MSDS 422 Group 6
Aaron Chen, Elvis Matos, Reese Mayer, Nikhil Prabhu

Module 8 Assignment 1: Dogs vs. Cats Redux: Kernel Edition

Introduction

Our research leverages advanced machine learning techniques, specifically Convolutional Neural Networks (CNNs), to classify images as either dogs or cats. This task is challenging for models due to the diversity in breeds, poses, lighting, and backgrounds, despite being simple for humans. Automating this classification has practical applications in social media, search engines, animal monitoring, and veterinary diagnostics. By conducting thorough Exploratory Data Analysis (EDA), building and tuning multiple CNN models, and evaluating them with cross-validation and performance metrics, the aim is to develop a robust model. The goal is to demonstrate CNNs' effectiveness in handling complex visual tasks and contribute to the broader field of image recognition.

Cross-Validation Design

We employed a stratified 5-fold cross-validation design to evaluate our logistic regression model for classifying images as dogs or cats. This method splits the dataset into five equal folds, maintaining the class distribution in each fold. During each iteration, four folds were used for training, and one fold was used for validation, with each fold serving as the validation set once. This approach ensures a robust and unbiased evaluation of model performance. The average accuracy across all folds was computed, providing a reliable estimate of the model's classification accuracy. This design helps mitigate overfitting and ensures the model's generalizability.

Exploratory Data Analysis (EDA)

Our Exploratory Data Analysis (EDA) confirmed a balanced dataset with 961 dog images and 1,039 cat images. Both training and validation sets maintained this balance, with approximately 48% dog images and 52% cat images. These proportions were visualized using bar and pie charts. Ensuring this balance is essential for training unbiased and robust Convolutional Neural Network (CNN) models. The EDA also helped identify potential anomalies and ensured proper data preprocessing, providing a solid foundation for model development.

Model Building and Hyperparameter Tuning:

The three models we chose to highlight were the simple CNN model, the dropout CNN model, and the batch normalization CNN model. After creating the models, we plotted the ROC and precision-recall graphs to visually compare training vs. validation performance. The best-performing models were the simple CNN model and the batch normalization CNN model, with accuracy scores of 0.80 and 0.87, respectively. Since the batch normalization CNN model achieved the highest accuracy of 0.87, we used it to make predictions on the testing set.

Output and Metrics Evaluation

Images were loaded into the best performing model to predict probabilities. These probabilities are then converted to binary predictions, which are added to the DataFrame as a new column, predict_label. The output shows that y_pred has been created with 391 batches processed, indicating predictions for all images.

When the data was uploaded to Kaggle, an unusual accuracy score of 114% was obtained. This anomaly occurred because a subset of the data was created to run the model. Due to limited processing power, the entire dataset could not be run on laptops, so a sample batch was created. This smaller batch can skew results, leading to underfitting. To address this issue in future steps, increasing the sample size could achieve a more accurate representation, resulting in a better accuracy score by reducing biases in the dataset.

Appendix:

NikhilPrabhu2025

CNN3.csv Complete (after deadline) · 3d ago	**11.45319**	**11.45319**

Assignment_8_Dogs_and_Cats

May 16, 2024

[263]:
```
import pandas as pd

import cv2
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split, LeaveOneOut,␣
 ↪cross_val_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.preprocessing.image import load_img, img_to_array

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import BatchNormalization
from sklearn.metrics import precision_score
```

[174]:
```
image_dir = "/Users/nikhil/Desktop/dogs-vs-cats-redux-kernels-edition/train"

filenames = os.listdir(image_dir)
labels = [x.split(".")[0] for x in filenames]

df_train = pd.DataFrame({"filename": filenames, "label": labels})

df_train.head()
```

[174]:
```
          filename label
0   dog.8011.jpg   dog
1   cat.5077.jpg   cat
2   dog.7322.jpg   dog
3   cat.2718.jpg   cat
4  cat.10151.jpg   cat
```

[175]:
```
df_train.count()
```

[175]:
```
filename    25000
label       25000
dtype: int64
```

1 1.) Cross Validation Design

[176]:
```
file_names = df_train['filename'].tolist()

train_files, val_files = train_test_split(file_names, test_size=0.2,␣
 ↪random_state=42)

train_set = df_train[df_train['filename'].isin(train_files)]
val_set = df_train[df_train['filename'].isin(val_files)]


X_train = train_set.drop('label', axis=1)
y_train = train_set['label']

X_val = val_set.drop('label', axis=1)
y_val = val_set['label']
## Remember to try Kfold and Logistic Regression
```

[201]:
```
data = df_train.sample(frac=0.1, random_state=42)  # Change the fraction as␣
 ↪needed

X = data['filename']
y = data['label']

n_splits = 5
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)
accuracy_scores = []

def load_images(filenames, directory, target_size=(150, 150)):
    images = []
    for file in filenames:
        img_path = os.path.join(directory, file)
        img = load_img(img_path, target_size=target_size)
```

```
        img_array = img_to_array(img)
        images.append(img_array)
    return np.array(images)
for train_index, val_index in skf.split(X, y):
    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    X_train_images = load_images(X_train, image_dir)
    X_val_images = load_images(X_val, image_dir)

    X_train_flat = X_train_images.reshape(X_train_images.shape[0], -1)
    X_val_flat = X_val_images.reshape(X_val_images.shape[0], -1)

    model = LogisticRegression(max_iter=1000)
    model.fit(X_train_flat, y_train)

    y_pred = model.predict(X_val_flat)
    accuracy = accuracy_score(y_val, y_pred)
    accuracy_scores.append(accuracy)
    print(f'Fold accuracy: {accuracy}')

mean_accuracy = np.mean(accuracy_scores)
print(f'Mean accuracy: {mean_accuracy}')
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Fold accuracy: 0.57
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Fold accuracy: 0.564
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Fold accuracy: 0.568
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Fold accuracy: 0.522
Fold accuracy: 0.506
Mean accuracy: 0.546
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

2 EDA

[203]:
```
X_train
```

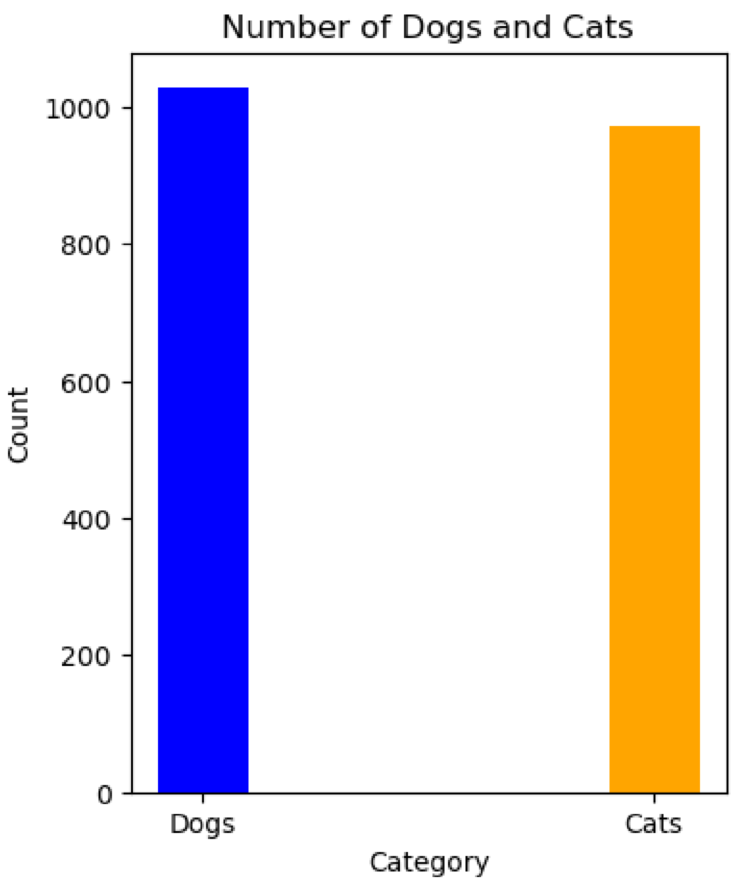
[203]:
```
6868      cat.1474.jpg
24016    dog.11287.jpg
9668      dog.8276.jpg
13640     cat.7227.jpg
14018     cat.2997.jpg
               …
907      dog.11705.jpg
4104     dog.10810.jpg
15308     dog.8290.jpg
13238    dog.12298.jpg
21567     cat.8996.jpg
Name: filename, Length: 2000, dtype: object
```

[204]:
```
y_train
```

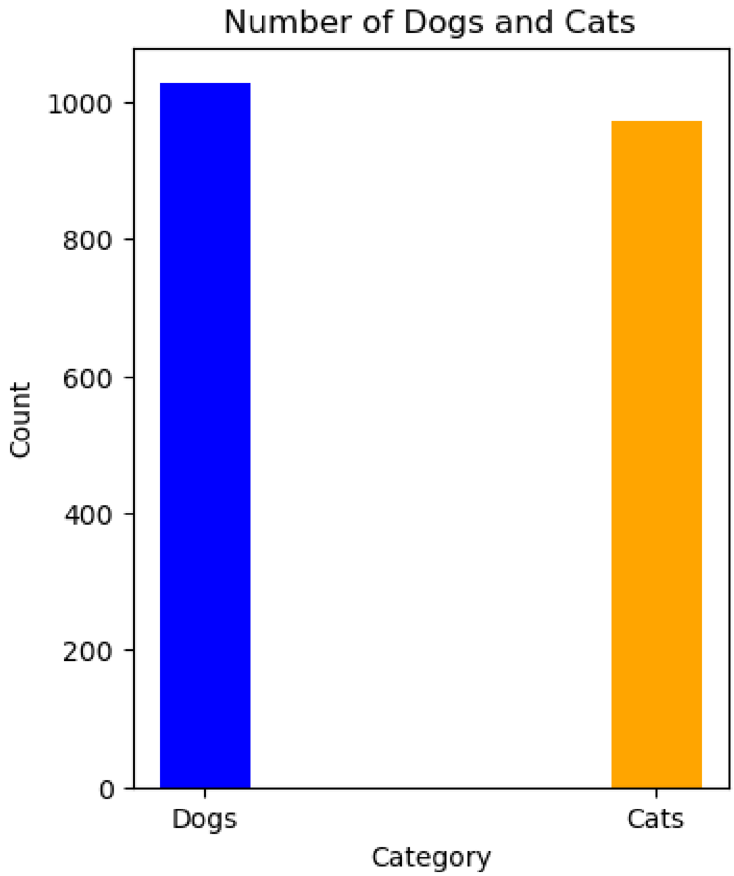
[204]:
```
6868     cat
24016    dog
9668     dog
13640    cat
14018    cat
        …
907      dog
4104     dog
15308    dog
13238    dog
21567    cat
Name: label, Length: 2000, dtype: object
```

[205]:
```
dog_count = y_train.str.contains('dog').sum()
cat_count = y_train.str.contains('cat').sum()

