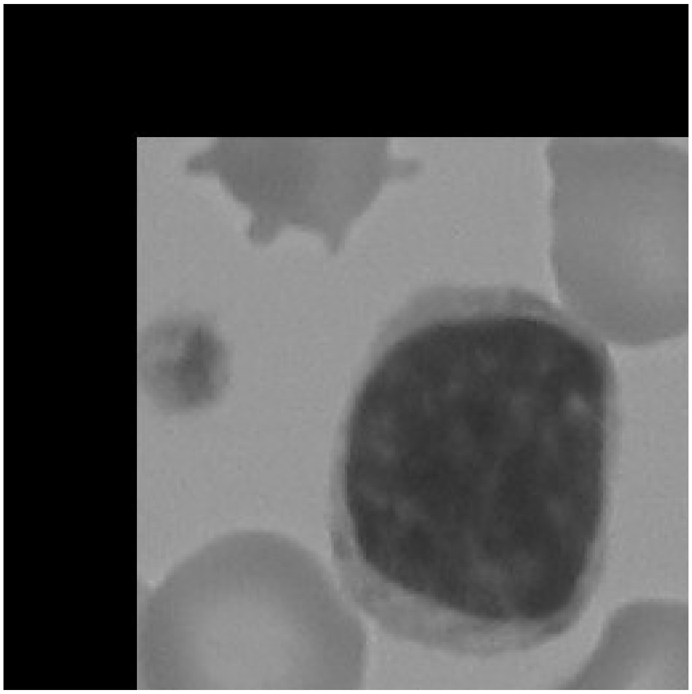
```
'{augmented_image_path}'. Please check the file path and format.")
        else:
            cv2_imshow(augmented_image)
```

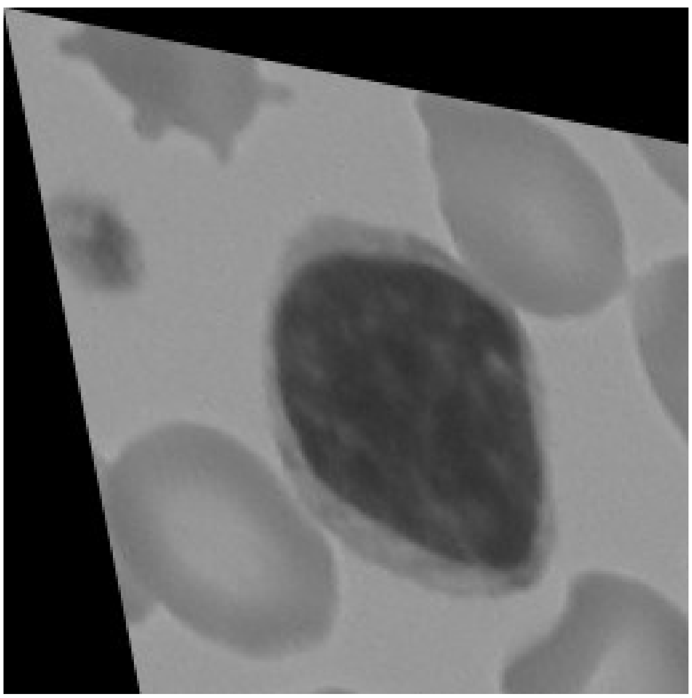












The given model architecture

This piece of code creates a sequential model using the Keras library.

Specifically, the model architecture is as follows:

1. Convolutional Layers:
   - The model includes five convolutional layers with different configurations.

- Each convolutional layer uses a 3x3 filter size and ReLU activation.
- The number of filters is set to 48 for all convolutional layers.
- The padding is set to 'same' for all convolutional layers.
- The input shape of the first convolutional layer is (257, 257, 3).

2. Max Pooling:
   - Some of the convolutional layers are followed by max pooling layers.
   - The max pooling layers use a 2x2 pool size and a stride of 2.
3. Flattening:
   - After the convolutional layers, the tensor output is flattened.
4. Fully Connected Layer:
   - A fully connected layer with 1000 units and ReLU activation is added.
5. Dropout Layer:
   - A dropout layer with a rate of 0.5 is added to prevent overfitting.
6. Output Layer:
   - An output layer with 2 units and softmax activation is added.

```
model = Sequential()

# Layer parameters
FS = (3, 3)  # Filter size
MP_size = (2, 2)  # Max-pooling size
NoF = 48  # Number of filters
P = 'same'  # Padding
activation = 'relu'  # Activation function

# Convolutional layers
layer_configs = [(2, 2, 2), (1, 1, 2), (5, 1, 1), (3, 1, 1), (3, 3, 
1)]
for i, (s, strides, pooling) in enumerate(layer_configs, start=1):
    model.add(Conv2D(NoF, FS, strides=strides, padding=P, 
activation=activation, input_shape=(257, 257, 3)))
    if pooling:
        model.add(MaxPooling2D(pool_size=MP_size, strides=2))

# Flatten the tensor output
model.add(Flatten())

# Add a Fully Connected layer
model.add(Dense(1000, activation='relu'))
# Add a Dropout layer
model.add(Dropout(0.5))

# Add an output layer
model.add(Dense(2, activation='softmax'))

# Model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 129, 129, 48)      1344      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 64, 64, 48)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 64, 64, 48)        20784     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 32, 32, 48)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 32, 32, 48)        20784     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 16, 16, 48)       0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 16, 16, 48)        20784     
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 8, 8, 48)         0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (None, 3, 3, 48)          20784     
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 1, 1, 48)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 48)                0         
                                                                 
 dense (Dense)               (None, 1000)              49000     
                                                                 
 dropout (Dropout)           (None, 1000)              0         
                                                                 
 dense_1 (Dense)             (None, 2)                 2002      
                                                                 
=================================================================
Total params: 135,482
Trainable params: 135,482
Non-trainable params: 0
_________________________________________________________________
```

Training Phase

1. The image size is set to (257, 257).
2. Lists are initialized to store the training and test data and labels.
3. The get_label function is defined to extract labels from filenames.
4. The pixel values of the images are normalized to the range [0, 1].
5. The labels are converted to one-hot encoded format.

