In [1]:
```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Den
from keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

In [8]:
```
# Load and preprocess the data
import numpy as np
import json

def load_data(train_file_path, test_file_path):
    """
    Load training and test data from JSON files and return as arrays.

    Parameters:
    train_file_path (str): Path to the training JSON file.
    test_file_path (str): Path to the test JSON file.

    Returns:
    tuple: X_train, y_train, X_test arrays
    """
    # Load the JSON data
    with open(train_file_path) as train_file:
        train_data = json.load(train_file)
    with open(test_file_path) as test_file:
        test_data = json.load(test_file)

    # Extract features and labels, convert to numpy arrays
    X_train = np.array([np.array(instance['band_1'], dtype=np.float32).reshape(7
    X_test = np.array([np.array(instance['band_1'], dtype=np.float32).reshape(75
    y_train = np.array([instance['is_iceberg'] for instance in train_data], dtyp

    return X_train, y_train, X_test

# Usage example:
# X_train, y_train, X_test = load_data('path/to/train.json', 'path/to/test.json'


def normalize(X):
    return (X - np.min(X)) / (np.max(X) - np.min(X))

def preprocess(X):
    X_normalized = np.array([normalize(image) for image in X])
    return X_normalized.reshape(-1, 75, 75, 1)

X_train, y_train, X_test = load_data(r'train.json\data\processed\train.json',r't
X_train = preprocess(X_train)
X_test = preprocess(X_test)

# Train-test split for validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
```

In [9]:
```
# Define model hyperparameters
batch_size = 32
num_epochs = 50
kernel_size = 3
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
conv_depth_3 = 128
dense_1 = 128
dense_2 = 1
drop_out = 0.3
weight_decay = 1e-4

# Model architecture with modifications
model = Sequential()
model.add(Conv2D(conv_depth_1, (kernel_size, kernel_size), input_shape=(75, 75,
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

model.add(Conv2D(conv_depth_2, (kernel_size, kernel_size), padding='same', kerne
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

model.add(Conv2D(conv_depth_3, (kernel_size, kernel_size), padding='same', kerne
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

model.add(Flatten())
model.add(Dense(dense_1, kernel_regularizer=l2(weight_decay)))
model.add(Activation('relu'))
model.add(Dropout(drop_out))

model.add(Dense(dense_2, activation='sigmoid'))

# Compile the model
optimizer = Adam(learning_rate=0.001)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accurac
```

```
c:\Users\sanba\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_co
nv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
yer. When using Sequential models, prefer using an `Input(shape)` object as the f
irst layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [10]:
```
# Model training
history = model.fit(
    X_train, y_train,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_data=(X_val, y_val),
    callbacks=[reduce_lr]
)

# Save model
model.save_weights('modified_model.weights.h5')
```

