Vivek Convultion Networking Assignment

Report: Exploring the Relationship Between Training Sample Size and Choice of Network for Cats vs. Dogs Classification

Objective

The goal of this project was to investigate the impact of training sample size on the performance of convolutional neural networks (CNNs) trained from scratch versus using pretrained networks. Specifically, we aimed to:

1. Apply CNNs to image data and explore overfitting reduction techniques.
2. Compare the performance of models trained from scratch and pretrained models across different training sample sizes (1500, 2000, and 2500).
3. Optimize model architectures and techniques to achieve the best performance.

Methodology

I trained two types of models on a subset of the Cats vs. Dogs dataset:

1. Scratch Model: A CNN model built from scratch with multiple convolutional layers, pooling layers, and dropout regularization to reduce overfitting.
2. Pretrained Model: A transfer learning approach using the ResNet50 architecture pretrained on the ImageNet dataset, with fine-tuning applied on the final layers.

Both models were trained on three different sample sizes: 1500, 2000, and 2500 training samples, with 500 validation and 500 test samples for each scenario. Techniques such as data augmentation, dropout, and regularization were applied to mitigate overfitting.

Results

The following table summarizes the performance of both models across different sample sizes:

Sample Size	Model Type	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
1500	Scratch	0.6007	0.562	1.8124	1.8398
1500	Pretrained	0.9473	0.966	0.1224	0.0985
2000	Scratch	0.6375	0.552	1.7352	2.1645
2000	Pretrained	0.9415	0.958	0.1472	0.1302
2500	Scratch	0.6264	0.616	1.3995	1.3633
2500	Pretrained	0.9448	0.966	0.1357	0.0932

Analysis and Key Findings

1. Performance of Scratch Models:
- The scratch models exhibited lower training and validation accuracy compared to the pretrained models across all sample sizes. The best validation accuracy achieved by the scratch model was 0.616 for the 2500 sample size.
- The models trained from scratch were prone to overfitting, as indicated by a significant gap between training accuracy and validation accuracy, particularly with the 2000 and 2500 sample sizes.
- Regularization techniques (such as dropout and data augmentation) were effective at mitigating overfitting, but their overall impact was limited compared to the performance of pretrained models.

2. Performance of Pretrained Models:
- Pretrained models consistently outperformed the scratch models, achieving validation accuracies of 0.958 to 0.966 across all sample sizes, with minimal overfitting. The validation losses were also significantly lower than those of the scratch models.
- Pretrained models demonstrated that even with smaller datasets (1500 samples), transfer learning provided excellent generalization, confirming that pretrained networks are highly effective for small to moderately sized datasets.

3. Impact of Sample Size:
- Increasing the training sample size improved the performance of both scratch and pretrained models, but the improvement was marginal for pretrained models since they already achieved high accuracy even with smaller sample sizes.
- Scratch models benefitted slightly more from larger sample sizes, but still underperformed compared to pretrained models. The training process for scratch models appeared to stabilize around the 2500 sample size but did not close the performance gap.

4. Overfitting and Regularization:
- Overfitting was more pronounced in scratch models, which had lower validation accuracy and higher validation loss compared to training loss. This suggests that scratch models required larger datasets and additional regularization techniques to improve generalization.
- Pretrained models, benefiting from their pretrained weights, were less susceptible to overfitting and required fewer overfitting prevention techniques.

Conclusion

In conclusion, this project highlights a clear relationship between training sample size and the choice of network:

- Pretrained Models: Transfer learning with pretrained models (like ResNet50) is highly effective, especially when working with small to moderate sample sizes. Pretrained models are able to generalize well with less data and require less fine-tuning compared to scratch models.

- Scratch Models: While scratch models can achieve reasonable performance, they require much larger datasets to avoid overfitting and reach the same level of generalization as pretrained models. For smaller datasets, scratch models are prone to overfitting and underperformance, even with regularization techniques applied.

Based on these findings, pretrained models are recommended when working with smaller datasets due to their superior performance and generalization ability. Scratch models may be suitable for larger datasets or when a custom architecture is required, but they demand more careful tuning and regularization.

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, Bat
from tensorflow.keras.regularizers import l2
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.models import Model
```

Loading Dataset

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
ount("/content/drive", force_remount=True).
```

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 [4]:
```
# 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 [5]:
```
# Function to split and organize dataset
def split_data_folders(base_dir, cat_images, dog_images, train_samples, validation_sam
    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)
```

```
    random.shuffle(cat_images)
    random.shuffle(dog_images)

    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 [6]:

```
# Image augmentations and data generators
def create_data_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 Optimized Scratch Model

In [7]:
```
# Improved Scratch Model
def build_optimized_scratch_model(image_size):
    model = Sequential([
        Conv2D(64, (3, 3), activation='relu', input_shape=(image_size[0], image_size[1
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu', kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(512, (3, 3), activation='relu', kernel_regularizer=l2(0.001)),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(512, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.5),  # Prevent overfitting
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Inception Model

In [8]:
```
# Pretrained InceptionV3 Model
def build_inception_pretrained_model(image_size):
    base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(image
    for layer in base_model.layers:
        layer.trainable = False  # Freeze the base model layers
    x = GlobalAveragePooling2D()(base_model.output)
    x = Dense(512, activation='relu')(x)
    Dropout(0.5)
    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
```

In [9]:
```
# Train the model and plot results
def train_and_evaluate_model(model, train_generator, validation_generator, epochs):
    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=validation_generator
    )
    return history
```

In [10]:
```
# Plot performance metrics
def plot_training_metrics(history):
```

```python
    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 [23]:

```python
# Plot accuracy comparison for scratch vs pretrained model for different sample sizes
def plot_accuracy_comparison(results, sample_size):
    # Access the history attributes correctly
    scratch_acc = results[sample_size]['scratch']['history'].history['accuracy']
    scratch_val_acc = results[sample_size]['scratch']['history'].history['val_accuracy
    pretrained_acc = results[sample_size]['pretrained']['history'].history['accuracy']
    pretrained_val_acc = results[sample_size]['pretrained']['history'].history['val_ac

    plt.plot(scratch_acc, label='Scratch Model Training Accuracy')
    plt.plot(scratch_val_acc, label='Scratch Model Validation Accuracy')
    plt.plot(pretrained_acc, label='Pretrained Model Training Accuracy')
    plt.plot(pretrained_val_acc, label='Pretrained Model Validation Accuracy')
    plt.title(f'Training vs Validation Accuracy for {sample_size} Samples')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

In [24]:

```python
# Plot loss comparison for scratch vs pretrained model for different sample sizes
def plot_loss_comparison(results, sample_size):
    # Access the history attributes correctly
    scratch_loss = results[sample_size]['scratch']['history'].history['loss']
    scratch_val_loss = results[sample_size]['scratch']['history'].history['val_loss']
    pretrained_loss = results[sample_size]['pretrained']['history'].history['loss']
    pretrained_val_loss = results[sample_size]['pretrained']['history'].history['val_l

    plt.plot(scratch_loss, label='Scratch Model Training Loss')
    plt.plot(scratch_val_loss, label='Scratch Model Validation Loss')
    plt.plot(pretrained_loss, label='Pretrained Model Training Loss')
    plt.plot(pretrained_val_loss, label='Pretrained Model Validation Loss')
    plt.title(f'Training vs Validation Loss for {sample_size} Samples')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```

In [15]:

```python
# Setting up parameters
image_size = (150, 150)
batch_size = 32
validation_samples = 500
test_samples = 500
cat_images = os.listdir(cat_folder_path)
dog_images = os.listdir(dog_folder_path)
```

In [16]:
```python
# Train and validate models for both 1500 and 2000 sample sizes
sample_sizes = [1500, 2000,2500]
results = {}
```

In [17]:
```python
scratch_model = build_optimized_scratch_model(image_size)
scratch_model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:1
07: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When u
sing Sequential models, prefer using an `Input(shape)` object as the first layer in t
he model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Pa
conv2d (Conv2D)	(None, 148, 148, 64)	
batch_normalization (BatchNormalization)	(None, 148, 148, 64)	
max_pooling2d (MaxPooling2D)	(None, 74, 74, 64)	
conv2d_1 (Conv2D)	(None, 72, 72, 128)	7
batch_normalization_1 (BatchNormalization)	(None, 72, 72, 128)	
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 128)	
conv2d_2 (Conv2D)	(None, 34, 34, 256)	29
batch_normalization_2 (BatchNormalization)	(None, 34, 34, 256)	
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 256)	
conv2d_3 (Conv2D)	(None, 15, 15, 512)	1,18
batch_normalization_3 (BatchNormalization)	(None, 15, 15, 512)	
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 512)	
flatten (Flatten)	(None, 25088)	
dense (Dense)	(None, 512)	12,84
dropout (Dropout)	(None, 512)	
dense_1 (Dense)	(None, 1)	

Total params: 14,400,897 (54.94 MB)

Trainable params: 14,398,977 (54.93 MB)