# Plotting the bar graph
plt.figure(figsize=(4, 5))
plt.bar(['Dogs', 'Cats'], [dog_count, cat_count], color=['blue', 'orange'],␣
 ↪width=0.2)
plt.title('Number of Dogs and Cats')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
print("Number of dogs:", dog_count)
print("Number of cats:", cat_count)

dog_count2 = y_val.str.contains('dog').sum()
cat_count2 = y_val.str.contains('cat').sum()

# Plotting the bar graph
plt.figure(figsize=(4, 5))
```

```
plt.bar(['Dogs', 'Cats'], [dog_count, cat_count], color=['blue', 'orange'],
    width=0.2)
plt.title('Number of Dogs and Cats')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
print("Number of dogs:", dog_count2)
print("Number of cats:", cat_count2)
```



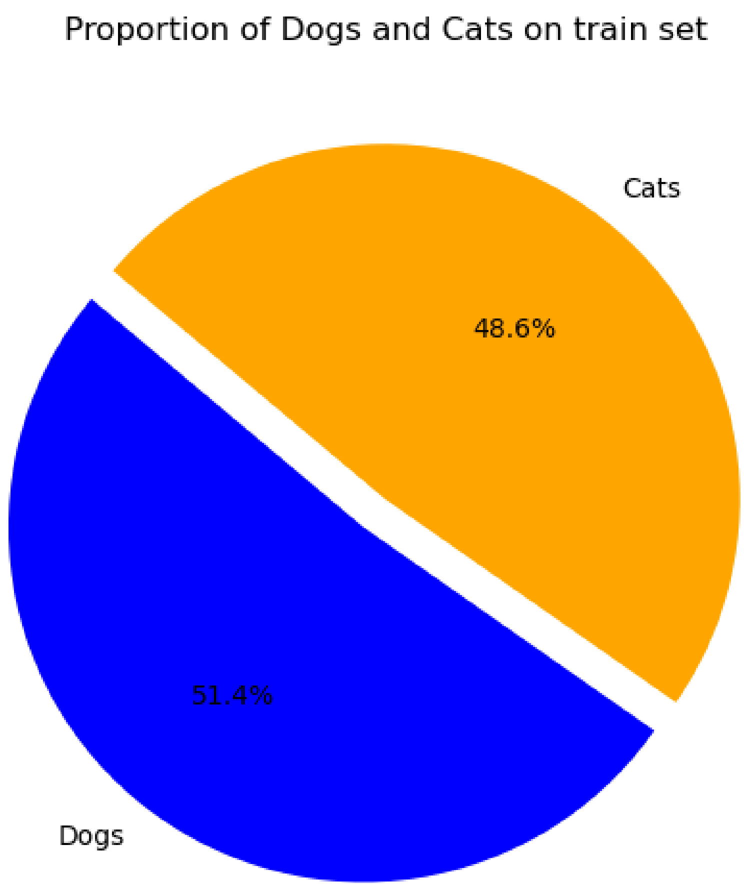

```
Number of dogs: 1028
Number of cats: 972
```

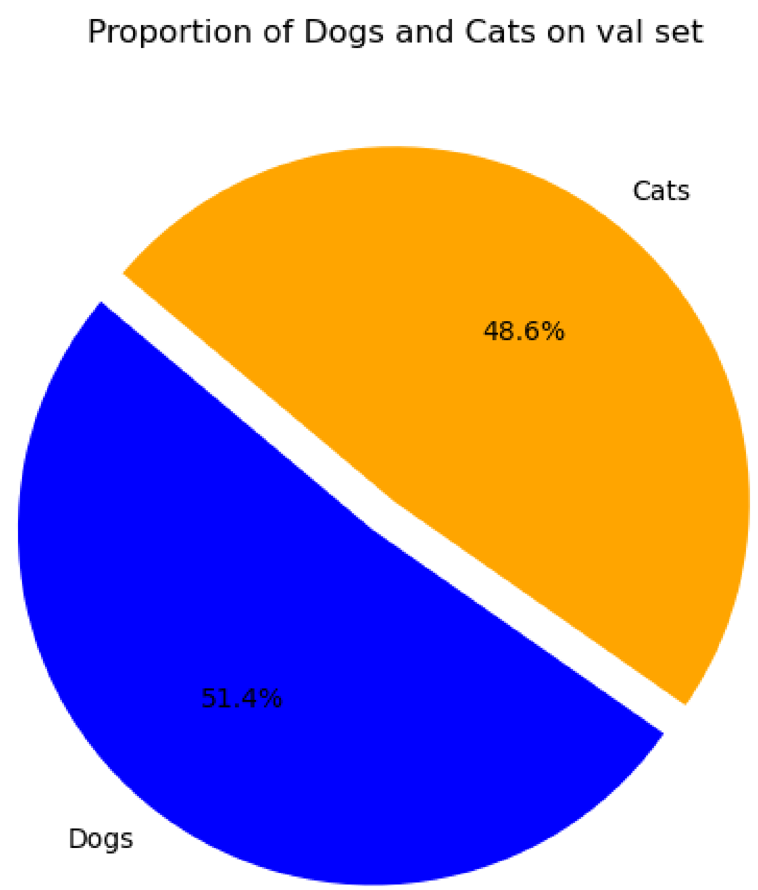



```
Number of dogs: 257
Number of cats: 243
```

[206]:
```
labels = ['Dogs', 'Cats']
sizes = [dog_count, cat_count]
colors = ['blue', 'orange']
explode = (0.1, 0)


plt.figure(figsize=(8, 6))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.
    ↪1f%%', startangle=140)
plt.title('Proportion of Dogs and Cats on train set')
plt.show()

labels = ['Dogs', 'Cats']
sizes = [dog_count2, cat_count2]
colors = ['blue', 'orange']
explode = (0.1, 0)
```

```python
plt.figure(figsize=(8, 6))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.
 ↪1f%%', startangle=140)
plt.title('Proportion of Dogs and Cats on val set')
plt.show()
```





3 CNN Models

[241]:
```
model_simple = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

[242]:
```
from tensorflow.keras.optimizers import Adam
```

```
model_simple.compile(optimizer=Adam(learning_rate=0.001),␣
↪loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history_simple = model_simple.fit(X_train_images, y_train_encoded, epochs=20,␣
↪batch_size=32,
                                  validation_data=(X_val_images, y_val_encoded),
                                  callbacks=[EarlyStopping(patience=3)])
```

```
Epoch 1/20
63/63                10s 156ms/step -
accuracy: 0.5285 - loss: 56.6451 - val_accuracy: 0.5620 - val_loss: 0.6814
Epoch 2/20
63/63                10s 161ms/step -
accuracy: 0.6614 - loss: 0.6261 - val_accuracy: 0.5640 - val_loss: 0.7244
Epoch 3/20
63/63                10s 158ms/step -
accuracy: 0.7168 - loss: 0.5402 - val_accuracy: 0.5880 - val_loss: 0.7242
Epoch 4/20
63/63                10s 161ms/step -
accuracy: 0.7043 - loss: 0.5638 - val_accuracy: 0.5660 - val_loss: 0.7063
```