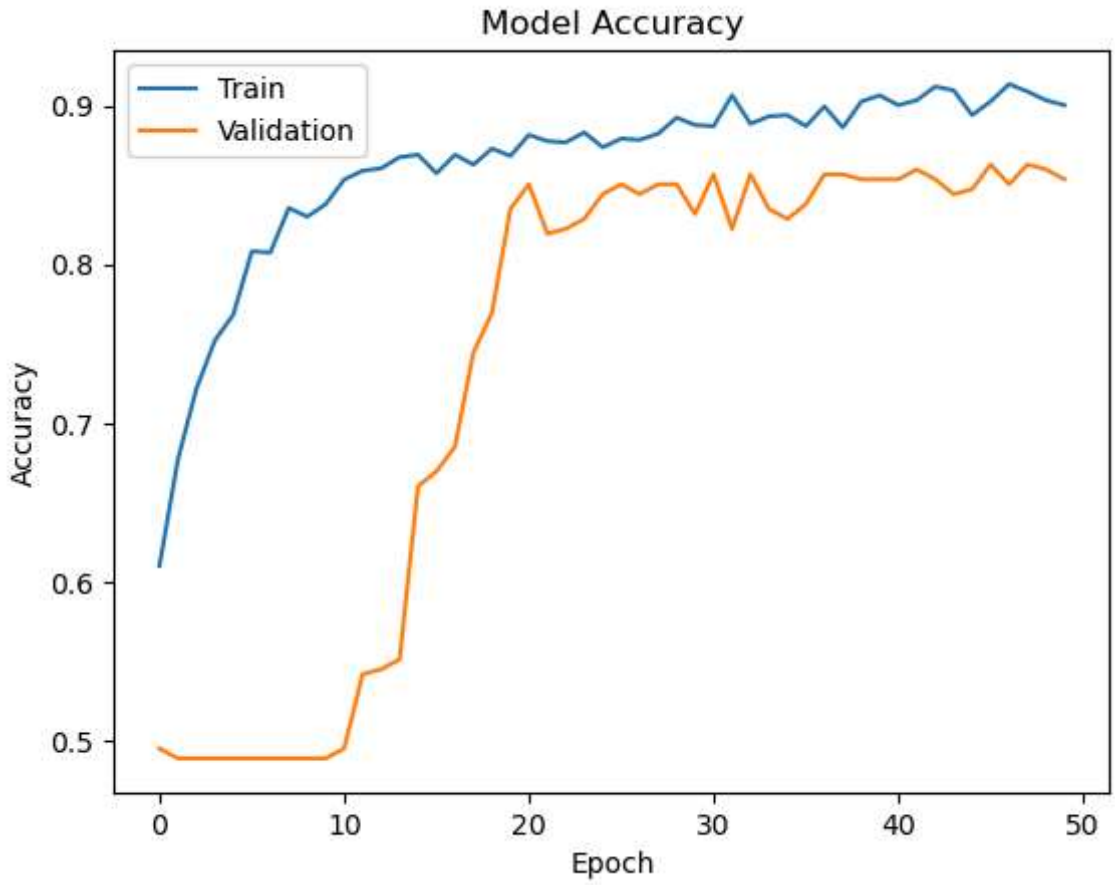
```
Epoch 1/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 19s 307ms/step - accuracy: 0.5809 - loss: 2.5957 - val
_accuracy: 0.4953 - val_loss: 0.7304 - learning_rate: 0.0010
Epoch 2/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 12s 293ms/step - accuracy: 0.6657 - loss: 0.6402 - val
_accuracy: 0.4891 - val_loss: 0.7416 - learning_rate: 0.0010
Epoch 3/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 11s 261ms/step - accuracy: 0.7124 - loss: 0.5966 - val
_accuracy: 0.4891 - val_loss: 0.7706 - learning_rate: 0.0010
Epoch 4/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 11s 255ms/step - accuracy: 0.7307 - loss: 0.5493 - val
_accuracy: 0.4891 - val_loss: 0.9636 - learning_rate: 0.0010
Epoch 5/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 11s 260ms/step - accuracy: 0.7634 - loss: 0.5021 - val
_accuracy: 0.4891 - val_loss: 0.8819 - learning_rate: 5.0000e-04
Epoch 6/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 11s 257ms/step - accuracy: 0.8102 - loss: 0.4660 - val
_accuracy: 0.4891 - val_loss: 0.9208 - learning_rate: 5.0000e-04
Epoch 7/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 11s 256ms/step - accuracy: 0.7970 - loss: 0.4819 - val
_accuracy: 0.4891 - val_loss: 0.9537 - learning_rate: 5.0000e-04
Epoch 8/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 10s 249ms/step - accuracy: 0.8278 - loss: 0.4253 - val
_accuracy: 0.4891 - val_loss: 0.8965 - learning_rate: 2.5000e-04
Epoch 9/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 10s 251ms/step - accuracy: 0.8215 - loss: 0.4527 - val
_accuracy: 0.4891 - val_loss: 0.8936 - learning_rate: 2.5000e-04
Epoch 10/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 11s 255ms/step - accuracy: 0.8472 - loss: 0.4222 - val
_accuracy: 0.4891 - val_loss: 0.9120 - learning_rate: 2.5000e-04
Epoch 11/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 10s 250ms/step - accuracy: 0.8592 - loss: 0.4046 - val
_accuracy: 0.4953 - val_loss: 0.8982 - learning_rate: 1.2500e-04
Epoch 12/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 11s 258ms/step - accuracy: 0.8625 - loss: 0.4136 - val
_accuracy: 0.5421 - val_loss: 0.8499 - learning_rate: 1.2500e-04
Epoch 13/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 12s 303ms/step - accuracy: 0.8614 - loss: 0.3870 - val
_accuracy: 0.5452 - val_loss: 0.8925 - learning_rate: 1.2500e-04
Epoch 14/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 11s 278ms/step - accuracy: 0.8830 - loss: 0.3640 - val
_accuracy: 0.5514 - val_loss: 0.9141 - learning_rate: 6.2500e-05
Epoch 15/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 10s 246ms/step - accuracy: 0.8799 - loss: 0.3885 - val
_accuracy: 0.6604 - val_loss: 0.6872 - learning_rate: 6.2500e-05
Epoch 16/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 221ms/step - accuracy: 0.8674 - loss: 0.3714 - val_
accuracy: 0.6698 - val_loss: 0.6683 - learning_rate: 6.2500e-05
Epoch 17/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 226ms/step - accuracy: 0.8797 - loss: 0.3592 - val_
accuracy: 0.6854 - val_loss: 0.6531 - learning_rate: 6.2500e-05
Epoch 18/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 221ms/step - accuracy: 0.8670 - loss: 0.3895 - val_
accuracy: 0.7445 - val_loss: 0.5261 - learning_rate: 6.2500e-05
Epoch 19/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 216ms/step - accuracy: 0.8800 - loss: 0.3628 - val_
accuracy: 0.7695 - val_loss: 0.4718 - learning_rate: 6.2500e-05
Epoch 20/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 214ms/step - accuracy: 0.8748 - loss: 0.3618 - val_
accuracy: 0.8349 - val_loss: 0.3978 - learning_rate: 6.2500e-05
```

