Tirumal Convultional Network Assignment Report

Introduction:

This assignment explored the performance of two different neural network architectures—one trained from scratch and the other using a pretrained ResNet model—on the task of classifying images of cats and dogs. The goal was to determine how varying the size of the training dataset impacts the performance of each model and to identify the relationship between training sample size and model architecture.

Methodology:

The task was divided into two parts:

1. Training from Scratch: A simple convolutional neural network (CNN) was designed and trained from scratch with different training sample sizes (1000, 1500, and 2500).
2. Pretrained ResNet: A ResNet-based model was fine-tuned using the same sample sizes to compare its performance against the scratch model.

For each experiment, the models were trained for 10 epochs, and performance metrics were captured for both the training set and the test set. Key metrics used were accuracy and loss for both the training and validation datasets.

Results:

Model	Training Sample Size	Test Accuracy (%)	Training Accuracy (%)	Validation Accuracy (%)	Validation Loss
Scratch CNN	1000	52.9	49.8	50	0.6911
Scratch CNN	1500	67.2	64.5	68.8	0.6274
Scratch CNN	2500	64	67.1	65.4	0.5964
Pretrained ResNet	1000	54	53.6	52	0.6833
Pretrained ResNet	2000	58	50.4	62.2	0.6816
Pretrained ResNet	2500	51.2	50.3	52.4	0.6931

Discussion:

1. Impact of Training Sample Size:

- For both models, increasing the training sample size generally improved the performance. The models showed higher accuracy with larger datasets, particularly the CNN trained from scratch, which saw a noticeable boost between 1000 and 1500 samples.
- Pretrained ResNet's performance, on the other hand, remained relatively stable with smaller sample sizes but did not show as significant improvements with increasing samples compared to the scratch model.

2. Model Architecture:

- Scratch Model: The CNN model trained from scratch benefited more significantly from an increase in training sample size. It showed marked improvement as the dataset size increased, particularly in the 1000 to 1500 range, where test accuracy jumped from 52.9% to 67.2%. However, further increases in sample size (2500) yielded diminishing returns, with accuracy plateauing around 64-65%.
- Pretrained ResNet Model: While the pretrained ResNet model started with a higher accuracy at smaller sample sizes, it did not experience the same level of improvement with larger datasets. Test accuracy for ResNet was highest at 57.6% for 2000 samples but dropped to 51.2% with 2500 samples, suggesting that fine-tuning the ResNet might have required a different approach or additional regularization techniques.

3. Regularization and Overfitting:

- Scratch Model: There was evidence of overfitting in the scratch model, especially as the training accuracy surpassed validation accuracy. Regularization techniques like dropout or early stopping may be beneficial to address this.
- Pretrained Model: The pretrained model's performance plateaued early and was less sensitive to the increasing data size, which could be due to the fact that the model's pretrained layers are already well-optimized for general features.

Conclusion:

The study found that the training sample size plays a crucial role in determining model performance, particularly for models trained from scratch. Pretrained models like ResNet showed less sensitivity to training sample size in this task, suggesting that fine-tuning with a larger dataset may require more advanced techniques.

Overall, the scratch model showed greater improvement with additional training data, while the pretrained ResNet maintained a more stable but slightly lower performance level across all sample sizes.

Loading Libraries

In [1]:
```
from google.colab import drive
import zipfile
import os
import shutil
import random
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
```

Loading Dataset

In [2]:
```
# Mount Google Drive
drive.mount('/content/drive')
```

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

In [3]:
```
# Define paths
base_dir = '/content/drive/MyDrive'
zip_file_path = os.path.join(base_dir, 'cats_vs_dogs_small_dataset.zip')
extracted_dir_path = os.path.join(base_dir, 'cats_vs_dogs_small_dataset')
```

In [4]:
```
# Unzip dataset
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extracted_dir_path)
```

In [13]:
```
# Dataset directories for 'cat' and 'dog' folders
cat_folder_path = os.path.join(extracted_dir_path, 'cats_vs_dogs_small_dataset/cat')
dog_folder_path = os.path.join(extracted_dir_path, 'cats_vs_dogs_small_dataset/dog')
```

Splitting The Dataset

In [6]:
```
# Function to split and organize dataset
def split_dataset(base_dir, cat_images, dog_images, train_samples, validation_samples,
    train_dir = os.path.join(base_dir, 'train')
    validation_dir = os.path.join(base_dir, 'validation')
    test_dir = os.path.join(base_dir, 'test')

    for d in [train_dir, validation_dir, test_dir]:
        shutil.rmtree(d, ignore_errors=True)
        os.makedirs(os.path.join(d, 'cat'), exist_ok=True)
        os.makedirs(os.path.join(d, 'dog'), exist_ok=True)

    # Randomly shuffle images
    random.shuffle(cat_images)
    random.shuffle(dog_images)
```

```
    # Copy images to appropriate directories
    def copy_images(src_dir, dst_dir, file_list):
        for file in file_list:
            src_path = os.path.join(src_dir, file)
            dst_path = os.path.join(dst_dir, file)
            shutil.copyfile(src_path, dst_path)

    copy_images(cat_folder_path, os.path.join(train_dir, 'cat'), cat_images[:train_sam
    copy_images(dog_folder_path, os.path.join(train_dir, 'dog'), dog_images[:train_sam

    copy_images(cat_folder_path, os.path.join(validation_dir, 'cat'),
                cat_images[train_samples // 2:train_samples // 2 + validation_samples
    copy_images(dog_folder_path, os.path.join(validation_dir, 'dog'),
                dog_images[train_samples // 2:train_samples // 2 + validation_samples

    copy_images(cat_folder_path, os.path.join(test_dir, 'cat'),
                cat_images[train_samples // 2 + validation_samples // 2:
                           train_samples // 2 + validation_samples // 2 + test_samples
    copy_images(dog_folder_path, os.path.join(test_dir, 'dog'),
                dog_images[train_samples // 2 + validation_samples // 2:
                           train_samples // 2 + validation_samples // 2 + test_samples

    return train_dir, validation_dir, test_dir
```

In [7]:

```
# Image augmentations and data generators
def create_generators(train_dir, validation_dir, test_dir, image_size, batch_size):
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    )
    validation_datagen = ImageDataGenerator(rescale=1./255)
    test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=image_size,
        batch_size=batch_size,
        class_mode='binary'
    )
    validation_generator = validation_datagen.flow_from_directory(
        validation_dir,
        target_size=image_size,
        batch_size=batch_size,
        class_mode='binary'
    )
    test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=image_size,
        batch_size=batch_size,
        class_mode='binary'
    )
    return train_generator, validation_generator, test_generator
```

Building Scartch Model

In [17]:
```python
# Build a convolutional model from scratch with Input layer
def build_scratch_model(image_size):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(image_size[0], image_size[1
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Building Pre Trained Models

In [9]:
```python
# Build a pretrained ResNet model
def build_pretrained_resnet(image_size):
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(image_si
    for layer in base_model.layers:
        layer.trainable = False
    x = Flatten()(base_model.output)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Model Training Function

In [23]:
```python
# Train the model and plot results
def train_model(model, train_generator, validation_generator, epochs):
    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=validation_generator
    )
    return history
```

Plotting Metrics

In [11]:
```python
# Plot performance metrics
def plot_metrics(history):
```

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

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

In [14]:
```
# Splitting dataset for training
test_samples = 500
validation_samples = 500
cat_images = os.listdir(cat_folder_path)
dog_images = os.listdir(dog_folder_path)
```

In [19]:
```
# Setting up parameters
batch_size = 20  # Set batch size
image_size = (150, 150)  # Set image size
```

Training from scratch with 1000 samples

In [20]:
```
# Step 1: Training from scratch with 1000 samples
train_samples_step1 = 1000
train_dir, validation_dir, test_dir = split_dataset(extracted_dir_path, cat_images, do
train_generator, validation_generator, test_generator = create_generators(train_dir, v
```

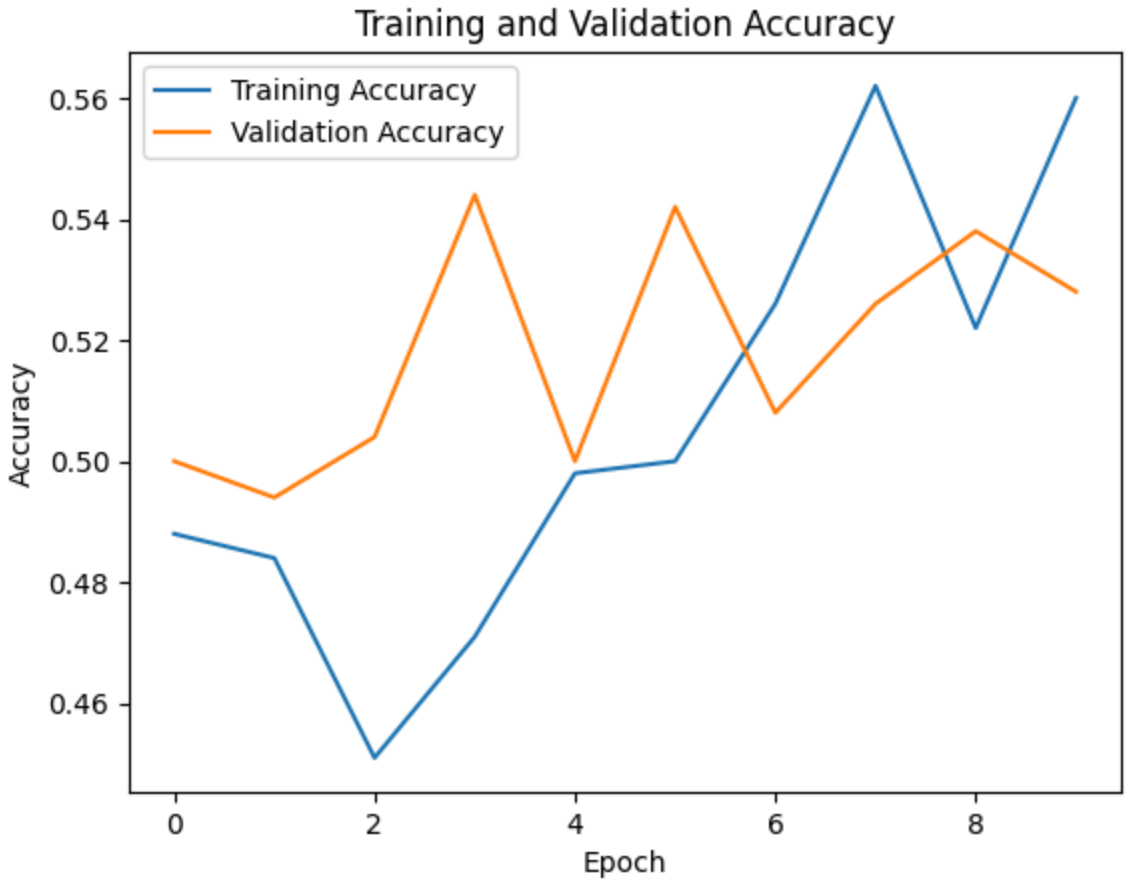
```
Found 1000 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
```

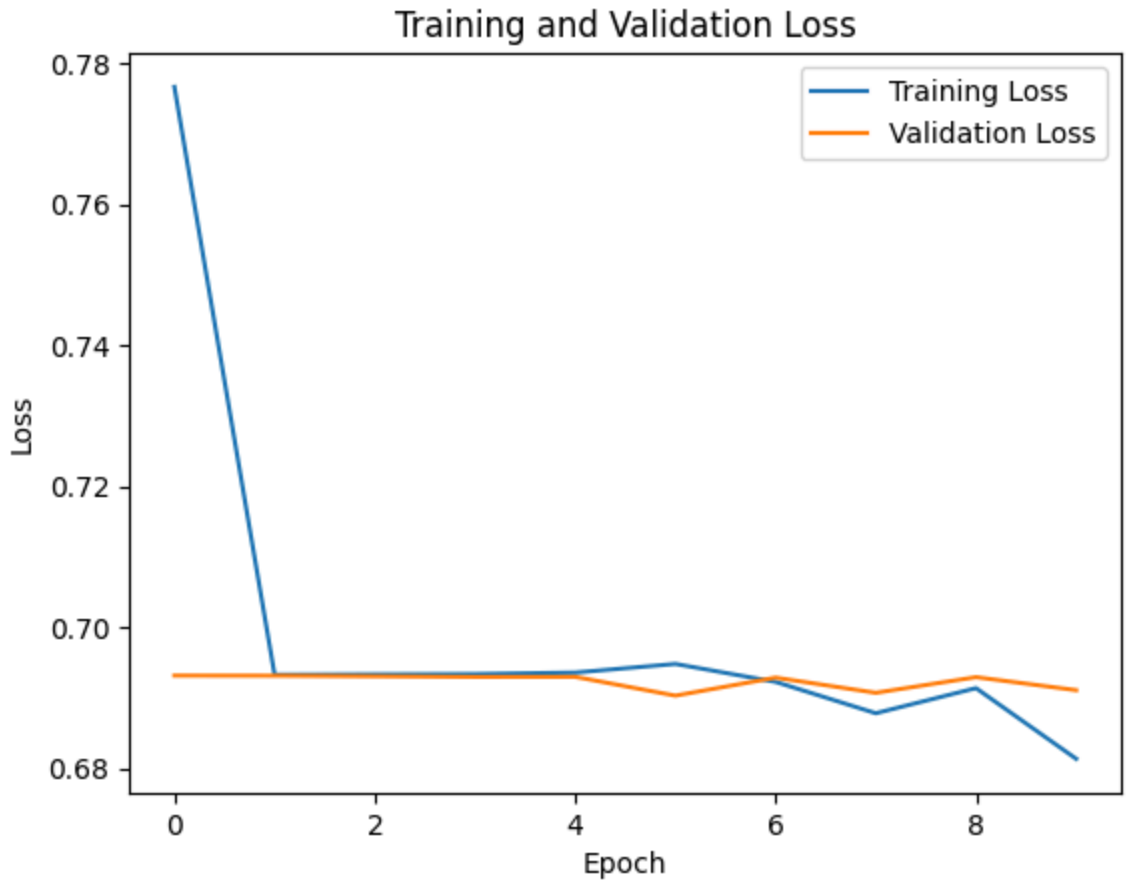
In [24]:
```
# Train from scratch
model_scratch_step1 = build_scratch_model(image_size)
history_scratch_step1 = train_model(model_scratch_step1, train_generator, validation_g
```

```
Epoch 1/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 277ms/step - accuracy: 0.4981 - loss: 0.9694 - val_acc
uracy: 0.5000 - val_loss: 0.6932
Epoch 2/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 247ms/step - accuracy: 0.5045 - loss: 0.6931 - val_acc
uracy: 0.4940 - val_loss: 0.6931
Epoch 3/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 20s 236ms/step - accuracy: 0.4395 - loss: 0.6934 - val_acc
uracy: 0.5040 - val_loss: 0.6931
Epoch 4/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 245ms/step - accuracy: 0.4739 - loss: 0.6933 - val_acc
uracy: 0.5440 - val_loss: 0.6930
Epoch 5/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 19s 219ms/step - accuracy: 0.5233 - loss: 0.6932 - val_acc
uracy: 0.5000 - val_loss: 0.6930
Epoch 6/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 21s 239ms/step - accuracy: 0.4963 - loss: 0.6935 - val_acc
uracy: 0.5420 - val_loss: 0.6903
Epoch 7/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 20s 226ms/step - accuracy: 0.5419 - loss: 0.6903 - val_acc
uracy: 0.5080 - val_loss: 0.6929
Epoch 8/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 217ms/step - accuracy: 0.5820 - loss: 0.6893 - val_acc
uracy: 0.5260 - val_loss: 0.6907
Epoch 9/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 222ms/step - accuracy: 0.5042 - loss: 0.6927 - val_acc
uracy: 0.5380 - val_loss: 0.6929
Epoch 10/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 234ms/step - accuracy: 0.5639 - loss: 0.6769 - val_acc
uracy: 0.5280 - val_loss: 0.6911
```

In [25]:

```
# Plot metrics for scratch model
plot_metrics(history_scratch_step1)
```





In [26]:
```
# Evaluate scratch model
test_loss_step1, test_accuracy_step1 = model_scratch_step1.evaluate(test_generator, st
```

```
print("Test accuracy (scratch, 1000 samples):", test_accuracy_step1)
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 89ms/step - accuracy: 0.5483 - loss: 0.6865
Test accuracy (scratch, 1000 samples): 0.5299999713897705
```

Training scratch model with 1500 samples

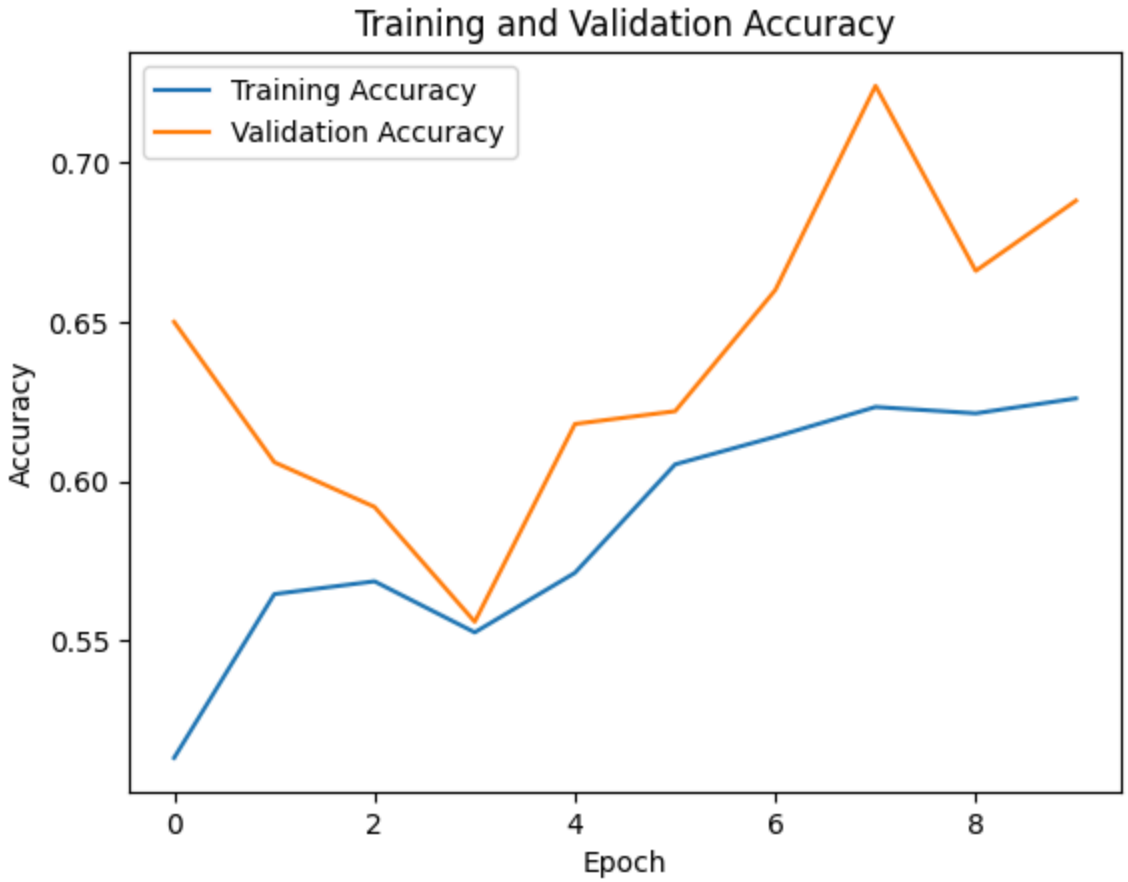
In [27]:
```
# Step 2: Training with 1500 samples
train_samples_step2 = 1500
train_dir, validation_dir, test_dir = split_dataset(extracted_dir_path, cat_images, do
train_generator, validation_generator, test_generator = create_generators(train_dir, v
```

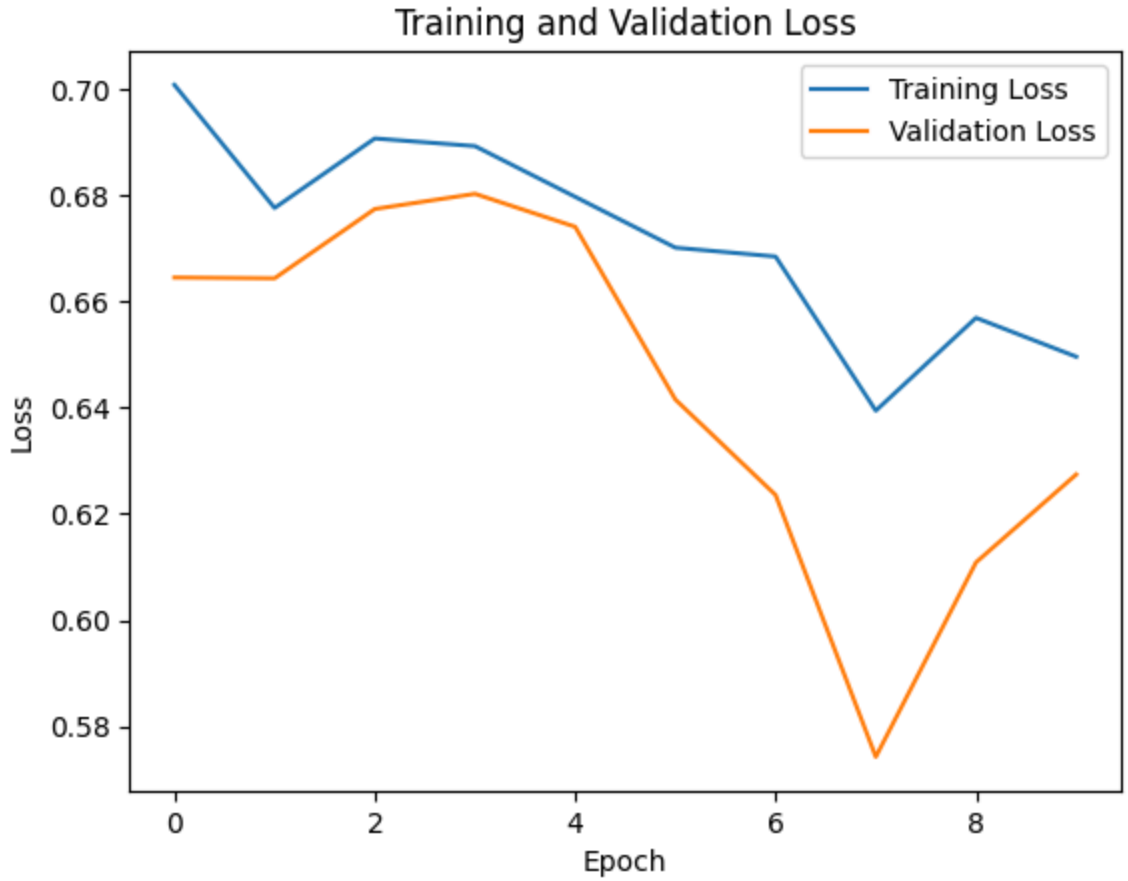
```
Found 1500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
```

In [28]:
```
# Train from scratch
model_scratch_step2 = build_scratch_model(image_size)
history_scratch_step2 = train_model(model_scratch_step2, train_generator, validation_g
```

```
Epoch 1/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 238ms/step - accuracy: 0.4882 - loss: 0.7231 - val_acc
uracy: 0.6500 - val_loss: 0.6646
Epoch 2/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 19s 234ms/step - accuracy: 0.5496 - loss: 0.6773 - val_acc
uracy: 0.6060 - val_loss: 0.6644
Epoch 3/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 21s 231ms/step - accuracy: 0.5567 - loss: 0.7097 - val_acc
uracy: 0.5920 - val_loss: 0.6775
Epoch 4/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 20s 230ms/step - accuracy: 0.5621 - loss: 0.6877 - val_acc
uracy: 0.5560 - val_loss: 0.6804
Epoch 5/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 21s 229ms/step - accuracy: 0.5626 - loss: 0.6814 - val_acc
uracy: 0.6180 - val_loss: 0.6742
Epoch 6/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 18s 227ms/step - accuracy: 0.6042 - loss: 0.6739 - val_acc
uracy: 0.6220 - val_loss: 0.6416
Epoch 7/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 21s 231ms/step - accuracy: 0.6119 - loss: 0.6677 - val_acc
uracy: 0.6600 - val_loss: 0.6236
Epoch 8/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 18s 226ms/step - accuracy: 0.6076 - loss: 0.6460 - val_acc
uracy: 0.7240 - val_loss: 0.5742
Epoch 9/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 18s 222ms/step - accuracy: 0.6161 - loss: 0.6619 - val_acc
uracy: 0.6660 - val_loss: 0.6109
Epoch 10/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 18s 228ms/step - accuracy: 0.6452 - loss: 0.6289 - val_acc
uracy: 0.6880 - val_loss: 0.6274
```

In [29]:
```
# Plot metrics for scratch model
plot_metrics(history_scratch_step2)
```





In [30]:
```
# Evaluate scratch model
test_loss_step2, test_accuracy_step2 = model_scratch_step2.evaluate(test_generator, st
```

```
print("Test accuracy (scratch, 1500 samples):", test_accuracy_step2)
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 92ms/step - accuracy: 0.7237 - loss: 0.6204
Test accuracy (scratch, 1500 samples): 0.671999990940094
```

Training scartch with 2500 samples

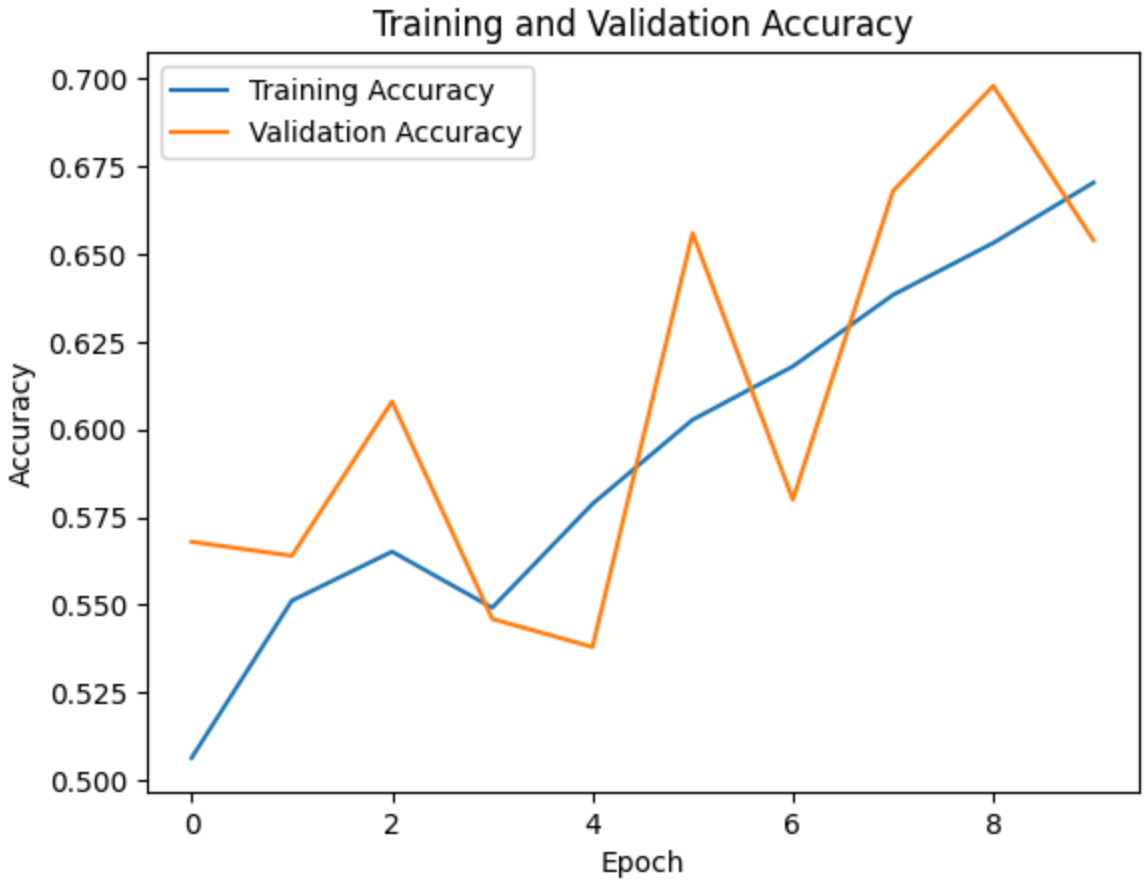
In [31]:
```
# Step 3: Training with 2500 samples
train_samples_step3 = 2500
train_dir, validation_dir, test_dir = split_dataset(extracted_dir_path, cat_images, do
train_generator, validation_generator, test_generator = create_generators(train_dir, v
```

```
Found 2500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
```

In [32]:
```
# Train from scratch
model_scratch_step3 = build_scratch_model(image_size)
history_scratch_step3 = train_model(model_scratch_step3, train_generator, validation_g
```

```
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 39s 262ms/step - accuracy: 0.4945 - loss: 0.7627 - val_a
ccuracy: 0.5680 - val_loss: 0.6888
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 30s 228ms/step - accuracy: 0.5510 - loss: 0.6926 - val_a
ccuracy: 0.5640 - val_loss: 0.6854
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 30s 233ms/step - accuracy: 0.5698 - loss: 0.6860 - val_a
ccuracy: 0.6080 - val_loss: 0.6711
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 30s 233ms/step - accuracy: 0.5539 - loss: 0.6877 - val_a
ccuracy: 0.5460 - val_loss: 0.6876
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 41s 231ms/step - accuracy: 0.5824 - loss: 0.6766 - val_a
ccuracy: 0.5380 - val_loss: 0.7204
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 30s 232ms/step - accuracy: 0.5941 - loss: 0.6707 - val_a
ccuracy: 0.6560 - val_loss: 0.6512
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 40s 225ms/step - accuracy: 0.6092 - loss: 0.6599 - val_a
ccuracy: 0.5800 - val_loss: 0.6381
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 29s 222ms/step - accuracy: 0.6378 - loss: 0.6512 - val_a
ccuracy: 0.6680 - val_loss: 0.6018
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 41s 220ms/step - accuracy: 0.6697 - loss: 0.6160 - val_a
ccuracy: 0.6980 - val_loss: 0.5964
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 41s 218ms/step - accuracy: 0.6711 - loss: 0.6066 - val_a
ccuracy: 0.6540 - val_loss: 0.6234
```

In [33]:
```
# Plot metrics for scratch model
plot_metrics(history_scratch_step3)
```



In [34]:
```
# Evaluate scratch model
test_loss_step3, test_accuracy_step3 = model_scratch_step3.evaluate(test_generator, st
```

```
print("Test accuracy (scratch, 2500 samples):", test_accuracy_step3)
```

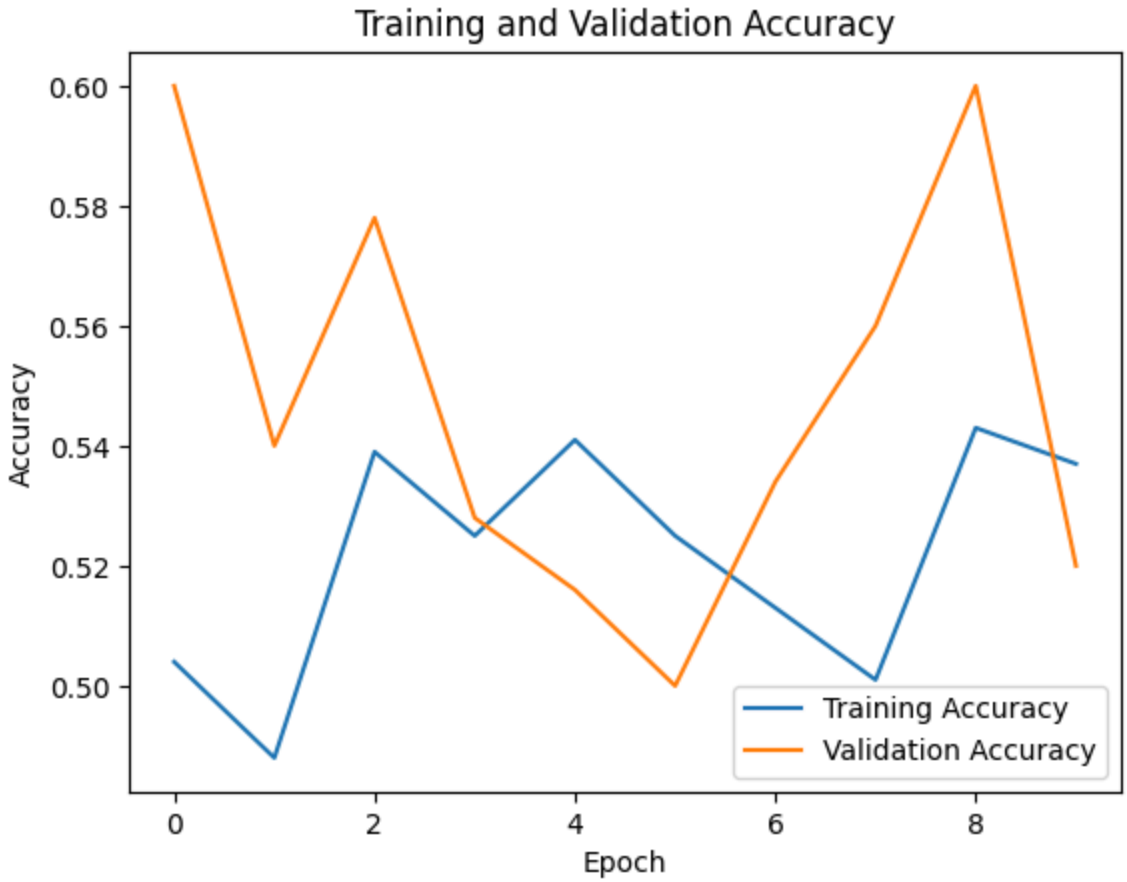
```
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 111ms/step - accuracy: 0.6439 - loss: 0.6347
Test accuracy (scratch, 2500 samples): 0.6399999856948853
```

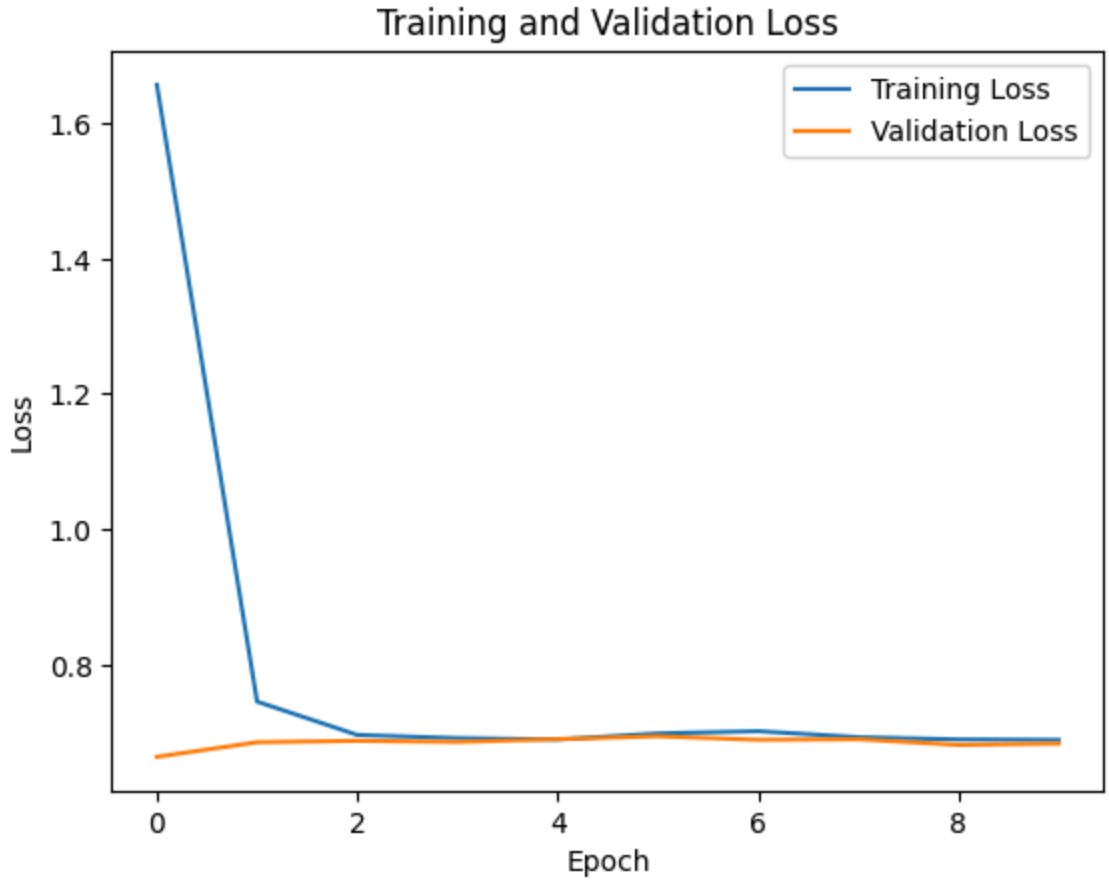
Training Pre Trained Model

In [37]:
```
# Step 4: Using a Pretrained Model

# Function to evaluate pretrained model with different training sample sizes
def evaluate_pretrained_model(train_samples, image_size, batch_size, validation_sample
    train_dir, validation_dir, test_dir = split_dataset(extracted_dir_path, cat_images
    train_generator, validation_generator, test_generator = create_generators(train_di

    model_resnet = build_pretrained_resnet(image_size)
    history_resnet = train_model(model_resnet, train_generator, validation_generator,

    # Plot metrics for ResNet model
    plot_metrics(history_resnet)

    # Evaluate pretrained ResNet model
    test_loss_resnet, test_accuracy_resnet = model_resnet.evaluate(test_generator, ste
    print(f"Test accuracy (ResNet, {train_samples} samples):", test_accuracy_resnet)
```

In [38]:
```
# Evaluate pretrained model with sample sizes 1000, 2000, and 2500
print("\nEvaluating Pretrained Model:")
evaluate_pretrained_model(1000, image_size, batch_size, validation_samples, test_sampl
```

```
Evaluating Pretrained Model:
Found 1000 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Epoch 1/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 31s 341ms/step - accuracy: 0.4969 - loss: 2.5380 - val_acc
uracy: 0.6000 - val_loss: 0.6637
Epoch 2/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 256ms/step - accuracy: 0.4803 - loss: 0.7654 - val_acc
uracy: 0.5400 - val_loss: 0.6852
Epoch 3/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 23s 288ms/step - accuracy: 0.5178 - loss: 0.6971 - val_acc
uracy: 0.5780 - val_loss: 0.6874
Epoch 4/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 19s 268ms/step - accuracy: 0.5374 - loss: 0.6918 - val_acc
uracy: 0.5280 - val_loss: 0.6858
Epoch 5/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 269ms/step - accuracy: 0.5244 - loss: 0.6895 - val_acc
uracy: 0.5160 - val_loss: 0.6898
Epoch 6/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 287ms/step - accuracy: 0.5278 - loss: 0.6984 - val_acc
uracy: 0.5000 - val_loss: 0.6942
Epoch 7/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 251ms/step - accuracy: 0.5073 - loss: 0.7110 - val_acc
uracy: 0.5340 - val_loss: 0.6888
Epoch 8/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 256ms/step - accuracy: 0.4763 - loss: 0.6937 - val_acc
uracy: 0.5600 - val_loss: 0.6897
Epoch 9/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 21s 260ms/step - accuracy: 0.5444 - loss: 0.6914 - val_acc
uracy: 0.6000 - val_loss: 0.6813
Epoch 10/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 20s 256ms/step - accuracy: 0.5368 - loss: 0.6885 - val_acc
uracy: 0.5200 - val_loss: 0.6833
```

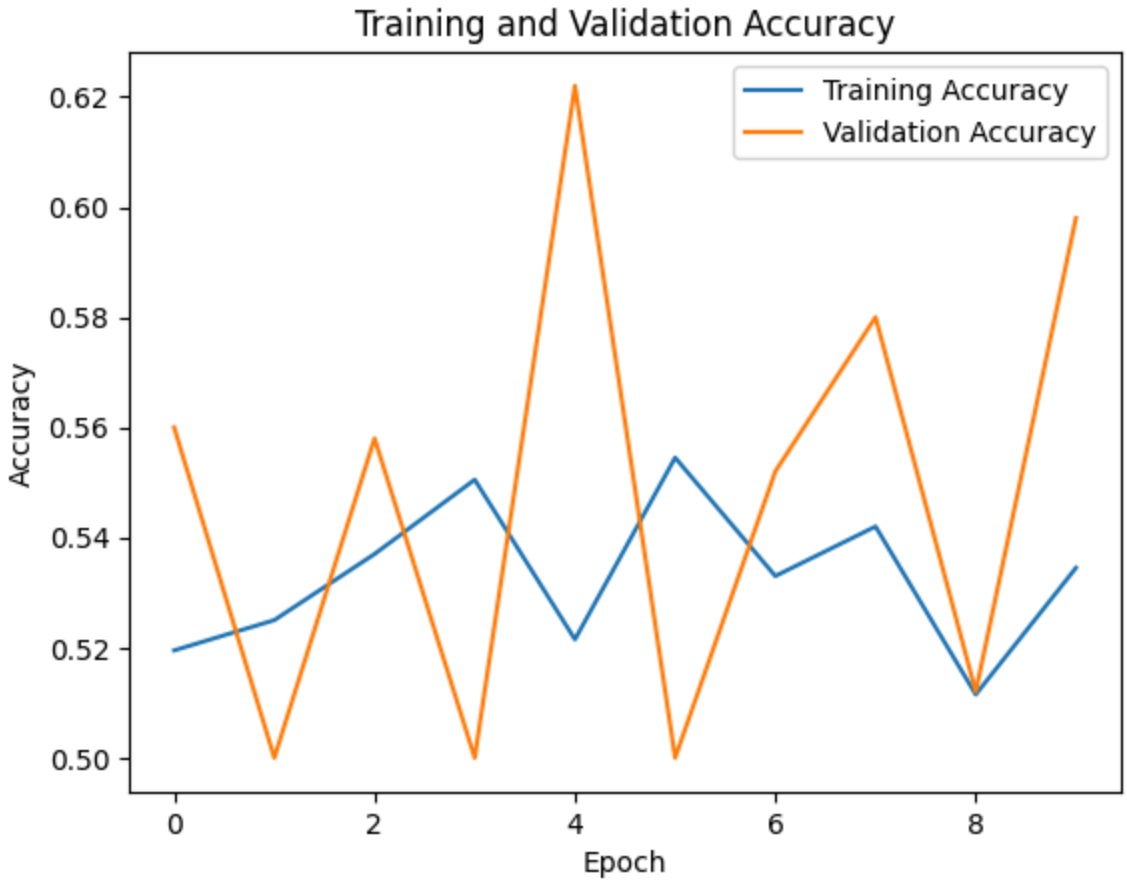


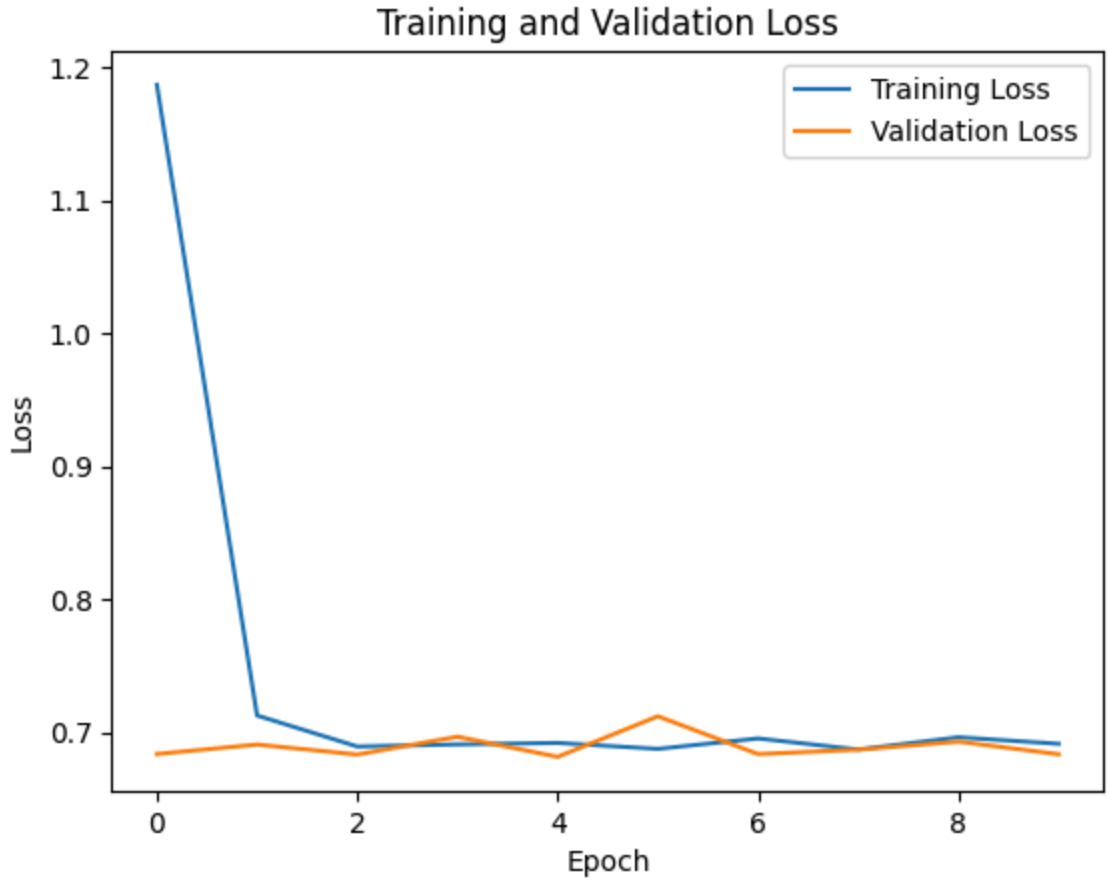



```
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 101ms/step - accuracy: 0.5531 - loss: 0.6825
Test accuracy (ResNet, 1000 samples): 0.5400000214576721
```

In [39]:
```
evaluate_pretrained_model(2000, image_size, batch_size, validation_samples, test_sampl
```

```
Found 2000 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 41s 296ms/step - accuracy: 0.5153 - loss: 1.6926 - val_a
ccuracy: 0.5600 - val_loss: 0.6837
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 32s 238ms/step - accuracy: 0.5363 - loss: 0.7121 - val_a
ccuracy: 0.5000 - val_loss: 0.6908
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 42s 245ms/step - accuracy: 0.5430 - loss: 0.6901 - val_a
ccuracy: 0.5580 - val_loss: 0.6833
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 41s 244ms/step - accuracy: 0.5581 - loss: 0.6914 - val_a
ccuracy: 0.5000 - val_loss: 0.6967
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 41s 242ms/step - accuracy: 0.5044 - loss: 0.6951 - val_a
ccuracy: 0.6220 - val_loss: 0.6816
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 42s 255ms/step - accuracy: 0.5474 - loss: 0.6872 - val_a
ccuracy: 0.5000 - val_loss: 0.7121
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 229ms/step - accuracy: 0.5048 - loss: 0.7012 - val_a
ccuracy: 0.5520 - val_loss: 0.6836
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 43s 253ms/step - accuracy: 0.5279 - loss: 0.6857 - val_a
ccuracy: 0.5800 - val_loss: 0.6870
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 27s 253ms/step - accuracy: 0.5041 - loss: 0.6992 - val_a
ccuracy: 0.5120 - val_loss: 0.6930
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 226ms/step - accuracy: 0.5062 - loss: 0.6934 - val_a
ccuracy: 0.5980 - val_loss: 0.6835
```

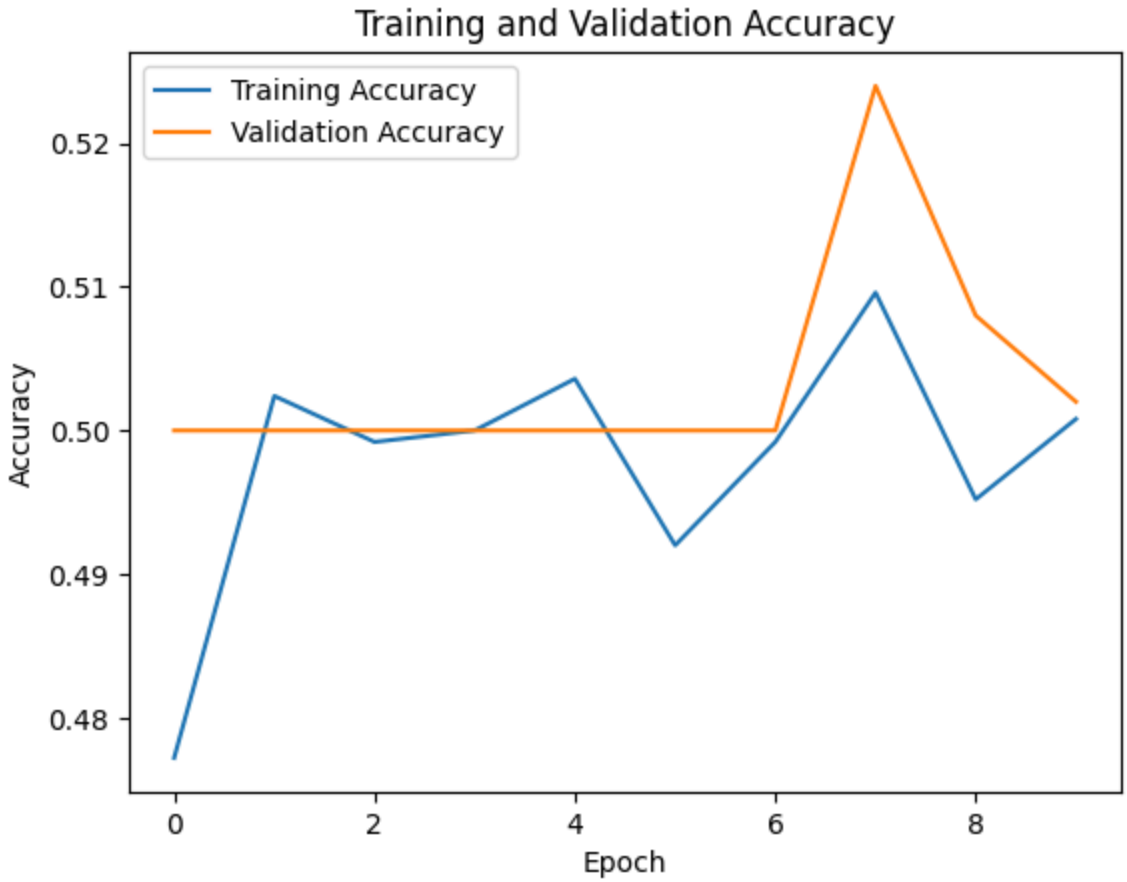


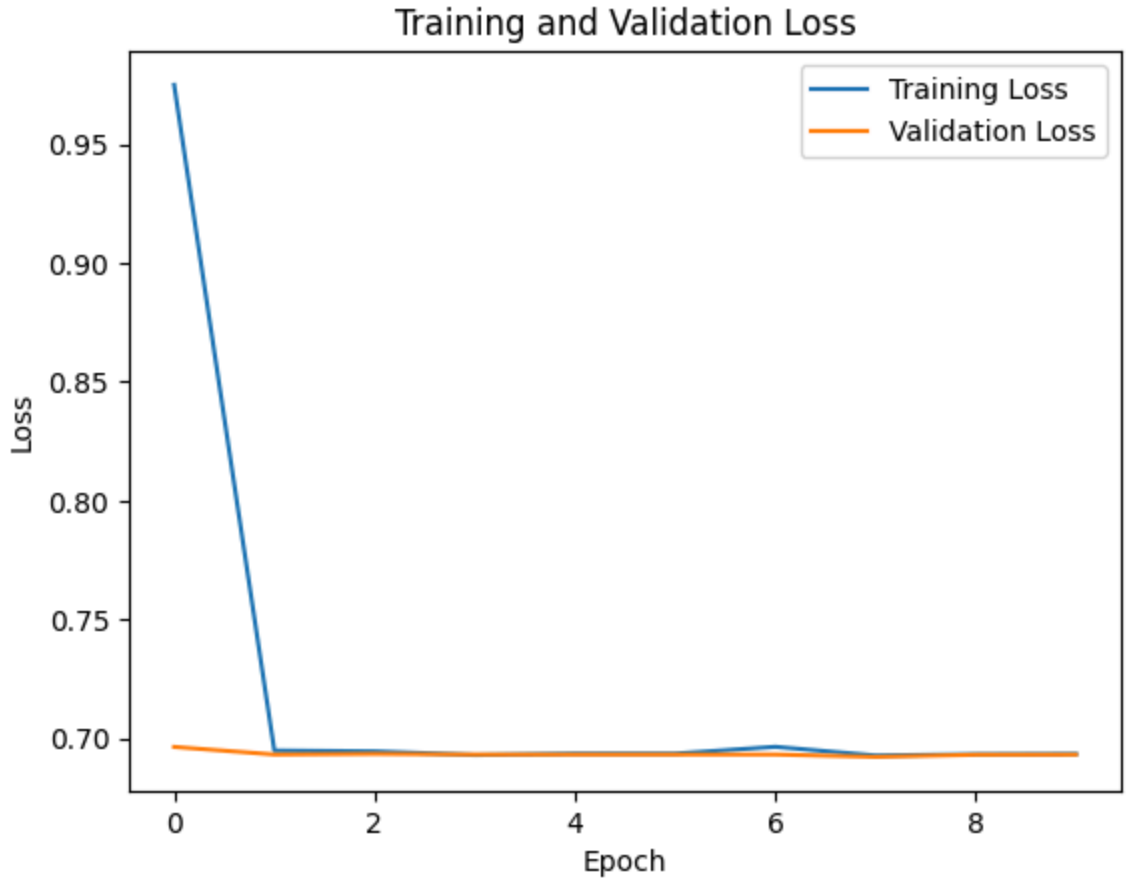



```
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 93ms/step - accuracy: 0.5487 - loss: 0.6881
Test accuracy (ResNet, 2000 samples): 0.5759999752044678
```

In [40]:
```
evaluate_pretrained_model(2500, image_size, batch_size, validation_samples, test_sampl
```

```
Found 2500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 47s 274ms/step - accuracy: 0.4718 - loss: 1.7947 - val_a
ccuracy: 0.5000 - val_loss: 0.6964
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 32s 243ms/step - accuracy: 0.5183 - loss: 0.6943 - val_a
ccuracy: 0.5000 - val_loss: 0.6931
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 40s 238ms/step - accuracy: 0.4821 - loss: 0.6941 - val_a
ccuracy: 0.5000 - val_loss: 0.6934
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 31s 235ms/step - accuracy: 0.5195 - loss: 0.6926 - val_a
ccuracy: 0.5000 - val_loss: 0.6932
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 44s 261ms/step - accuracy: 0.5194 - loss: 0.6933 - val_a
ccuracy: 0.5000 - val_loss: 0.6932
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 32s 242ms/step - accuracy: 0.4797 - loss: 0.6938 - val_a
ccuracy: 0.5000 - val_loss: 0.6931
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 31s 234ms/step - accuracy: 0.4996 - loss: 0.6944 - val_a
ccuracy: 0.5000 - val_loss: 0.6931
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 30s 231ms/step - accuracy: 0.5166 - loss: 0.6930 - val_a
ccuracy: 0.5240 - val_loss: 0.6922
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 30s 231ms/step - accuracy: 0.4925 - loss: 0.6929 - val_a
ccuracy: 0.5080 - val_loss: 0.6931
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 33s 255ms/step - accuracy: 0.5034 - loss: 0.6932 - val_a
ccuracy: 0.5020 - val_loss: 0.6931
```






```
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.5174 - loss: 0.6925
Test accuracy (ResNet, 2500 samples): 0.5120000243186951
```