Phase 5 Project Submission

Student Name: Tai Ngoc Bui

Student Pace: Flex

Scheduled Project Review: December 7th, 2024

Instructor Name: Mark Barbour

1. Business Understanding

This project focuses on developing a robust Convolutional Neural Network (CNN) capable of classifying skin lesions as malignant or benign. Skin cancer is one of the most prevalent cancers worldwide, and early detection significantly improves treatment outcomes and survival rates. Despite advancements in medical technology, many cases are either detected too late or misdiagnosed due to the limitations of traditional diagnostic methods. By leveraging deep learning techniques, I aim to provide a reliable, efficient, and scalable solution to assist dermatologists and healthcare professionals in diagnosing skin cancer.

This topic is personally significant as it combines two areas of great interest: leveraging artificial intelligence to solve real-world problems and contributing to public health initiatives. On a broader level, this research has societal implications, potentially saving lives and reducing healthcare costs. The target audience for this project includes not only potential patients of skin cancer but also dermatologists and oncologists seeking diagnostic support tools, as well as the whole healthcare system. To successfully complete this project, I relied on numerous studies and projects highlighting the performance of CNNs like ResNet, InceptionNet and MobileNet for skin cancer detection. Moreover, studies on transfer learning to improve model performance with limited datasets are also utilized in this project.

This model will use recall rate as the main metric to prioritize the ability of the model to correctly identify all positive cases (e.g. malignant cancer). In medical diagnoses, a false negative (failing to detect cancer when it is present) can have severe consequences, such as delayed treatment or worsened prognosis. Recall ensures tha the model minimizes false negatives, even if it occasionally produces false positives. Missing a malignant case is riskier than flagging a benign case as malignant, as false positives can often be corrected through follow-up procedures.

2. Data Understanding

The dataset used in this project is sourced from Kaggle's "Skin Cancer: Malignant vs. Benign" (https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign (https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign)). It consists of two folders: train and test with images (224x244) of the two types of moles including benign skin moles and malignant skin moles. It is curated for machine learning research and adheres to ethical standards for data use. The primary features include images of skin lesions data and categorical labels indicating whether the lesion is benign or malignant. Several researchers have used this dataset to train and evaluate machine learning models for skin cancer detection.

3. Exploratory Analysis

a. Load Data and Libraries

In [ ]:
```
# Run this code if data is not already available from your local machine
!kaggle datasets download -d fanconic/skin-cancer-malignant-vs-benign
```

```
Dataset URL: https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign (https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign)
License(s): unknown
skin-cancer-malignant-vs-benign.zip: Skipping, found more recently modified local copy (use --force to force download)
```

In [ ]:
```
# Unzipping downloaded data
! unzip skin-cancer-malignant-vs-benign.zip
```

In [8]:
```
!wget https://raw.githubusercontent.com/taingocbui/phase5_project/main/models/hybrid_xgboost_model.pkl
```

```
--2025-01-11 23:50:08--  https://raw.githubusercontent.com/taingocbui/phase5_project/main/models/hybrid_xgboost_model.pkl (https://raw.githubusercontent.com/taingocbui/phase5_project/main/models/hybrid_xgboost_model.pkl)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 196062 (191K) [application/octet-stream]
Saving to: ‘models/hybrid_xgboost_model.pkl’

hybrid_xgboost_mode 100%[===================>] 191.47K  --.-KB/s    in 0.1s

2025-01-11 23:50:09 (1.31 MB/s) - ‘models/hybrid_xgboost_model.pkl’ saved [196062/196062]
```

In [42]:
```python
# Load all libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, GlobalAveragePooling2D
from keras.optimizers import Adam
from keras.metrics import AUC
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc, accuracy_score
from sklearn.model_selection import train_test_split
import xgboost as xgb
import joblib
import pickle
%matplotlib inline
```

b. Exploratory Analsysis

I will create a function to plot a given number of images from each of the class. This function should require a path to image folder, the title for the plot, and the number of images users want to display.

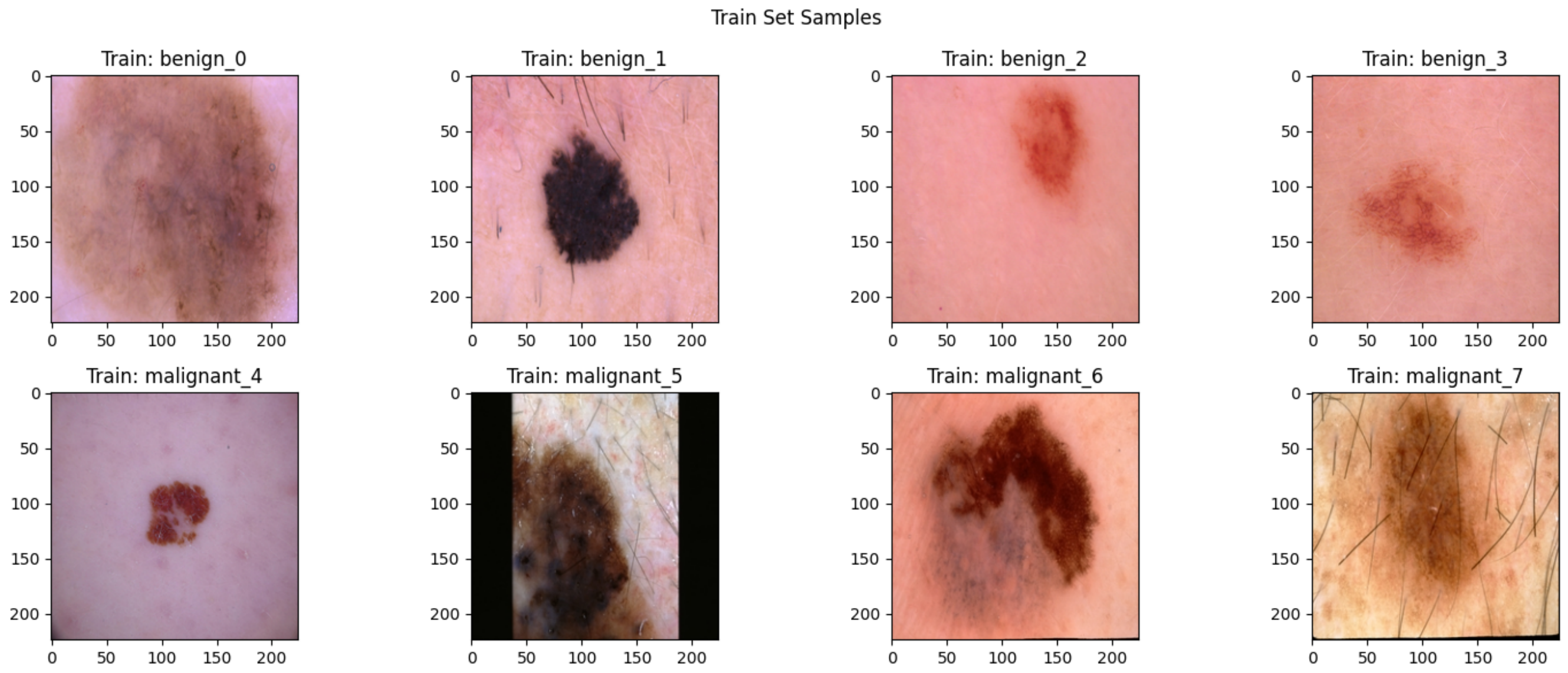
In [10]:

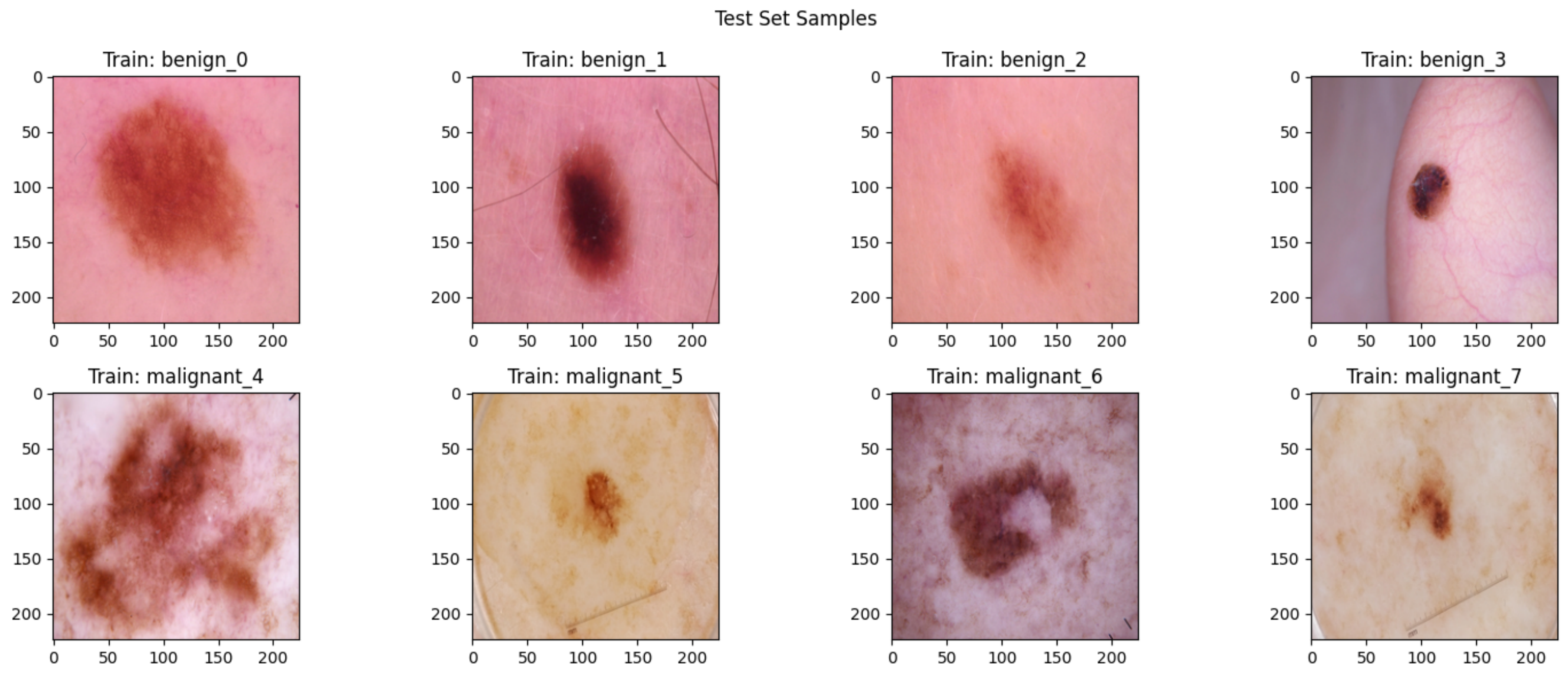
```
# Create path for train and test sets
_dir_train = 'data/train'
_dir_test = 'data/test'
```

Type *Markdown* and LaTeX: α^2

In [11]:
```
# Function to plot images from a directory
def plot_images_from_directory(directory, title, num):
    '''This function takes in a string path, a string title name, and number of images to be displayed
    The output will plot half of num of images for each class
    '''
    class_names = os.listdir(directory)
    num_classes = len(class_names)
    l = list()
    col = num//2
    for i, class_name in enumerate(class_names):
        class_dir = os.path.join(directory, class_name)
         # Take the n/2 number of images from each class
        for n in range(col):
            image_path = os.path.join(class_dir, os.listdir(class_dir)[n])
            l.append((image_path, class_name))

    fig, ax = plt.subplots(2,num//2, figsize = (15,6))

    for i in range(len(l)):
        r = i//col
        c = i%col
        img = load_img(l[i][0], target_size = (224, 224))
        ax[r,c].imshow(img)
        ax[r,c].set_title(f'Train: {l[i][1]}_{i}')

    plt.suptitle(title)
    plt.tight_layout()
    plt.show();
```

```python

plot_images_from_directory(_dir_train,'Train Set Samples', 8)
plot_images_from_directory(_dir_test,'Test Set Samples', 8)
```





4. Image Preprocessing

In [12]:

```
# Image Augmentation using ImageDataGenerator
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range = 0.2,
        zoom_range = 0.2,
        rotation_range=30,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip = True,
        validation_split = 0.2,
        fill_mode='nearest'
)

# For test set, image augmentation will not be applied to prevent overfitting
test_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 32 using train_datagen generator
training_set = train_datagen.flow_from_directory(
    _dir_train,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset = 'training'
)

validation_set = train_datagen.flow_from_directory(
    _dir_train,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset = 'validation'
)

# Flow validation images in batches of 32 using test_datagen generator
test_set = test_datagen.flow_from_directory(
    _dir_test,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle = 'False'
)

class_indices = training_set.class_indices
print("Class Indices:", class_indices)
```

```
Found 2110 images belonging to 2 classes.
Found 527 images belonging to 2 classes.
Found 660 images belonging to 2 classes.
Class Indices: {'benign': 0, 'malignant': 1}
```

5. Build a Convolutional Neural Networks (CNNs) as Baseline Model

a. Build a Base Model Structure

We start by building a Convolutional Neural Networks (CNNs) as our baseline model. It is designed to automatically learn and extract useful features from the input data, making them highly effective for image classification task as in this case.

The key components of a CNN architecture are:

- Convolutional Layers: These layers apply a set of learnable filters (or kernels) to the input data, extracting local features and patterns. The filters are learned during the training process.
- Pooling Layers: These layers reduce the spatial size of the feature maps, making the representation more manageable and invariant to small translations in the input.
- Fully Connected Layers: These layers connect all neurons from the previous layer to produce the final output, such as class probabilities in a classification task.

In [13]:
```
# Create an instance of the Sequential model
base_model = Sequential()
```

I will use 3 Convolutional layers with increasing filter sizes (32, 64, 128). The "relu" activation function is used in each of the convolution layer to introduce non-linearity, enabling the network to learn complex patterns and relationships in data. Early Convolutional layers detect simple features such as edges, corners, or textures. As the network goes deeper, it needs to learn more complex patterns, such as shapes, structures, and semantic representations. Increasing the number of filters allows the network to capture more varied and complex patterns. Filters act as feature detectors. More filters mean the model can learn a richer set of features at each layer, improving its ability to generalize across diverse input data. On the other hand, starting with too many filters in early layers can lead to overfitting or unnecessary computational overhead since these layers only need to learn simple features. A gradual increase ensures a smooth transition between layers and allows the network to build on the features learned in previous layers. Followed each convolutional layer, a pooling layer is used to reduce the spatial dimensions of the feature maps.

After Convolutional and Pooling layers, a Dense layer (fully connected) combines these features into a high-dimensional representation. A layer with 256 neurons provides a balance between model capacity and computational efficiency. This size is large enough to capture meaningful patterns but small enough to avoid overfitting.