In [18]:
```
pretrained_model = build_inception_pretrained_model(image_size)
pretrained_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/in
ception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connectec
input_layer_1 (InputLayer)	(None, 150, 150, 3)	0	-
conv2d_4 (Conv2D)	(None, 74, 74, 32)	864	input_lay
batch_normalization_4 (BatchNormalization)	(None, 74, 74, 32)	96	conv2d_4
activation (Activation)	(None, 74, 74, 32)	0	batch_noı
conv2d_5 (Conv2D)	(None, 72, 72, 32)	9,216	activatic
batch_normalization_5 (BatchNormalization)	(None, 72, 72, 32)	96	conv2d_5
activation_1 (Activation)	(None, 72, 72, 32)	0	batch_noı
conv2d_6 (Conv2D)	(None, 72, 72, 64)	18,432	activatic
batch_normalization_6 (BatchNormalization)	(None, 72, 72, 64)	192	conv2d_6
activation_2 (Activation)	(None, 72, 72, 64)	0	batch_noı
max_pooling2d_4 (MaxPooling2D)	(None, 35, 35, 64)	0	activatic
conv2d_7 (Conv2D)	(None, 35, 35, 80)	5,120	max_pooli
batch_normalization_7 (BatchNormalization)	(None, 35, 35, 80)	240	conv2d_7
activation_3 (Activation)	(None, 35, 35, 80)	0	batch_noı
conv2d_8 (Conv2D)	(None, 33, 33, 192)	138,240	activatic
batch_normalization_8 (BatchNormalization)	(None, 33, 33, 192)	576	conv2d_8
activation_4 (Activation)	(None, 33, 33, 192)	0	batch_noı
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 192)	0	activatic
conv2d_12 (Conv2D)	(None, 16, 16, 64)	12,288	max_pooli
batch_normalization_12 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_12
activation_8 (Activation)	(None, 16, 16, 64)	0	batch_noı
conv2d_10 (Conv2D)	(None, 16, 16, 48)	9,216	max_pooli
conv2d_13 (Conv2D)	(None, 16, 16, 96)	55,296	activatic

batch_normalization_10 (BatchNormalization)	(None, 16, 16, 48)	144	conv2d_1(
batch_normalization_13 (BatchNormalization)	(None, 16, 16, 96)	288	conv2d_1:
activation_6 (Activation)	(None, 16, 16, 48)	0	batch_nor
activation_9 (Activation)	(None, 16, 16, 96)	0	batch_nor
average_pooling2d (AveragePooling2D)	(None, 16, 16, 192)	0	max_pool:
conv2d_9 (Conv2D)	(None, 16, 16, 64)	12,288	max_pool:
conv2d_11 (Conv2D)	(None, 16, 16, 64)	76,800	activatic
conv2d_14 (Conv2D)	(None, 16, 16, 96)	82,944	activatic
conv2d_15 (Conv2D)	(None, 16, 16, 32)	6,144	average_
batch_normalization_9 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_9
batch_normalization_11 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_1:
batch_normalization_14 (BatchNormalization)	(None, 16, 16, 96)	288	conv2d_1
batch_normalization_15 (BatchNormalization)	(None, 16, 16, 32)	96	conv2d_1!
activation_5 (Activation)	(None, 16, 16, 64)	0	batch_nor
activation_7 (Activation)	(None, 16, 16, 64)	0	batch_nor
activation_10 (Activation)	(None, 16, 16, 96)	0	batch_nor
activation_11 (Activation)	(None, 16, 16, 32)	0	batch_nor
mixed0 (Concatenate)	(None, 16, 16, 256)	0	activatic activatic activatic activatic
conv2d_19 (Conv2D)	(None, 16, 16, 64)	16,384	mixed0[0]
batch_normalization_19 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_1!
activation_15 (Activation)	(None, 16, 16, 64)	0	batch_nor

conv2d_17 (Conv2D)	(None, 16, 16, 48)	12,288	mixed0[0
conv2d_20 (Conv2D)	(None, 16, 16, 96)	55,296	activatio
batch_normalization_17 (BatchNormalization)	(None, 16, 16, 48)	144	conv2d_1
batch_normalization_20 (BatchNormalization)	(None, 16, 16, 96)	288	conv2d_2
activation_13 (Activation)	(None, 16, 16, 48)	0	batch_nor
activation_16 (Activation)	(None, 16, 16, 96)	0	batch_nor
average_pooling2d_1 (AveragePooling2D)	(None, 16, 16, 256)	0	mixed0[0
conv2d_16 (Conv2D)	(None, 16, 16, 64)	16,384	mixed0[0
conv2d_18 (Conv2D)	(None, 16, 16, 64)	76,800	activatio
conv2d_21 (Conv2D)	(None, 16, 16, 96)	82,944	activatio
conv2d_22 (Conv2D)	(None, 16, 16, 64)	16,384	average_
batch_normalization_16 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_1
batch_normalization_18 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_1
batch_normalization_21 (BatchNormalization)	(None, 16, 16, 96)	288	conv2d_2
batch_normalization_22 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_2
activation_12 (Activation)	(None, 16, 16, 64)	0	batch_nor
activation_14 (Activation)	(None, 16, 16, 64)	0	batch_nor
activation_17 (Activation)	(None, 16, 16, 96)	0	batch_nor
activation_18 (Activation)	(None, 16, 16, 64)	0	batch_nor
mixed1 (Concatenate)	(None, 16, 16, 288)	0	activatio activatio activatio activatio
conv2d_26 (Conv2D)	(None, 16, 16, 64)	18,432	mixed1[0

batch_normalization_26 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_26
activation_22 (Activation)	(None, 16, 16, 64)	0	batch_nor
conv2d_24 (Conv2D)	(None, 16, 16, 48)	13,824	mixed1[0
conv2d_27 (Conv2D)	(None, 16, 16, 96)	55,296	activatio
batch_normalization_24 (BatchNormalization)	(None, 16, 16, 48)	144	conv2d_24
batch_normalization_27 (BatchNormalization)	(None, 16, 16, 96)	288	conv2d_27
activation_20 (Activation)	(None, 16, 16, 48)	0	batch_nor
activation_23 (Activation)	(None, 16, 16, 96)	0	batch_nor
average_pooling2d_2 (AveragePooling2D)	(None, 16, 16, 288)	0	mixed1[0
conv2d_23 (Conv2D)	(None, 16, 16, 64)	18,432	mixed1[0
conv2d_25 (Conv2D)	(None, 16, 16, 64)	76,800	activatio
conv2d_28 (Conv2D)	(None, 16, 16, 96)	82,944	activatio
conv2d_29 (Conv2D)	(None, 16, 16, 64)	18,432	average_p
batch_normalization_23 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_23
batch_normalization_25 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_25
batch_normalization_28 (BatchNormalization)	(None, 16, 16, 96)	288	conv2d_28
batch_normalization_29 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_29
activation_19 (Activation)	(None, 16, 16, 64)	0	batch_nor
activation_21 (Activation)	(None, 16, 16, 64)	0	batch_nor
activation_24 (Activation)	(None, 16, 16, 96)	0	batch_nor
activation_25 (Activation)	(None, 16, 16, 64)	0	batch_nor

mixed2 (Concatenate)	(None, 16, 16, 288)	0	activatic activatic activatic activatic
conv2d_31 (Conv2D)	(None, 16, 16, 64)	18,432	mixed2[0
batch_normalization_31 (BatchNormalization)	(None, 16, 16, 64)	192	conv2d_3
activation_27 (Activation)	(None, 16, 16, 64)	0	batch_no
conv2d_32 (Conv2D)	(None, 16, 16, 96)	55,296	activatic
batch_normalization_32 (BatchNormalization)	(None, 16, 16, 96)	288	conv2d_3
activation_28 (Activation)	(None, 16, 16, 96)	0	batch_no
conv2d_30 (Conv2D)	(None, 7, 7, 384)	995,328	mixed2[0
conv2d_33 (Conv2D)	(None, 7, 7, 96)	82,944	activatic
batch_normalization_30 (BatchNormalization)	(None, 7, 7, 384)	1,152	conv2d_3
batch_normalization_33 (BatchNormalization)	(None, 7, 7, 96)	288	conv2d_3
activation_26 (Activation)	(None, 7, 7, 384)	0	batch_no
activation_29 (Activation)	(None, 7, 7, 96)	0	batch_no
max_pooling2d_6 (MaxPooling2D)	(None, 7, 7, 288)	0	mixed2[0
mixed3 (Concatenate)	(None, 7, 7, 768)	0	activatic activatic max_pooli
conv2d_38 (Conv2D)	(None, 7, 7, 128)	98,304	mixed3[0
batch_normalization_38 (BatchNormalization)	(None, 7, 7, 128)	384	conv2d_3
activation_34 (Activation)	(None, 7, 7, 128)	0	batch_no
conv2d_39 (Conv2D)	(None, 7, 7, 128)	114,688	activatic
batch_normalization_39 (BatchNormalization)	(None, 7, 7, 128)	384	conv2d_3

activation_35 (Activation)	(None, 7, 7, 128)	0	batch_nor
conv2d_35 (Conv2D)	(None, 7, 7, 128)	98,304	mixed3[0
conv2d_40 (Conv2D)	(None, 7, 7, 128)	114,688	activatic
batch_normalization_35 (BatchNormalization)	(None, 7, 7, 128)	384	conv2d_35
batch_normalization_40 (BatchNormalization)	(None, 7, 7, 128)	384	conv2d_4(
activation_31 (Activation)	(None, 7, 7, 128)	0	batch_nor
activation_36 (Activation)	(None, 7, 7, 128)	0	batch_nor
conv2d_36 (Conv2D)	(None, 7, 7, 128)	114,688	activatic
conv2d_41 (Conv2D)	(None, 7, 7, 128)	114,688	activatic
batch_normalization_36 (BatchNormalization)	(None, 7, 7, 128)	384	conv2d_3(
batch_normalization_41 (BatchNormalization)	(None, 7, 7, 128)	384	conv2d_4:
activation_32 (Activation)	(None, 7, 7, 128)	0	batch_nor
activation_37 (Activation)	(None, 7, 7, 128)	0	batch_nor
average_pooling2d_3 (AveragePooling2D)	(None, 7, 7, 768)	0	mixed3[0
conv2d_34 (Conv2D)	(None, 7, 7, 192)	147,456	mixed3[0
conv2d_37 (Conv2D)	(None, 7, 7, 192)	172,032	activatic
conv2d_42 (Conv2D)	(None, 7, 7, 192)	172,032	activatic
conv2d_43 (Conv2D)	(None, 7, 7, 192)	147,456	average_
batch_normalization_34 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_34
batch_normalization_37 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_3
batch_normalization_42 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_4:
batch_normalization_43	(None, 7, 7, 192)	576	conv2d_4:

(BatchNormalization)			
activation_30 (Activation)	(None, 7, 7, 192)	0	batch_nor
activation_33 (Activation)	(None, 7, 7, 192)	0	batch_nor
activation_38 (Activation)	(None, 7, 7, 192)	0	batch_nor
activation_39 (Activation)	(None, 7, 7, 192)	0	batch_nor
mixed4 (Concatenate)	(None, 7, 7, 768)	0	activatic activatic activatic activatic
conv2d_48 (Conv2D)	(None, 7, 7, 160)	122,880	mixed4[0]
batch_normalization_48 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_48
activation_44 (Activation)	(None, 7, 7, 160)	0	batch_nor
conv2d_49 (Conv2D)	(None, 7, 7, 160)	179,200	activatic
batch_normalization_49 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_49
activation_45 (Activation)	(None, 7, 7, 160)	0	batch_nor
conv2d_45 (Conv2D)	(None, 7, 7, 160)	122,880	mixed4[0]
conv2d_50 (Conv2D)	(None, 7, 7, 160)	179,200	activatic
batch_normalization_45 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_45
batch_normalization_50 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_50
activation_41 (Activation)	(None, 7, 7, 160)	0	batch_nor
activation_46 (Activation)	(None, 7, 7, 160)	0	batch_nor
conv2d_46 (Conv2D)	(None, 7, 7, 160)	179,200	activatic
conv2d_51 (Conv2D)	(None, 7, 7, 160)	179,200	activatic
batch_normalization_46 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_46

batch_normalization_51 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_5
activation_42 (Activation)	(None, 7, 7, 160)	0	batch_nor
activation_47 (Activation)	(None, 7, 7, 160)	0	batch_nor
average_pooling2d_4 (AveragePooling2D)	(None, 7, 7, 768)	0	mixed4[0
conv2d_44 (Conv2D)	(None, 7, 7, 192)	147,456	mixed4[0
conv2d_47 (Conv2D)	(None, 7, 7, 192)	215,040	activatic
conv2d_52 (Conv2D)	(None, 7, 7, 192)	215,040	activatic
conv2d_53 (Conv2D)	(None, 7, 7, 192)	147,456	average_
batch_normalization_44 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_4
batch_normalization_47 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_4
batch_normalization_52 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_5
batch_normalization_53 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_5
activation_40 (Activation)	(None, 7, 7, 192)	0	batch_nor
activation_43 (Activation)	(None, 7, 7, 192)	0	batch_nor
activation_48 (Activation)	(None, 7, 7, 192)	0	batch_nor
activation_49 (Activation)	(None, 7, 7, 192)	0	batch_nor
mixed5 (Concatenate)	(None, 7, 7, 768)	0	activatic activatic activatic activatic
conv2d_58 (Conv2D)	(None, 7, 7, 160)	122,880	mixed5[0
batch_normalization_58 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_5
activation_54 (Activation)	(None, 7, 7, 160)	0	batch_nor

conv2d_59 (Conv2D)	(None, 7, 7, 160)	179,200	activatio
batch_normalization_59 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_5
activation_55 (Activation)	(None, 7, 7, 160)	0	batch_nor
conv2d_55 (Conv2D)	(None, 7, 7, 160)	122,880	mixed5[0
conv2d_60 (Conv2D)	(None, 7, 7, 160)	179,200	activatio
batch_normalization_55 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_5
batch_normalization_60 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_6
activation_51 (Activation)	(None, 7, 7, 160)	0	batch_nor
activation_56 (Activation)	(None, 7, 7, 160)	0	batch_nor
conv2d_56 (Conv2D)	(None, 7, 7, 160)	179,200	activatio
conv2d_61 (Conv2D)	(None, 7, 7, 160)	179,200	activatio
batch_normalization_56 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_5
batch_normalization_61 (BatchNormalization)	(None, 7, 7, 160)	480	conv2d_6
activation_52 (Activation)	(None, 7, 7, 160)	0	batch_nor
activation_57 (Activation)	(None, 7, 7, 160)	0	batch_nor
average_pooling2d_5 (AveragePooling2D)	(None, 7, 7, 768)	0	mixed5[0
conv2d_54 (Conv2D)	(None, 7, 7, 192)	147,456	mixed5[0
conv2d_57 (Conv2D)	(None, 7, 7, 192)	215,040	activatio
conv2d_62 (Conv2D)	(None, 7, 7, 192)	215,040	activatio
conv2d_63 (Conv2D)	(None, 7, 7, 192)	147,456	average_
batch_normalization_54 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_5
batch_normalization_57 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_5

batch_normalization_62 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_6
batch_normalization_63 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_6
activation_50 (Activation)	(None, 7, 7, 192)	0	batch_no
activation_53 (Activation)	(None, 7, 7, 192)	0	batch_no
activation_58 (Activation)	(None, 7, 7, 192)	0	batch_no
activation_59 (Activation)	(None, 7, 7, 192)	0	batch_no
mixed6 (Concatenate)	(None, 7, 7, 768)	0	activati activati activati activati
conv2d_68 (Conv2D)	(None, 7, 7, 192)	147,456	mixed6[0
batch_normalization_68 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_6
activation_64 (Activation)	(None, 7, 7, 192)	0	batch_no
conv2d_69 (Conv2D)	(None, 7, 7, 192)	258,048	activati
batch_normalization_69 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_6
activation_65 (Activation)	(None, 7, 7, 192)	0	batch_no
conv2d_65 (Conv2D)	(None, 7, 7, 192)	147,456	mixed6[0
conv2d_70 (Conv2D)	(None, 7, 7, 192)	258,048	activati
batch_normalization_65 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_6
batch_normalization_70 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_7
activation_61 (Activation)	(None, 7, 7, 192)	0	batch_no
activation_66 (Activation)	(None, 7, 7, 192)	0	batch_no
conv2d_66 (Conv2D)	(None, 7, 7, 192)	258,048	activati

conv2d_71 (Conv2D)	(None, 7, 7, 192)	258,048	activati
batch_normalization_66 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_6
batch_normalization_71 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_7
activation_62 (Activation)	(None, 7, 7, 192)	0	batch_no
activation_67 (Activation)	(None, 7, 7, 192)	0	batch_no
average_pooling2d_6 (AveragePooling2D)	(None, 7, 7, 768)	0	mixed6[0
conv2d_64 (Conv2D)	(None, 7, 7, 192)	147,456	mixed6[0
conv2d_67 (Conv2D)	(None, 7, 7, 192)	258,048	activati
conv2d_72 (Conv2D)	(None, 7, 7, 192)	258,048	activati
conv2d_73 (Conv2D)	(None, 7, 7, 192)	147,456	average_
batch_normalization_64 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_6
batch_normalization_67 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_6
batch_normalization_72 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_7
batch_normalization_73 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_7
activation_60 (Activation)	(None, 7, 7, 192)	0	batch_no
activation_63 (Activation)	(None, 7, 7, 192)	0	batch_no
activation_68 (Activation)	(None, 7, 7, 192)	0	batch_no
activation_69 (Activation)	(None, 7, 7, 192)	0	batch_no
mixed7 (Concatenate)	(None, 7, 7, 768)	0	activati activati activati activati
conv2d_76 (Conv2D)	(None, 7, 7, 192)	147,456	mixed7[0

batch_normalization_76 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_76
activation_72 (Activation)	(None, 7, 7, 192)	0	batch_nor
conv2d_77 (Conv2D)	(None, 7, 7, 192)	258,048	activatio
batch_normalization_77 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_77
activation_73 (Activation)	(None, 7, 7, 192)	0	batch_nor
conv2d_74 (Conv2D)	(None, 7, 7, 192)	147,456	mixed7[0]
conv2d_78 (Conv2D)	(None, 7, 7, 192)	258,048	activatio
batch_normalization_74 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_74
batch_normalization_78 (BatchNormalization)	(None, 7, 7, 192)	576	conv2d_78
activation_70 (Activation)	(None, 7, 7, 192)	0	batch_nor
activation_74 (Activation)	(None, 7, 7, 192)	0	batch_nor
conv2d_75 (Conv2D)	(None, 3, 3, 320)	552,960	activatio
conv2d_79 (Conv2D)	(None, 3, 3, 192)	331,776	activatio
batch_normalization_75 (BatchNormalization)	(None, 3, 3, 320)	960	conv2d_75
batch_normalization_79 (BatchNormalization)	(None, 3, 3, 192)	576	conv2d_79
activation_71 (Activation)	(None, 3, 3, 320)	0	batch_nor
activation_75 (Activation)	(None, 3, 3, 192)	0	batch_nor
max_pooling2d_7 (MaxPooling2D)	(None, 3, 3, 768)	0	mixed7[0]
mixed8 (Concatenate)	(None, 3, 3, 1280)	0	activatio activatio max_pooli
conv2d_84 (Conv2D)	(None, 3, 3, 448)	573,440	mixed8[0]
batch_normalization_84 (BatchNormalization)	(None, 3, 3, 448)	1,344	conv2d_84

activation_80 (Activation)	(None, 3, 3, 448)	0	batch_nor
conv2d_81 (Conv2D)	(None, 3, 3, 384)	491,520	mixed8[0
conv2d_85 (Conv2D)	(None, 3, 3, 384)	1,548,288	activatic
batch_normalization_81 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_81
batch_normalization_85 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_85
activation_77 (Activation)	(None, 3, 3, 384)	0	batch_nor
activation_81 (Activation)	(None, 3, 3, 384)	0	batch_nor
conv2d_82 (Conv2D)	(None, 3, 3, 384)	442,368	activatic
conv2d_83 (Conv2D)	(None, 3, 3, 384)	442,368	activatic
conv2d_86 (Conv2D)	(None, 3, 3, 384)	442,368	activatic
conv2d_87 (Conv2D)	(None, 3, 3, 384)	442,368	activatic
average_pooling2d_7 (AveragePooling2D)	(None, 3, 3, 1280)	0	mixed8[0
conv2d_80 (Conv2D)	(None, 3, 3, 320)	409,600	mixed8[0
batch_normalization_82 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_82
batch_normalization_83 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_83
batch_normalization_86 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_86
batch_normalization_87 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_87
conv2d_88 (Conv2D)	(None, 3, 3, 192)	245,760	average_p
batch_normalization_80 (BatchNormalization)	(None, 3, 3, 320)	960	conv2d_80
activation_78 (Activation)	(None, 3, 3, 384)	0	batch_nor
activation_79 (Activation)	(None, 3, 3, 384)	0	batch_nor
activation_82	(None, 3, 3, 384)	0	batch_nor

(Activation)			
activation_83 (Activation)	(None, 3, 3, 384)	0	batch_nor
batch_normalization_88 (BatchNormalization)	(None, 3, 3, 192)	576	conv2d_88
activation_76 (Activation)	(None, 3, 3, 320)	0	batch_nor
mixed9_0 (Concatenate)	(None, 3, 3, 768)	0	activatic activatic
concatenate (Concatenate)	(None, 3, 3, 768)	0	activatic activatic
activation_84 (Activation)	(None, 3, 3, 192)	0	batch_nor
mixed9 (Concatenate)	(None, 3, 3, 2048)	0	activatic mixed9_0[concatena activatic
conv2d_93 (Conv2D)	(None, 3, 3, 448)	917,504	mixed9[0]
batch_normalization_93 (BatchNormalization)	(None, 3, 3, 448)	1,344	conv2d_93
activation_89 (Activation)	(None, 3, 3, 448)	0	batch_nor
conv2d_90 (Conv2D)	(None, 3, 3, 384)	786,432	mixed9[0]
conv2d_94 (Conv2D)	(None, 3, 3, 384)	1,548,288	activatic
batch_normalization_90 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_90
batch_normalization_94 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_94
activation_86 (Activation)	(None, 3, 3, 384)	0	batch_nor
activation_90 (Activation)	(None, 3, 3, 384)	0	batch_nor
conv2d_91 (Conv2D)	(None, 3, 3, 384)	442,368	activatic
conv2d_92 (Conv2D)	(None, 3, 3, 384)	442,368	activatic
conv2d_95 (Conv2D)	(None, 3, 3, 384)	442,368	activatic
conv2d_96 (Conv2D)	(None, 3, 3, 384)	442,368	activatic

average_pooling2d_8 (AveragePooling2D)	(None, 3, 3, 2048)	0	mixed9[0
conv2d_89 (Conv2D)	(None, 3, 3, 320)	655,360	mixed9[0
batch_normalization_91 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_9
batch_normalization_92 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_9
batch_normalization_95 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_9
batch_normalization_96 (BatchNormalization)	(None, 3, 3, 384)	1,152	conv2d_9
conv2d_97 (Conv2D)	(None, 3, 3, 192)	393,216	average_
batch_normalization_89 (BatchNormalization)	(None, 3, 3, 320)	960	conv2d_8
activation_87 (Activation)	(None, 3, 3, 384)	0	batch_no
activation_88 (Activation)	(None, 3, 3, 384)	0	batch_no
activation_91 (Activation)	(None, 3, 3, 384)	0	batch_no
activation_92 (Activation)	(None, 3, 3, 384)	0	batch_no
batch_normalization_97 (BatchNormalization)	(None, 3, 3, 192)	576	conv2d_9
activation_85 (Activation)	(None, 3, 3, 320)	0	batch_no
mixed9_1 (Concatenate)	(None, 3, 3, 768)	0	activati activati
concatenate_1 (Concatenate)	(None, 3, 3, 768)	0	activati activati
activation_93 (Activation)	(None, 3, 3, 192)	0	batch_no
mixed10 (Concatenate)	(None, 3, 3, 2048)	0	activati mixed9_1[concaten activati
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	mixed10[

dense_2 (Dense)	(None, 512)	1,049,088	global_av
dense_3 (Dense)	(None, 1)	513	dense_2[(

Total params: 22,852,385 (87.17 MB)

Trainable params: 1,049,601 (4.00 MB)

Model Training

In [20]:

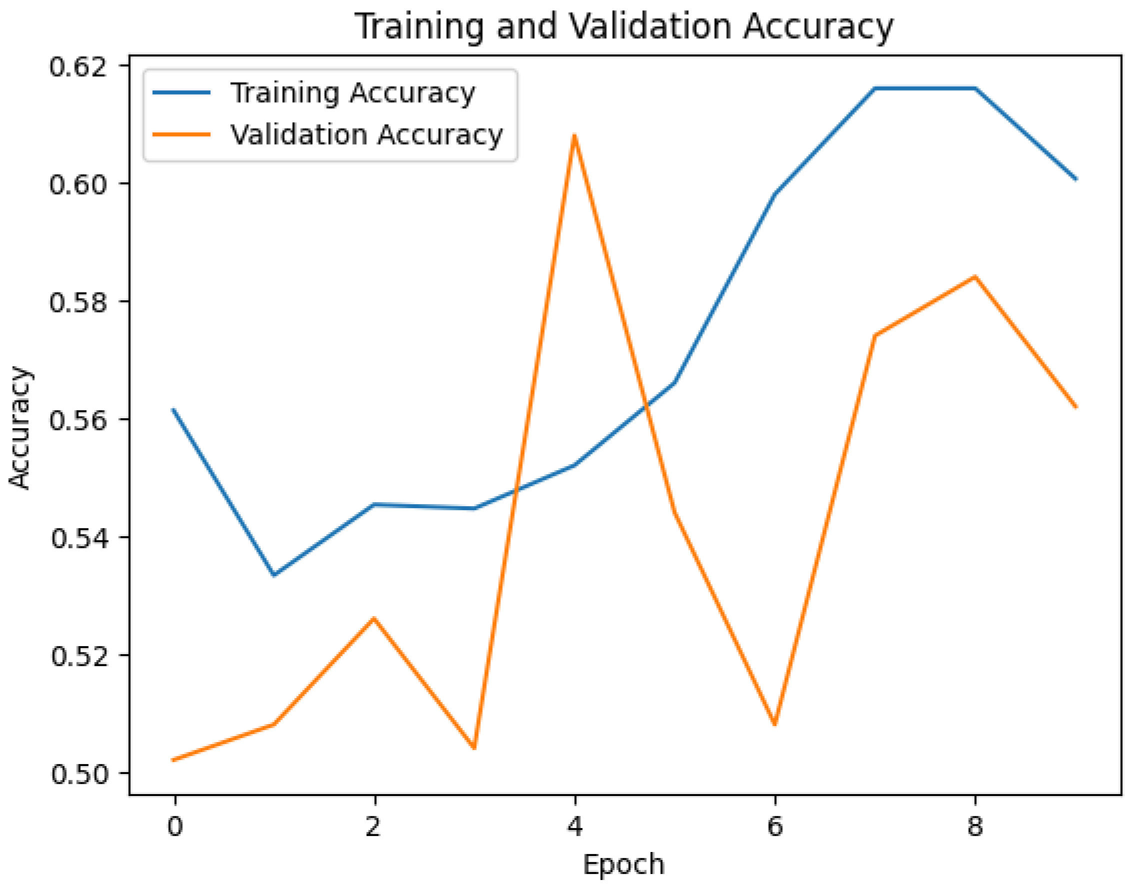
```
for sample_size in sample_sizes:
    # Split dataset for current sample size
    train_dir, validation_dir, test_dir = split_data_folders(extracted_dir_path, cat_i
    train_generator, validation_generator, test_generator = create_data_generators(tra

    # Train scratch model
    scratch_model = build_optimized_scratch_model(image_size)
    history_scratch = train_and_evaluate_model(scratch_model, train_generator, validat
    test_loss_scratch, test_accuracy_scratch = scratch_model.evaluate(test_generator)

    # Train pretrained InceptionV3 model
    pretrained_model = build_inception_pretrained_model(image_size)
    history_pretrained = train_and_evaluate_model(pretrained_model, train_generator, v
    test_loss_pretrained, test_accuracy_pretrained = pretrained_model.evaluate(test_ge

    # Store results for later comparison
    results[sample_size] = {
        'scratch': {'model': scratch_model, 'history': history_scratch, 'test_loss': t
        'pretrained': {'model': pretrained_model, 'history': history_pretrained, 'test
    }

    # Plot training and validation performance
    print(f"\nResults for CNN Model {sample_size} samples:")
    plot_training_metrics(history_scratch)
    print(f"\nResults for Pre Trained Model {sample_size} samples:")
    plot_training_metrics(history_pretrained)
```

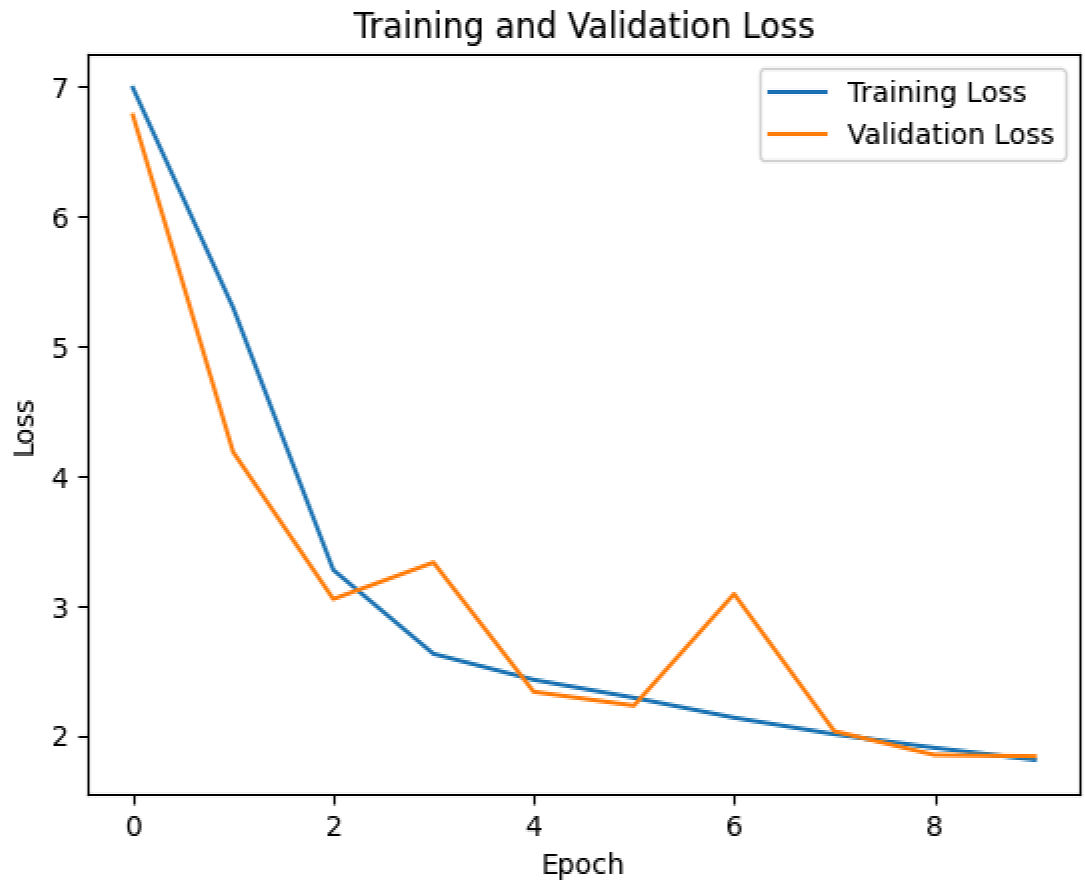
```
Found 1500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:1
07: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When u
sing Sequential models, prefer using an `Input(shape)` object as the first layer in t
he model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

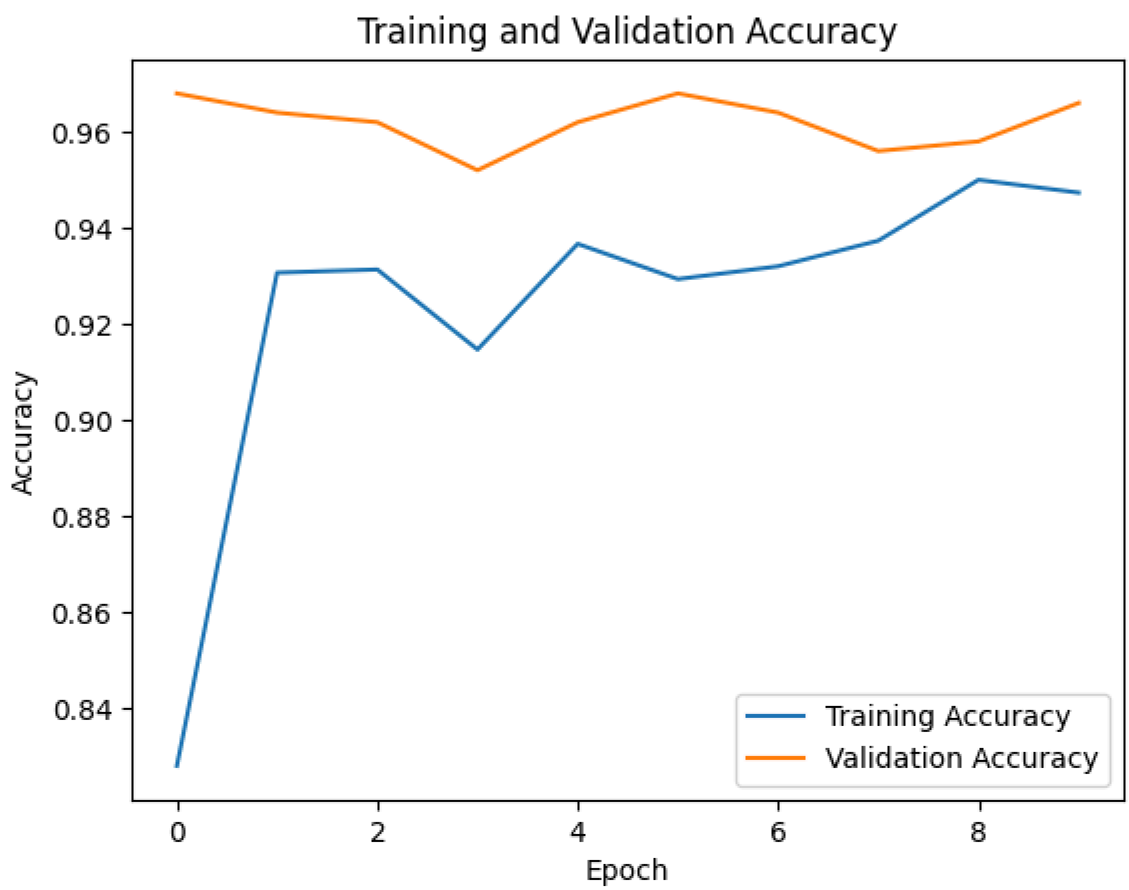
```
Epoch 1/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 32s 482ms/step - accuracy: 0.5472 - loss: 7.4070 - val_acc
uracy: 0.5020 - val_loss: 6.7758
Epoch 2/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 20s 362ms/step - accuracy: 0.5454 - loss: 5.8942 - val_acc
uracy: 0.5080 - val_loss: 4.1834
Epoch 3/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 19s 351ms/step - accuracy: 0.5697 - loss: 3.6160 - val_acc
uracy: 0.5260 - val_loss: 3.0504
Epoch 4/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 20s 368ms/step - accuracy: 0.5357 - loss: 2.6757 - val_acc
uracy: 0.5040 - val_loss: 3.3322
Epoch 5/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 19s 355ms/step - accuracy: 0.5310 - loss: 2.4724 - val_acc
uracy: 0.6080 - val_loss: 2.3365
Epoch 6/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 21s 364ms/step - accuracy: 0.5584 - loss: 2.3234 - val_acc
uracy: 0.5440 - val_loss: 2.2301
Epoch 7/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 20s 350ms/step - accuracy: 0.5928 - loss: 2.1722 - val_acc
uracy: 0.5080 - val_loss: 3.0899
Epoch 8/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 20s 359ms/step - accuracy: 0.6407 - loss: 2.0324 - val_acc
uracy: 0.5740 - val_loss: 2.0331
Epoch 9/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 22s 422ms/step - accuracy: 0.6032 - loss: 1.9424 - val_acc
uracy: 0.5840 - val_loss: 1.8505
Epoch 10/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 41s 420ms/step - accuracy: 0.6155 - loss: 1.8153 - val_acc
uracy: 0.5620 - val_loss: 1.8398
16/16 ━━━━━━━━━━━━━━━━━━━━ 3s 163ms/step - accuracy: 0.5343 - loss: 1.8164
Epoch 1/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 61s 943ms/step - accuracy: 0.7164 - loss: 2.2800 - val_acc
uracy: 0.9680 - val_loss: 0.0916
Epoch 2/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 21s 404ms/step - accuracy: 0.9205 - loss: 0.1786 - val_acc
uracy: 0.9640 - val_loss: 0.1082
Epoch 3/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 19s 358ms/step - accuracy: 0.9159 - loss: 0.2050 - val_acc
uracy: 0.9620 - val_loss: 0.0905
Epoch 4/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 23s 415ms/step - accuracy: 0.9263 - loss: 0.1524 - val_acc
uracy: 0.9520 - val_loss: 0.0908
Epoch 5/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 19s 354ms/step - accuracy: 0.9374 - loss: 0.1591 - val_acc
uracy: 0.9620 - val_loss: 0.0844
Epoch 6/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 23s 417ms/step - accuracy: 0.9214 - loss: 0.1613 - val_acc
uracy: 0.9680 - val_loss: 0.0863
Epoch 7/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 19s 358ms/step - accuracy: 0.9408 - loss: 0.1312 - val_acc
uracy: 0.9640 - val_loss: 0.1005
Epoch 8/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 19s 360ms/step - accuracy: 0.9451 - loss: 0.1290 - val_acc
uracy: 0.9560 - val_loss: 0.1121
Epoch 9/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 20s 343ms/step - accuracy: 0.9544 - loss: 0.1147 - val_acc
uracy: 0.9580 - val_loss: 0.0883
Epoch 10/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 18s 339ms/step - accuracy: 0.9530 - loss: 0.1092 - val_acc
```

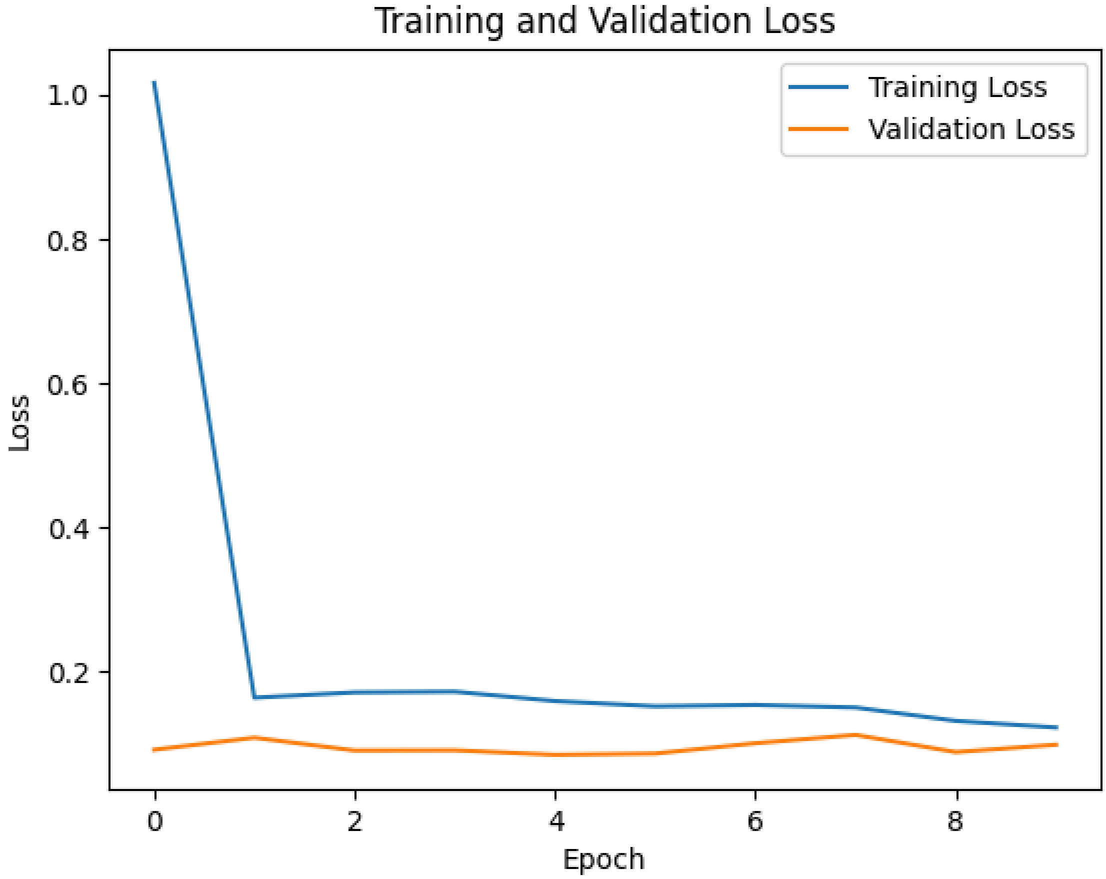
```
uracy: 0.9660 - val_loss: 0.0985
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 149ms/step - accuracy: 0.9601 - loss: 0.0988

Results for CNN Model 1500 samples:
```





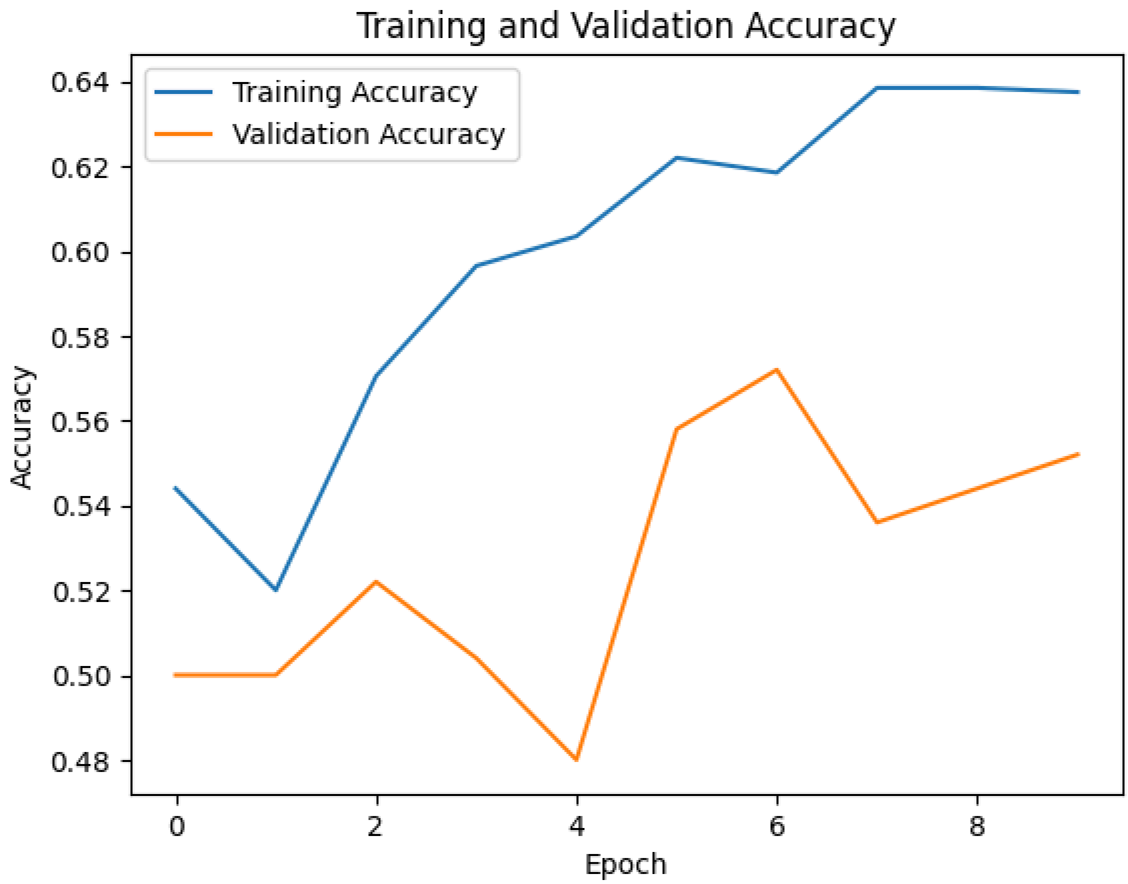

```
Results for Pre Trained Model 1500 samples:
```

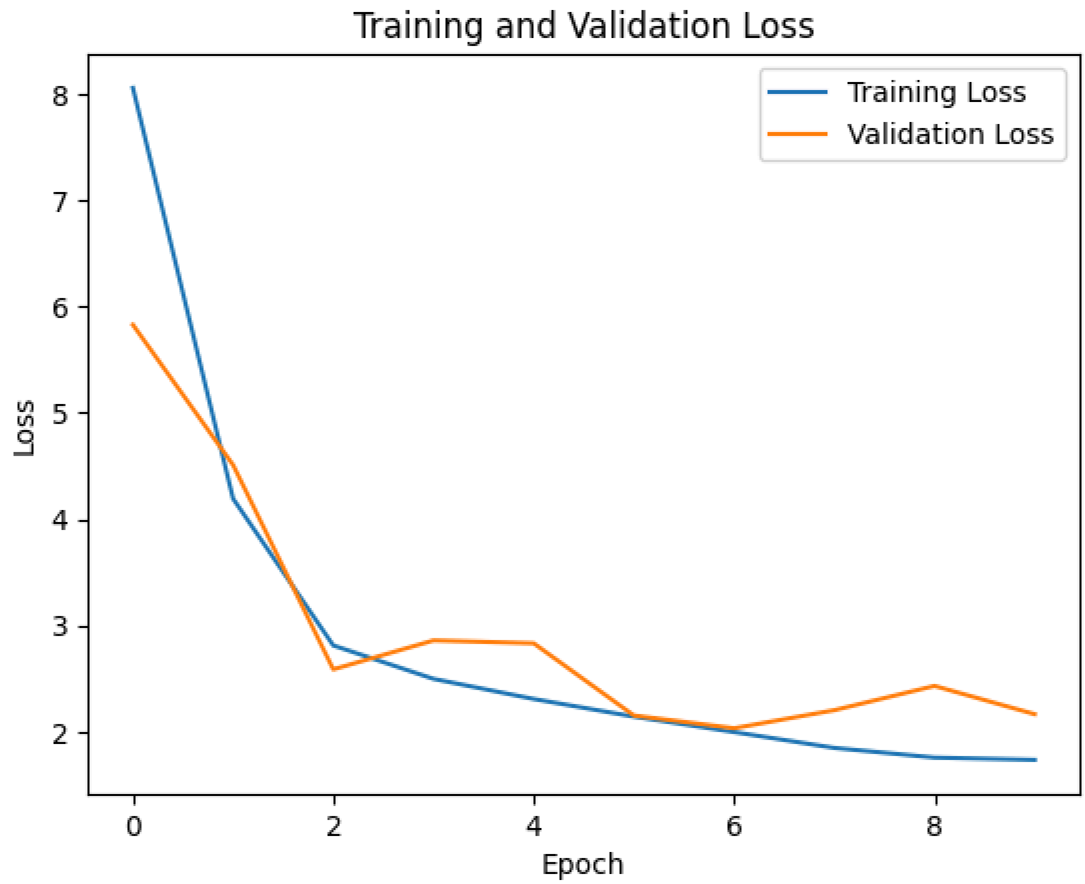



Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8

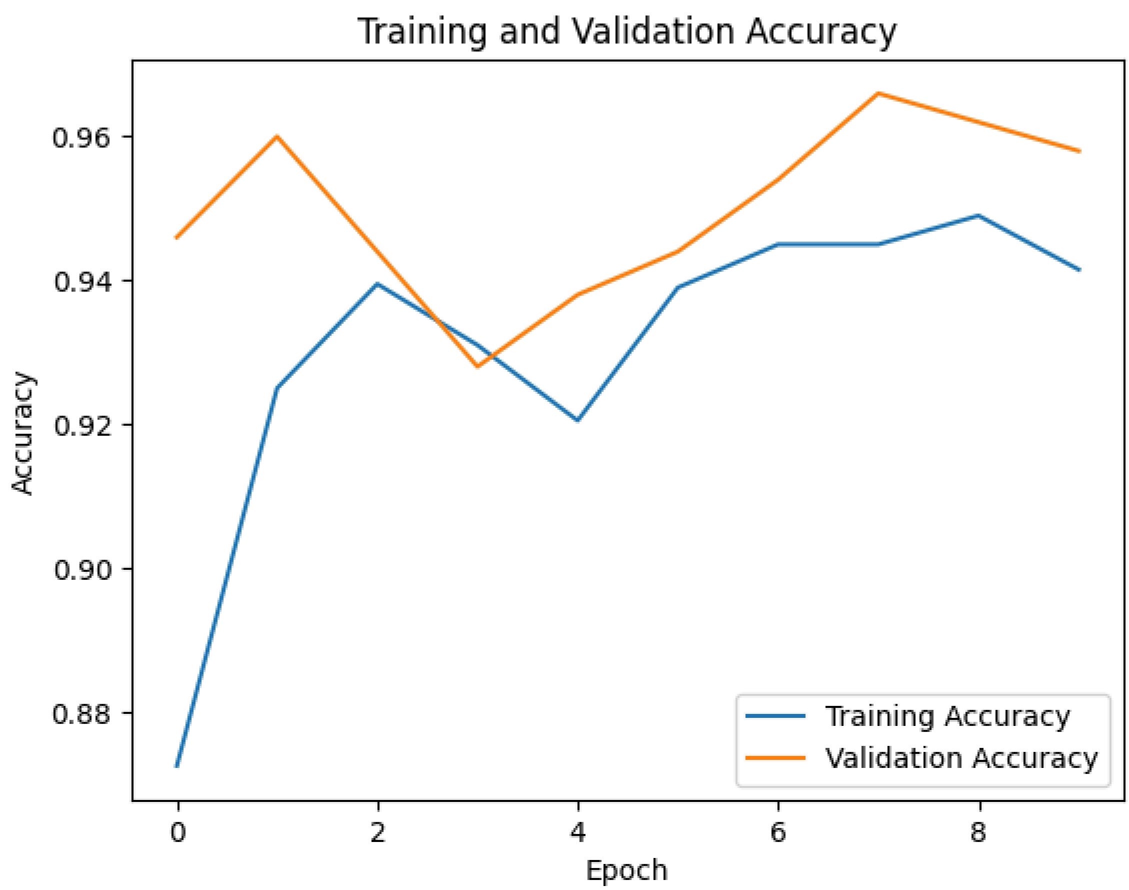
```
Found 2000 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 42s 519ms/step - accuracy: 0.5371 - loss: 9.3586 - val_acc
uracy: 0.5000 - val_loss: 5.8278
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 29s 363ms/step - accuracy: 0.5154 - loss: 4.8557 - val_acc
uracy: 0.5000 - val_loss: 4.5087
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 351ms/step - accuracy: 0.5600 - loss: 2.9523 - val_acc
uracy: 0.5220 - val_loss: 2.5862
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 42s 361ms/step - accuracy: 0.6022 - loss: 2.5199 - val_acc
uracy: 0.5040 - val_loss: 2.8582
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 356ms/step - accuracy: 0.6036 - loss: 2.3387 - val_acc
uracy: 0.4800 - val_loss: 2.8316
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 365ms/step - accuracy: 0.6289 - loss: 2.1685 - val_acc
uracy: 0.5580 - val_loss: 2.1522
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 364ms/step - accuracy: 0.6198 - loss: 2.0263 - val_acc
uracy: 0.5720 - val_loss: 2.0322
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 365ms/step - accuracy: 0.6357 - loss: 1.8809 - val_acc
uracy: 0.5360 - val_loss: 2.2022
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 342ms/step - accuracy: 0.6400 - loss: 1.7775 - val_acc
uracy: 0.5440 - val_loss: 2.4304
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 337ms/step - accuracy: 0.6430 - loss: 1.7204 - val_acc
uracy: 0.5520 - val_loss: 2.1645
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 138ms/step - accuracy: 0.6269 - loss: 1.9761
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 48s 506ms/step - accuracy: 0.8193 - loss: 0.7808 - val_acc
uracy: 0.9460 - val_loss: 0.1630
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 350ms/step - accuracy: 0.9277 - loss: 0.1984 - val_acc
uracy: 0.9600 - val_loss: 0.1108
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 332ms/step - accuracy: 0.9316 - loss: 0.1667 - val_acc
uracy: 0.9440 - val_loss: 0.1672
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 351ms/step - accuracy: 0.9430 - loss: 0.1343 - val_acc
uracy: 0.9280 - val_loss: 0.1747
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 350ms/step - accuracy: 0.9056 - loss: 0.2149 - val_acc
uracy: 0.9380 - val_loss: 0.1887
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 43s 386ms/step - accuracy: 0.9339 - loss: 0.1473 - val_acc
uracy: 0.9440 - val_loss: 0.1384
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 373ms/step - accuracy: 0.9485 - loss: 0.1252 - val_acc
uracy: 0.9540 - val_loss: 0.1103
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 327ms/step - accuracy: 0.9397 - loss: 0.1436 - val_acc
uracy: 0.9660 - val_loss: 0.0969
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 338ms/step - accuracy: 0.9462 - loss: 0.1346 - val_acc
```

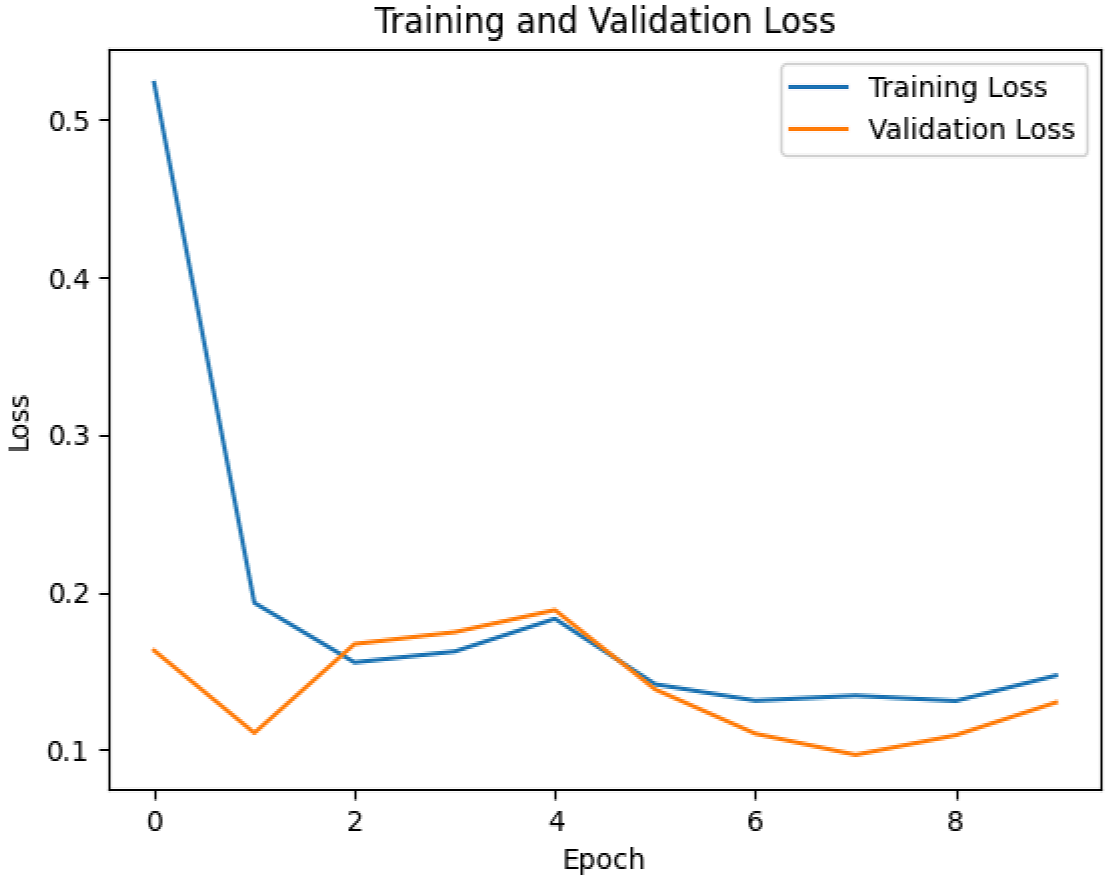
```
uracy: 0.9620 - val_loss: 0.1094
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 334ms/step - accuracy: 0.9426 - loss: 0.1443 - val_acc
uracy: 0.9580 - val_loss: 0.1302
16/16 ━━━━━━━━━━━━━━━━━━━━ 3s 190ms/step - accuracy: 0.9621 - loss: 0.1031

Results for CNN Model 2000 samples:
```





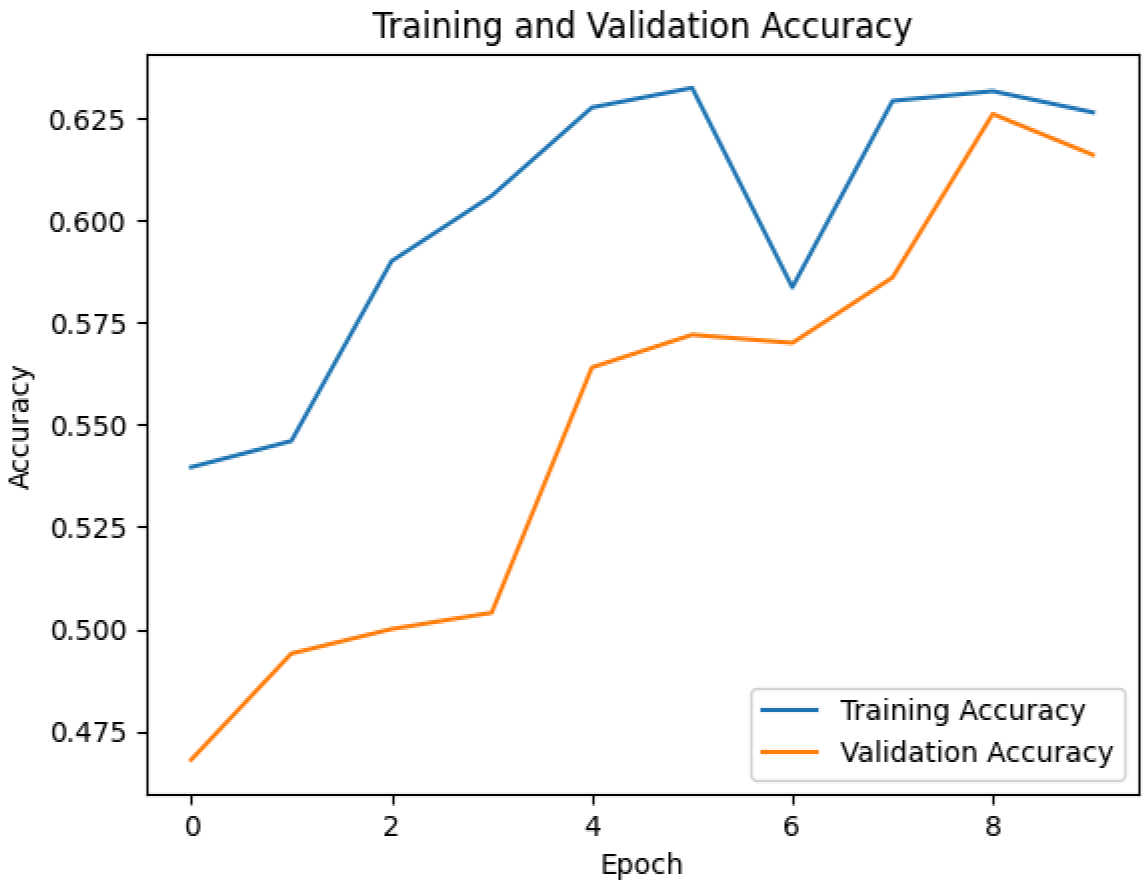
Results for Pre Trained Model 2000 samples:

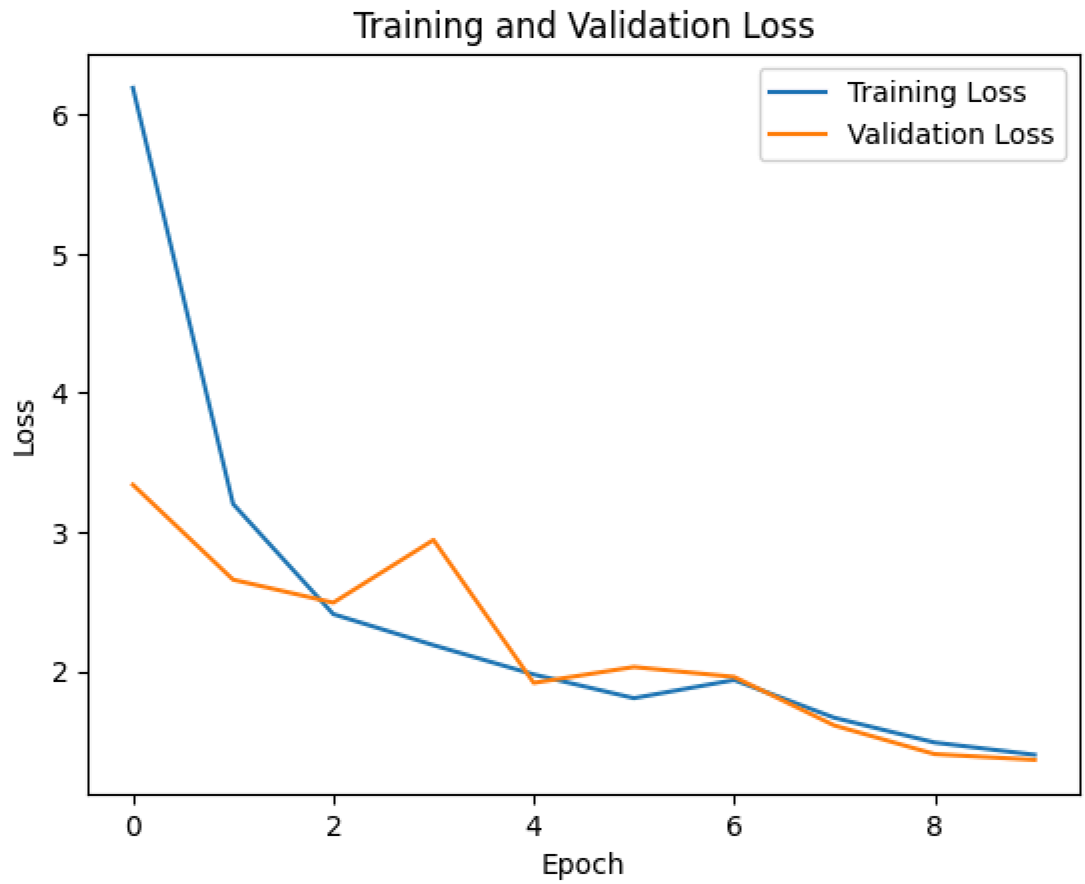



Training and Validation Loss
Training Loss
Validation Loss
Loss
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
Epoch

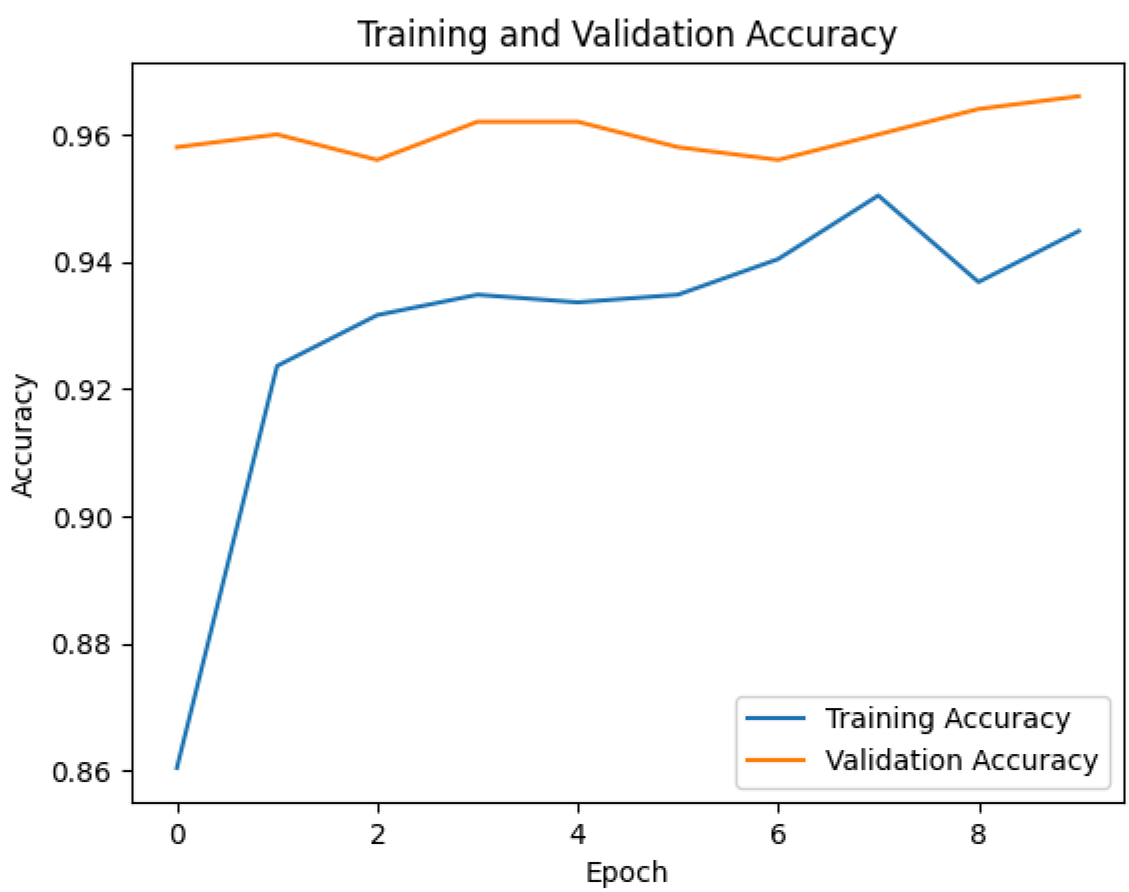
```
Found 2500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
Epoch 1/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 46s 475ms/step - accuracy: 0.5313 - loss: 6.8356 - val_acc
uracy: 0.4680 - val_loss: 3.3398
Epoch 2/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 31s 367ms/step - accuracy: 0.5419 - loss: 3.7844 - val_acc
uracy: 0.4940 - val_loss: 2.6578
Epoch 3/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 31s 366ms/step - accuracy: 0.5785 - loss: 2.4775 - val_acc
uracy: 0.5000 - val_loss: 2.4924
Epoch 4/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 31s 367ms/step - accuracy: 0.6107 - loss: 2.2325 - val_acc
uracy: 0.5040 - val_loss: 2.9432
Epoch 5/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 31s 360ms/step - accuracy: 0.6394 - loss: 2.0145 - val_acc
uracy: 0.5640 - val_loss: 1.9163
Epoch 6/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 40s 359ms/step - accuracy: 0.6264 - loss: 1.8425 - val_acc
uracy: 0.5720 - val_loss: 2.0305
Epoch 7/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 33s 373ms/step - accuracy: 0.5675 - loss: 2.0198 - val_acc
uracy: 0.5700 - val_loss: 1.9591
Epoch 8/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 379ms/step - accuracy: 0.6313 - loss: 1.7120 - val_acc
uracy: 0.5860 - val_loss: 1.6105
Epoch 9/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 31s 361ms/step - accuracy: 0.6317 - loss: 1.5143 - val_acc
uracy: 0.6260 - val_loss: 1.4046
Epoch 10/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 363ms/step - accuracy: 0.6450 - loss: 1.3951 - val_acc
uracy: 0.6160 - val_loss: 1.3633
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 146ms/step - accuracy: 0.6096 - loss: 1.3484
Epoch 1/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 54s 507ms/step - accuracy: 0.7597 - loss: 1.9802 - val_acc
uracy: 0.9580 - val_loss: 0.0997
Epoch 2/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 30s 350ms/step - accuracy: 0.9270 - loss: 0.1758 - val_acc
uracy: 0.9600 - val_loss: 0.1051
Epoch 3/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 30s 352ms/step - accuracy: 0.9276 - loss: 0.1675 - val_acc
uracy: 0.9560 - val_loss: 0.1132
Epoch 4/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 345ms/step - accuracy: 0.9450 - loss: 0.1422 - val_acc
uracy: 0.9620 - val_loss: 0.0888
Epoch 5/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 349ms/step - accuracy: 0.9338 - loss: 0.1544 - val_acc
uracy: 0.9620 - val_loss: 0.1071
Epoch 6/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 30s 348ms/step - accuracy: 0.9348 - loss: 0.1516 - val_acc
uracy: 0.9580 - val_loss: 0.1040
Epoch 7/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 344ms/step - accuracy: 0.9419 - loss: 0.1516 - val_acc
uracy: 0.9560 - val_loss: 0.1044
Epoch 8/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 343ms/step - accuracy: 0.9485 - loss: 0.1343 - val_acc
uracy: 0.9600 - val_loss: 0.1222
Epoch 9/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 339ms/step - accuracy: 0.9363 - loss: 0.1484 - val_acc
```

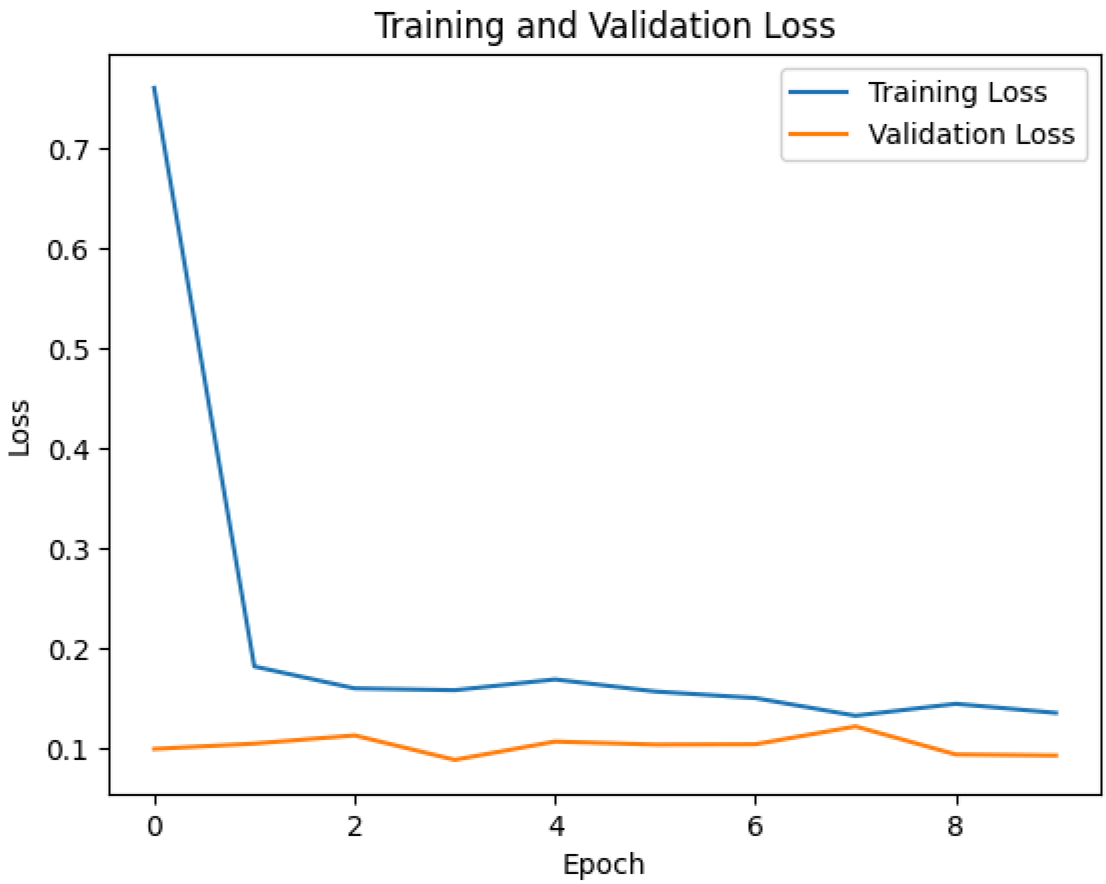
```
uracy: 0.9640 - val_loss: 0.0942
Epoch 10/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 41s 334ms/step - accuracy: 0.9410 - loss: 0.1451 - val_acc
uracy: 0.9660 - val_loss: 0.0932
16/16 ━━━━━━━━━━━━━━━━━━━━ 3s 159ms/step - accuracy: 0.9840 - loss: 0.0584

Results for CNN Model 2500 samples:
```





Results for Pre Trained Model 2500 samples:

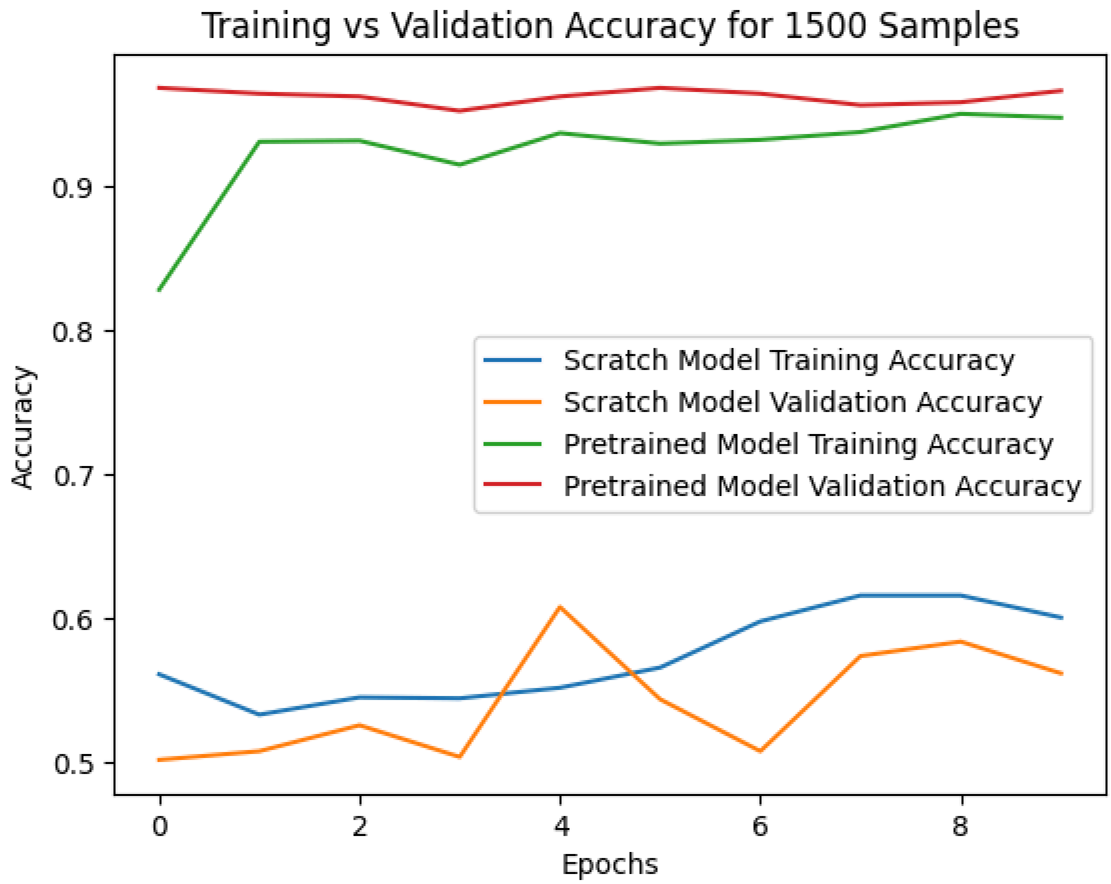




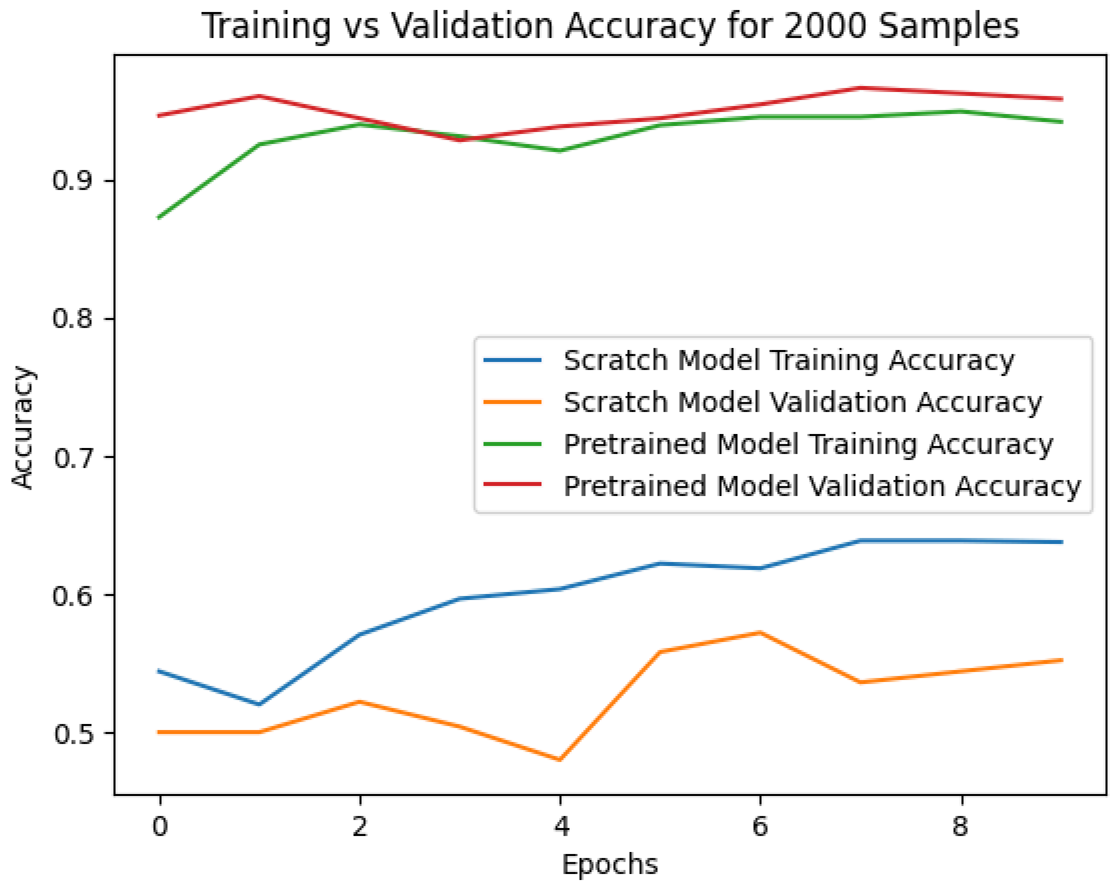
Model Comparison

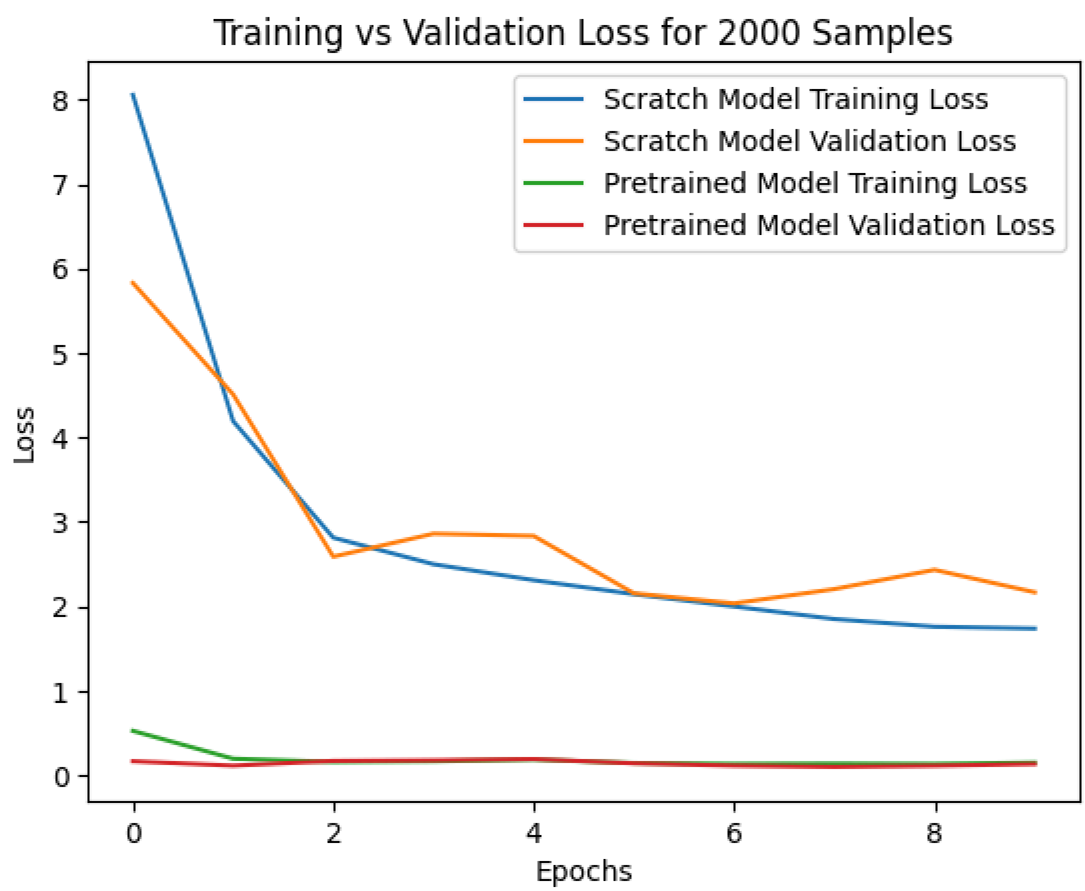
In [25]:
```
# Plot accuracy and loss comparison for both models for all sample sizes
for sample_size in sample_sizes:
    print(f"\nResults for {sample_size} samples:")
    plot_accuracy_comparison(results, sample_size)
    plot_loss_comparison(results, sample_size)
```

```
Results for 1500 samples:
```

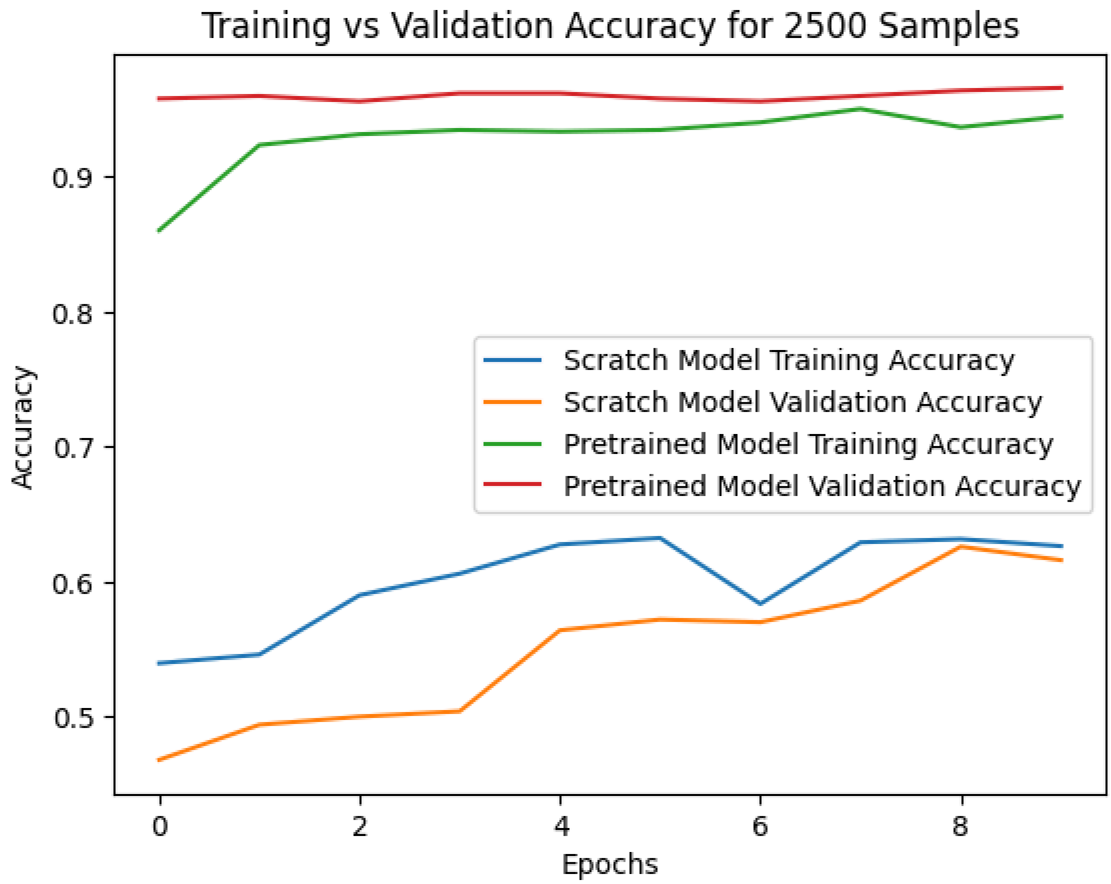



```
Results for 2000 samples:
```






```
Results for 2500 samples:
```



In [30]:

```
import pandas as pd
def display_results_table(results, sample_sizes):
```

```
    # Prepare lists to store the data
    data = {
        'Sample Size': [],
        'Model Type': [],
        'Training Accuracy': [],
        'Validation Accuracy': [],
        'Training Loss': [],
        'Validation Loss': []
    }

    for sample_size in sample_sizes:
        # Scratch Model Results
        scratch_acc = results[sample_size]['scratch']['history'].history['accuracy'][-
        scratch_val_acc = results[sample_size]['scratch']['history'].history['val_accu
        scratch_loss = results[sample_size]['scratch']['history'].history['loss'][-1]
        scratch_val_loss = results[sample_size]['scratch']['history'].history['val_los

        # Pretrained Model Results
        pretrained_acc = results[sample_size]['pretrained']['history'].history['accura
        pretrained_val_acc = results[sample_size]['pretrained']['history'].history['va
        pretrained_loss = results[sample_size]['pretrained']['history'].history['loss'
        pretrained_val_loss = results[sample_size]['pretrained']['history'].history['v

        # Append data for scratch model
        data['Sample Size'].append(sample_size)
        data['Model Type'].append('Scratch')
        data['Training Accuracy'].append(scratch_acc)
        data['Validation Accuracy'].append(scratch_val_acc)
        data['Training Loss'].append(scratch_loss)
        data['Validation Loss'].append(scratch_val_loss)

        # Append data for pretrained model
        data['Sample Size'].append(sample_size)
        data['Model Type'].append('Pretrained')
        data['Training Accuracy'].append(pretrained_acc)
        data['Validation Accuracy'].append(pretrained_val_acc)
        data['Training Loss'].append(pretrained_loss)
        data['Validation Loss'].append(pretrained_val_loss)

    # Create a DataFrame
    df = pd.DataFrame(data)

    # Print the DataFrame
    return df
```

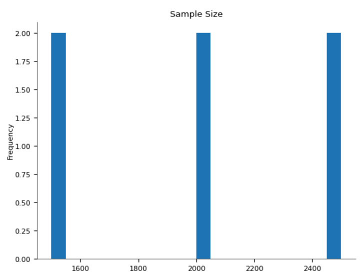
In [31]:

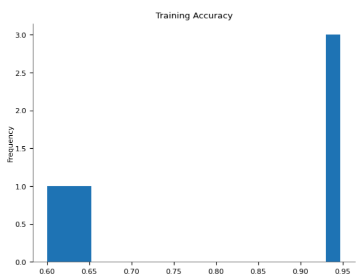
```
df = display_results_table(results, sample_sizes)
df
```

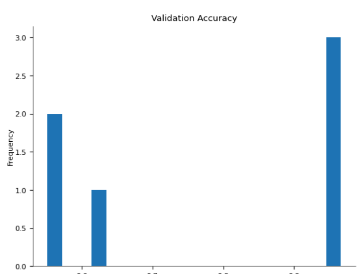
Out[31]:

	Sample Size	Model Type	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0	1500	Scratch	0.600667	0.562	1.812383	1.839815
1	1500	Pretrained	0.947333	0.966	0.122357	0.098470
2	2000	Scratch	0.637500	0.552	1.735171	2.164541
3	2000	Pretrained	0.941500	0.958	0.147226	0.130177
4	2500	Scratch	0.626400	0.616	1.399527	1.363309
5	2500	Pretrained	0.944800	0.966	0.135690	0.093156

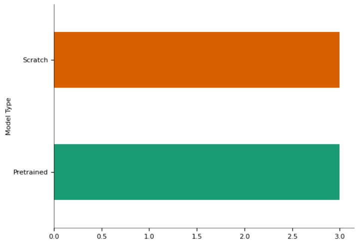
Distributions



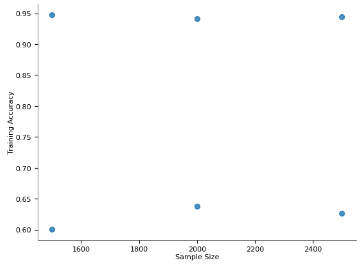


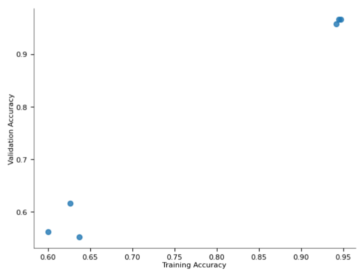


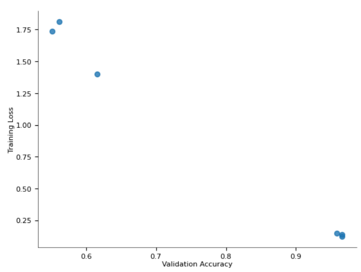
Categorical distributions

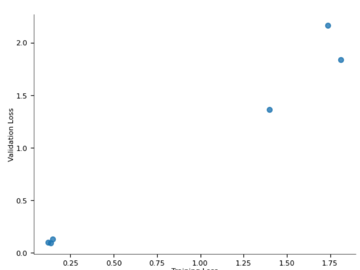


2-d distributions

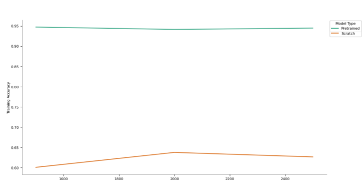


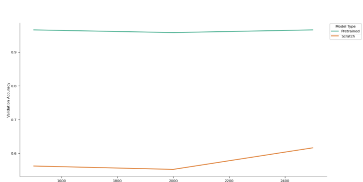


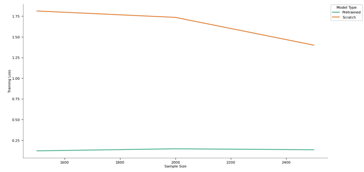


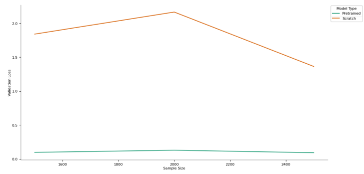


Time series

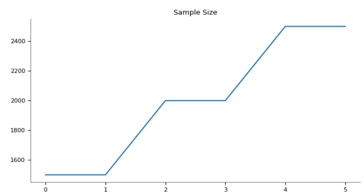


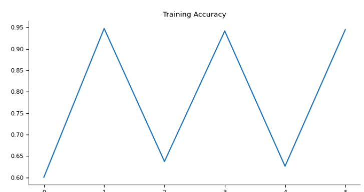


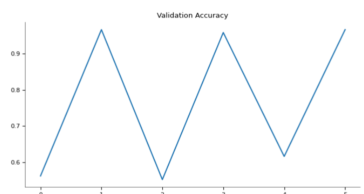


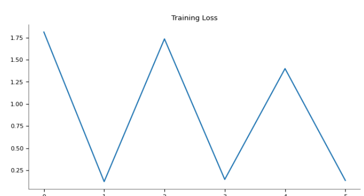


Values



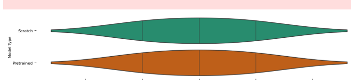




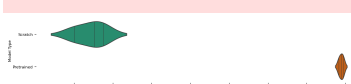


Faceted distributions

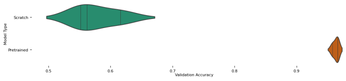
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```




```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```