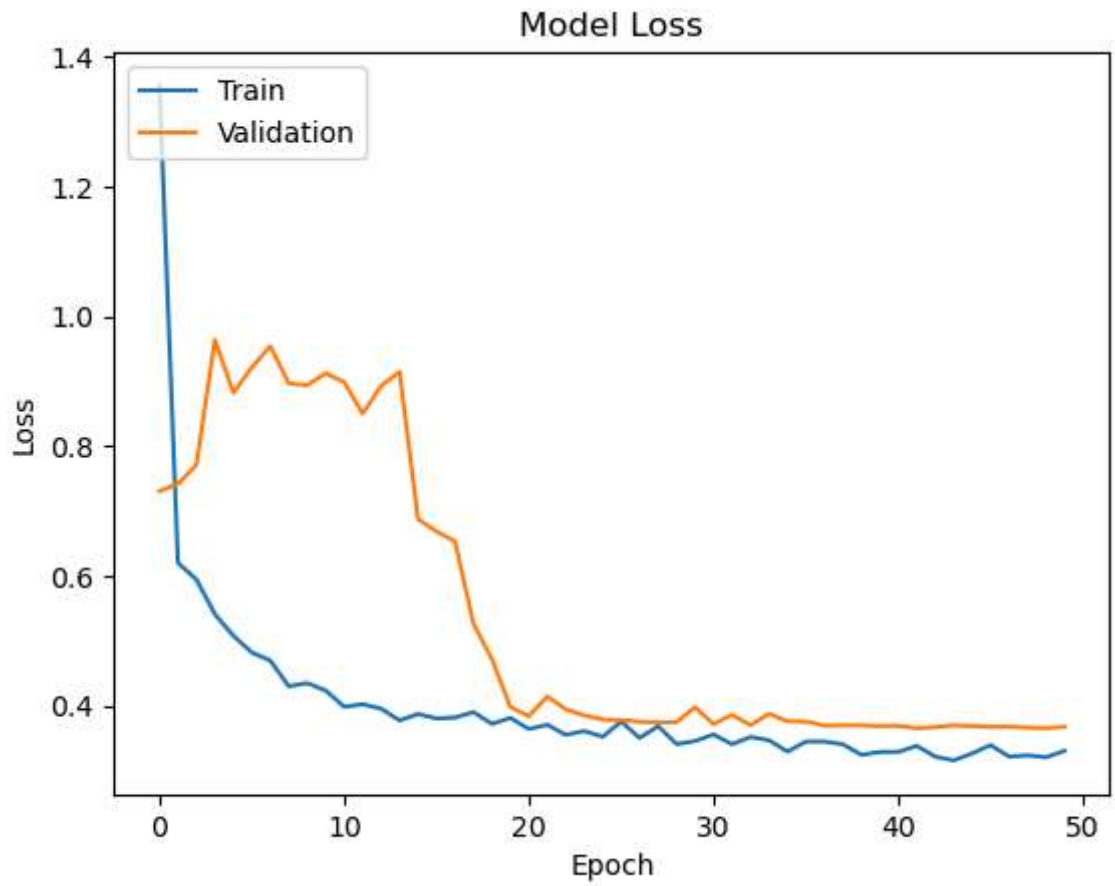
```
Epoch 21/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 214ms/step - accuracy: 0.8808 - loss: 0.3688 - val_
accuracy: 0.8505 - val_loss: 0.3838 - learning_rate: 6.2500e-05
Epoch 22/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 213ms/step - accuracy: 0.8879 - loss: 0.3551 - val_
accuracy: 0.8193 - val_loss: 0.4141 - learning_rate: 6.2500e-05
Epoch 23/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 214ms/step - accuracy: 0.8819 - loss: 0.3481 - val_
accuracy: 0.8224 - val_loss: 0.3944 - learning_rate: 6.2500e-05
Epoch 24/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 216ms/step - accuracy: 0.8686 - loss: 0.3691 - val_
accuracy: 0.8287 - val_loss: 0.3848 - learning_rate: 6.2500e-05
Epoch 25/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 217ms/step - accuracy: 0.8695 - loss: 0.3678 - val_
accuracy: 0.8442 - val_loss: 0.3786 - learning_rate: 3.1250e-05
Epoch 26/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 218ms/step - accuracy: 0.8888 - loss: 0.3675 - val_
accuracy: 0.8505 - val_loss: 0.3774 - learning_rate: 3.1250e-05
Epoch 27/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 212ms/step - accuracy: 0.8780 - loss: 0.3507 - val_
accuracy: 0.8442 - val_loss: 0.3751 - learning_rate: 3.1250e-05
Epoch 28/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 212ms/step - accuracy: 0.8802 - loss: 0.3694 - val_
accuracy: 0.8505 - val_loss: 0.3740 - learning_rate: 3.1250e-05
Epoch 29/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 213ms/step - accuracy: 0.8965 - loss: 0.3256 - val_
accuracy: 0.8505 - val_loss: 0.3745 - learning_rate: 3.1250e-05
Epoch 30/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 213ms/step - accuracy: 0.8993 - loss: 0.3300 - val_
accuracy: 0.8318 - val_loss: 0.3975 - learning_rate: 3.1250e-05
Epoch 31/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 212ms/step - accuracy: 0.8880 - loss: 0.3651 - val_
accuracy: 0.8567 - val_loss: 0.3713 - learning_rate: 3.1250e-05
Epoch 32/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 210ms/step - accuracy: 0.9058 - loss: 0.3451 - val_
accuracy: 0.8224 - val_loss: 0.3856 - learning_rate: 3.1250e-05
Epoch 33/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 215ms/step - accuracy: 0.8714 - loss: 0.3706 - val_
accuracy: 0.8567 - val_loss: 0.3698 - learning_rate: 3.1250e-05
Epoch 34/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 10s 203ms/step - accuracy: 0.8937 - loss: 0.3483 - val
_accuracy: 0.8349 - val_loss: 0.3874 - learning_rate: 3.1250e-05
Epoch 35/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 10s 250ms/step - accuracy: 0.8826 - loss: 0.3382 - val
_accuracy: 0.8287 - val_loss: 0.3758 - learning_rate: 3.1250e-05