In [14]:

```python
# Setting up 3 convolutional layers and 3 pooling layers
base_model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same',
           input_shape = (224,224,3)))
base_model.add(MaxPool2D(pool_size = (2,2), strides = 2))
base_model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu', padding = 'same'))
base_model.add(MaxPool2D(pool_size = (2,2), strides = 2))
base_model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu', padding = 'same'))
base_model.add(MaxPool2D(pool_size = (2,2), strides = 2))
# Flatten layer will flatten previous layers output into single vector
base_model.add(Flatten())

# Add a fully connected layer
base_model.add(Dense(units = 256, activation='relu'))

# Add an output layer with sigmoid as activation function
base_model.add(Dense(units = 1, activation='sigmoid'))
base_model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_
dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━
│ conv2d (Conv2D)                      │ (None, 224, 224, 32)        │            89
├──────────────────────────────────────┼─────────────────────────────┼───────────────
│ max_pooling2d (MaxPooling2D)         │ (None, 112, 112, 32)        │
├──────────────────────────────────────┼─────────────────────────────┼───────────────
│ conv2d_1 (Conv2D)                    │ (None, 112, 112, 64)        │         18,49
├──────────────────────────────────────┼─────────────────────────────┼───────────────
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 56, 56, 64)          │
├──────────────────────────────────────┼─────────────────────────────┼───────────────
│ conv2d_2 (Conv2D)                    │ (None, 56, 56, 128)         │         73,85
├──────────────────────────────────────┼─────────────────────────────┼───────────────
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 28, 28, 128)         │
├──────────────────────────────────────┼─────────────────────────────┼───────────────
│ flatten (Flatten)                    │ (None, 100352)              │
├──────────────────────────────────────┼─────────────────────────────┼───────────────
│ dense (Dense)                        │ (None, 256)                 │     25,690,36
├──────────────────────────────────────┼─────────────────────────────┼───────────────
│ dense_1 (Dense)                      │ (None, 1)                   │            25
└──────────────────────────────────────┴─────────────────────────────┴───────────────

 Total params: 25,783,873 (98.36 MB)

 Trainable params: 25,783,873 (98.36 MB)

 Non-trainable params: 0 (0.00 B)
```

b. Compile Base Model

Compiling a CNN model involves specifying the loss function, optimizer, and evaluation metrics. I use binary cross entropy as the loss function, a common loss function for binary classification. With the optimizer, I used Adam optimization, a commonly used optimizer. Adam optimization not only incorporates the advantages of two popular optimization techniques, the Momentum and RMSProp, but also helps the model converge faster compared to vanilla SGD by efficiently navigating through flat or steep regions of the loss surface. Lastly, I used both "Accuracy" and AUC (Area under the curve) as the metric to measure the effectiveness of the model.

In [15]:

```
# Compile the model

base_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy', AUC(name='auc')]
)
```

I also use 2 callbacks EarlyStopping and ModelCheckpoint to prevent overfitting, save computational resources by halting training when improvements plateau, and ensure that the best model (based on validation loss) is saved during training. The model trains for up to 50 epochs but may stop earlier if validation loss does not improve for 5 epochs due to EarlyStopping.

In [16]:

```python
# Early stopping and model checkpoints
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True)

history = base_model.fit(
    training_set,
    # Number of steps (batches) in one epoch. Computed as the total samples divided by batch size.
    steps_per_epoch=training_set.samples // training_set.batch_size,
    epochs=50,
    validation_data=validation_set,
    validation_steps=validation_set.samples // validation_set.batch_size,
    callbacks=[early_stopping]
)
```

```
Epoch 1/50

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class sh
ould call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do n
ot pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()

65/65 ━━━━━━━━━━━━━━━━━━━━ 337s 5s/step - accuracy: 0.6415 - auc: 0.6588 - loss: 1.0788 - val_accuracy: 0.7539 - val_auc: 0.7938 - val_los
s: 0.5867
Epoch 2/50
 1/65 ━━━━━━━━━━━━━━━━━━━━ 4:56 5s/step - accuracy: 0.7667 - auc: 0.8304 - loss: 0.5562

/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or gene
rator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()` function when building your dataset.
  self.gen.throw(typ, value, traceback)
```

```
65/65 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.7667 - auc: 0.8304 - loss: 0.5562 - val_accuracy: 0.7333 - val_auc: 0.8519 - val_loss
: 0.5182
Epoch 3/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 373s 5s/step - accuracy: 0.7273 - auc: 0.8355 - loss: 0.4944 - val_accuracy: 0.7031 - val_auc: 0.8092 - val_los
s: 0.5661
Epoch 4/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.7500 - auc: 0.8398 - loss: 0.4216 - val_accuracy: 0.8000 - val_auc: 0.7857 - val_los
s: 0.4765
Epoch 5/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 324s 5s/step - accuracy: 0.7789 - auc: 0.8627 - loss: 0.4465 - val_accuracy: 0.7793 - val_auc: 0.8241 - val_los
s: 0.4996
Epoch 6/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9062 - auc: 0.9479 - loss: 0.2288 - val_accuracy: 0.8667 - val_auc: 0.8929 - val_loss
: 0.3596
Epoch 7/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 331s 5s/step - accuracy: 0.7886 - auc: 0.8824 - loss: 0.4096 - val_accuracy: 0.7754 - val_auc: 0.8331 - val_los
s: 0.4903
Epoch 8/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.8438 - auc: 0.8843 - loss: 0.3691 - val_accuracy: 0.6667 - val_auc: 0.6250 - val_loss
: 0.6563
Epoch 9/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 322s 5s/step - accuracy: 0.7967 - auc: 0.8972 - loss: 0.3839 - val_accuracy: 0.7637 - val_auc: 0.8283 - val_los
s: 0.5002
Epoch 10/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.7188 - auc: 0.8398 - loss: 0.4439 - val_accuracy: 0.8667 - val_auc: 0.9821 - val_loss
: 0.2378
Epoch 11/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 329s 5s/step - accuracy: 0.7886 - auc: 0.8767 - loss: 0.4177 - val_accuracy: 0.7578 - val_auc: 0.8333 - val_los
s: 0.5439
Epoch 12/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 15s 173ms/step - accuracy: 0.8125 - auc: 0.8373 - loss: 0.4761 - val_accuracy: 0.8667 - val_auc: 0.9444 - val_l
oss: 0.3014
Epoch 13/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 323s 5s/step - accuracy: 0.8117 - auc: 0.9085 - loss: 0.3686 - val_accuracy: 0.7754 - val_auc: 0.8379 - val_los
s: 0.4965
Epoch 14/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.7500 - auc: 0.8938 - loss: 0.4016 - val_accuracy: 0.9333 - val_auc: 0.9167 - val_loss
: 0.2637
Epoch 15/50
65/65 ━━━━━━━━━━━━━━━━━━━━ 340s 5s/step - accuracy: 0.8169 - auc: 0.9083 - loss: 0.3664 - val_accuracy: 0.7656 - val_auc: 0.8350 - val_los
s: 0.5237
```

```
Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 10.
```

