In [1]:
```
from transformers import CLIPProcessor, CLIPModel
import pandas as pd
from PIL import Image
import os
import torch
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dropout, BatchNormalization

import numpy as np
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt
```

```
G:\anaconda3\lib\site-packages\transformers\utils\generic.py:311: UserWarning: tor
ch.utils._pytree._register_pytree_node is deprecated. Please use torch.utils._pytr
ee.register_pytree_node instead.
  torch.utils._pytree._register_pytree_node(
```

In [2]:
```
file_path = "C:\\Users\\alan\\Medical Image Project\\combine_data\\BrEaST-Lesions-U
text_data = pd.read_excel(file_path)
```

Build NLP model

In [64]:
```
# filter out rows where 'Classification' is not 'benign' or 'malignant'
filtered_text_data = text_data[(text_data['Classification'] == 'benign') | (text_da

# Now, combine the relevant textual columns into a single text field per case
text_columns = [col for col in filtered_text_data.columns if col not in ['Image_fil
filtered_text_data['combined_text'] = filtered_text_data[text_columns].apply(lambda
```

```
C:\Users\alan\AppData\Local\Temp\ipykernel_11740\3001372267.py:6: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_text_data['combined_text'] = filtered_text_data[text_columns].apply(lam
bda x: ' '.join(x.dropna().astype(str)), axis=1)
```

In [65]:
```
model_name = "openai/clip-vit-base-patch32"
processor = CLIPProcessor.from_pretrained(model_name)
model = CLIPModel.from_pretrained(model_name)

text_inputs = processor(text=filtered_text_data["combined_text"].tolist(), padding=
```

In [66]:
```
text_embeddings = model.get_text_features(**text_inputs)
```

In [67]:
```
def get_image_embedding(image_path, processor, model):
    # Load and process the image
    image = Image.open(image_path).convert("RGB")
    inputs = processor(images=image, return_tensors="pt")

    # Generate embedding
    with torch.no_grad():  # Ensure no gradients are calculated
        image_embedding = model.get_image_features(**inputs)
```

```
    return image_embedding
```

In [68]:
```
# add label
labels = filtered_text_data['Classification'].values
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)  # Converts labels to numerica
```

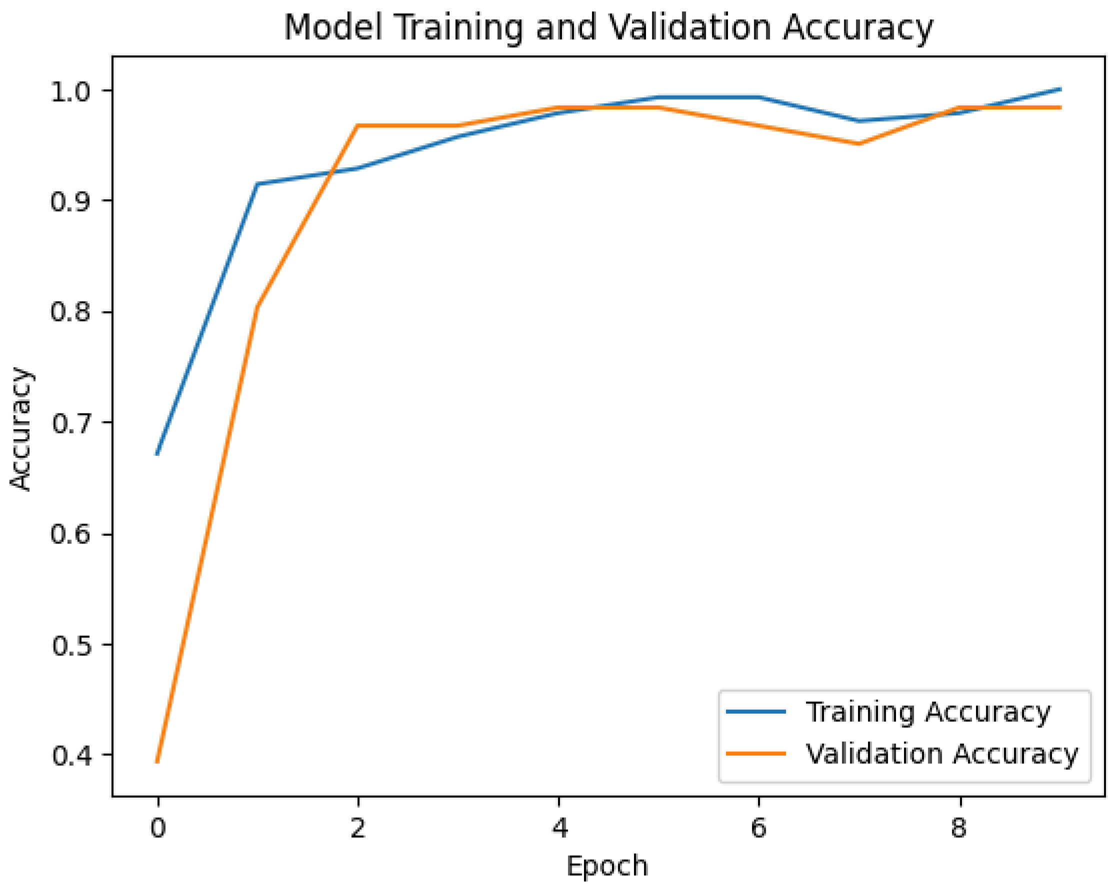
In [69]:
```
# Convert it to a NumPy array
X = text_embeddings.detach().numpy()
```