Epoch 36/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 229ms/step - accuracy: 0.8905 - loss: 0.3505 - val_
accuracy: 0.8380 - val_loss: 0.3755 - learning_rate: 3.1250e-05
Epoch 37/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 214ms/step - accuracy: 0.9012 - loss: 0.3467 - val_
accuracy: 0.8567 - val_loss: 0.3691 - learning_rate: 1.5625e-05
Epoch 38/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 211ms/step - accuracy: 0.8974 - loss: 0.3294 - val_
accuracy: 0.8567 - val_loss: 0.3697 - learning_rate: 1.5625e-05
Epoch 39/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 210ms/step - accuracy: 0.9040 - loss: 0.3137 - val_
accuracy: 0.8536 - val_loss: 0.3695 - learning_rate: 1.5625e-05
Epoch 40/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 214ms/step - accuracy: 0.9012 - loss: 0.3278 - val_
accuracy: 0.8536 - val_loss: 0.3683 - learning_rate: 1.5625e-05
```

```
Epoch 41/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 211ms/step - accuracy: 0.9082 - loss: 0.3288 - val_
accuracy: 0.8536 - val_loss: 0.3689 - learning_rate: 1.5625e-05
Epoch 42/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 209ms/step - accuracy: 0.9090 - loss: 0.3318 - val_
accuracy: 0.8598 - val_loss: 0.3652 - learning_rate: 1.5625e-05
Epoch 43/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 210ms/step - accuracy: 0.9058 - loss: 0.3325 - val_
accuracy: 0.8536 - val_loss: 0.3668 - learning_rate: 1.5625e-05
Epoch 44/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 213ms/step - accuracy: 0.9024 - loss: 0.3213 - val_
accuracy: 0.8442 - val_loss: 0.3696 - learning_rate: 1.5625e-05
Epoch 45/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 217ms/step - accuracy: 0.8936 - loss: 0.3344 - val_
accuracy: 0.8474 - val_loss: 0.3685 - learning_rate: 1.5625e-05
Epoch 46/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 211ms/step - accuracy: 0.9141 - loss: 0.3321 - val_
accuracy: 0.8629 - val_loss: 0.3672 - learning_rate: 1.0000e-05
Epoch 47/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 10s 244ms/step - accuracy: 0.9119 - loss: 0.3144 - val
_accuracy: 0.8505 - val_loss: 0.3674 - learning_rate: 1.0000e-05
Epoch 48/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 221ms/step - accuracy: 0.9129 - loss: 0.3305 - val_
accuracy: 0.8629 - val_loss: 0.3662 - learning_rate: 1.0000e-05
Epoch 49/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 215ms/step - accuracy: 0.9176 - loss: 0.3091 - val_
accuracy: 0.8598 - val_loss: 0.3653 - learning_rate: 1.0000e-05
Epoch 50/50
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 215ms/step - accuracy: 0.8897 - loss: 0.3483 - val_
accuracy: 0.8536 - val_loss: 0.3673 - learning_rate: 1.0000e-05
```