c. Evaluate the Model

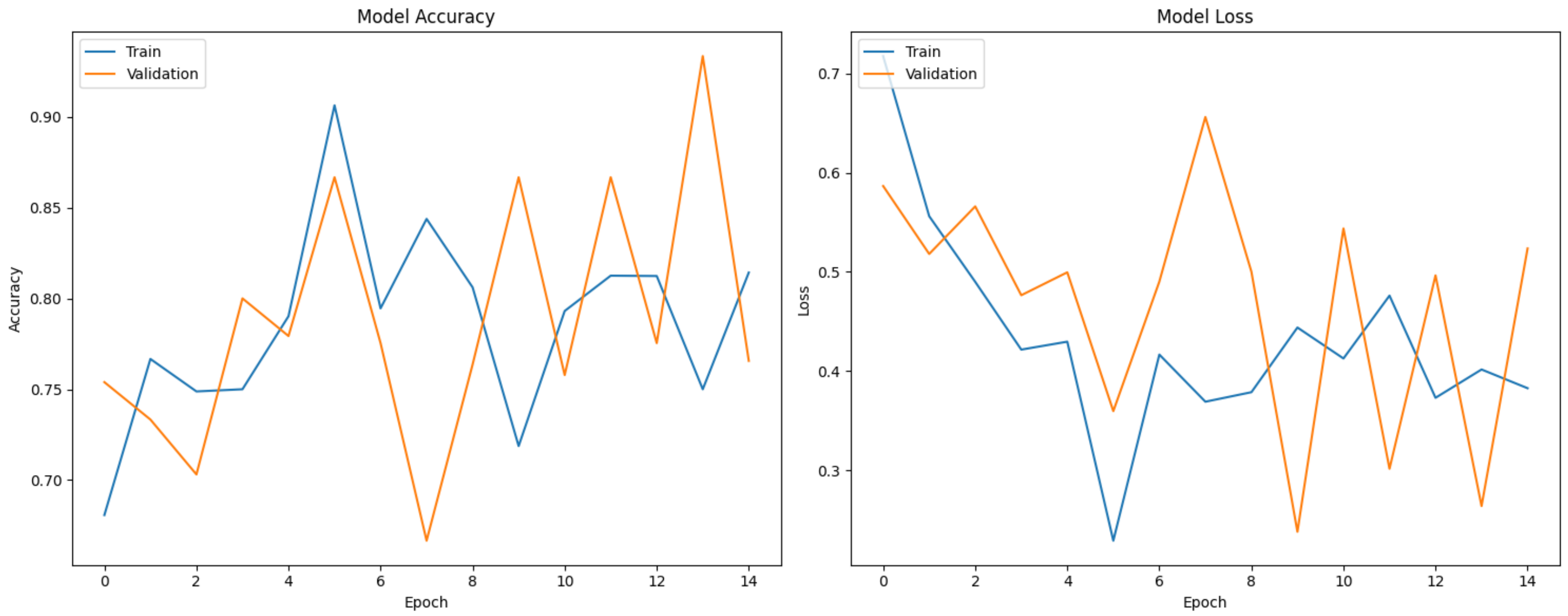
In [17]:

```python
# Evaluate the model
vloss, vaccuracy, *_ = base_model.evaluate(validation_set)
print(f'Validation Loss: {vloss:.4f}')
print(f'Validation Accuracy: {vaccuracy:.4f}')

tloss, taccuracy, *_ = base_model.evaluate(test_set)
print(f'Test set Loss: {tloss:.4f}')
print(f'Test set Accuracy: {taccuracy:.4f}')
```

```
17/17 ━━━━━━━━━━━━━━━━━━━━ 27s 2s/step - accuracy: 0.8055 - auc: 0.8266 - loss: 0.4750
Validation Loss: 0.4921
Validation Accuracy: 0.7856
21/21 ━━━━━━━━━━━━━━━━━━━━ 26s 1s/step - accuracy: 0.8028 - auc: 0.8806 - loss: 0.4090
Test set Loss: 0.4086
Test set Accuracy: 0.7955
```

This CNN model shows a decent performance with the test set with 79.55% accuracy and AUC is 0.8806. Next, will graph the training set and validation set's accuracy and loss metrics.

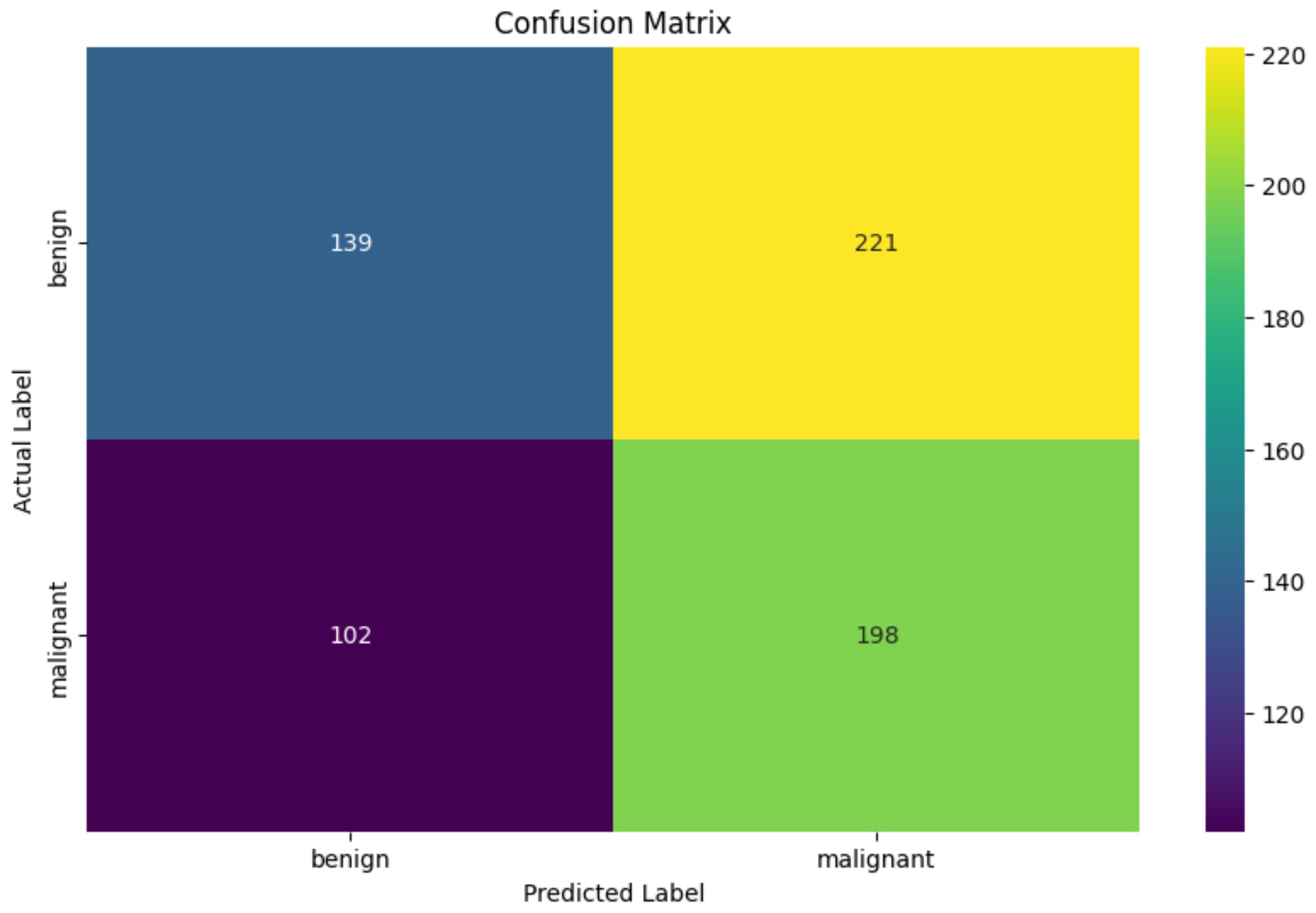
In [18]:
```python
fig, (ax1,ax2) = plt.subplots(nrows = 1, ncols = 2, figsize=(15, 6))

# Plot training & validation accuracy values
ax1.plot(history.history['accuracy'])
ax1.plot(history.history['val_accuracy'])
ax1.set_title('Model Accuracy')
ax1.set_ylabel('Accuracy')
ax1.set_xlabel('Epoch')
ax1.legend(['Train', 'Validation'], loc='upper left')

# # Plot training & validation loss values
ax2.plot(history.history['loss'])
ax2.plot(history.history['val_loss'])
ax2.set_title('Model Loss')
ax2.set_ylabel('Loss')
ax2.set_xlabel('Epoch')
ax2.legend(['Train', 'Validation'], loc='upper left')

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



d. Generate Confusion Matrix and Additional Metrics

In [19]:
```
predictions = base_model.predict(test_set)
y_pred = y_pred = np.where(predictions > 0.5, 1, 0)