6. The model is compiled using a binary cross-entropy loss function, Adam optimizer with a learning rate of 0.0001, and accuracy as the metric.
7. Early stopping is defined with a patience of 20 epochs and restores the best weights.
8. The model is trained using the training data, with a batch size of 100 and a maximum of 500 epochs. The validation data is provided to monitor performance and apply early stopping.
9. The model is evaluated on the test data, and the test loss and accuracy are printed.
10. The model's training history is saved to a file using pickle.

```
train_path = '/content/drive/MyDrive/image processing/train set'
test_path = '/content/drive/MyDrive/image processing/test set'

# The image size
image_size = (257, 257)

# Prepare data
X_train = []  # List to store the training image data
y_train = []  # List to store the corresponding training labels

X_test = []  # List to store the test image data
y_test = []  # List to store the corresponding test labels

# Defining the get_label function according to the data explanation
def get_label(filename):
    label = int(filename.split('_')[-1].split('.')[0].split()[0])
    return label

# Read the images from the train dataset folder and resize them
for filename in os.listdir(train_path):
    filepath = os.path.join(train_path, filename)
    img = cv2.imread(filepath)
    img = cv2.resize(img, image_size)  # Resize the image
    X_train.append(img)
    y_train.append(get_label(filename))

# Read the images from the test dataset folder and resize them
for filename in os.listdir(test_path):
    filepath = os.path.join(test_path, filename)
    img = cv2.imread(filepath)
    img = cv2.resize(img, image_size)  # Resize the image
    X_test.append(img)
    y_test.append(get_label(filename))

# Convert the lists to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
```

```
# Normalize the pixel values to the range [0, 1]
X_train = X_train / 255.0
X_test = X_test / 255.0

# Convert labels to one-hot encoded format
num_classes = 2
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Compile the model
learning_rate = 0.0001
loss_function = BinaryCrossentropy()
optimizer = Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss=loss_function,
metrics=['accuracy'])

# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=20,
restore_best_weights=True)

# Training the model with early stopping
batch_size = 100
epochs = 500
history = model.fit(X_train, y_train, batch_size=batch_size,
epochs=epochs, validation_data=(X_test, y_test),
callbacks=[early_stopping])

# Evaluating on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

with open('/content/drive/MyDrive/image processing/history.pickle',
'wb') as file:
    pickle.dump(history.history, file)
```

```
Epoch 1/500
13/13 [==============================] - 15s 217ms/step - loss: 0.6931
- accuracy: 0.4936 - val_loss: 0.6936 - val_accuracy: 0.4679
Epoch 2/500
13/13 [==============================] - 1s 92ms/step - loss: 0.6925 -
accuracy: 0.5088 - val_loss: 0.6937 - val_accuracy: 0.4679
Epoch 3/500
13/13 [==============================] - 1s 103ms/step - loss: 0.6919
- accuracy: 0.5112 - val_loss: 0.6940 - val_accuracy: 0.4679
Epoch 4/500
13/13 [==============================] - 1s 101ms/step - loss: 0.6915
- accuracy: 0.5080 - val_loss: 0.6942 - val_accuracy: 0.4679
Epoch 5/500
```

```
13/13 [==============================] - 1s 91ms/step - loss: 0.6901 -
accuracy: 0.5088 - val_loss: 0.6935 - val_accuracy: 0.4679
Epoch 6/500
13/13 [==============================] - 1s 92ms/step - loss: 0.6881 -
accuracy: 0.5136 - val_loss: 0.6920 - val_accuracy: 0.4679
Epoch 7/500
13/13 [==============================] - 1s 100ms/step - loss: 0.6841
- accuracy: 0.5136 - val_loss: 0.6887 - val_accuracy: 0.4712
Epoch 8/500
13/13 [==============================] - 2s 132ms/step - loss: 0.6758
- accuracy: 0.5513 - val_loss: 0.6805 - val_accuracy: 0.6218
Epoch 9/500
13/13 [==============================] - 2s 137ms/step - loss: 0.6656
- accuracy: 0.6506 - val_loss: 0.6647 - val_accuracy: 0.6378
Epoch 10/500
13/13 [==============================] - 1s 100ms/step - loss: 0.6456
- accuracy: 0.6675 - val_loss: 0.6566 - val_accuracy: 0.6442
Epoch 11/500
13/13 [==============================] - 1s 104ms/step - loss: 0.6247
- accuracy: 0.6867 - val_loss: 0.6655 - val_accuracy: 0.6410
Epoch 12/500
13/13 [==============================] - 1s 95ms/step - loss: 0.6084 -
accuracy: 0.6899 - val_loss: 0.6326 - val_accuracy: 0.6538
Epoch 13/500
13/13 [==============================] - 1s 92ms/step - loss: 0.5901 -
accuracy: 0.6899 - val_loss: 0.6506 - val_accuracy: 0.6603
Epoch 14/500
13/13 [==============================] - 1s 94ms/step - loss: 0.5753 -
accuracy: 0.7091 - val_loss: 0.6085 - val_accuracy: 0.6731
Epoch 15/500
13/13 [==============================] - 1s 103ms/step - loss: 0.5659
- accuracy: 0.7091 - val_loss: 0.5988 - val_accuracy: 0.6955
Epoch 16/500
13/13 [==============================] - 1s 94ms/step - loss: 0.5501 -
accuracy: 0.7155 - val_loss: 0.5902 - val_accuracy: 0.6955
Epoch 17/500
13/13 [==============================] - 1s 93ms/step - loss: 0.5427 -
accuracy: 0.7324 - val_loss: 0.5842 - val_accuracy: 0.7115
Epoch 18/500
13/13 [==============================] - 2s 120ms/step - loss: 0.5398
- accuracy: 0.7332 - val_loss: 0.5779 - val_accuracy: 0.7019
Epoch 19/500
13/13 [==============================] - 2s 132ms/step - loss: 0.5259
- accuracy: 0.7348 - val_loss: 0.5733 - val_accuracy: 0.7051
Epoch 20/500
13/13 [==============================] - 2s 119ms/step - loss: 0.5148
- accuracy: 0.7428 - val_loss: 0.5670 - val_accuracy: 0.7115
Epoch 21/500
13/13 [==============================] - 1s 94ms/step - loss: 0.5066 -
accuracy: 0.7364 - val_loss: 0.5675 - val_accuracy: 0.7212
```

```
Epoch 22/500
13/13 [==============================] - 1s 103ms/step - loss: 0.5077
- accuracy: 0.7508 - val_loss: 0.5772 - val_accuracy: 0.7276
Epoch 23/500
13/13 [==============================] - 1s 93ms/step - loss: 0.4887 -
accuracy: 0.7492 - val_loss: 0.5454 - val_accuracy: 0.7372
Epoch 24/500
13/13 [==============================] - 1s 102ms/step - loss: 0.4800
- accuracy: 0.7604 - val_loss: 0.5340 - val_accuracy: 0.7244
Epoch 25/500
13/13 [==============================] - 1s 94ms/step - loss: 0.4653 -
accuracy: 0.7812 - val_loss: 0.5311 - val_accuracy: 0.7179
Epoch 26/500
13/13 [==============================] - 1s 93ms/step - loss: 0.4718 -
accuracy: 0.7708 - val_loss: 0.5270 - val_accuracy: 0.7244
Epoch 27/500
13/13 [==============================] - 1s 94ms/step - loss: 0.4440 -
accuracy: 0.7788 - val_loss: 0.5374 - val_accuracy: 0.7532
Epoch 28/500
13/13 [==============================] - 1s 111ms/step - loss: 0.4499
- accuracy: 0.7772 - val_loss: 0.5031 - val_accuracy: 0.7404
Epoch 29/500
13/13 [==============================] - 2s 133ms/step - loss: 0.4456
- accuracy: 0.7877 - val_loss: 0.5429 - val_accuracy: 0.7468
Epoch 30/500
13/13 [==============================] - 2s 135ms/step - loss: 0.4373
- accuracy: 0.7845 - val_loss: 0.5082 - val_accuracy: 0.7628
Epoch 31/500
13/13 [==============================] - 1s 94ms/step - loss: 0.4252 -
accuracy: 0.7949 - val_loss: 0.4915 - val_accuracy: 0.7724
Epoch 32/500
13/13 [==============================] - 1s 106ms/step - loss: 0.4163
- accuracy: 0.7973 - val_loss: 0.4893 - val_accuracy: 0.7756
Epoch 33/500
13/13 [==============================] - 1s 103ms/step - loss: 0.4246
- accuracy: 0.7965 - val_loss: 0.4849 - val_accuracy: 0.7500
Epoch 34/500
13/13 [==============================] - 1s 109ms/step - loss: 0.4022
- accuracy: 0.7957 - val_loss: 0.4781 - val_accuracy: 0.7821
Epoch 35/500
13/13 [==============================] - 2s 129ms/step - loss: 0.3903
- accuracy: 0.8157 - val_loss: 0.4778 - val_accuracy: 0.7917
Epoch 36/500
13/13 [==============================] - 1s 95ms/step - loss: 0.3806 -
accuracy: 0.8141 - val_loss: 0.4533 - val_accuracy: 0.8109
Epoch 37/500
13/13 [==============================] - 1s 95ms/step - loss: 0.3671 -
accuracy: 0.8365 - val_loss: 0.4863 - val_accuracy: 0.7821
Epoch 38/500
13/13 [==============================] - 2s 123ms/step - loss: 0.3632
```

```
- accuracy: 0.8245 - val_loss: 0.4378 - val_accuracy: 0.8173
Epoch 39/500
13/13 [==============================] - 2s 132ms/step - loss: 0.3501
- accuracy: 0.8389 - val_loss: 0.4813 - val_accuracy: 0.7788
Epoch 40/500
13/13 [==============================] - 2s 128ms/step - loss: 0.3511
- accuracy: 0.8405 - val_loss: 0.4300 - val_accuracy: 0.8269
Epoch 41/500
13/13 [==============================] - 1s 94ms/step - loss: 0.3276 -
accuracy: 0.8614 - val_loss: 0.4550 - val_accuracy: 0.7981
Epoch 42/500
13/13 [==============================] - 1s 95ms/step - loss: 0.3419 -
accuracy: 0.8486 - val_loss: 0.4238 - val_accuracy: 0.8269
Epoch 43/500
13/13 [==============================] - 1s 95ms/step - loss: 0.3138 -
accuracy: 0.8558 - val_loss: 0.4150 - val_accuracy: 0.8365
Epoch 44/500
13/13 [==============================] - 1s 95ms/step - loss: 0.3145 -
accuracy: 0.8686 - val_loss: 0.4095 - val_accuracy: 0.8365
Epoch 45/500
13/13 [==============================] - 1s 94ms/step - loss: 0.3072 -
accuracy: 0.8678 - val_loss: 0.4221 - val_accuracy: 0.8141
Epoch 46/500
13/13 [==============================] - 1s 103ms/step - loss: 0.3037
- accuracy: 0.8750 - val_loss: 0.4011 - val_accuracy: 0.8429
Epoch 47/500
13/13 [==============================] - 1s 106ms/step - loss: 0.2810
- accuracy: 0.8862 - val_loss: 0.3943 - val_accuracy: 0.8333
Epoch 48/500
13/13 [==============================] - 1s 112ms/step - loss: 0.2833