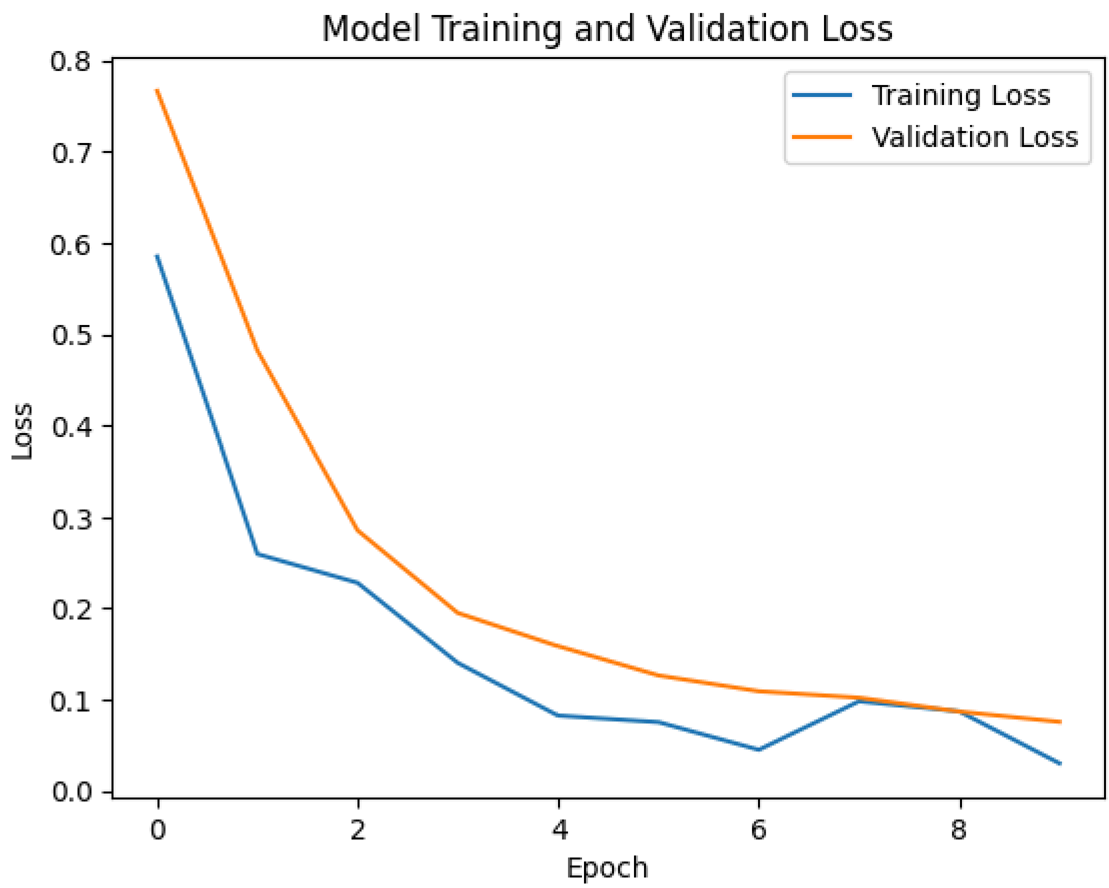
In [70]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, encoded_labels, test_size=0.
```

In [71]:
```
model = Sequential([
    Dense(128, activation='relu', input_dim=X_train.shape[1]),
    Dropout(0.2),                 # Dropout layer for regularization
    Dense(64, activation='relu'),
    BatchNormalization(),         # BatchNormalization layer for normalization
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