# Confusion Matrix
cm = confusion_matrix(test_set.classes, y_pred)
class_names = list(test_set.class_indices.keys())

# Classification Report
cr = classification_report(test_set.classes, y_pred, target_names=class_names, output_dict=True)

# ROC Curve
fpr, tpr, _ = roc_curve(test_set.classes, predictions)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="viridis", xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show();

print('Classification Report:')
for key, value in cr.items():
    if isinstance(value, dict):
        print(f'\nClass: {key}')
        for k, v in value.items():
            print(f'{k}: {v}')
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 26s 1s/step
```



```
Classification Report:

Class: benign
precision: 0.5767634854771784
recall: 0.3861111111111111
f1-score: 0.46256239600665555
support: 360.0

Class: malignant
precision: 0.47255369928400953
recall: 0.66
f1-score: 0.5507649513212796
support: 300.0

Class: macro avg
precision: 0.524658592380594
recall: 0.5230555555555556
f1-score: 0.5066636736639676
support: 660.0

Class: weighted avg
precision: 0.5293954008439199
recall: 0.5106060606060606
f1-score: 0.5026544666042119
support: 660.0
```

The recall rate for the model is 66% despite accuracy achieves 79.55%. For a medical detection problem, this is not a quite acceptable recall rate. I will need to find a way to improve recall rate of this model. Here, I will utilize the ResNet50 model, a popular model for medical detection with small data set.

6. ResNet50

ResNet50 is a deep convolutional neural network (CNN) with 50 layers, part of the Residual Network (ResNet) family, designed to address the vanishing gradient problem and enable the training of very deep networks. It uses residual connections (skip connections) to allow gradients to flow through the network more effectively. I wil use the same image augmentation techniques used in baseline model. All train, validation and test set have same proportion split.

a. Image Preprocessing

In [26]:

```
# Image Augmentation using ImageDataGenerator
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range = 0.2,
        zoom_range = 0.2,
        rotation_range=30,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip = True,
        fill_mode='nearest',
        validation_split = 0.2
)

# For test set, image augmentation will not be applied to prevent overfitting
test_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 32 using train_datagen generator
training_set = train_datagen.flow_from_directory(
    _dir_train,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset = 'training'
)

validation_set = train_datagen.flow_from_directory(
    _dir_train,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset = 'validation'
)

# Flow validation images in batches of 32 using test_datagen generator
test_set = test_datagen.flow_from_directory(
    _dir_test,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle = 'False'
)

class_indices = training_set.class_indices
print("Class Indices:", class_indices)
```

```
Found 2110 images belonging to 2 classes.
Found 527 images belonging to 2 classes.
Found 660 images belonging to 2 classes.
Class Indices: {'benign': 0, 'malignant': 1}
```

The reason I decide to use RestNet 50 is due to its proven success in various medical image classification tasks ResNet50 is well-suited for medical image detection problems, including tasks like skin cancer classification, due to its innovative architecture and ability to handle complex features in image data. On the other hand, I also prevents the weights of the pre-trained layers from being updated during training. Freezing ensures that the base model serves as a fixed feature extractor while the custom layers learn task-specific patterns.

b. Load ResNet50 and Add Customize Layers

I do not use the ResNet50's last layer. Here, I decide to customize the last layer by adding a dense layer with

256 nodes and a relu activation function. The output layer will use a sigmoid activation function, a common activation used in binary classification. This last customize layer will ensure all features are considered.

In [27]:

```
# Load ResNet50
RN = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Prevents the weights of the pre-trained layers from being updated during training
for layer in RN.layers:
    layer.trainable = False

# Add custom classification layers
x = RN.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)  # Add a dense layer with 256 units
predictions = Dense(1, activation='sigmoid')(x)

# Combine base model and custom layers
model = Model(inputs=RN.input, outputs=predictions)
```

c. Train Model

In [28]:

```python
# Compile the model
model.compile(
    optimizer='adam',
    loss= 'binary_crossentropy',
    metrics=['accuracy', AUC(name='auc')]
)

# Train the model
history = model.fit(
    training_set,
    steps_per_epoch=training_set.samples // training_set.batch_size,
    validation_data=validation_set,
    validation_steps=validation_set.samples // validation_set.batch_size,
    epochs=30,
    callbacks=[early_stopping]
)

```

```
Epoch 1/30

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class sh
ould call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do n
ot pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()

65/65 ━━━━━━━━━━━━━━━━━━━━ 533s 8s/step - accuracy: 0.5311 - auc: 0.5348 - loss: 0.7381 - val_accuracy: 0.5547 - val_auc: 0.8445 - val_los
s: 0.6688
Epoch 2/30
 1/65 ━━━━━━━━━━━━━━━━━━━━ 5:38 5s/step - accuracy: 0.5938 - auc: 0.9028 - loss: 0.6582
```

```
/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or gene
rator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()` function when building your dataset.
  self.gen.throw(typ, value, traceback)


65/65 ━━━━━━━━━━━━━━━━━━━━ 8s 41ms/step - accuracy: 0.5938 - auc: 0.9028 - loss: 0.6582 - val_accuracy: 0.8667 - val_auc: 0.9107 - val_los
s: 0.6655
Epoch 3/30
65/65 ━━━━━━━━━━━━━━━━━━━━ 580s 8s/step - accuracy: 0.5697 - auc: 0.5613 - loss: 0.7038 - val_accuracy: 0.5488 - val_auc: 0.8285 - val_los
s: 0.6648
Epoch 4/30
65/65 ━━━━━━━━━━━━━━━━━━━━ 9s 66ms/step - accuracy: 0.5312 - auc: 0.9216 - loss: 0.6667 - val_accuracy: 0.6667 - val_auc: 0.8800 - val_los
s: 0.6085
Epoch 5/30
65/65 ━━━━━━━━━━━━━━━━━━━━ 519s 8s/step - accuracy: 0.5864 - auc: 0.6067 - loss: 0.6752 - val_accuracy: 0.7598 - val_auc: 0.8305 - val_los
s: 0.6546
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 1.
```

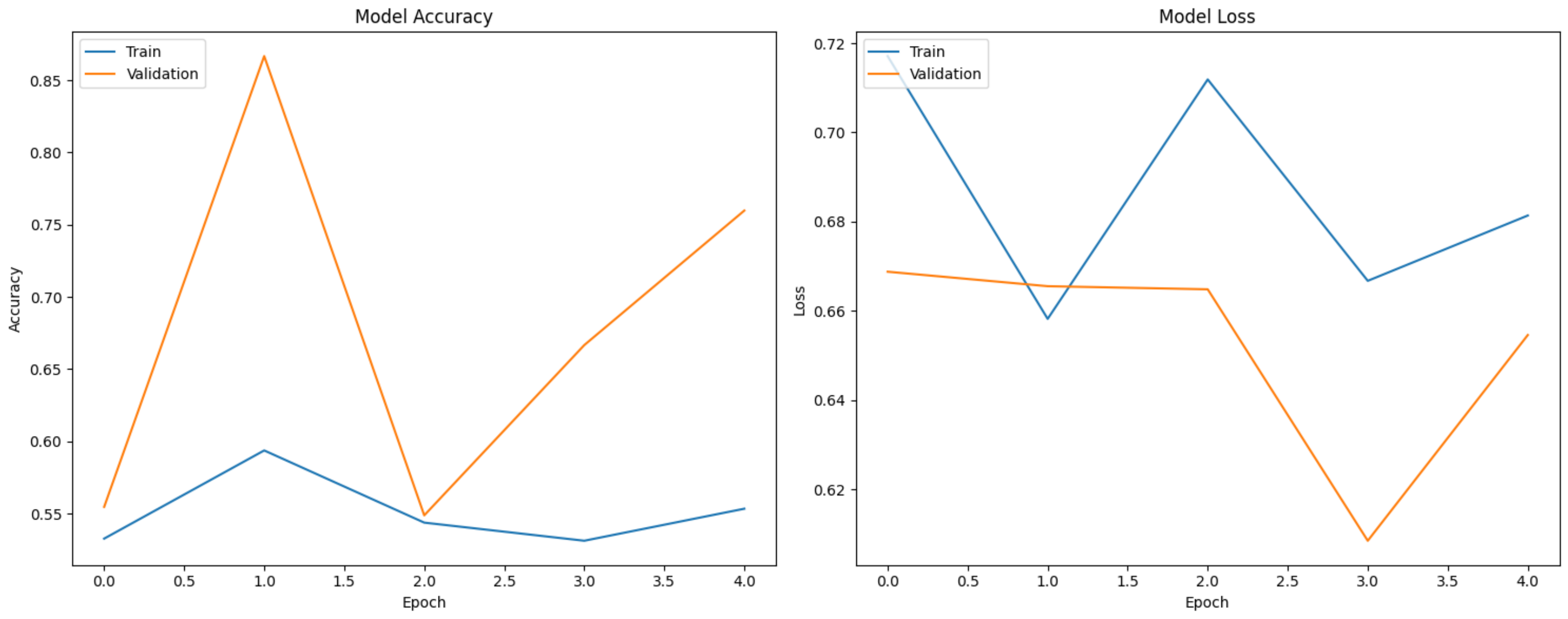
d. Evaluate Model

In [29]:
```python
# Evaluate the model
vloss, vaccuracy, *_ = model.evaluate(validation_set)
print(f'Validation Loss: {vloss:.4f}')
print(f'Validation Accuracy: {vaccuracy:.4f}')