In [11]:

```python
# Plotting accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plotting loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Prediction example (optional)
predictions = model.predict(X_test)
print(predictions[:5])
```






```
264/264 ━━━━━━━━━━━━━━━━━━━━ 13s 49ms/step
[[0.10773922]
 [0.80737066]
 [0.18277249]
 [0.9919188 ]
 [0.94463503]]
```

In [12]:
```
# Train-test split for validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
```

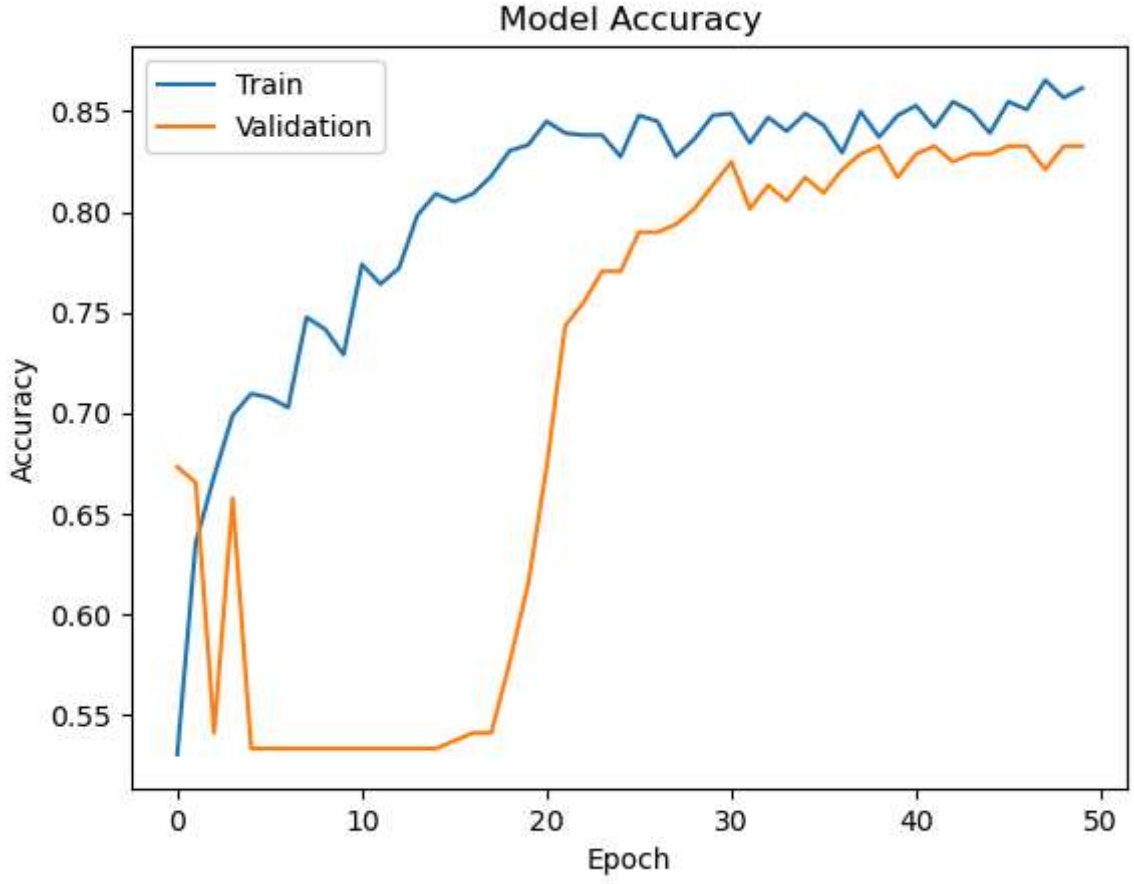
In [13]:
```
# Define model hyperparameters
batch_size = 32
num_epochs = 50
kernel_size = 3
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
conv_depth_3 = 128
dense_1 = 128
dense_2 = 1
drop_out = 0.3
weight_decay = 1e-4

# Model architecture with modifications
model = Sequential()
model.add(Conv2D(conv_depth_1, (kernel_size, kernel_size), input_shape=(75, 75,
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

model.add(Conv2D(conv_depth_2, (kernel_size, kernel_size), padding='same', kerne
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

model.add(Conv2D(conv_depth_3, (kernel_size, kernel_size), padding='same', kerne
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

model.add(Flatten())
model.add(Dense(dense_1, kernel_regularizer=l2(weight_decay)))
model.add(Activation('relu'))
model.add(Dropout(drop_out))

model.add(Dense(dense_2, activation='sigmoid'))

# Compile the model
optimizer = Adam(learning_rate=0.001)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accurac

# Model training
history = model.fit(
    X_train, y_train,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_data=(X_val, y_val),
    callbacks=[reduce_lr]
)

# Save model
model.save_weights('modified_model.weights.h5')

# Plotting accuracy
plt.plot(history.history['accuracy'])
```

```
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 12s 228ms/step - accuracy: 0.4999 - loss: 5.4829 - val
_accuracy: 0.6732 - val_loss: 0.7146 - learning_rate: 0.0010
Epoch 2/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 221ms/step - accuracy: 0.6245 - loss: 0.9461 - val_
accuracy: 0.6654 - val_loss: 0.7249 - learning_rate: 0.0010
Epoch 3/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 8s 239ms/step - accuracy: 0.6794 - loss: 0.6412 - val_
accuracy: 0.5409 - val_loss: 0.7152 - learning_rate: 0.0010
Epoch 4/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 214ms/step - accuracy: 0.6942 - loss: 0.5893 - val_
accuracy: 0.6576 - val_loss: 0.7132 - learning_rate: 0.0010
Epoch 5/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 216ms/step - accuracy: 0.6945 - loss: 0.5942 - val_
accuracy: 0.5331 - val_loss: 0.7401 - learning_rate: 0.0010