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

# Train the model with validation data
history = model.fit(X_train, y_train, epochs=10, batch_size=10, validation_split=0.

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)

print(f"Test Loss: {test_loss}\nTest Accuracy: {test_accuracy}")
```

```
Epoch 1/10
14/14 [==============================] - 1s 13ms/step - loss: 0.5851 - accuracy:
0.6714 - val_loss: 0.7666 - val_accuracy: 0.3934
Epoch 2/10
14/14 [==============================] - 0s 4ms/step - loss: 0.2597 - accuracy: 0.
9143 - val_loss: 0.4822 - val_accuracy: 0.8033
Epoch 3/10
14/14 [==============================] - 0s 4ms/step - loss: 0.2281 - accuracy: 0.
9286 - val_loss: 0.2856 - val_accuracy: 0.9672
Epoch 4/10
14/14 [==============================] - 0s 4ms/step - loss: 0.1404 - accuracy: 0.
9571 - val_loss: 0.1951 - val_accuracy: 0.9672
Epoch 5/10
14/14 [==============================] - 0s 4ms/step - loss: 0.0826 - accuracy: 0.
9786 - val_loss: 0.1586 - val_accuracy: 0.9836
Epoch 6/10
14/14 [==============================] - 0s 4ms/step - loss: 0.0754 - accuracy: 0.
9929 - val_loss: 0.1266 - val_accuracy: 0.9836
Epoch 7/10
14/14 [==============================] - 0s 4ms/step - loss: 0.0451 - accuracy: 0.
9929 - val_loss: 0.1093 - val_accuracy: 0.9672
Epoch 8/10
14/14 [==============================] - 0s 4ms/step - loss: 0.0983 - accuracy: 0.
9714 - val_loss: 0.1022 - val_accuracy: 0.9508
Epoch 9/10
14/14 [==============================] - 0s 4ms/step - loss: 0.0873 - accuracy: 0.
9786 - val_loss: 0.0871 - val_accuracy: 0.9836
Epoch 10/10
14/14 [==============================] - 0s 5ms/step - loss: 0.0303 - accuracy: 1.
0000 - val_loss: 0.0758 - val_accuracy: 0.9836
2/2 [==============================] - 0s 3ms/step - loss: 0.0328 - accuracy: 1.00
00
Test Loss: 0.03280746564269066
Test Accuracy: 1.0
```

In [72]:
```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()

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





In [73]:
```
model.save('C:/Users/alan/Medical Image Project/CLIP_NLP.h5')
```

In [74]:
```
from tensorflow.keras.models import load_model
dir = 'C:/Users/alan/Medical Image Project'
nlp_model = load_model(dir +'/CLIP_NLP.h5')
```

In [75]:
```
text_predictions = nlp_model.predict(X_test)
```

```
2/2 [==============================] - 0s 2ms/step
```