tloss, taccuracy, *_ = model.evaluate(test_set)
print(f'Test set Loss: {tloss:.4f}')
print(f'Test set Accuracy: {taccuracy:.4f}')
```

```
17/17 ━━━━━━━━━━━━━━━━━━━━ 105s 6s/step - accuracy: 0.5382 - auc: 0.8066 - loss: 0.6732
Validation Loss: 0.6693
Validation Accuracy: 0.5541
21/21 ━━━━━━━━━━━━━━━━━━━━ 125s 6s/step - accuracy: 0.5574 - auc: 0.7803 - loss: 0.6719
Test set Loss: 0.6684
Test set Accuracy: 0.5682
```

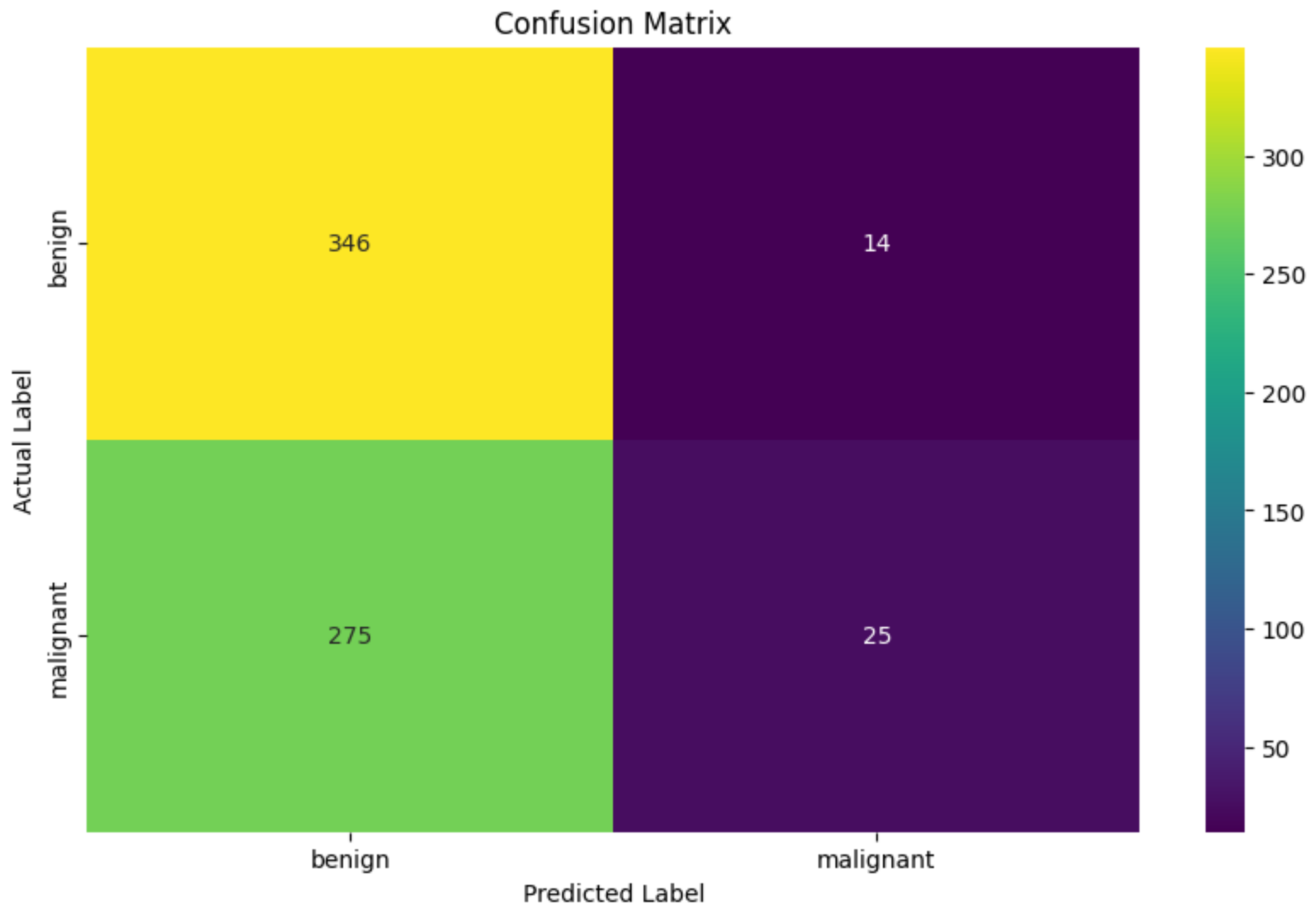
In [30]:

```
fig, (ax1,ax2) = plt.subplots(nrows = 1, ncols = 2, figsize=(15, 6))

# Plot training & validation accuracy values
ax1.plot(history.history['accuracy'])
ax1.plot(history.history['val_accuracy'])
ax1.set_title('Model Accuracy')
ax1.set_ylabel('Accuracy')
ax1.set_xlabel('Epoch')
ax1.legend(['Train', 'Validation'], loc='upper left')

# # Plot training & validation loss values
ax2.plot(history.history['loss'])
ax2.plot(history.history['val_loss'])
ax2.set_title('Model Loss')
ax2.set_ylabel('Loss')
ax2.set_xlabel('Epoch')
ax2.legend(['Train', 'Validation'], loc='upper left')

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


Model Accuracy
Train
Validation
Accuracy
Epoch
Model Loss
Train
Validation
Loss
Epoch