[244]:
```
model_dropout = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

[245]:
```
from tensorflow.keras.optimizers import Adam

model_dropout.compile(optimizer=Adam(learning_rate=0.001),␣
↪loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history_dropout = model_dropout.fit(X_train_images, y_train_encoded, epochs=20,␣
↪batch_size=32,
                                  validation_data=(X_val_images, y_val_encoded),
```

```
                                  callbacks=[EarlyStopping(patience=3)])
```

```
Epoch 1/20
63/63               11s 166ms/step -
accuracy: 0.4952 - loss: 32.5069 - val_accuracy: 0.5220 - val_loss: 0.6927
Epoch 2/20
63/63               11s 175ms/step -
accuracy: 0.5212 - loss: 0.6966 - val_accuracy: 0.5180 - val_loss: 0.6923
Epoch 3/20
63/63               10s 166ms/step -
accuracy: 0.5537 - loss: 0.6871 - val_accuracy: 0.5500 - val_loss: 0.6755
Epoch 4/20
63/63               10s 166ms/step -
accuracy: 0.5795 - loss: 0.6770 - val_accuracy: 0.5040 - val_loss: 0.6919
Epoch 5/20
63/63               11s 168ms/step -
accuracy: 0.5413 - loss: 0.6874 - val_accuracy: 0.5160 - val_loss: 0.6943
Epoch 6/20
63/63               11s 169ms/step -
accuracy: 0.5364 - loss: 0.6706 - val_accuracy: 0.5460 - val_loss: 0.6916
```

[248]:
```
model_batchnorm = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

[249]:
```
from tensorflow.keras.optimizers import Adam

model_batchnorm.compile(optimizer=Adam(learning_rate=0.001),
 ↪loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history_batchnorm = model_batchnorm.fit(X_train_images, y_train_encoded,
 ↪epochs=20, batch_size=32,
                                  validation_data=(X_val_images, y_val_encoded),
                                  callbacks=[EarlyStopping(patience=3)])
```

```
Epoch 1/20
```

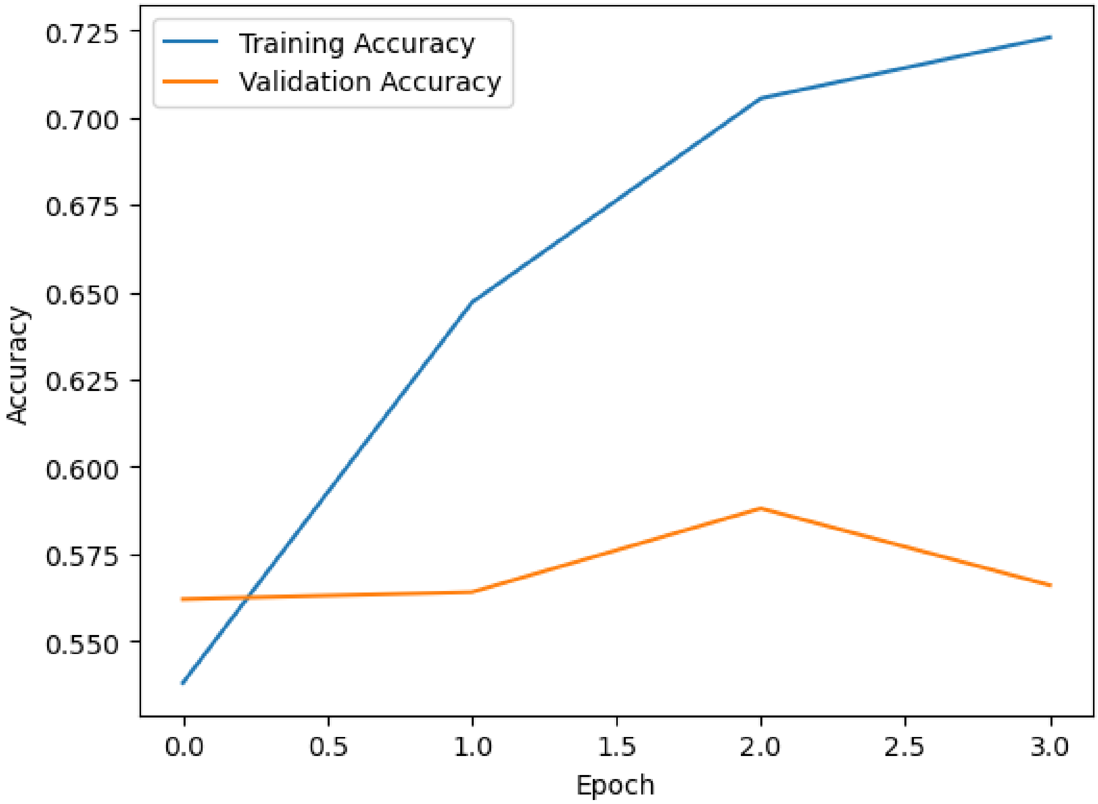
```
63/63                18s 279ms/step -
accuracy: 0.5733 - loss: 3.3558 - val_accuracy: 0.5120 - val_loss: 0.7315
Epoch 2/20
63/63                17s 273ms/step -
accuracy: 0.6932 - loss: 0.6617 - val_accuracy: 0.6600 - val_loss: 0.9443
Epoch 3/20
63/63                17s 273ms/step -
accuracy: 0.7890 - loss: 0.4335 - val_accuracy: 0.6620 - val_loss: 0.6897
Epoch 4/20
63/63                17s 272ms/step -
accuracy: 0.8608 - loss: 0.3036 - val_accuracy: 0.6840 - val_loss: 0.7000
Epoch 5/20
63/63                17s 272ms/step -
accuracy: 0.9198 - loss: 0.2054 - val_accuracy: 0.6440 - val_loss: 2.6688
Epoch 6/20
63/63                18s 286ms/step -
accuracy: 0.9576 - loss: 0.1283 - val_accuracy: 0.6620 - val_loss: 1.0419
```

[264]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
 ↪roc_auc_score

# Evaluate model performance on training and validation data
loss_train, accuracy_train = model_simple.evaluate(X_train_images,␣
 ↪y_train_encoded)
loss_val, accuracy_val = model_simple.evaluate(X_val_images, y_val_encoded)

print("Training Loss:", loss_train)
print("Training Accuracy:", accuracy_train)
print("Validation Loss:", loss_val)
print("Validation Accuracy:", accuracy_val)

# Plot training history
plt.plot(history_simple.history['accuracy'], label='Training Accuracy')
plt.plot(history_simple.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

y_train_pred = np.where(model_simple.predict(X_train_images) > 0.5, 1, 0)
y_val_pred = np.where(model_simple.predict(X_val_images) > 0.5, 1, 0)

conf_matrix_train = confusion_matrix(y_train_encoded, y_train_pred)
conf_matrix_val = confusion_matrix(y_val_encoded, y_val_pred)

print('Confusion Matrix - Training:')
```

```python
print(conf_matrix_train)
print('Confusion Matrix - Validation:')
print(conf_matrix_val)

print('Classification Report - Training:')
print(classification_report(y_train_encoded, y_train_pred))
print('Classification Report - Validation:')
print(classification_report(y_val_encoded, y_val_pred))


fpr_train, tpr_train, _ = roc_curve(y_train_encoded, model_simple.
 ↪predict(X_train_images))
fpr_val, tpr_val, _ = roc_curve(y_val_encoded, model_simple.
 ↪predict(X_val_images))

auc_train = roc_auc_score(y_train_encoded, model_simple.predict(X_train_images))
auc_val = roc_auc_score(y_val_encoded, model_simple.predict(X_val_images))

plt.plot(fpr_train, tpr_train, label=f'Training ROC Curve (AUC = {auc_train:.
 ↪2f})')
plt.plot(fpr_val, tpr_val, label=f'Validation ROC Curve (AUC = {auc_val:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Model Simple')
plt.legend()
plt.show()
thresholds = np.linspace(0, 1, 100)
precisions = []
for threshold in thresholds:
    y_val_pred_thresholded = (model_simple.predict(X_val_images) > threshold).
 ↪astype(int)
    precisions.append(precision_score(y_val_encoded, y_val_pred_thresholded))

# Plot precision graph
plt.plot(thresholds, precisions, label='Precision', color='blue')
plt.xlabel('Threshold')
plt.ylabel('Precision')
plt.title('Precision vs Threshold')
plt.legend()
plt.grid(True)
plt.show()
```

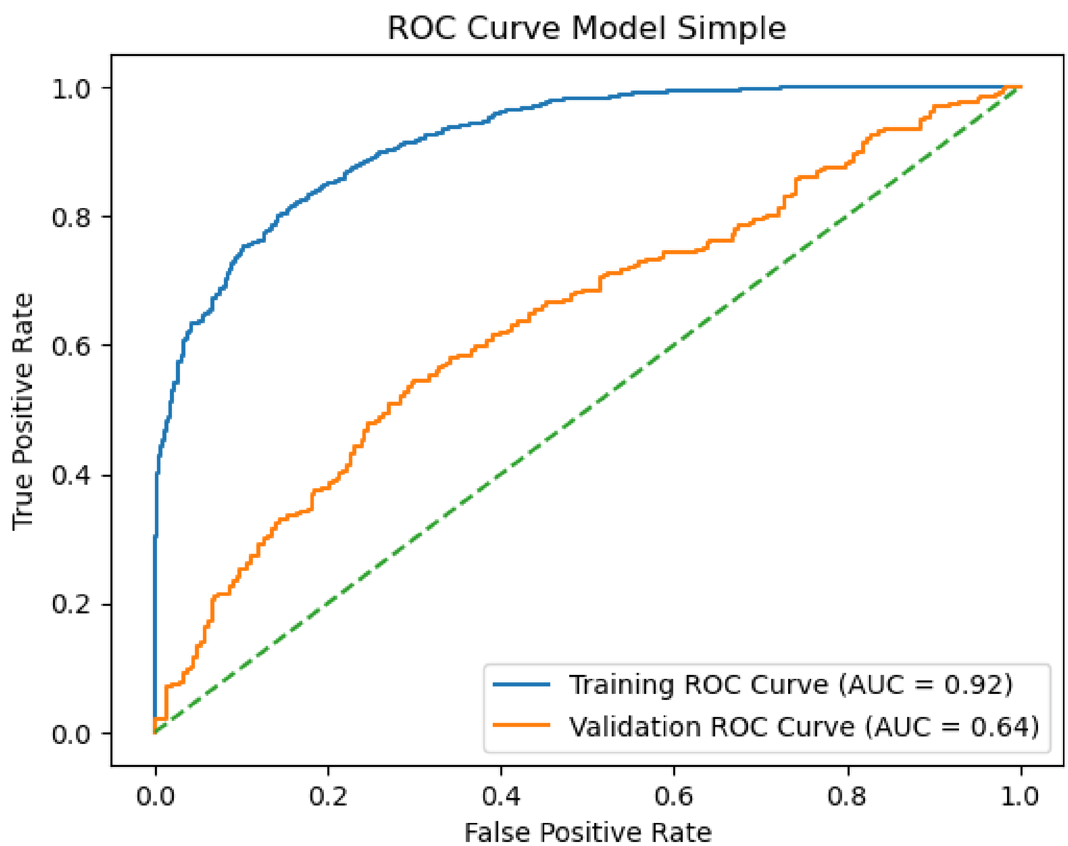
```
63/63              3s 41ms/step -
accuracy: 0.7900 - loss: 0.4520
16/16              1s 39ms/step -
accuracy: 0.5981 - loss: 0.6660
```

```
Training Loss: 0.4397807717323303
Training Accuracy: 0.8044999837875366
Validation Loss: 0.7062530517578125
Validation Accuracy: 0.5659999847412109
```




```
63/63              3s 42ms/step
16/16              1s 38ms/step
Confusion Matrix - Training:
[[656 316]
 [ 75 953]]
Confusion Matrix - Validation:
[[ 91 152]
 [ 65 192]]
Classification Report - Training:
              precision    recall  f1-score   support

           0       0.90      0.67      0.77       972
           1       0.75      0.93      0.83      1028

    accuracy                           0.80      2000
   macro avg       0.82      0.80      0.80      2000
weighted avg       0.82      0.80      0.80      2000
```

```
Classification Report - Validation:
              precision    recall  f1-score   support

           0       0.58      0.37      0.46       243
           1       0.56      0.75      0.64       257

    accuracy                           0.57       500
   macro avg       0.57      0.56      0.55       500
weighted avg       0.57      0.57      0.55       500

63/63                 2s 39ms/step
16/16                 1s 38ms/step
63/63                 3s 40ms/step
16/16                 1s 38ms/step
```

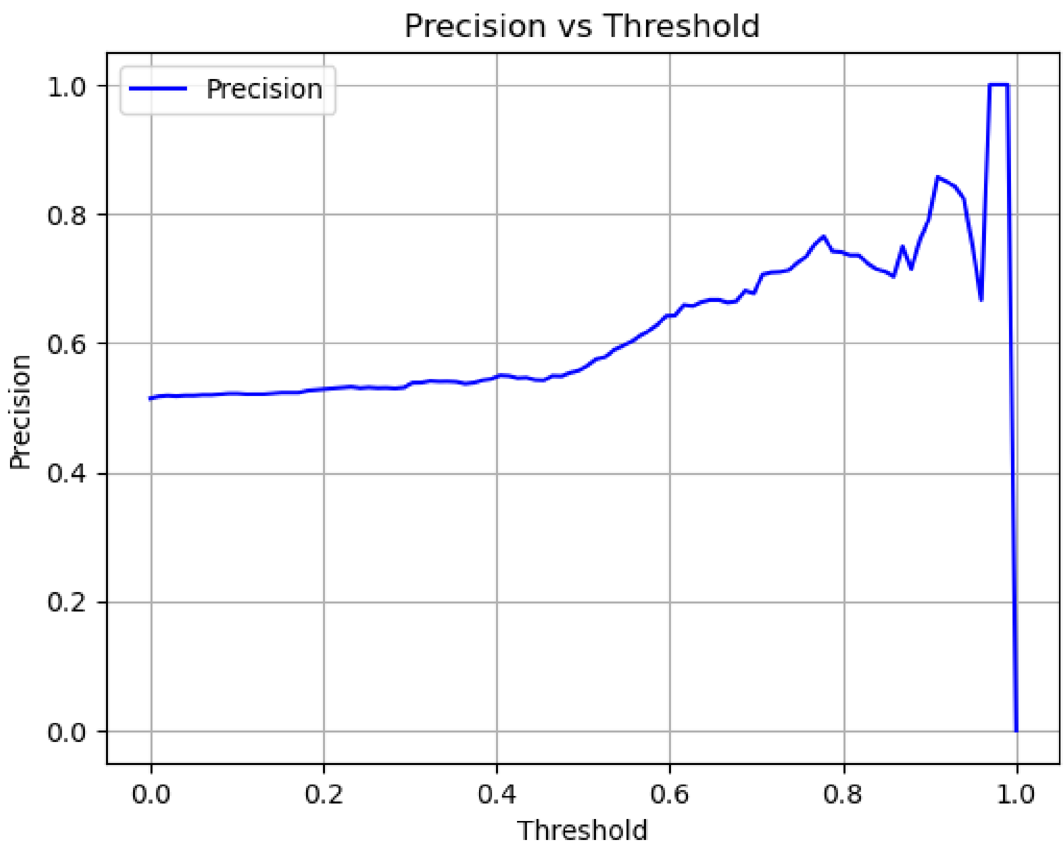



```
16/16                 1s 39ms/step
16/16                 1s 40ms/step
16/16                 1s 39ms/step
16/16                 1s 38ms/step
16/16                 1s 38ms/step
```

```
16/16            1s 37ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 41ms/step
16/16            1s 40ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 39ms/step
16/16            1s 38ms/step
16/16            1s 40ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 39ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 37ms/step
16/16            1s 39ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 42ms/step
16/16            1s 43ms/step
16/16            1s 41ms/step
16/16            1s 40ms/step
16/16            1s 45ms/step
16/16            1s 42ms/step
16/16            1s 41ms/step
16/16            1s 39ms/step
16/16            1s 41ms/step
16/16            1s 41ms/step
16/16            1s 41ms/step
16/16            1s 42ms/step
16/16            1s 45ms/step
16/16            1s 39ms/step
16/16            1s 39ms/step
16/16            1s 40ms/step
16/16            1s 39ms/step
16/16            1s 39ms/step
16/16            1s 42ms/step
```

```
16/16                    1s 40ms/step
16/16                    1s 40ms/step
16/16                    1s 39ms/step
16/16                    1s 40ms/step
16/16                    1s 40ms/step
16/16                    1s 40ms/step
16/16                    1s 39ms/step
16/16                    1s 39ms/step
16/16                    1s 40ms/step
16/16                    1s 39ms/step
16/16                    1s 40ms/step
16/16                    1s 41ms/step
16/16                    1s 40ms/step
16/16                    1s 40ms/step
16/16                    1s 39ms/step
16/16                    1s 39ms/step
16/16                    1s 39ms/step
16/16                    1s 40ms/step
16/16                    1s 41ms/step
16/16                    1s 40ms/step
16/16                    1s 40ms/step
16/16                    1s 40ms/step
16/16                    1s 42ms/step
16/16                    1s 39ms/step
16/16                    1s 40ms/step
16/16                    1s 39ms/step
16/16                    1s 39ms/step
16/16                    1s 42ms/step
16/16                    1s 44ms/step
16/16                    1s 40ms/step
16/16                    1s 42ms/step
16/16                    1s 43ms/step
16/16                    1s 42ms/step
16/16                    1s 42ms/step
16/16                    1s 42ms/step
16/16                    1s 40ms/step
16/16                    1s 41ms/step
16/16                    1s 40ms/step
16/16                    1s 42ms/step
16/16                    1s 40ms/step
16/16                    1s 40ms/step
16/16                    1s 39ms/step
16/16                    1s 39ms/step
16/16                    1s 39ms/step
16/16                    1s 39ms/step
16/16                    1s 39ms/step
16/16                    1s 38ms/step
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

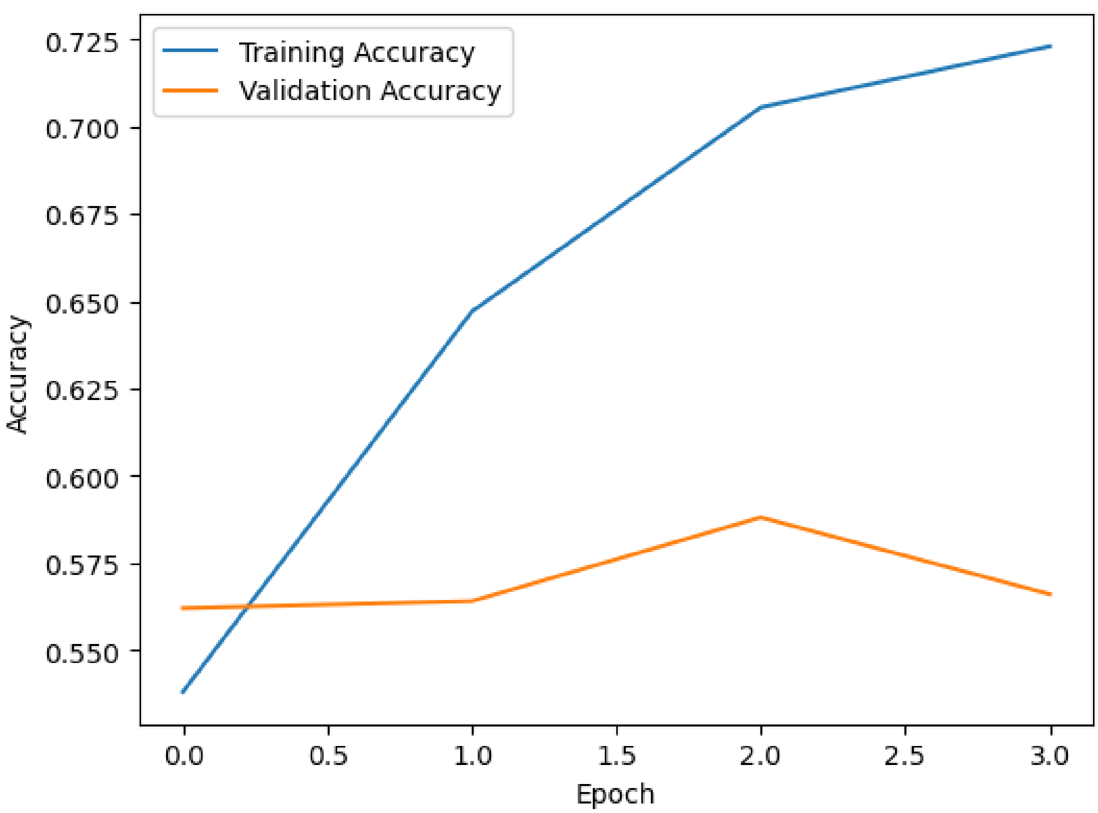


[265]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
↪roc_auc_score

# Evaluate model performance on training and validation data
loss_train, accuracy_train = model_dropout.evaluate(X_train_images,␣
↪y_train_encoded)
loss_val, accuracy_val = model_dropout.evaluate(X_val_images, y_val_encoded)

print("Training Loss:", loss_train)
print("Training Accuracy:", accuracy_train)
print("Validation Loss:", loss_val)
print("Validation Accuracy:", accuracy_val)
```

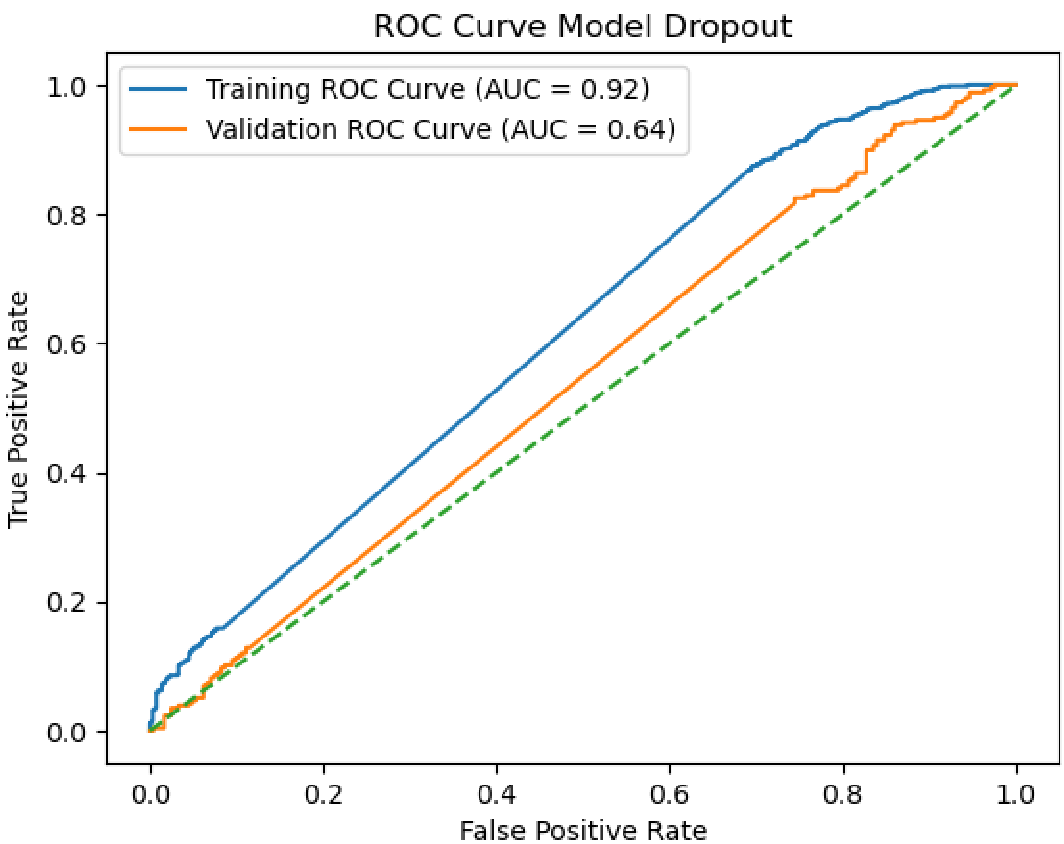
```python
# Plot training history
plt.plot(history_simple.history['accuracy'], label='Training Accuracy')
plt.plot(history_simple.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

y_train_pred = np.where(model_dropout.predict(X_train_images) > 0.5, 1, 0)
y_val_pred = np.where(model_dropout.predict(X_val_images) > 0.5, 1, 0)

conf_matrix_train = confusion_matrix(y_train_encoded, y_train_pred)
conf_matrix_val = confusion_matrix(y_val_encoded, y_val_pred)

print('Confusion Matrix - Training:')
print(conf_matrix_train)
print('Confusion Matrix - Validation:')
print(conf_matrix_val)

print('Classification Report - Training:')
print(classification_report(y_train_encoded, y_train_pred))
print('Classification Report - Validation:')
print(classification_report(y_val_encoded, y_val_pred))


fpr_train, tpr_train, _ = roc_curve(y_train_encoded, model_dropout.
 ↪predict(X_train_images))
fpr_val, tpr_val, _ = roc_curve(y_val_encoded, model_dropout.
 ↪predict(X_val_images))

auc_train = roc_auc_score(y_train_encoded, model_simple.predict(X_train_images))
auc_val = roc_auc_score(y_val_encoded, model_simple.predict(X_val_images))

plt.plot(fpr_train, tpr_train, label=f'Training ROC Curve (AUC = {auc_train:.
 ↪2f})')
plt.plot(fpr_val, tpr_val, label=f'Validation ROC Curve (AUC = {auc_val:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Model Dropout')
plt.legend()
plt.show()

thresholds = np.linspace(0, 1, 100)
precisions = []
for threshold in thresholds:
```

```
    y_val_pred_thresholded = (model_dropout.predict(X_val_images) > threshold).
↪astype(int)
    precisions.append(precision_score(y_val_encoded, y_val_pred_thresholded))

# Plot precision graph
plt.plot(thresholds, precisions, label='Precision', color='blue')
plt.xlabel('Threshold')
plt.ylabel('Precision')
plt.title('Precision vs Threshold')
plt.legend()
plt.grid(True)
plt.show()
```

```
63/63                3s 40ms/step -
accuracy: 0.5629 - loss: 0.6637
16/16                1s 38ms/step -
accuracy: 0.5777 - loss: 0.6764
Training Loss: 0.6577760577201843
Training Accuracy: 0.5789999961853027
Validation Loss: 0.6915680766105652
Validation Accuracy: 0.5460000038146973
```



```
63/63            3s 40ms/step
16/16            1s 38ms/step
Confusion Matrix - Training:
[[184 788]
 [ 54 974]]
Confusion Matrix - Validation:
[[ 40 203]
 [ 24 233]]
Classification Report - Training:
              precision    recall  f1-score   support

           0       0.77      0.19      0.30       972
           1       0.55      0.95      0.70      1028

    accuracy                           0.58      2000
   macro avg       0.66      0.57      0.50      2000
weighted avg       0.66      0.58      0.51      2000

Classification Report - Validation:
              precision    recall  f1-score   support

           0       0.62      0.16      0.26       243
           1       0.53      0.91      0.67       257

    accuracy                           0.55       500
   macro avg       0.58      0.54      0.47       500
weighted avg       0.58      0.55      0.47       500

63/63            2s 39ms/step
16/16            1s 39ms/step
63/63            3s 42ms/step
16/16            1s 42ms/step
```




```
16/16                   1s 39ms/step
16/16                   1s 38ms/step
16/16                   1s 38ms/step
16/16                   1s 38ms/step
16/16                   1s 38ms/step
16/16                   1s 44ms/step
16/16                   1s 45ms/step
16/16                   1s 41ms/step
16/16                   1s 40ms/step
16/16                   1s 39ms/step
16/16                   1s 41ms/step
16/16                   1s 39ms/step
16/16                   1s 39ms/step
16/16                   1s 40ms/step
16/16                   1s 38ms/step
16/16                   1s 39ms/step
16/16                   1s 39ms/step
16/16                   1s 38ms/step
16/16                   1s 38ms/step
16/16                   1s 40ms/step
```

```
16/16                  1s 40ms/step
16/16                  1s 41ms/step
16/16                  1s 39ms/step
16/16                  1s 40ms/step
16/16                  1s 38ms/step
16/16                  1s 41ms/step
16/16                  1s 39ms/step
16/16                  1s 39ms/step
16/16                  1s 42ms/step
16/16                  1s 40ms/step
16/16                  1s 38ms/step
16/16                  1s 38ms/step
16/16                  1s 39ms/step
16/16                  1s 38ms/step
16/16                  1s 43ms/step
16/16                  1s 40ms/step
16/16                  1s 42ms/step
16/16                  1s 40ms/step
16/16                  1s 46ms/step
16/16                  1s 40ms/step
16/16                  1s 39ms/step
16/16                  1s 39ms/step
16/16                  1s 39ms/step
16/16                  1s 38ms/step
16/16                  1s 38ms/step
16/16                  1s 40ms/step
16/16                  1s 39ms/step
16/16                  1s 41ms/step
16/16                  1s 39ms/step
16/16                  1s 39ms/step
16/16                  1s 38ms/step
16/16                  1s 39ms/step
16/16                  1s 42ms/step
16/16                  1s 41ms/step
16/16                  1s 44ms/step
16/16                  1s 38ms/step
16/16                  1s 41ms/step
16/16                  1s 39ms/step
16/16                  1s 40ms/step
16/16                  1s 39ms/step
16/16                  1s 38ms/step
16/16                  1s 41ms/step
16/16                  1s 40ms/step
16/16                  1s 46ms/step
16/16                  1s 40ms/step
16/16                  1s 38ms/step
16/16                  1s 39ms/step
16/16                  1s 41ms/step
```

```
16/16            1s 40ms/step
16/16            1s 40ms/step
16/16            1s 40ms/step
16/16            1s 41ms/step
16/16            1s 43ms/step
16/16            1s 39ms/step
16/16            1s 40ms/step
16/16            1s 41ms/step
16/16            1s 43ms/step
16/16            1s 43ms/step
16/16            1s 39ms/step
16/16            1s 38ms/step
16/16            1s 38ms/step
16/16            1s 42ms/step
16/16            1s 44ms/step
16/16            1s 41ms/step
16/16            1s 40ms/step
16/16            1s 39ms/step
16/16            1s 39ms/step
16/16            1s 39ms/step
16/16            1s 39ms/step
16/16            1s 41ms/step
16/16            1s 39ms/step
16/16            1s 39ms/step
16/16            1s 41ms/step
 3/16            0s 45ms/step
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
16/16            1s 41ms/step
 3/16            0s 41ms/step
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
16/16            1s 39ms/step
 3/16            0s 39ms/step
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
16/16            1s 41ms/step
 3/16            0s 43ms/step
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
16/16            1s 41ms/step
 3/16            0s 39ms/step
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
16/16            1s 40ms/step
 3/16            0s 40ms/step
```

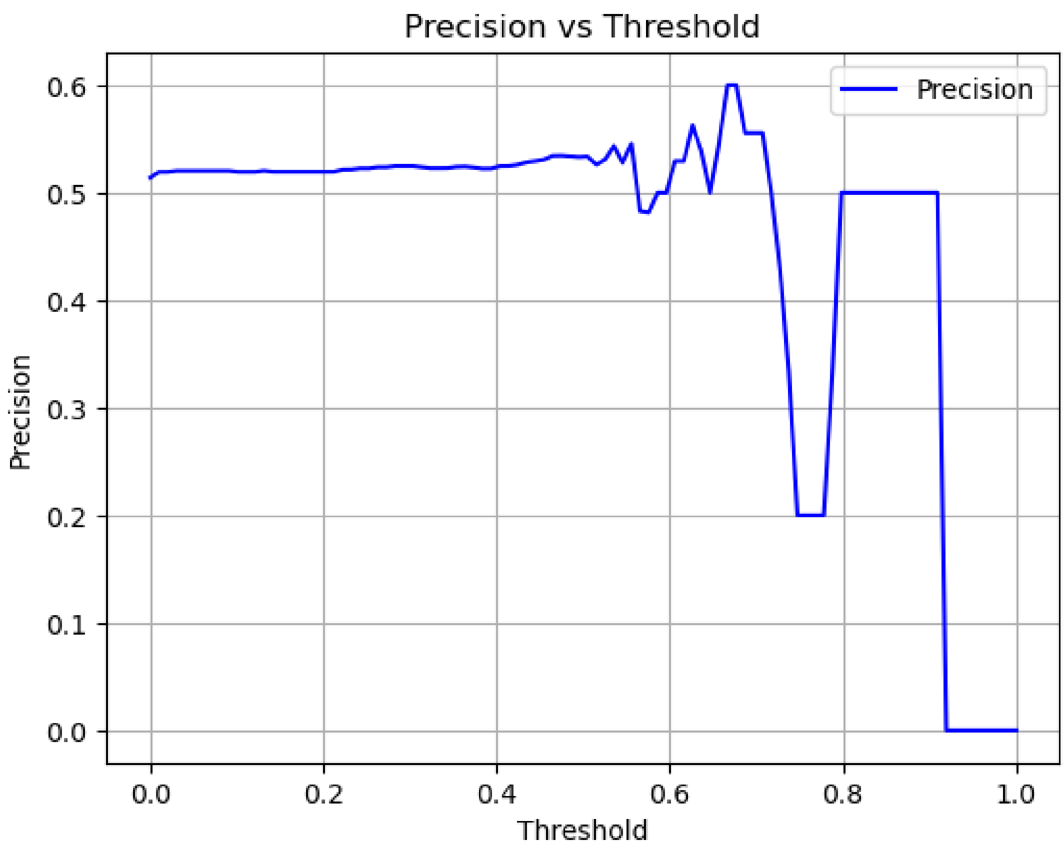
```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
16/16            1s 39ms/step
 3/16            0s 43ms/step
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
16/16            1s 38ms/step
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

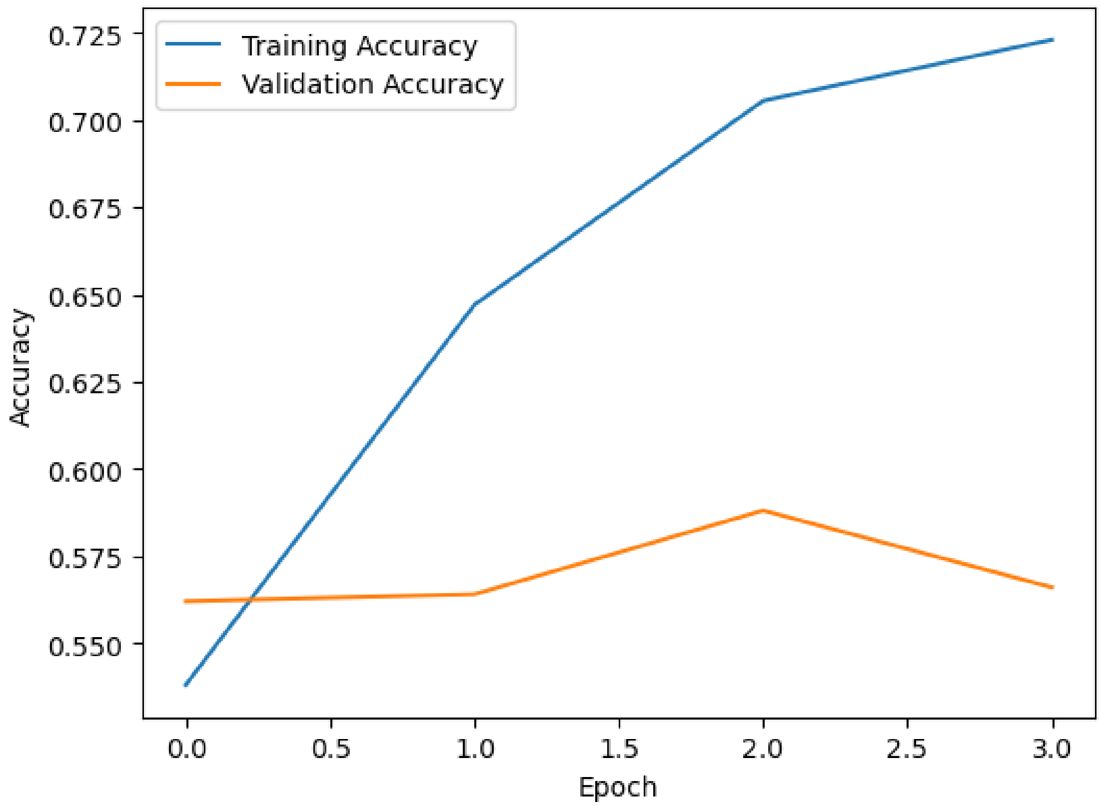


[266]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
↪roc_auc_score

# Evaluate model performance on training and validation data
loss_train, accuracy_train = model_batchnorm.evaluate(X_train_images,␣
↪y_train_encoded)
loss_val, accuracy_val = model_batchnorm.evaluate(X_val_images, y_val_encoded)

print("Training Loss:", loss_train)
print("Training Accuracy:", accuracy_train)
print("Validation Loss:", loss_val)
print("Validation Accuracy:", accuracy_val)

# Plot training history
plt.plot(history_simple.history['accuracy'], label='Training Accuracy')
plt.plot(history_simple.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
```

```
plt.legend()
plt.show()

y_train_pred = np.where(model_batchnorm.predict(X_train_images) > 0.5, 1, 0)
y_val_pred = np.where(model_batchnorm.predict(X_val_images) > 0.5, 1, 0)

conf_matrix_train = confusion_matrix(y_train_encoded, y_train_pred)
conf_matrix_val = confusion_matrix(y_val_encoded, y_val_pred)

print('Confusion Matrix - Training:')
print(conf_matrix_train)
print('Confusion Matrix - Validation:')
print(conf_matrix_val)

print('Classification Report - Training:')
print(classification_report(y_train_encoded, y_train_pred))
print('Classification Report - Validation:')
print(classification_report(y_val_encoded, y_val_pred))


fpr_train, tpr_train, _ = roc_curve(y_train_encoded, model_batchnorm.
 ↪predict(X_train_images))
fpr_val, tpr_val, _ = roc_curve(y_val_encoded, model_batchnorm.
 ↪predict(X_val_images))

auc_train = roc_auc_score(y_train_encoded, model_batchnorm.
 ↪predict(X_train_images))
auc_val = roc_auc_score(y_val_encoded, model_batchnorm.predict(X_val_images))

plt.plot(fpr_train, tpr_train, label=f'Training ROC Curve (AUC = {auc_train:.
 ↪2f})')
plt.plot(fpr_val, tpr_val, label=f'Validation ROC Curve (AUC = {auc_val:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Model Batch Normilization')
plt.legend()
plt.show()

thresholds = np.linspace(0, 1, 100)
precisions = []
for threshold in thresholds:
    y_val_pred_thresholded = (model_batchnorm.predict(X_val_images) >␣
 ↪threshold).astype(int)
    precisions.append(precision_score(y_val_encoded, y_val_pred_thresholded))

# Plot precision graph
```

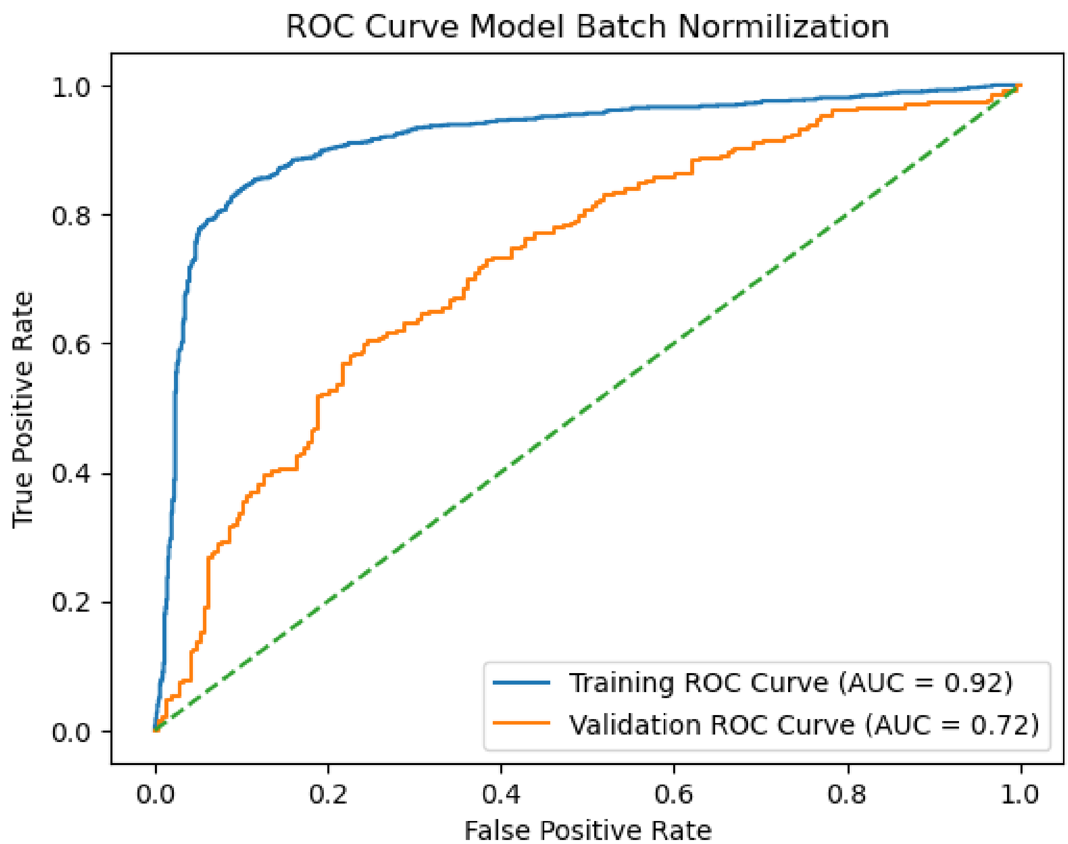
```
plt.plot(thresholds, precisions, label='Precision', color='blue')
plt.xlabel('Threshold')
plt.ylabel('Precision')
plt.title('Precision vs Threshold')
plt.legend()
plt.grid(True)
plt.show()
```

```
63/63                3s 50ms/step -
accuracy: 0.8674 - loss: 0.4589
16/16                1s 47ms/step -
accuracy: 0.6594 - loss: 1.1167
Training Loss: 0.45821356773376465
Training Accuracy: 0.8659999966621399
Validation Loss: 1.0418514013290405
Validation Accuracy: 0.6620000004768372
```




```
63/63                3s 46ms/step
16/16                1s 46ms/step
Confusion Matrix - Training:
[[852 120]
 [148 880]]
```

```
Confusion Matrix - Validation:
[[164  79]
 [ 90 167]]
Classification Report - Training:
              precision    recall  f1-score   support

           0       0.85      0.88      0.86       972
           1       0.88      0.86      0.87      1028

    accuracy                           0.87      2000
   macro avg       0.87      0.87      0.87      2000
weighted avg       0.87      0.87      0.87      2000

Classification Report - Validation:
              precision    recall  f1-score   support

           0       0.65      0.67      0.66       243
           1       0.68      0.65      0.66       257

    accuracy                           0.66       500
   macro avg       0.66      0.66      0.66       500
weighted avg       0.66      0.66      0.66       500

63/63                 3s 45ms/step
16/16                 1s 44ms/step
63/63                 3s 46ms/step
16/16                 1s 44ms/step
```

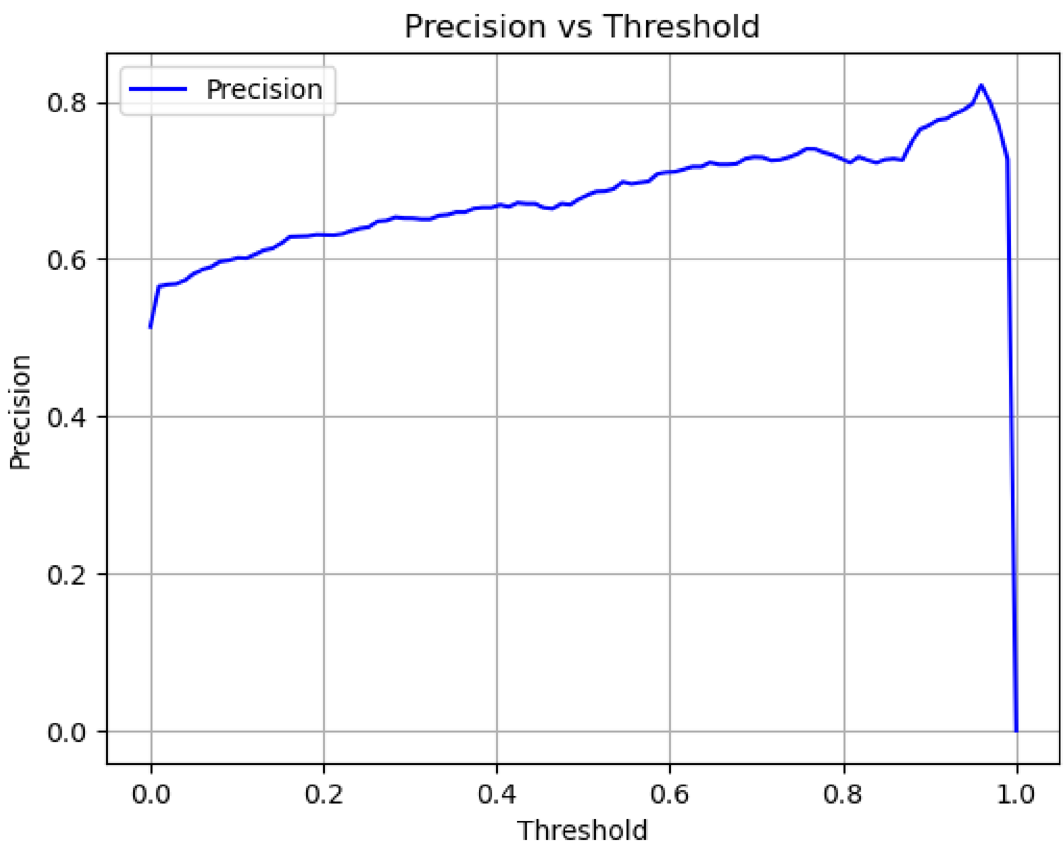



```
16/16                  1s 47ms/step
16/16                  1s 48ms/step
16/16                  1s 45ms/step
16/16                  1s 45ms/step
16/16                  1s 47ms/step
16/16                  1s 45ms/step
16/16                  1s 45ms/step
16/16                  1s 47ms/step
16/16                  1s 50ms/step
16/16                  1s 45ms/step
16/16                  1s 48ms/step
16/16                  1s 46ms/step
16/16                  1s 44ms/step
16/16                  1s 50ms/step
16/16                  1s 47ms/step
16/16                  1s 45ms/step
16/16                  1s 51ms/step
16/16                  1s 45ms/step
16/16                  1s 46ms/step
16/16                  1s 47ms/step
```

```
16/16                  1s 45ms/step
16/16                  1s 46ms/step
16/16                  1s 45ms/step
16/16                  1s 44ms/step
16/16                  1s 44ms/step
16/16                  1s 44ms/step
16/16                  1s 44ms/step
16/16                  1s 44ms/step
16/16                  1s 46ms/step
16/16                  1s 45ms/step
16/16                  1s 48ms/step
16/16                  1s 52ms/step
16/16                  1s 49ms/step
16/16                  1s 45ms/step
16/16                  1s 44ms/step
16/16                  1s 50ms/step
16/16                  1s 51ms/step
16/16                  1s 46ms/step
16/16                  1s 46ms/step
16/16                  1s 48ms/step
16/16                  1s 45ms/step
16/16                  1s 44ms/step
16/16                  1s 48ms/step
16/16                  1s 45ms/step
16/16                  1s 46ms/step
16/16                  1s 45ms/step
16/16                  1s 45ms/step
16/16                  1s 46ms/step
16/16                  1s 45ms/step
16/16                  1s 47ms/step
16/16                  1s 47ms/step
16/16                  1s 47ms/step
16/16                  1s 46ms/step
16/16                  1s 47ms/step
16/16                  1s 46ms/step
16/16                  1s 46ms/step
16/16                  1s 45ms/step
16/16                  1s 45ms/step
16/16                  1s 47ms/step
16/16                  1s 44ms/step
16/16                  1s 46ms/step
16/16                  1s 51ms/step
16/16                  1s 55ms/step
16/16                  1s 48ms/step
16/16                  1s 45ms/step
16/16                  1s 48ms/step
16/16                  1s 47ms/step
16/16                  1s 49ms/step
```

```
16/16            1s 46ms/step
16/16            1s 46ms/step
16/16            1s 45ms/step
16/16            1s 45ms/step
16/16            1s 47ms/step
16/16            1s 53ms/step
16/16            1s 47ms/step
16/16            1s 49ms/step
16/16            1s 45ms/step
16/16            1s 46ms/step
16/16            1s 46ms/step
16/16            1s 49ms/step
16/16            1s 46ms/step
16/16            1s 44ms/step
16/16            1s 45ms/step
16/16            1s 48ms/step
16/16            1s 44ms/step
16/16            1s 46ms/step
16/16            1s 44ms/step
16/16            1s 45ms/step
16/16            1s 50ms/step
16/16            1s 45ms/step
16/16            1s 46ms/step
16/16            1s 49ms/step
16/16            1s 45ms/step
16/16            1s 45ms/step
16/16            1s 47ms/step
16/16            1s 44ms/step
16/16            1s 46ms/step
16/16            1s 45ms/step
16/16            1s 45ms/step
16/16            1s 46ms/step
```

```
/Library/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



[324]:
```python
image_test = "/Users/nikhil/Desktop/dogs-vs-cats-redux-kernels-edition/test"


df_test = pd.DataFrame({"filename": filenames})

df_test.head()
df_test['id'] = df_test['filename'].str.extract('(\d+)').astype(int)
df_sorted = df_test.sort_values(by='id')
df_sorted.drop(columns=['id'], inplace=True)
df_sorted
```

[324]:

	filename
11768	1.jpg
10969	2.jpg
11663	3.jpg
9385	4.jpg
10128	5.jpg
…	…
1157	12496.jpg
1212	12497.jpg

```
12131  12498.jpg
12077  12499.jpg
10162  12500.jpg

[12500 rows x 1 columns]
```

[325]:
```
X_test_images = load_images(df_sorted['filename'], image_test)



y_pred = model_batchnorm.predict(X_test_images)  # Predict probabilities
y_pred = (y_pred_prob >= 0.5).astype(int)  # Convert probabilities to binary␣
↪predictions

df_sorted['predicted_label'] = y_pred

# Alternatively, you can create a new DataFrame with filenames and predicted␣
↪labels
y_pred
```

```
391/391                18s 46ms/step
```

[325]:
```
array([[0],
       [1],
       [0],
       …,
       [1],
       [1],
       [0]])
```

[332]:
```
len(y_pred)
```

[332]:
```
12500
```

[333]:
```
predictions_df = pd.DataFrame(y_pred, columns=['label'])

# Print or use predictions_df as needed
print(predictions_df)
```

```
       label
0          0
1          1
2          0
3          1
4          0
…        …
12495      0
12496      0
12497      1
```

```
12498      1
12499      0

[12500 rows x 1 columns]
```

[334]:
```python
y_test_df = pd.DataFrame(predictions_df, columns=['label'])
y_test_df
```

[334]:

	label
0	0
1	1
2	0
3	1
4	0
…	…
12495	0
12496	0
12497	1
12498	1
12499	0

[12500 rows x 1 columns]

[335]:
```python
y_test_df['id'] = range(1, len(y_test_df) + 1)

print(y_test_df)
```

```
       label     id
0          0      1
1          1      2
2          0      3
3          1      4
4          0      5
…        …     …
12495      0  12496
12496      0  12497
12497      1  12498
12498      1  12499
12499      0  12500

[12500 rows x 2 columns]
```

[336]:
```python
reversed_df = y_test_df.iloc[:, ::-1]

print(reversed_df)
```

```
          id  label
0          1      0
1          2      1
```

```
2          3      0
3          4      1
4          5      0
…        …    …
12495  12496      0
12496  12497      0
12497  12498      1
12498  12499      1
12499  12500      0

[12500 rows x 2 columns]
```

[338]:
```
csv_file_path = 'CNN.csv'

reversed_df.to_csv(csv_file_path, index=False)

print("DataFrame has been exported to:", csv_file_path)
```

```
DataFrame has been exported to: CNN.csv
```

[ ]:
```

```