Build Image Model

In [30]:
```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing import image
import numpy as np
import os

base_model = VGG16(weights='imagenet', include_top=False)  # Load VGG16 without the
base_path = "C:\\Users\\alan\\Medical Image Project\\combine_data"
def get_cnn_features(img_path):
    img = image.load_img(img_path, target_size=(224, 224))  # Resize image
    img_array = image.img_to_array(img)
    expanded_img_array = np.expand_dims(img_array, axis=0)
    preprocessed_img = preprocess_input(expanded_img_array)  # Preprocess the image
    features = base_model.predict(preprocessed_img)
    flattened_features = features.flatten()  # Flatten the features to a 1D array
    return flattened_features

cnn_embeddings =[]
for filename in filtered_text_data['Image_filename']:
    image_path = os.path.join(base_path, filename)
    embedding = get_cnn_features(image_path)
    cnn_embeddings.append(embedding)
```

```
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 102ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 92ms/step
```

```
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 114ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 118ms/step
```

```
1/1 [==============================] - 0s 102ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 114ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 92ms/step
```

```
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 102ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 102ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 102ms/step
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 106ms/step
```

In [31]:

```
# Assuming cnn_embeddings is a numpy array of your embeddings and encoded_labels ar
X_train, X_test, y_train, y_test = train_test_split(cnn_embeddings, encoded_labels,
# Convert X_train and y_train to NumPy arrays if they're not already
```

```
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
```

In [32]:
```
model = Sequential([
    Dense(512, activation='relu', input_shape=(25088,)),  # Adjust the layer sizes
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Use 'sigmoid' for binary classification
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',  # Use 'binary_crossentropy' for binary c
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_16 (Dense)            (None, 512)               12845568

 dropout_8 (Dropout)         (None, 512)               0

 dense_17 (Dense)            (None, 256)               131328

 dropout_9 (Dropout)         (None, 256)               0

 dense_18 (Dense)            (None, 1)                 257

=================================================================
Total params: 12,977,153
Trainable params: 12,977,153
Non-trainable params: 0
_________________________________________________________________
```

In [33]:
```
history = model.fit(X_train, y_train,
                    epochs=12,
                    batch_size=16,
                    validation_split=0.2)  # Using part of the training data for va
```

```
Epoch 1/12
10/10 [==============================] - 2s 124ms/step - loss: 9.9782 - accuracy:
0.5625 - val_loss: 10.2540 - val_accuracy: 0.5610
Epoch 2/12
10/10 [==============================] - 1s 112ms/step - loss: 10.0610 - accuracy:
0.7437 - val_loss: 11.2402 - val_accuracy: 0.6098
Epoch 3/12
10/10 [==============================] - 1s 114ms/step - loss: 13.4882 - accuracy:
0.6938 - val_loss: 6.1236 - val_accuracy: 0.6098
Epoch 4/12
10/10 [==============================] - 1s 110ms/step - loss: 8.0270 - accuracy:
0.7063 - val_loss: 15.0287 - val_accuracy: 0.6341
Epoch 5/12
10/10 [==============================] - 1s 107ms/step - loss: 6.3014 - accuracy:
0.8062 - val_loss: 9.3737 - val_accuracy: 0.6341
Epoch 6/12
10/10 [==============================] - 1s 112ms/step - loss: 3.2962 - accuracy:
0.8750 - val_loss: 11.3189 - val_accuracy: 0.5610
Epoch 7/12
10/10 [==============================] - 1s 117ms/step - loss: 2.7004 - accuracy:
0.8687 - val_loss: 17.5292 - val_accuracy: 0.6098
Epoch 8/12
10/10 [==============================] - 1s 105ms/step - loss: 2.6028 - accuracy:
0.8875 - val_loss: 6.8667 - val_accuracy: 0.6098
Epoch 9/12
10/10 [==============================] - 1s 117ms/step - loss: 2.7840 - accuracy:
0.9187 - val_loss: 6.8324 - val_accuracy: 0.7317
Epoch 10/12
10/10 [==============================] - 1s 106ms/step - loss: 1.7091 - accuracy:
0.9438 - val_loss: 5.6002 - val_accuracy: 0.6098
Epoch 11/12
10/10 [==============================] - 1s 117ms/step - loss: 1.8911 - accuracy:
0.9312 - val_loss: 14.0695 - val_accuracy: 0.5854
Epoch 12/12
10/10 [==============================] - 1s 113ms/step - loss: 0.4353 - accuracy:
0.9563 - val_loss: 9.2975 - val_accuracy: 0.6585
```