In [31]:
```
predictions = model.predict(test_set)
y_pred = np.where(predictions > 0.5, 1, 0)

# Confusion Matrix
cm = confusion_matrix(test_set.classes, y_pred)
class_names = list(test_set.class_indices.keys())

# Classification Report
cr = classification_report(test_set.classes, y_pred, target_names=class_names, output_dict=True)

# ROC Curve
fpr, tpr, _ = roc_curve(test_set.classes, predictions)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="viridis", xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show();

print('Classification Report:')
for key, value in cr.items():
    if isinstance(value, dict):
        print(f'\nClass: {key}')
        for k, v in value.items():
            print(f'{k}: {v}')
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 138s 6s/step
```



```
Classification Report:

Class: benign
precision: 0.5571658615136876
recall: 0.9611111111111111
f1-score: 0.7054026503567788
support: 360.0

Class: malignant
precision: 0.6410256410256411
recall: 0.08333333333333333
f1-score: 0.14749262536873156
support: 300.0

Class: macro avg
precision: 0.5990957512696644
recall: 0.5222222222222223
f1-score: 0.4264476378627552
support: 660.0

Class: weighted avg
precision: 0.5952839431100301
recall: 0.5621212121212121
f1-score: 0.45180718445312096
support: 660.0
```

With the use of Resnet50 model, there is no improvement in both recall rate for the malignant class and accuracy. In fact, both the recall rate and accuracy drastically dipped (8% recall rate and 56.82%). The cause of such low recall rate may result from model complexity. With fewer layers and simpler architecture, shallow CNNs are less complex and thus easier to train on smaller or less complex datasets. They may find a good balance between bias and variance. On the other hand, being a more complex model, ResNet50 can capture more intricate patterns, which is beneficial for large and complex datasets. However, for simpler tasks or datasets, this complexity might not translate into better performance.

7. Hybrid Model with XGBoost and ResNet50

In the final part of this project, I will create a hybrid model, combining ResNet50's feature extraction process with the XGBoost model. XGBoost (eXtreme Gradient Boosting) is an efficient and scalable implementation of the gradient boosting algorithm. It builds an ensemble of decision trees, where each tree corrects the errors of the previous ones, optimizing a loss function. A hybrid model combining ResNet50 for feature extraction and XGBoost for classification often performs better than using ResNet50 alone for end-to-end classification. While ResNet50 specializes in extracting high-level features from images (e.g., edges, textures, shapes), XGBoost excels in handling structured data and learning complex decision boundaries efficiently.

a. Load ResNet50 for Feature Extraction

In [32]:

```
# Excludes the final fully connected (classification) layer.
RN_model = ResNet50(weights='imagenet', include_top=False, pooling='avg')

# Applies ResNet50-specific preprocessing, such as scaling pixel values to match ImageNet training.
datagen = ImageDataGenerator(preprocessing_function=preprocess_input)


# shuffle=False to maintains the original order of images, important for linking features with labels l
training_set = datagen.flow_from_directory(
    _dir_train,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)

# Passes images through ResNet50 to extract high-level features.
features = RN_model.predict(training_set)
labels = training_set.classes

# Split Data for Training and Testing
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=50)
```

```
Found 2637 images belonging to 2 classes.


/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class sh
ould call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do n
ot pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

```
83/83 ━━━━━━━━━━━━━━━━━━━━ 509s 6s/step
```

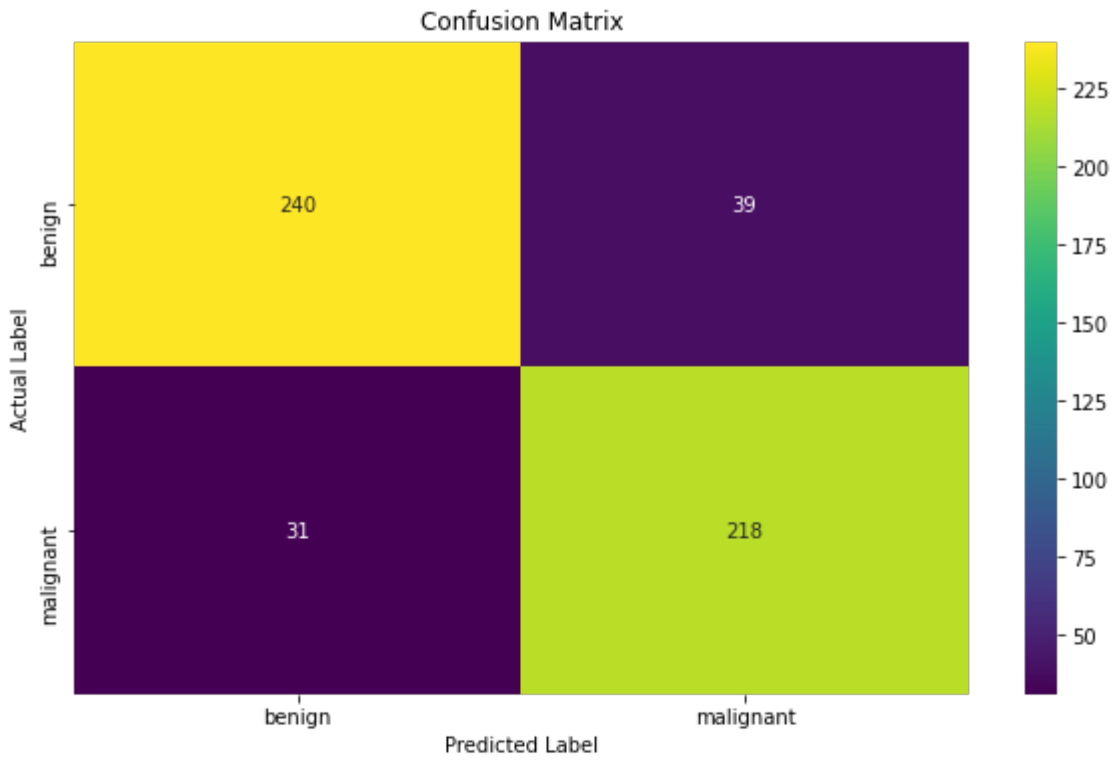
b. Train XGBoost Model with ResNet50 Feature Extraction

In [33]:
```python
# A gradient boosting decision tree algorithm for binary classification problem with logistic loss.
xgb_model = xgb.XGBClassifier(objective='binary:logistic',
#                               Builds 100 trees in the ensemble.
                              n_estimators=100,
#                               Controls the contribution of each tree to the final prediction
                              learning_rate=0.1,
                              max_depth=5, random_state=42)
xgb_model.fit(X_train, y_train)
# Predict labels for the test set
y_pred = xgb_model.predict(X_test)
```

c. Evaluate Hybrid Model

In [ ]:

```
# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)
class_names = list(test_set.class_indices.keys())
# Plot the confusion matrix
plt.figure(figsize=(10, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="viridis", xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show();

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
print('Classification Report:')
print(classification_report(y_test, y_pred))

```