- accuracy: 0.8846 - val_loss: 0.4080 - val_accuracy: 0.8397
Epoch 49/500
13/13 [==============================] - 2s 133ms/step - loss: 0.2717
- accuracy: 0.8862 - val_loss: 0.3841 - val_accuracy: 0.8397
Epoch 50/500
13/13 [==============================] - 2s 174ms/step - loss: 0.2799
- accuracy: 0.8830 - val_loss: 0.5061 - val_accuracy: 0.7821
Epoch 51/500
13/13 [==============================] - 2s 124ms/step - loss: 0.2761
- accuracy: 0.8862 - val_loss: 0.3958 - val_accuracy: 0.8173
Epoch 52/500
13/13 [==============================] - 1s 97ms/step - loss: 0.2521 -
accuracy: 0.9006 - val_loss: 0.3691 - val_accuracy: 0.8558
Epoch 53/500
13/13 [==============================] - 1s 94ms/step - loss: 0.2438 -
accuracy: 0.9103 - val_loss: 0.3859 - val_accuracy: 0.8429
Epoch 54/500
13/13 [==============================] - 1s 104ms/step - loss: 0.2506
- accuracy: 0.8942 - val_loss: 0.3801 - val_accuracy: 0.8365
Epoch 55/500
```

```
13/13 [==============================] - 1s 104ms/step - loss: 0.2570
- accuracy: 0.8902 - val_loss: 0.3702 - val_accuracy: 0.8462
Epoch 56/500
13/13 [==============================] - 1s 105ms/step - loss: 0.2642
- accuracy: 0.8886 - val_loss: 0.3669 - val_accuracy: 0.8590
Epoch 57/500
13/13 [==============================] - 1s 93ms/step - loss: 0.2268 -
accuracy: 0.9167 - val_loss: 0.3609 - val_accuracy: 0.8654
Epoch 58/500
13/13 [==============================] - 2s 157ms/step - loss: 0.2325
- accuracy: 0.9095 - val_loss: 0.3885 - val_accuracy: 0.8365
Epoch 59/500
13/13 [==============================] - 2s 134ms/step - loss: 0.2183
- accuracy: 0.9143 - val_loss: 0.3497 - val_accuracy: 0.8654
Epoch 60/500
13/13 [==============================] - 1s 95ms/step - loss: 0.2113 -
accuracy: 0.9191 - val_loss: 0.3464 - val_accuracy: 0.8654
Epoch 61/500
13/13 [==============================] - 1s 94ms/step - loss: 0.2278 -
accuracy: 0.9071 - val_loss: 0.3981 - val_accuracy: 0.8462
Epoch 62/500
13/13 [==============================] - 1s 95ms/step - loss: 0.2457 -
accuracy: 0.8902 - val_loss: 0.4621 - val_accuracy: 0.8173
Epoch 63/500
13/13 [==============================] - 1s 104ms/step - loss: 0.2264
- accuracy: 0.9022 - val_loss: 0.3533 - val_accuracy: 0.8494
Epoch 64/500
13/13 [==============================] - 1s 97ms/step - loss: 0.1988 -
accuracy: 0.9279 - val_loss: 0.3422 - val_accuracy: 0.8494
Epoch 65/500
13/13 [==============================] - 1s 93ms/step - loss: 0.1921 -
accuracy: 0.9311 - val_loss: 0.3313 - val_accuracy: 0.8686
Epoch 66/500
13/13 [==============================] - 1s 103ms/step - loss: 0.1834
- accuracy: 0.9335 - val_loss: 0.3410 - val_accuracy: 0.8686
Epoch 67/500
13/13 [==============================] - 1s 103ms/step - loss: 0.1791
- accuracy: 0.9431 - val_loss: 0.3542 - val_accuracy: 0.8654
Epoch 68/500
13/13 [==============================] - 2s 133ms/step - loss: 0.1747
- accuracy: 0.9447 - val_loss: 0.3320 - val_accuracy: 0.8718
Epoch 69/500
13/13 [==============================] - 2s 130ms/step - loss: 0.1699
- accuracy: 0.9391 - val_loss: 0.3197 - val_accuracy: 0.8686
Epoch 70/500
13/13 [==============================] - 1s 104ms/step - loss: 0.1684
- accuracy: 0.9407 - val_loss: 0.3519 - val_accuracy: 0.8526
Epoch 71/500
13/13 [==============================] - 1s 96ms/step - loss: 0.1683 -
accuracy: 0.9399 - val_loss: 0.3978 - val_accuracy: 0.8494
```

```
Epoch 72/500
13/13 [==============================] - 1s 95ms/step - loss: 0.1675 -
accuracy: 0.9367 - val_loss: 0.3358 - val_accuracy: 0.8654
Epoch 73/500
13/13 [==============================] - 1s 96ms/step - loss: 0.1604 -
accuracy: 0.9455 - val_loss: 0.3408 - val_accuracy: 0.8686
Epoch 74/500
13/13 [==============================] - 1s 104ms/step - loss: 0.1495
- accuracy: 0.9495 - val_loss: 0.3281 - val_accuracy: 0.8750
Epoch 75/500
13/13 [==============================] - 1s 105ms/step - loss: 0.1536
- accuracy: 0.9463 - val_loss: 0.3724 - val_accuracy: 0.8590
Epoch 76/500
13/13 [==============================] - 1s 106ms/step - loss: 0.1529
- accuracy: 0.9511 - val_loss: 0.3297 - val_accuracy: 0.8750
Epoch 77/500
13/13 [==============================] - 1s 97ms/step - loss: 0.1638 -
accuracy: 0.9311 - val_loss: 0.3299 - val_accuracy: 0.8782
Epoch 78/500
13/13 [==============================] - 2s 120ms/step - loss: 0.1470
- accuracy: 0.9479 - val_loss: 0.3513 - val_accuracy: 0.8654
Epoch 79/500
13/13 [==============================] - 2s 134ms/step - loss: 0.1410
- accuracy: 0.9471 - val_loss: 0.3328 - val_accuracy: 0.8750
Epoch 80/500
13/13 [==============================] - 2s 127ms/step - loss: 0.1302
- accuracy: 0.9559 - val_loss: 0.3243 - val_accuracy: 0.8750
Epoch 81/500
13/13 [==============================] - 1s 96ms/step - loss: 0.1336 -
accuracy: 0.9511 - val_loss: 0.3653 - val_accuracy: 0.8718
Epoch 82/500
13/13 [==============================] - 1s 103ms/step - loss: 0.1371
- accuracy: 0.9503 - val_loss: 0.3248 - val_accuracy: 0.8782
Epoch 83/500
13/13 [==============================] - 1s 103ms/step - loss: 0.1284
- accuracy: 0.9559 - val_loss: 0.3180 - val_accuracy: 0.8846
Epoch 84/500
13/13 [==============================] - 1s 93ms/step - loss: 0.1224 -
accuracy: 0.9575 - val_loss: 0.3549 - val_accuracy: 0.8686
Epoch 85/500
13/13 [==============================] - 1s 95ms/step - loss: 0.1320 -
accuracy: 0.9559 - val_loss: 0.3268 - val_accuracy: 0.8974
Epoch 86/500
13/13 [==============================] - 1s 105ms/step - loss: 0.1189
- accuracy: 0.9663 - val_loss: 0.3154 - val_accuracy: 0.8878
Epoch 87/500
13/13 [==============================] - 1s 103ms/step - loss: 0.1249
- accuracy: 0.9575 - val_loss: 0.3408 - val_accuracy: 0.8910
Epoch 88/500
13/13 [==============================] - 1s 118ms/step - loss: 0.1153
```

```
- accuracy: 0.9663 - val_loss: 0.3740 - val_accuracy: 0.8622
Epoch 89/500
13/13 [==============================] - 2s 138ms/step - loss: 0.1122
- accuracy: 0.9599 - val_loss: 0.3372 - val_accuracy: 0.8846
Epoch 90/500
13/13 [==============================] - 2s 122ms/step - loss: 0.1110
- accuracy: 0.9567 - val_loss: 0.3364 - val_accuracy: 0.8814
Epoch 91/500
13/13 [==============================] - 1s 105ms/step - loss: 0.1000
- accuracy: 0.9696 - val_loss: 0.3495 - val_accuracy: 0.8718
Epoch 92/500
13/13 [==============================] - 1s 95ms/step - loss: 0.1022 -
accuracy: 0.9679 - val_loss: 0.3168 - val_accuracy: 0.8910
Epoch 93/500
13/13 [==============================] - 1s 94ms/step - loss: 0.0983 -
accuracy: 0.9696 - val_loss: 0.3126 - val_accuracy: 0.9006
Epoch 94/500
13/13 [==============================] - 1s 103ms/step - loss: 0.0925
- accuracy: 0.9744 - val_loss: 0.3484 - val_accuracy: 0.8750
Epoch 95/500
13/13 [==============================] - 1s 95ms/step - loss: 0.1001 -
accuracy: 0.9679 - val_loss: 0.3149 - val_accuracy: 0.8910
Epoch 96/500
13/13 [==============================] - 1s 104ms/step - loss: 0.0943
- accuracy: 0.9647 - val_loss: 0.3287 - val_accuracy: 0.8846
Epoch 97/500
13/13 [==============================] - 2s 133ms/step - loss: 0.0974
- accuracy: 0.9671 - val_loss: 0.3251 - val_accuracy: 0.8878
Epoch 98/500
13/13 [==============================] - 2s 171ms/step - loss: 0.0875
- accuracy: 0.9736 - val_loss: 0.3304 - val_accuracy: 0.9103
Epoch 99/500
13/13 [==============================] - 2s 133ms/step - loss: 0.0874
- accuracy: 0.9728 - val_loss: 0.3325 - val_accuracy: 0.8910
Epoch 100/500
13/13 [==============================] - 2s 126ms/step - loss: 0.0833
- accuracy: 0.9728 - val_loss: 0.3432 - val_accuracy: 0.8878
Epoch 101/500
13/13 [==============================] - 1s 104ms/step - loss: 0.1023
- accuracy: 0.9647 - val_loss: 0.3194 - val_accuracy: 0.9006
Epoch 102/500
13/13 [==============================] - 1s 105ms/step - loss: 0.1129
- accuracy: 0.9599 - val_loss: 0.3278 - val_accuracy: 0.9038
Epoch 103/500
13/13 [==============================] - 1s 93ms/step - loss: 0.0782 -
accuracy: 0.9760 - val_loss: 0.3310 - val_accuracy: 0.8910
Epoch 104/500
13/13 [==============================] - 1s 103ms/step - loss: 0.0753
- accuracy: 0.9784 - val_loss: 0.3289 - val_accuracy: 0.8974
Epoch 105/500
```

```
13/13 [==============================] - 1s 95ms/step - loss: 0.0687 -
accuracy: 0.9816 - val_loss: 0.3262 - val_accuracy: 0.9038
Epoch 106/500
13/13 [==============================] - 1s 95ms/step - loss: 0.0750 -
accuracy: 0.9752 - val_loss: 0.4269 - val_accuracy: 0.8590
Epoch 107/500
13/13 [==============================] - 1s 96ms/step - loss: 0.0865 -
accuracy: 0.9671 - val_loss: 0.3409 - val_accuracy: 0.9006
Epoch 108/500
13/13 [==============================] - 1s 105ms/step - loss: 0.0764
- accuracy: 0.9728 - val_loss: 0.3191 - val_accuracy: 0.8974
Epoch 109/500
13/13 [==============================] - 2s 135ms/step - loss: 0.0773
- accuracy: 0.9784 - val_loss: 0.3517 - val_accuracy: 0.8846
Epoch 110/500
13/13 [==============================] - 2s 135ms/step - loss: 0.0704
- accuracy: 0.9808 - val_loss: 0.3765 - val_accuracy: 0.8622
Epoch 111/500
13/13 [==============================] - 1s 105ms/step - loss: 0.0666
- accuracy: 0.9816 - val_loss: 0.3334 - val_accuracy: 0.9038
Epoch 112/500
13/13 [==============================] - 1s 96ms/step - loss: 0.0651 -
accuracy: 0.9808 - val_loss: 0.3629 - val_accuracy: 0.8814
Epoch 113/500
13/13 [==============================] - 1s 95ms/step - loss: 0.0602 -
accuracy: 0.9832 - val_loss: 0.3621 - val_accuracy: 0.8846
10/10 [==============================] - 1s 39ms/step - loss: 0.3126 -
accuracy: 0.9006
Test Loss: 0.31264758110046387
Test Accuracy: 0.9006410241127014
```

Plot of the loss and accuracy of the given model

```
# Load the history object from the file
with open('/content/drive/MyDrive/image processing/history.pickle',
'rb') as file:
    history = pickle.load(file)

# Plot of model's loss and accuracy
plt.figure(figsize=(12, 4))

# Plot of training and validation loss
plt.subplot(1, 2, 1)
plt.plot(history['loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot of training and validation accuracy
```

```
plt.subplot(1, 2, 2)
plt.plot(history['accuracy'], label='Training Accuracy')
plt.plot(history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

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

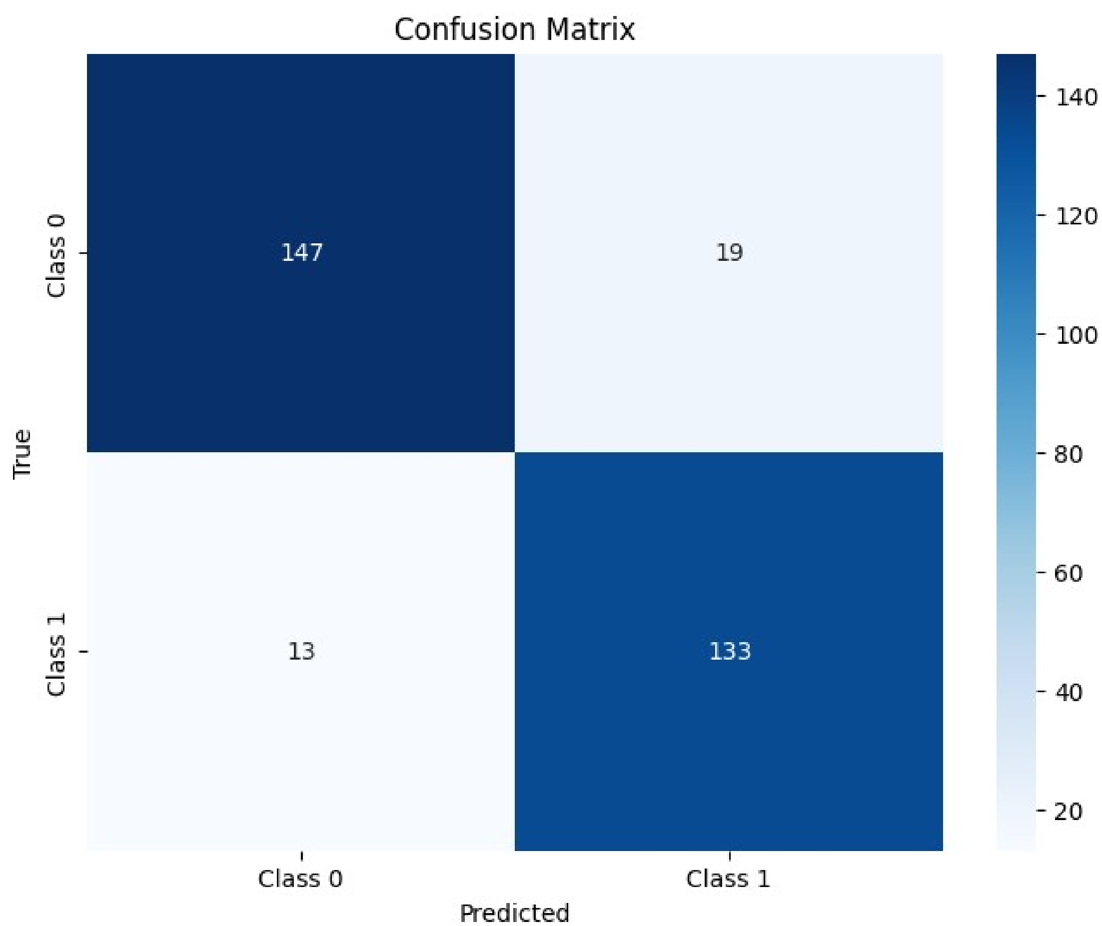
Confusion matrix

```
# Predictions on the test data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Confusion matrix
cm = confusion_matrix(y_test_classes, y_pred_classes)

# Performance metrics
loss, accuracy = model.evaluate(X_test, y_test)
precision = cm[1, 1] / (cm[1, 1] + cm[0, 1])
recall = cm[1, 1] / (cm[1, 1] + cm[1, 0])
specificity = cm[0, 0] / (cm[0, 0] + cm[0, 1])

print("Confusion Matrix:")
print(cm)
print("\nLoss:", loss)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("Specificity:", specificity)
```

```
10/10 [==============================] - 0s 26ms/step
10/10 [==============================] - 0s 22ms/step - loss: 0.3174 -
accuracy: 0.8974
Confusion Matrix:
```

```
[[147  19]
 [ 13 133]]

Loss: 0.31736159324645996
Accuracy: 0.8974359035491943
Precision: 0.875
Recall: 0.910958904109589
Specificity: 0.8855421686746988
```

```
labels = ['Class 0', 'Class 1']

# Plot of confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



Based on the confusion matrix and performance metrics, the accuracy of 0.8974 indicates that the model achieved a high overall correct prediction rate on the test dataset. The precision of 0.875 suggests that when the model predicts an instance as positive, there is an 87.5% chance that it is actually a true positive. This indicates a good ability of the model to minimize false positives. The recall of 0.911 implies that the model identified approximately 91.10% of the actual positive instances correctly. The specificity of 0.8855 indicates that the model correctly identified around 88.55% of the negative instances.

Overall, the model demonstrates strong performance, with high accuracy and balanced precision and recall values. It is effective in correctly classifying both positive and negative instances. The loss value also suggests a good fit of the model to the test data.

Changing the network structure

I added another fully connected layer (Dense(500, activation='relu')) after the first fully connected layer. I also added a dropout layer (Dropout(0.5)) after the second fully connected layer. These changes increase the depth of the model and provide more capacity to learn complex patterns in the data.

```
model = Sequential()

# Common layer parameters
FS = (3, 3)  # Filter size
MP_size = (2, 2)  # Max-pooling size
NoF = 48  # Number of filters
P = 'same'  # Padding
activation = 'relu'  # Activation function

# Convolutional layers
layer_configs = [(2, 2, 2), (1, 1, 2), (5, 1, 1), (3, 1, 1), (3, 3, 
1)]
for i, (s, strides, pooling) in enumerate(layer_configs, start=1):
    model.add(Conv2D(NoF, FS, strides=strides, padding=P, 
activation=activation, input_shape=(257, 257, 3)))
    if pooling:
        model.add(MaxPooling2D(pool_size=MP_size, strides=2))

# Flatten the tensor output
model.add(Flatten())

# Add a Fully Connected layer
model.add(Dense(1000, activation='relu'))
# Add a Dropout layer
model.add(Dropout(0.5))

# Add another Fully Connected layer
model.add(Dense(500, activation='relu'))
# Add another Dropout layer
model.add(Dropout(0.5))

# Add an output layer
```

```
model.add(Dense(2, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 129, 129, 48)      1344

 max_pooling2d (MaxPooling2D  (None, 64, 64, 48)       0
 )

 conv2d_1 (Conv2D)           (None, 64, 64, 48)        20784

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 48)       0
 2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 48)        20784

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 48)       0
 2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 48)        20784

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 48)         0
 2D)

 conv2d_4 (Conv2D)           (None, 3, 3, 48)          20784

 max_pooling2d_4 (MaxPooling  (None, 1, 1, 48)         0
 2D)

 flatten (Flatten)           (None, 48)                0

 dense (Dense)               (None, 1000)              49000

 dropout (Dropout)           (None, 1000)              0

 dense_1 (Dense)             (None, 500)               500500

 dropout_1 (Dropout)         (None, 500)               0

 dense_2 (Dense)             (None, 2)                 1002

=================================================================
Total params: 634,982
Trainable params: 634,982
Non-trainable params: 0
_________________________________________________________________
```

I only altered the batch size and set it to 200.

```
# Compile the model
learning_rate = 0.0001
loss_function = BinaryCrossentropy()
optimizer = Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss=loss_function,
metrics=['accuracy'])

# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=20,
restore_best_weights=True)

# Train the model with early stopping
batch_size = 200
epochs = 500
history = model.fit(X_train, y_train, batch_size=batch_size,
epochs=epochs, validation_data=(X_test, y_test),
callbacks=[early_stopping])

test_loss, test_accuracy = model.evaluate(X_test, y_test)

# Test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

with open('/content/drive/MyDrive/image
processing/history_modified.pickle', 'wb') as file:
    pickle.dump(history.history, file)
```

```
Epoch 1/500
7/7 [==============================] - 19s 517ms/step - loss: 0.6933 -
accuracy: 0.4848 - val_loss: 0.6938 - val_accuracy: 0.4679
Epoch 2/500
7/7 [==============================] - 1s 173ms/step - loss: 0.6918 -
accuracy: 0.5032 - val_loss: 0.6937 - val_accuracy: 0.4679
Epoch 3/500
7/7 [==============================] - 1s 178ms/step - loss: 0.6927 -
accuracy: 0.5096 - val_loss: 0.6941 - val_accuracy: 0.4679
Epoch 4/500
7/7 [==============================] - 1s 186ms/step - loss: 0.6915 -
accuracy: 0.5136 - val_loss: 0.6940 - val_accuracy: 0.4679
Epoch 5/500
7/7 [==============================] - 1s 175ms/step - loss: 0.6906 -
accuracy: 0.5072 - val_loss: 0.6951 - val_accuracy: 0.4679
Epoch 6/500
7/7 [==============================] - 2s 222ms/step - loss: 0.6907 -
accuracy: 0.5080 - val_loss: 0.6966 - val_accuracy: 0.4679
Epoch 7/500
7/7 [==============================] - 2s 231ms/step - loss: 0.6904 -
```

```
accuracy: 0.5080 - val_loss: 0.6945 - val_accuracy: 0.4679
Epoch 8/500
7/7 [==============================] - 1s 214ms/step - loss: 0.6896 -
accuracy: 0.5080 - val_loss: 0.6926 - val_accuracy: 0.4679
Epoch 9/500
7/7 [==============================] - 1s 185ms/step - loss: 0.6881 -
accuracy: 0.5152 - val_loss: 0.6912 - val_accuracy: 0.4679
Epoch 10/500
7/7 [==============================] - 1s 174ms/step - loss: 0.6864 -
accuracy: 0.5393 - val_loss: 0.6894 - val_accuracy: 0.5000
Epoch 11/500
7/7 [==============================] - 1s 177ms/step - loss: 0.6842 -
accuracy: 0.5553 - val_loss: 0.6879 - val_accuracy: 0.5353
Epoch 12/500
7/7 [==============================] - 1s 188ms/step - loss: 0.6809 -
accuracy: 0.5849 - val_loss: 0.6839 - val_accuracy: 0.6026
Epoch 13/500
7/7 [==============================] - 1s 188ms/step - loss: 0.6752 -
accuracy: 0.6106 - val_loss: 0.6784 - val_accuracy: 0.6090
Epoch 14/500
7/7 [==============================] - 1s 185ms/step - loss: 0.6678 -
accuracy: 0.6266 - val_loss: 0.6682 - val_accuracy: 0.6442
Epoch 15/500
7/7 [==============================] - 1s 188ms/step - loss: 0.6554 -
accuracy: 0.6707 - val_loss: 0.6626 - val_accuracy: 0.6603
Epoch 16/500
7/7 [==============================] - 1s 194ms/step - loss: 0.6372 -
accuracy: 0.6891 - val_loss: 0.6446 - val_accuracy: 0.6538
Epoch 17/500
7/7 [==============================] - 2s 302ms/step - loss: 0.6243 -
accuracy: 0.6651 - val_loss: 0.6830 - val_accuracy: 0.6538
Epoch 18/500
7/7 [==============================] - 1s 219ms/step - loss: 0.6231 -
accuracy: 0.6707 - val_loss: 0.6215 - val_accuracy: 0.6603
Epoch 19/500
7/7 [==============================] - 1s 178ms/step - loss: 0.5973 -
accuracy: 0.6931 - val_loss: 0.6193 - val_accuracy: 0.6731
Epoch 20/500
7/7 [==============================] - 1s 178ms/step - loss: 0.5827 -
accuracy: 0.6987 - val_loss: 0.6112 - val_accuracy: 0.6763
Epoch 21/500
7/7 [==============================] - 1s 176ms/step - loss: 0.5654 -
accuracy: 0.7260 - val_loss: 0.5978 - val_accuracy: 0.6699
Epoch 22/500
7/7 [==============================] - 1s 187ms/step - loss: 0.5625 -
accuracy: 0.7147 - val_loss: 0.5959 - val_accuracy: 0.6955
Epoch 23/500
7/7 [==============================] - 1s 186ms/step - loss: 0.5461 -
accuracy: 0.7300 - val_loss: 0.6075 - val_accuracy: 0.7083
Epoch 24/500
```

```
7/7 [==============================] - 1s 177ms/step - loss: 0.5408 -
accuracy: 0.7308 - val_loss: 0.5704 - val_accuracy: 0.6859
Epoch 25/500
7/7 [==============================] - 1s 185ms/step - loss: 0.5299 -
accuracy: 0.7244 - val_loss: 0.5872 - val_accuracy: 0.7083
Epoch 26/500
7/7 [==============================] - 1s 181ms/step - loss: 0.5291 -
accuracy: 0.7436 - val_loss: 0.5513 - val_accuracy: 0.7212
Epoch 27/500
7/7 [==============================] - 2s 226ms/step - loss: 0.5080 -
accuracy: 0.7572 - val_loss: 0.5512 - val_accuracy: 0.7115
Epoch 28/500
7/7 [==============================] - 2s 291ms/step - loss: 0.5146 -
accuracy: 0.7564 - val_loss: 0.5418 - val_accuracy: 0.6987
Epoch 29/500
7/7 [==============================] - 1s 177ms/step - loss: 0.5146 -
accuracy: 0.7484 - val_loss: 0.5314 - val_accuracy: 0.7244
Epoch 30/500
7/7 [==============================] - 1s 188ms/step - loss: 0.4967 -
accuracy: 0.7604 - val_loss: 0.5576 - val_accuracy: 0.7179
Epoch 31/500
7/7 [==============================] - 1s 179ms/step - loss: 0.4875 -
accuracy: 0.7668 - val_loss: 0.5550 - val_accuracy: 0.7340
Epoch 32/500
7/7 [==============================] - 1s 179ms/step - loss: 0.4912 -
accuracy: 0.7636 - val_loss: 0.5553 - val_accuracy: 0.7244
Epoch 33/500
7/7 [==============================] - 2s 211ms/step - loss: 0.4902 -
accuracy: 0.7708 - val_loss: 0.5436 - val_accuracy: 0.6795
Epoch 34/500
7/7 [==============================] - 1s 189ms/step - loss: 0.4936 -
accuracy: 0.7500 - val_loss: 0.5292 - val_accuracy: 0.7212
Epoch 35/500
7/7 [==============================] - 1s 188ms/step - loss: 0.4638 -
accuracy: 0.7700 - val_loss: 0.5398 - val_accuracy: 0.7276
Epoch 36/500
7/7 [==============================] - 2s 251ms/step - loss: 0.4603 -
accuracy: 0.7821 - val_loss: 0.5045 - val_accuracy: 0.7468
Epoch 37/500
7/7 [==============================] - 2s 288ms/step - loss: 0.4560 -
accuracy: 0.7933 - val_loss: 0.5496 - val_accuracy: 0.7276
Epoch 38/500
7/7 [==============================] - 1s 210ms/step - loss: 0.4489 -
accuracy: 0.7893 - val_loss: 0.4998 - val_accuracy: 0.7564
Epoch 39/500
7/7 [==============================] - 1s 175ms/step - loss: 0.4335 -
accuracy: 0.8053 - val_loss: 0.4929 - val_accuracy: 0.7628
Epoch 40/500
7/7 [==============================] - 1s 189ms/step - loss: 0.4341 -
accuracy: 0.8013 - val_loss: 0.4903 - val_accuracy: 0.7724
```

```
Epoch 41/500
7/7 [==============================] - 1s 187ms/step - loss: 0.4203 -
accuracy: 0.8005 - val_loss: 0.4787 - val_accuracy: 0.7628
Epoch 42/500
7/7 [==============================] - 1s 171ms/step - loss: 0.4147 -
accuracy: 0.8045 - val_loss: 0.4838 - val_accuracy: 0.7756
Epoch 43/500
7/7 [==============================] - 1s 176ms/step - loss: 0.3974 -
accuracy: 0.8117 - val_loss: 0.4751 - val_accuracy: 0.7756
Epoch 44/500
7/7 [==============================] - 1s 177ms/step - loss: 0.4186 -
accuracy: 0.8061 - val_loss: 0.4960 - val_accuracy: 0.7564
Epoch 45/500
7/7 [==============================] - 1s 178ms/step - loss: 0.3895 -
accuracy: 0.8285 - val_loss: 0.5059 - val_accuracy: 0.7628
Epoch 46/500
7/7 [==============================] - 2s 244ms/step - loss: 0.3956 -
accuracy: 0.8237 - val_loss: 0.4658 - val_accuracy: 0.7756
Epoch 47/500
7/7 [==============================] - 2s 243ms/step - loss: 0.3862 -
accuracy: 0.8269 - val_loss: 0.4921 - val_accuracy: 0.7564
Epoch 48/500
7/7 [==============================] - 1s 219ms/step - loss: 0.3714 -
accuracy: 0.8269 - val_loss: 0.4573 - val_accuracy: 0.7788
Epoch 49/500
7/7 [==============================] - 1s 187ms/step - loss: 0.3707 -
accuracy: 0.8309 - val_loss: 0.4559 - val_accuracy: 0.7853
Epoch 50/500
7/7 [==============================] - 1s 190ms/step - loss: 0.3608 -
accuracy: 0.8357 - val_loss: 0.4411 - val_accuracy: 0.7853
Epoch 51/500
7/7 [==============================] - 1s 189ms/step - loss: 0.3461 -
accuracy: 0.8478 - val_loss: 0.4443 - val_accuracy: 0.7949
Epoch 52/500
7/7 [==============================] - 1s 188ms/step - loss: 0.3408 -
accuracy: 0.8526 - val_loss: 0.4405 - val_accuracy: 0.7981
Epoch 53/500
7/7 [==============================] - 1s 188ms/step - loss: 0.3436 -
accuracy: 0.8510 - val_loss: 0.4498 - val_accuracy: 0.7821
Epoch 54/500
7/7 [==============================] - 1s 190ms/step - loss: 0.3340 -
accuracy: 0.8574 - val_loss: 0.4423 - val_accuracy: 0.7821
Epoch 55/500
7/7 [==============================] - 1s 178ms/step - loss: 0.3199 -
accuracy: 0.8638 - val_loss: 0.4333 - val_accuracy: 0.8109
Epoch 56/500
7/7 [==============================] - 1s 202ms/step - loss: 0.3170 -
accuracy: 0.8694 - val_loss: 0.4468 - val_accuracy: 0.7853
Epoch 57/500
7/7 [==============================] - 2s 253ms/step - loss: 0.3266 -
```

```
accuracy: 0.8550 - val_loss: 0.4368 - val_accuracy: 0.7949
Epoch 58/500
7/7 [==============================] - 2s 230ms/step - loss: 0.3181 -
accuracy: 0.8702 - val_loss: 0.4225 - val_accuracy: 0.7885
Epoch 59/500
7/7 [==============================] - 1s 175ms/step - loss: 0.2865 -
accuracy: 0.8958 - val_loss: 0.4945 - val_accuracy: 0.7853
Epoch 60/500
7/7 [==============================] - 1s 177ms/step - loss: 0.3066 -
accuracy: 0.8686 - val_loss: 0.4260 - val_accuracy: 0.8045
Epoch 61/500
7/7 [==============================] - 1s 178ms/step - loss: 0.2761 -
accuracy: 0.8934 - val_loss: 0.4173 - val_accuracy: 0.8237
Epoch 62/500
7/7 [==============================] - 1s 176ms/step - loss: 0.2673 -
accuracy: 0.8990 - val_loss: 0.4176 - val_accuracy: 0.8269
Epoch 63/500
7/7 [==============================] - 1s 190ms/step - loss: 0.2787 -
accuracy: 0.8830 - val_loss: 0.4169 - val_accuracy: 0.8109
Epoch 64/500
7/7 [==============================] - 1s 186ms/step - loss: 0.2535 -
accuracy: 0.9119 - val_loss: 0.4934 - val_accuracy: 0.7788
Epoch 65/500
7/7 [==============================] - 1s 189ms/step - loss: 0.2669 -
accuracy: 0.9022 - val_loss: 0.4038 - val_accuracy: 0.8269
Epoch 66/500
7/7 [==============================] - 1s 181ms/step - loss: 0.2446 -
accuracy: 0.9062 - val_loss: 0.3982 - val_accuracy: 0.8237
Epoch 67/500
7/7 [==============================] - 2s 285ms/step - loss: 0.2493 -
accuracy: 0.9087 - val_loss: 0.4083 - val_accuracy: 0.8333
Epoch 68/500
7/7 [==============================] - 2s 289ms/step - loss: 0.2293 -
accuracy: 0.9183 - val_loss: 0.4199 - val_accuracy: 0.8109
Epoch 69/500
7/7 [==============================] - 1s 177ms/step - loss: 0.2397 -
accuracy: 0.9087 - val_loss: 0.4133 - val_accuracy: 0.8462
Epoch 70/500
7/7 [==============================] - 1s 188ms/step - loss: 0.2278 -
accuracy: 0.9135 - val_loss: 0.4026 - val_accuracy: 0.8397
Epoch 71/500
7/7 [==============================] - 1s 190ms/step - loss: 0.2226 -
accuracy: 0.9183 - val_loss: 0.3965 - val_accuracy: 0.8462
Epoch 72/500
7/7 [==============================] - 1s 190ms/step - loss: 0.2216 -
accuracy: 0.9223 - val_loss: 0.3945 - val_accuracy: 0.8397
Epoch 73/500
7/7 [==============================] - 1s 177ms/step - loss: 0.2124 -
accuracy: 0.9199 - val_loss: 0.4137 - val_accuracy: 0.8301
Epoch 74/500
```

```
7/7 [==============================] - 1s 189ms/step - loss: 0.2024 -
accuracy: 0.9207 - val_loss: 0.4017 - val_accuracy: 0.8494
Epoch 75/500
7/7 [==============================] - 1s 188ms/step - loss: 0.1964 -
accuracy: 0.9319 - val_loss: 0.3948 - val_accuracy: 0.8526
Epoch 76/500
7/7 [==============================] - 1s 200ms/step - loss: 0.1898 -
accuracy: 0.9375 - val_loss: 0.3963 - val_accuracy: 0.8558
Epoch 77/500
7/7 [==============================] - 2s 285ms/step - loss: 0.1891 -
accuracy: 0.9335 - val_loss: 0.4197 - val_accuracy: 0.8301
Epoch 78/500
7/7 [==============================] - 2s 315ms/step - loss: 0.1968 -
accuracy: 0.9191 - val_loss: 0.3948 - val_accuracy: 0.8462
Epoch 79/500
7/7 [==============================] - 2s 311ms/step - loss: 0.1884 -
accuracy: 0.9367 - val_loss: 0.4015 - val_accuracy: 0.8462
Epoch 80/500
7/7 [==============================] - 1s 180ms/step - loss: 0.1979 -
accuracy: 0.9287 - val_loss: 0.4613 - val_accuracy: 0.8365
Epoch 81/500
7/7 [==============================] - 1s 175ms/step - loss: 0.1927 -
accuracy: 0.9287 - val_loss: 0.5364 - val_accuracy: 0.8237
Epoch 82/500
7/7 [==============================] - 1s 174ms/step - loss: 0.2190 -
accuracy: 0.9111 - val_loss: 0.4911 - val_accuracy: 0.8301
Epoch 83/500
7/7 [==============================] - 1s 185ms/step - loss: 0.1945 -
accuracy: 0.9271 - val_loss: 0.4796 - val_accuracy: 0.8397
Epoch 84/500
7/7 [==============================] - 1s 177ms/step - loss: 0.1798 -
accuracy: 0.9359 - val_loss: 0.4058 - val_accuracy: 0.8397
Epoch 85/500
7/7 [==============================] - 1s 185ms/step - loss: 0.1728 -
accuracy: 0.9367 - val_loss: 0.4094 - val_accuracy: 0.8590
Epoch 86/500
7/7 [==============================] - 2s 235ms/step - loss: 0.1695 -
accuracy: 0.9407 - val_loss: 0.4080 - val_accuracy: 0.8494
Epoch 87/500
7/7 [==============================] - 2s 261ms/step - loss: 0.1540 -
accuracy: 0.9447 - val_loss: 0.4291 - val_accuracy: 0.8397
Epoch 88/500
7/7 [==============================] - 2s 278ms/step - loss: 0.1634 -
accuracy: 0.9479 - val_loss: 0.4216 - val_accuracy: 0.8397
Epoch 89/500
7/7 [==============================] - 1s 218ms/step - loss: 0.1426 -
accuracy: 0.9567 - val_loss: 0.4034 - val_accuracy: 0.8462
Epoch 90/500
7/7 [==============================] - 1s 178ms/step - loss: 0.1371 -
accuracy: 0.9527 - val_loss: 0.3975 - val_accuracy: 0.8558
```

```
Epoch 91/500
7/7 [==============================] - 1s 189ms/step - loss: 0.1362 -
accuracy: 0.9535 - val_loss: 0.3976 - val_accuracy: 0.8590
Epoch 92/500
7/7 [==============================] - 1s 183ms/step - loss: 0.1335 -
accuracy: 0.9527 - val_loss: 0.3992 - val_accuracy: 0.8590
10/10 [==============================] - 1s 40ms/step - loss: 0.3945 -
accuracy: 0.8397
Test Loss: 0.3945079743862152
Test Accuracy: 0.8397436141967773
```

As you see the accuracy on the train set is lower than before but the accuracy on the test set dropped more significantly meaning that adding more depth to our model has caused over fitting instead of improvement.

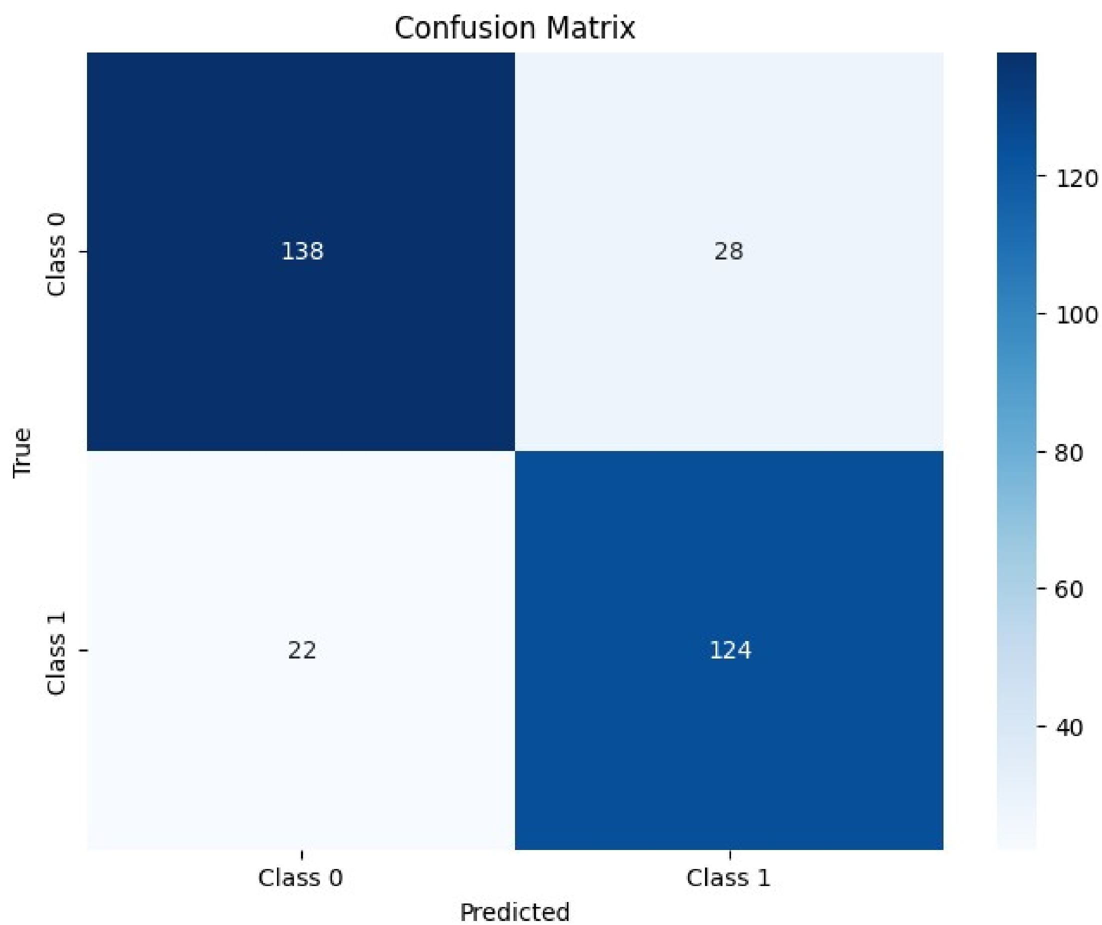
```
with open('/content/drive/MyDrive/image
processing/history_modified.pickle', 'rb') as file:
    history = pickle.load(file)

# Plot the model's loss and accuracy
plt.figure(figsize=(12, 4))

# Plot the training and validation loss
plt.subplot(1, 2, 1)
plt.plot(history['loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot the training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(history['accuracy'], label='Training Accuracy')
plt.plot(history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

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

The occurrence of overfitting is observed in these plots, as the validation loss ceases to decrease while the train loss continues to decrease steadily.

```
# Predictions on the test data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Confusion matrix
cm = confusion_matrix(y_test_classes, y_pred_classes)

# Performance metrics
loss, accuracy = model.evaluate(X_test, y_test)
precision = cm[1, 1] / (cm[1, 1] + cm[0, 1])
recall = cm[1, 1] / (cm[1, 1] + cm[1, 0])
specificity = cm[0, 0] / (cm[0, 0] + cm[0, 1])

print("Confusion Matrix:")
print(cm)
print("\nLoss:", loss)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("Specificity:", specificity)
```

```
10/10 [==============================] - 0s 17ms/step
10/10 [==============================] - 0s 22ms/step - loss: 0.3945 -
accuracy: 0.8397
Confusion Matrix:
[[138  28]
 [ 22 124]]

Loss: 0.3945079743862152
Accuracy: 0.8397436141967773
Precision: 0.8157894736842105
Recall: 0.8493150684931506
Specificity: 0.8313253012048193
```

```python
# Define the labels for the confusion matrix
labels = ['Class 0', 'Class 1']

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels,
yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



Compared to the confusion matrix of the given model:

1. Accuracy: The second confusion matrix has a lower accuracy of 0.8397 compared to the first matrix's accuracy of 0.8974.

2. Precision: The precision in the second matrix is 0.8158, while the precision in the first matrix is 0.875. The given model achieved a higher precision, indicating that it had a better ability to minimize false positive predictions.

3. Recall: The recall in the second matrix is 0.8493, while the recall in the first matrix is 0.911. The first model had a higher recall, indicating that it correctly identified a greater proportion of the actual positive instances.

4. Specificity: The specificity in the second matrix is 0.8313, while the specificity in the first matrix is 0.8855.

Overall, the given model, represented by the first confusion matrix, performed better than the second model. It achieved higher accuracy, precision, recall and specificity. This indicates that the first model had a beeter performance in correctly classifying instances and avoiding false predictions compared to the second model.