Epoch 6/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 210ms/step - accuracy: 0.7241 - loss: 0.5488 - val_
accuracy: 0.5331 - val_loss: 0.8211 - learning_rate: 0.0010
Epoch 7/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 214ms/step - accuracy: 0.7236 - loss: 0.5523 - val_
accuracy: 0.5331 - val_loss: 0.8690 - learning_rate: 0.0010
Epoch 8/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 211ms/step - accuracy: 0.7406 - loss: 0.5589 - val_
accuracy: 0.5331 - val_loss: 0.9033 - learning_rate: 5.0000e-04
Epoch 9/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 219ms/step - accuracy: 0.7429 - loss: 0.5463 - val_
accuracy: 0.5331 - val_loss: 0.8189 - learning_rate: 5.0000e-04
Epoch 10/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 210ms/step - accuracy: 0.7288 - loss: 0.5543 - val_
accuracy: 0.5331 - val_loss: 0.8465 - learning_rate: 5.0000e-04
Epoch 11/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 209ms/step - accuracy: 0.7785 - loss: 0.5199 - val_
accuracy: 0.5331 - val_loss: 0.9286 - learning_rate: 2.5000e-04
Epoch 12/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 209ms/step - accuracy: 0.7597 - loss: 0.4948 - val_
accuracy: 0.5331 - val_loss: 1.1116 - learning_rate: 2.5000e-04
Epoch 13/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 211ms/step - accuracy: 0.7561 - loss: 0.4937 - val_
accuracy: 0.5331 - val_loss: 1.2812 - learning_rate: 2.5000e-04
Epoch 14/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 206ms/step - accuracy: 0.7825 - loss: 0.4434 - val_
accuracy: 0.5331 - val_loss: 1.3986 - learning_rate: 1.2500e-04
Epoch 15/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 215ms/step - accuracy: 0.8095 - loss: 0.4420 - val_
accuracy: 0.5331 - val_loss: 1.4874 - learning_rate: 1.2500e-04
Epoch 16/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 215ms/step - accuracy: 0.7988 - loss: 0.4576 - val_
accuracy: 0.5370 - val_loss: 1.3904 - learning_rate: 1.2500e-04
Epoch 17/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 208ms/step - accuracy: 0.7807 - loss: 0.4528 - val_
accuracy: 0.5409 - val_loss: 1.2923 - learning_rate: 6.2500e-05
Epoch 18/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 208ms/step - accuracy: 0.8186 - loss: 0.4230 - val_
accuracy: 0.5409 - val_loss: 1.2340 - learning_rate: 6.2500e-05
Epoch 19/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 209ms/step - accuracy: 0.8132 - loss: 0.4369 - val_
accuracy: 0.5759 - val_loss: 1.0517 - learning_rate: 6.2500e-05
Epoch 20/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 208ms/step - accuracy: 0.8193 - loss: 0.4265 - val_
accuracy: 0.6148 - val_loss: 0.9028 - learning_rate: 3.1250e-05
```

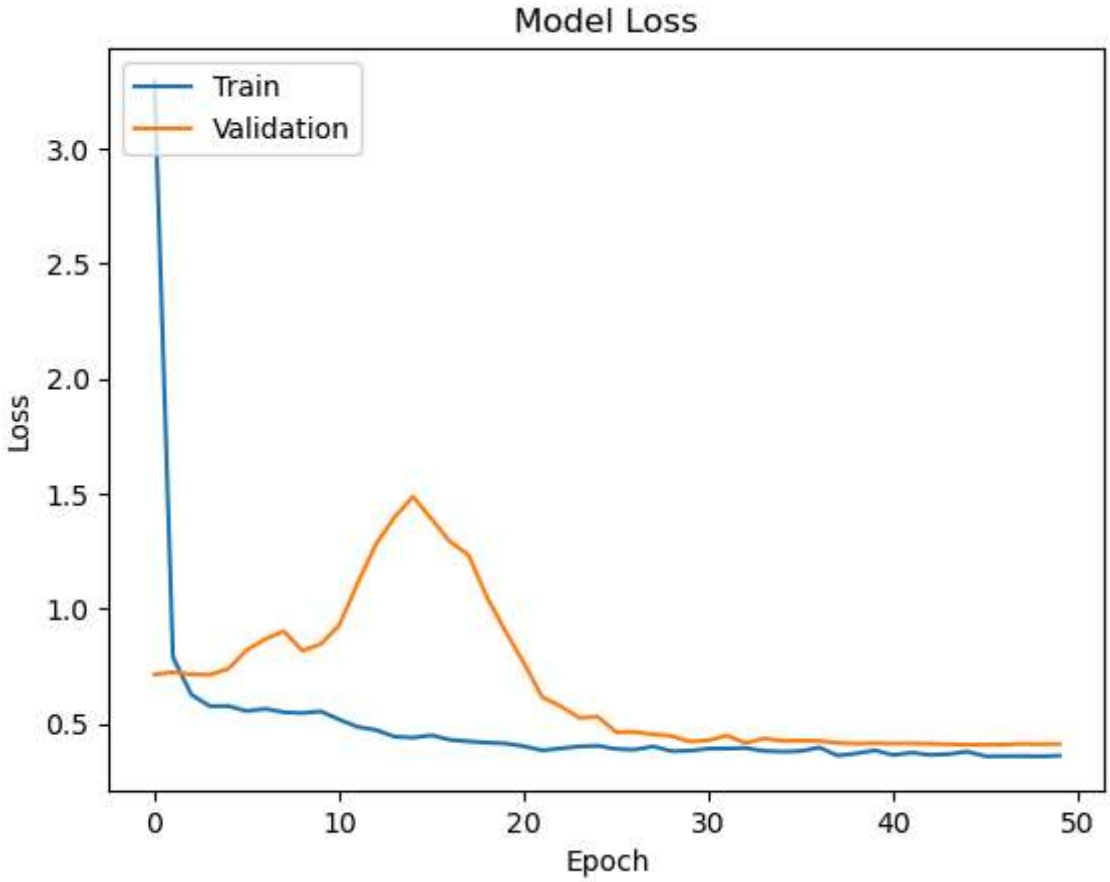
```
Epoch 21/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 208ms/step - accuracy: 0.8369 - loss: 0.3972 - val_
accuracy: 0.6732 - val_loss: 0.7651 - learning_rate: 3.1250e-05
Epoch 22/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 210ms/step - accuracy: 0.8383 - loss: 0.3802 - val_
accuracy: 0.7432 - val_loss: 0.6153 - learning_rate: 3.1250e-05
Epoch 23/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 211ms/step - accuracy: 0.8387 - loss: 0.3938 - val_
accuracy: 0.7549 - val_loss: 0.5771 - learning_rate: 3.1250e-05
Epoch 24/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 207ms/step - accuracy: 0.8394 - loss: 0.3925 - val_
accuracy: 0.7704 - val_loss: 0.5263 - learning_rate: 3.1250e-05
Epoch 25/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 210ms/step - accuracy: 0.8245 - loss: 0.4087 - val_
accuracy: 0.7704 - val_loss: 0.5312 - learning_rate: 3.1250e-05
Epoch 26/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 209ms/step - accuracy: 0.8502 - loss: 0.3944 - val_
accuracy: 0.7899 - val_loss: 0.4641 - learning_rate: 3.1250e-05
Epoch 27/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 206ms/step - accuracy: 0.8667 - loss: 0.3630 - val_
accuracy: 0.7899 - val_loss: 0.4650 - learning_rate: 3.1250e-05
Epoch 28/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 212ms/step - accuracy: 0.8238 - loss: 0.3927 - val_
accuracy: 0.7938 - val_loss: 0.4543 - learning_rate: 3.1250e-05
Epoch 29/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 210ms/step - accuracy: 0.8281 - loss: 0.3801 - val_
accuracy: 0.8016 - val_loss: 0.4476 - learning_rate: 3.1250e-05
Epoch 30/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 207ms/step - accuracy: 0.8490 - loss: 0.3730 - val_
accuracy: 0.8132 - val_loss: 0.4231 - learning_rate: 3.1250e-05
Epoch 31/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 208ms/step - accuracy: 0.8522 - loss: 0.3835 - val_
accuracy: 0.8249 - val_loss: 0.4292 - learning_rate: 3.1250e-05
Epoch 32/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 209ms/step - accuracy: 0.8446 - loss: 0.3842 - val_
accuracy: 0.8016 - val_loss: 0.4499 - learning_rate: 3.1250e-05
Epoch 33/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 213ms/step - accuracy: 0.8359 - loss: 0.4148 - val_
accuracy: 0.8132 - val_loss: 0.4167 - learning_rate: 3.1250e-05
Epoch 34/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 210ms/step - accuracy: 0.8462 - loss: 0.3731 - val_
accuracy: 0.8054 - val_loss: 0.4367 - learning_rate: 3.1250e-05
Epoch 35/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 215ms/step - accuracy: 0.8420 - loss: 0.3817 - val_
accuracy: 0.8171 - val_loss: 0.4263 - learning_rate: 3.1250e-05
Epoch 36/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 207ms/step - accuracy: 0.8433 - loss: 0.3898 - val_
accuracy: 0.8093 - val_loss: 0.4275 - learning_rate: 3.1250e-05
Epoch 37/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 8s 256ms/step - accuracy: 0.8439 - loss: 0.3793 - val_
accuracy: 0.8210 - val_loss: 0.4259 - learning_rate: 1.5625e-05
Epoch 38/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 9s 263ms/step - accuracy: 0.8468 - loss: 0.3618 - val_
accuracy: 0.8288 - val_loss: 0.4173 - learning_rate: 1.5625e-05
Epoch 39/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 9s 264ms/step - accuracy: 0.8143 - loss: 0.3974 - val_
accuracy: 0.8327 - val_loss: 0.4141 - learning_rate: 1.5625e-05
Epoch 40/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 8s 233ms/step - accuracy: 0.8396 - loss: 0.3859 - val_
accuracy: 0.8171 - val_loss: 0.4160 - learning_rate: 1.5625e-05
```

```
Epoch 41/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 221ms/step - accuracy: 0.8620 - loss: 0.3517 - val_
accuracy: 0.8288 - val_loss: 0.4143 - learning_rate: 1.5625e-05
Epoch 42/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 218ms/step - accuracy: 0.8446 - loss: 0.3761 - val_
accuracy: 0.8327 - val_loss: 0.4152 - learning_rate: 1.5625e-05
Epoch 43/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 210ms/step - accuracy: 0.8603 - loss: 0.3681 - val_
accuracy: 0.8249 - val_loss: 0.4138 - learning_rate: 1.0000e-05
Epoch 44/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 212ms/step - accuracy: 0.8545 - loss: 0.3593 - val_
accuracy: 0.8288 - val_loss: 0.4116 - learning_rate: 1.0000e-05
Epoch 45/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 214ms/step - accuracy: 0.8366 - loss: 0.3784 - val_
accuracy: 0.8288 - val_loss: 0.4097 - learning_rate: 1.0000e-05
Epoch 46/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 212ms/step - accuracy: 0.8414 - loss: 0.3689 - val_
accuracy: 0.8327 - val_loss: 0.4096 - learning_rate: 1.0000e-05
Epoch 47/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 213ms/step - accuracy: 0.8626 - loss: 0.3538 - val_
accuracy: 0.8327 - val_loss: 0.4102 - learning_rate: 1.0000e-05
Epoch 48/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 214ms/step - accuracy: 0.8652 - loss: 0.3459 - val_
accuracy: 0.8210 - val_loss: 0.4138 - learning_rate: 1.0000e-05
Epoch 49/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 8s 232ms/step - accuracy: 0.8523 - loss: 0.3621 - val_
accuracy: 0.8327 - val_loss: 0.4120 - learning_rate: 1.0000e-05
Epoch 50/50
33/33 ━━━━━━━━━━━━━━━━━━━━ 8s 245ms/step - accuracy: 0.8671 - loss: 0.3479 - val_
accuracy: 0.8327 - val_loss: 0.4133 - learning_rate: 1.0000e-05
```



In [14]:
```
# Plotting loss
plt.plot(history.history['loss'])
```

```
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



In [15]:
```
# Prediction example (optional)
predictions = model.predict(X_test)
print(predictions[:5])
```

```
264/264 ━━━━━━━━━━━━━━━━━━━━ 11s 42ms/step
[[0.13961528]
 [0.29620218]
 [0.01766438]
 [0.9715492 ]
 [0.253062  ]]
```