```
Accuracy: 0.8674
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.86      0.87       279
           1       0.85      0.88      0.86       249

    accuracy                           0.87       528
   macro avg       0.87      0.87      0.87       528
weighted avg       0.87      0.87      0.87       528
```

This hybrid model performed significantly better than both the base line CNN model and the ResNet50 model. This hybrid model achieve a 88% recall rate and 86.74% accuracy rate with the test set. Fine-tuning ResNet50 end-to-end on a small dataset can lead to overfitting due to the large number of parameters. Using ResNet50 as a frozen feature extractor reduces the risk of overfitting by focusing only on pre-trained feature extraction. XGBoost, with regularization parameters, helps generalize well even on small datasets.

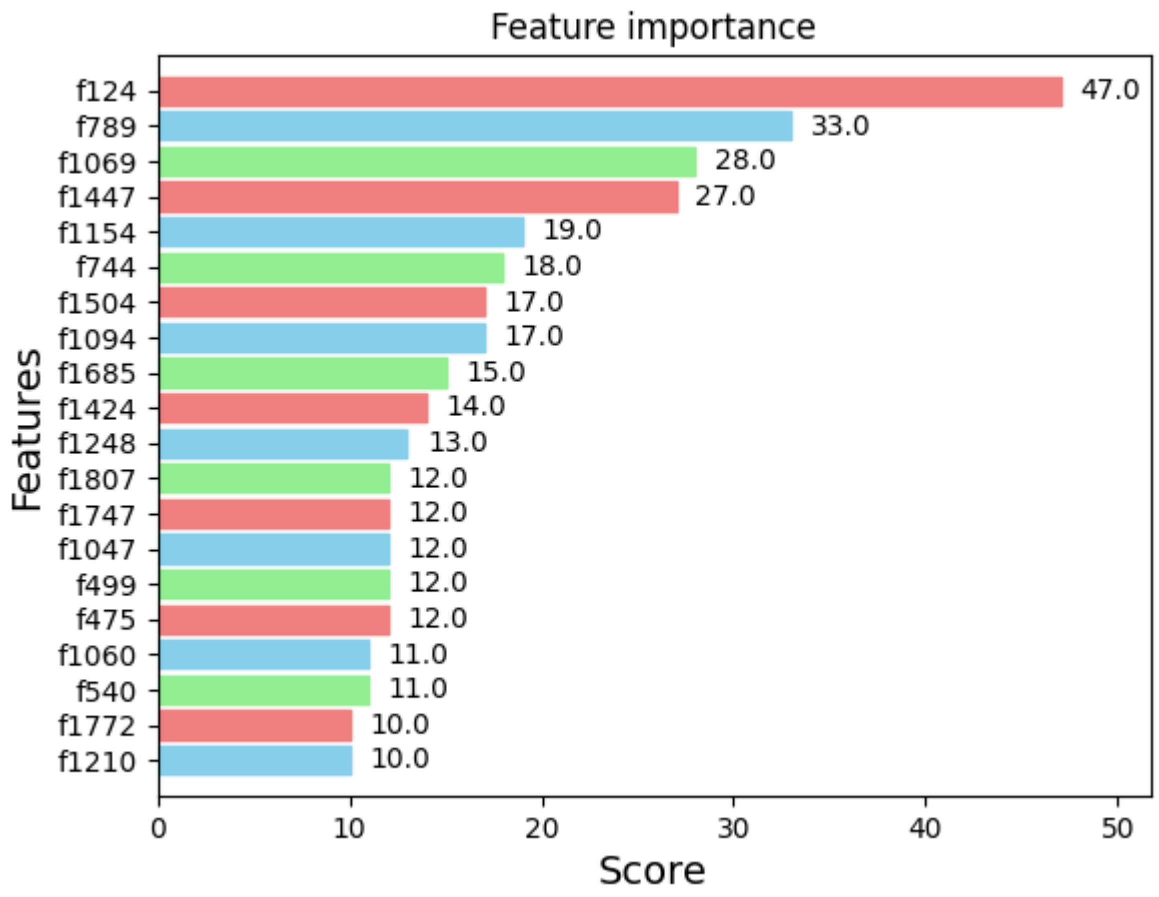
d. Feature Importance

I will visualize the feature importance of the xgb model. The plot will display the top 20 contributing features from the ResNet50 feature extraction step (used by XGBoost). Each bar will represent the importance of a feature, with a clear title, labels, and colors. The importance ranks how these high-dimensional features contribute to predictions.

In [34]:

```python
# Visualize Feature Importance - Ranks features based on their usage in splits across trees
plt.figure(figsize=(15, 10))
ax = xgb.plot_importance(xgb_model, importance_type='weight',
                         max_num_features=20,  # Adjust the number of features to display
                         color='skyblue',
                         height=0.8)

# Customize the plot to make it colorfu
ax.set_xlabel('Score', fontsize=14)
ax.set_ylabel('Features', fontsize=14)
ax.grid(False)


# Add a color theme
for i, bar in enumerate(ax.patches):
    if i % 3 == 0:
        bar.set_color('skyblue')
    elif i % 3 == 1:
        bar.set_color('lightcoral')
    else:
        bar.set_color('lightgreen')

plt.show();
```

```
<Figure size 1500x1000 with 0 Axes>
```



Out of all 3 models, the hybrid model appear to perform best with good recall rate and accuracy rate. I will save this hybrid model as the final model for this skin cancer detection problem.

In [35]:

```
# Save the XGBoost model

joblib.dump(xgb_model, 'models/hybrid_xgboost_model.pkl')
```

```
['models/hybrid_xgboost_model.pkl']
```

For the purpose of transfer learning, I provide the following codes for potential users to utilize the final hybrid model for their own photos. The process here including converting image into array, preprocess image with ResNet50's preprocessing input function, extract features using ResNet50, and finally predict using the loaded model.

In [48]:

```
# Load the photo names into a list of photo_files for testing
_dir_predict_photo = 'data/test/malignant'
photo_files = [f for f in os.listdir(_dir_predict_photo)]
```

In [49]:

```
# Load the pre-trained model (ResNet50 + XGBoost)
with open('models/hybrid_xgboost_model.pkl', 'rb') as file:
    loaded_model = pickle.load(file)

# Load ResNet50 model for feature extraction (excluding the top layer)
resnet_model = ResNet50(weights='imagenet', include_top=False, pooling='avg')
```

In [50]:

```
# Function to preprocess a single image
def preprocess_image(img_path):
    # Load the image and convert to an array
    img = load_img(img_path, target_size=(224, 224))
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)

    # Preprocessing the image array using ResNet50 preprocess_input function
    return preprocess_input(img_array)
```

In [53]:
```
# Function to predict using the loaded model
def predict(img_path):
    # Preprocess the image
    img_array = preprocess_image(img_path)

    # Extract features using ResNet50
    features = resnet_model.predict(img_array)

    # Use the XGBoost model to make predictions
    prediction = loaded_model.predict(features)
    return prediction
```

In [55]:
```
# Sample usage
for i in range(10):
  predicted_label = predict(os.path.join(_dir_predict_photo, photo_files[i]))
  print(f"Predicted Label: {predicted_label}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 401ms/step
Predicted Label: [1]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 346ms/step
Predicted Label: [1]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 359ms/step
Predicted Label: [1]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 316ms/step
Predicted Label: [1]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 362ms/step
Predicted Label: [0]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 353ms/step
Predicted Label: [0]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 281ms/step
Predicted Label: [1]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
Predicted Label: [1]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 204ms/step
Predicted Label: [1]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 338ms/step
Predicted Label: [1]
```

By using the pre-trained hybrid XGBoost model, we got 80% accuracy rate out of 10 predictions.

9. Conclusion

Based on our analysis and testing with different models, I want to recommend a hybrid model combining ResNet50's feature extraction process with the XGBoost model for this cancer detection problem. This final hybrid model not only achieved the highest accuracy rate among the 3 models, it also maximized the recall rate, prioritize the ability of the model to correctly identify all positive cases. In medical diagnoses, a false negative (failing to detect cancer when it is present) can have severe consequences, such as delayed treatment or worsened prognosis. Recall ensures that the model minimizes false negatives, even if it occasionally produces false positives.

10. Future Works

To better improve the quality of this project, I will extend this project by investigate other popular models used in medical detection field such as VGG16, InceptionV3 or MobileNetV3. Moreover, a hybridd between these new models may potentially improve the recall rate for this skin cancer detection problem.

In [ ]:

```
```