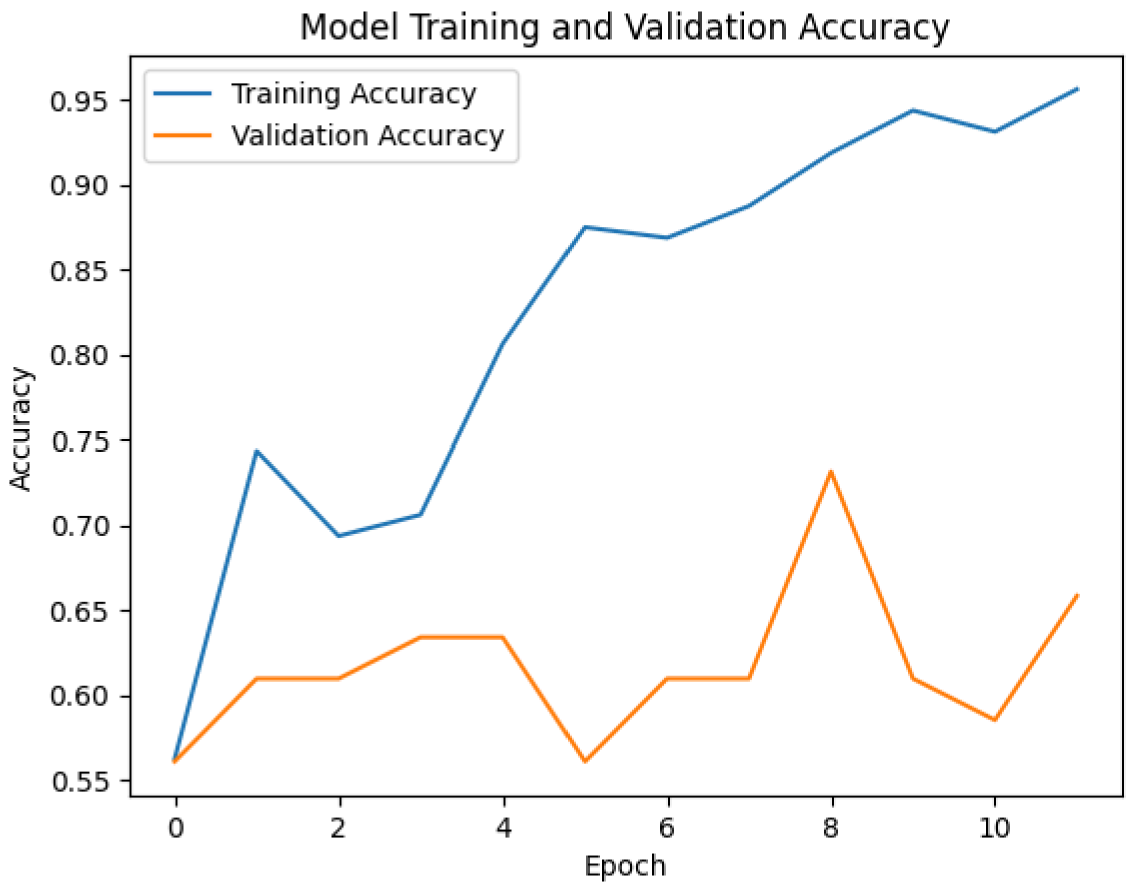
In [34]:

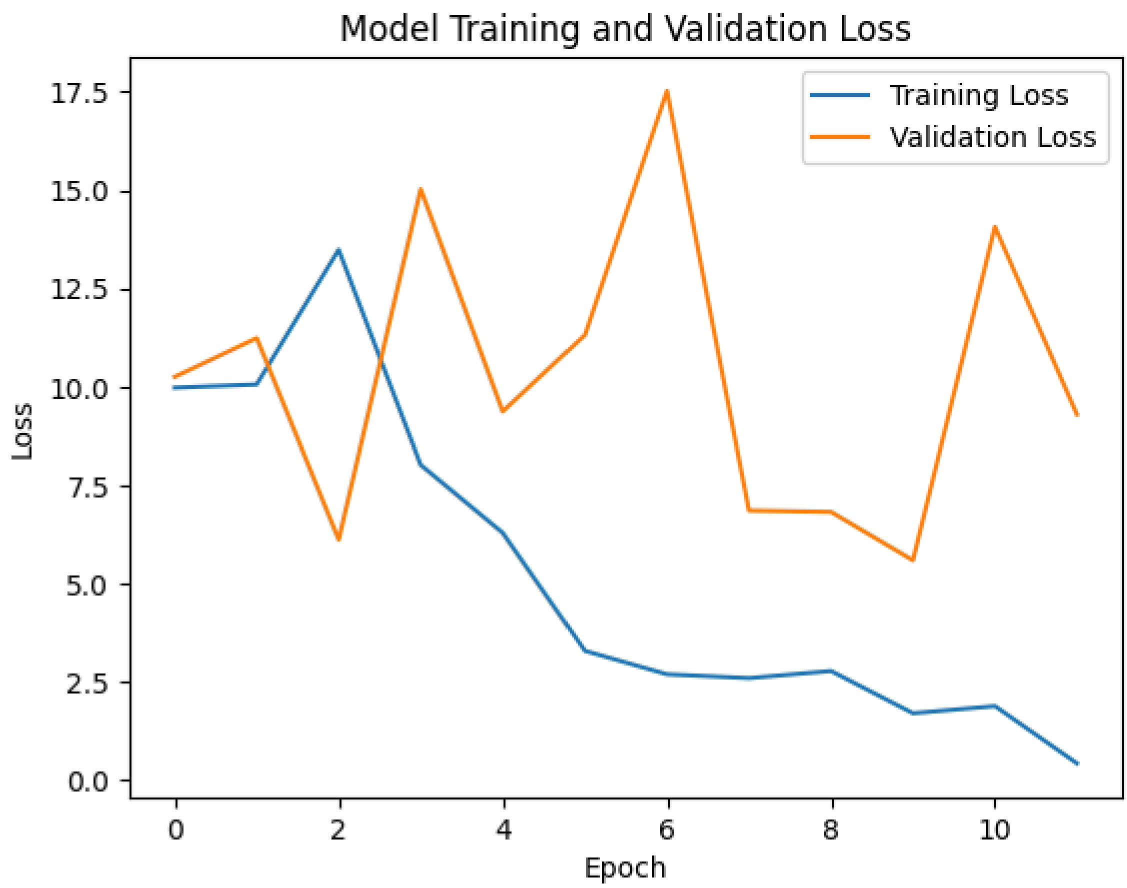
```python
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")
```

```
2/2 [==============================] - 0s 9ms/step - loss: 6.4449 - accuracy: 0.78
43
Test Accuracy: 0.7843137383460999
```

In [35]:

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

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





In [36]:
```
model.save('C:/Users/alan/Medical Image Project/Simple_CNN.h5')
```

In [37]:
```
dir = 'C:/Users/alan/Medical Image Project'
cnn_model = load_model(dir +'/Simple_CNN.h5')
```

In [38]:
```
image_predictions = cnn_model.predict(X_test)
```

```
2/2 [==============================] - 0s 7ms/step
```

In [48]:
```
# Ensure predictions are in the same scale/format, e.g., probabilities
# Example: weighted average favoring one set of predictions
weight_text = 0.6
weight_image = 0.4
final_predictions = (text_predictions * weight_text) + (image_predictions * weight_
```

In [49]:
```
# Assuming a threshold of 0.5 for binary classification
threshold = 0.5
predicted_labels = np.where(final_predictions > threshold, 1, 0)
```

In [50]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Evaluate metrics
accuracy = accuracy_score(y_test, predicted_labels)
precision = precision_score(y_test, predicted_labels)
recall = recall_score(y_test, predicted_labels)
f1 = f1_score(y_test, predicted_labels)

# Print the evaluation metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

# Confusion Matrix
cm = confusion_matrix(y_test, predicted_labels)
print("Confusion Matrix:")
print(cm)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
Confusion Matrix:
[[35  0]
 [ 0 16]]
```

In [ ]:

In [ ]: