```
from google.colab import drive
drive. mount('/content/drive')
```

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

```
import zipfile
with zipfile.ZipFile("/content/drive/MyDrive/Colab Notebooks/cats_vs_dogs_small.zip","r") as dataset_zip:
    dataset_zip.extractall("/content")
```

```
import os
dataset_directory = "/content/cats_vs_dogs_small"
print( "Contents of the base directory:", os.listdir(dataset_directory))
```

```
Contents of the base directory: ['test', 'validation', 'train']
```

```
pip install tensorflow
```

```
Collecting tensorflow
  Downloading tensorflow-2.19.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.1 kB)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Collecting astunparse>=1.6.0 (from tensorflow)
  Downloading astunparse-1.6.3-py2.py3-none-any.whl.metadata (4.4 kB)
Collecting flatbuffers>=24.3.25 (from tensorflow)
  Downloading flatbuffers-25.2.10-py2.py3-none-any.whl.metadata (875 bytes)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.
Collecting google-pasta>=0.1.1 (from tensorflow)
  Downloading google_pasta-0.2.0-py3-none-any.whl.metadata (814 bytes)
Collecting libclang>=13.0.0 (from tensorflow)
  Downloading libclang-18.1.1-py2.py3-none-manylinux2010_x86_64.whl.metadata (5.2 kB)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/pyt
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Collecting tensorboard~=2.19.0 (from tensorflow)
  Downloading tensorboard-2.19.0-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.5.1)
Collecting tensorflow-io-gcs-filesystem>=0.23.1 (from tensorflow)
  Downloading tensorflow_io_gcs_filesystem-0.37.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (14 kB)
Collecting wheel<1.0,>=0.23.0 (from astunparse>=1.6.0->tensorflow)
  Downloading wheel-0.45.1-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorf
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (
Requirement already satisfied: markdown>=2.6.8 in /usr/lib/python3/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.3.6)
Collecting tensorboard-data-server<0.8.0,>=0.7.0 (from tensorboard~=2.19.0->tensorflow)
  Downloading tensorboard_data_server-0.7.2-py3-none-manylinux_2_31_x86_64.whl.metadata (1.1 kB)
Collecting werkzeug>=1.0.1 (from tensorboard~=2.19.0->tensorflow)
  Downloading werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard~=2.19.
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflo
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0-
Downloading tensorflow-2.19.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (644.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 644.9/644.9 MB 1.6 MB/s eta 0:00:00
Downloading astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Downloading flatbuffers-25.2.10-py2.py3-none-any.whl (30 kB)
Downloading google_pasta-0.2.0-py3-none-any.whl (57 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.5/57.5 kB 3.8 MB/s eta 0:00:00
Downloading libclang-18.1.1-py2.py3-none-manylinux2010_x86_64.whl (24.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.5/24.5 MB 69.5 MB/s eta 0:00:00
Downloading tensorboard-2.19.0-py3-none-any.whl (5.5 MB)
```

```
import numpy as np
import matplotlib.pyplot as plot
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, applications

train_dataset = '/content/cats_vs_dogs_small/train'
val_dataset = '/content/cats_vs_dogs_small/validation'
test_dataset = '/content/cats_vs_dogs_small/test'



# Load datasets
training_set = keras.preprocessing.image_dataset_from_directory(
    train_dataset,
    image_size=(180, 180),
    batch_size=32)
```

```
Found 2000 files belonging to 2 classes.
```

```
validation_set = keras.preprocessing.image_dataset_from_directory(
    val_dataset,
    image_size=(180, 180),
    batch_size=32)
```

```
Found 1000 files belonging to 2 classes.
```

```
testing_set = keras.preprocessing.image_dataset_from_directory(
    test_dataset,
    image_size=(180, 180),
    batch_size=32)
```

```
Found 1000 files belonging to 2 classes.
```

```
# Display sample images
def show_images(dataset):
    plot.figure(figsize=(10, 10))
    for images, labels in dataset.take(1):
        for i in range(9):
            ax = plot.subplot(3, 3, i + 1)
            plot.imshow(images[i].numpy().astype("uint8"))



# Display sample images
def show_images(dataset):
    plot.figure(figsize=(10, 10))
    for images, labels in dataset.take(1):
        for i in range(9):
            ax = plot.subplot(3, 3, i + 1)
            plot.imshow(images[i].numpy().astype("uint8"))
            plot.title("Cat" if labels[i] == 0 else "Dog")
            plot.axis("off")
    plot.show()



# Call the function to display images
show_images(training_set)
```


```
# Function to create a CNN model from scratch
def cnn_scratch_model():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(180, 180, 3)),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model


# Data augmentation and preprocessing with sample size parameter
def get_data_generators(train_dataset, val_dataset, batch_size, num_samples=None):  # Add num_samples parameter with default None
    train_datagen = keras.preprocessing.image.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',
    )
    val_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
```

```
    train_generator = train_datagen.flow_from_directory(
        train_dataset,
        target_size=(180, 180),
        batch_size=batch_size,
        class_mode='binary',
    )
    val_generator = val_datagen.flow_from_directory(
        val_dataset,
        target_size=(180, 180),
        batch_size=batch_size,
        class_mode='binary',
    )
    return train_generator, val_generator  # Return both generators



# Train the CNN model
def train_model(model, train_generator, val_generator, epochs=30):
    history = model.fit(
        train_generator,
        validation_data=val_generator,
        epochs=epochs
    )
    return history



# Step 1: Train CNN model from scratch
train_gen_1, val_gen_1 = get_data_generators(train_dataset, val_dataset, batch_size=32)
model_A = cnn_scratch_model()
history_A = train_model(model_A, train_gen_1, val_gen_1)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`i
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cla
  self._warn_if_super_not_called()
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 379ms/step - accuracy: 0.5178 - loss: 1.0551 - val_accuracy: 0.5010 - val_loss: 0.6918
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 373ms/step - accuracy: 0.5080 - loss: 0.6968 - val_accuracy: 0.5000 - val_loss: 0.6929
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 370ms/step - accuracy: 0.4951 - loss: 0.6936 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 376ms/step - accuracy: 0.4932 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.5014 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.4893 - loss: 0.6931 - val_accuracy: 0.5010 - val_loss: 0.6909
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 377ms/step - accuracy: 0.4883 - loss: 0.6930 - val_accuracy: 0.5380 - val_loss: 0.6887
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.5263 - loss: 0.6897 - val_accuracy: 0.5250 - val_loss: 0.6904
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.5301 - loss: 0.6877 - val_accuracy: 0.5630 - val_loss: 0.6828
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.5705 - loss: 0.6761 - val_accuracy: 0.5370 - val_loss: 0.6825
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.5374 - loss: 0.6832 - val_accuracy: 0.5310 - val_loss: 0.6884
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.5682 - loss: 0.6815 - val_accuracy: 0.5770 - val_loss: 0.6751
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.5751 - loss: 0.6730 - val_accuracy: 0.5970 - val_loss: 0.6792
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 374ms/step - accuracy: 0.5383 - loss: 0.6838 - val_accuracy: 0.5700 - val_loss: 0.6771
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.5839 - loss: 0.6765 - val_accuracy: 0.6130 - val_loss: 0.6548
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 373ms/step - accuracy: 0.5908 - loss: 0.6743 - val_accuracy: 0.6160 - val_loss: 0.6638
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.6196 - loss: 0.6602 - val_accuracy: 0.6060 - val_loss: 0.6625
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 373ms/step - accuracy: 0.6326 - loss: 0.6559 - val_accuracy: 0.5850 - val_loss: 0.6614
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.5988 - loss: 0.6597 - val_accuracy: 0.6490 - val_loss: 0.6386
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.6232 - loss: 0.6602 - val_accuracy: 0.6280 - val_loss: 0.6528
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.6503 - loss: 0.6411 - val_accuracy: 0.6260 - val_loss: 0.6439
Epoch 22/30
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 376ms/step - accuracy: 0.6565 - loss: 0.6274 - val_accuracy: 0.6790 - val_loss: 0.5947
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 366ms/step - accuracy: 0.6471 - loss: 0.6214 - val_accuracy: 0.6640 - val_loss: 0.6214
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 374ms/step - accuracy: 0.6667 - loss: 0.6109 - val_accuracy: 0.7050 - val_loss: 0.5652
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 370ms/step - accuracy: 0.6776 - loss: 0.6057 - val_accuracy: 0.6860 - val_loss: 0.6019
Epoch 26/30
```

```
# Step 2: Increase training samples to 1500
train_gen_2, val_gen_2 = get_data_generators(train_dataset, val_dataset, batch_size=32, num_samples=1500)
model_B = cnn_scratch_model()
history_B = train_model(model_B, train_gen_2, val_gen_2)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 373ms/step - accuracy: 0.4861 - loss: 0.9703 - val_accuracy: 0.5000 - val_loss: 0.6940
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.5385 - loss: 0.6882 - val_accuracy: 0.5640 - val_loss: 0.6878
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 373ms/step - accuracy: 0.6064 - loss: 0.6768 - val_accuracy: 0.6070 - val_loss: 0.6446
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.6095 - loss: 0.6619 - val_accuracy: 0.5570 - val_loss: 0.6654
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.6280 - loss: 0.6486 - val_accuracy: 0.6150 - val_loss: 0.6475
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 366ms/step - accuracy: 0.6549 - loss: 0.6209 - val_accuracy: 0.6230 - val_loss: 0.6341
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 370ms/step - accuracy: 0.6626 - loss: 0.6237 - val_accuracy: 0.6350 - val_loss: 0.6418
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 366ms/step - accuracy: 0.6837 - loss: 0.6095 - val_accuracy: 0.7200 - val_loss: 0.5425
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.7083 - loss: 0.5674 - val_accuracy: 0.7340 - val_loss: 0.5320
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.7102 - loss: 0.5695 - val_accuracy: 0.7420 - val_loss: 0.5202
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.7073 - loss: 0.5681 - val_accuracy: 0.7010 - val_loss: 0.5528
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.7038 - loss: 0.5624 - val_accuracy: 0.7040 - val_loss: 0.5453
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 363ms/step - accuracy: 0.7320 - loss: 0.5394 - val_accuracy: 0.7160 - val_loss: 0.5374
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.6963 - loss: 0.5776 - val_accuracy: 0.7360 - val_loss: 0.5337
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 366ms/step - accuracy: 0.7069 - loss: 0.5616 - val_accuracy: 0.7330 - val_loss: 0.5227
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.7352 - loss: 0.5354 - val_accuracy: 0.7150 - val_loss: 0.5583
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 374ms/step - accuracy: 0.7479 - loss: 0.5346 - val_accuracy: 0.7410 - val_loss: 0.5046
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.7218 - loss: 0.5439 - val_accuracy: 0.7260 - val_loss: 0.5390
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.7119 - loss: 0.5451 - val_accuracy: 0.7660 - val_loss: 0.4772
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.7267 - loss: 0.5295 - val_accuracy: 0.7490 - val_loss: 0.5073
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 367ms/step - accuracy: 0.7262 - loss: 0.5305 - val_accuracy: 0.7590 - val_loss: 0.4871
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 373ms/step - accuracy: 0.7244 - loss: 0.5137 - val_accuracy: 0.7450 - val_loss: 0.5094
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 365ms/step - accuracy: 0.7413 - loss: 0.5179 - val_accuracy: 0.7720 - val_loss: 0.4911
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.7478 - loss: 0.5195 - val_accuracy: 0.7410 - val_loss: 0.5118
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.7093 - loss: 0.5392 - val_accuracy: 0.7160 - val_loss: 0.5389
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 370ms/step - accuracy: 0.7583 - loss: 0.5098 - val_accuracy: 0.7730 - val_loss: 0.4795
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.7563 - loss: 0.4934 - val_accuracy: 0.7450 - val_loss: 0.5371
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.7131 - loss: 0.5542 - val_accuracy: 0.7670 - val_loss: 0.4965
```

```
# Step 3: Use the full 2000 samples
train_gen_3, val_gen_3 = get_data_generators(train_dataset, val_dataset, batch_size=32, num_samples=2000)
model_C = cnn_scratch_model()
history_C = train_model(model_C, train_gen_3, val_gen_3)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 369ms/step - accuracy: 0.5191 - loss: 0.8191 - val_accuracy: 0.5120 - val_loss: 0.6928
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.5113 - loss: 0.6925 - val_accuracy: 0.5280 - val_loss: 0.6912
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 370ms/step - accuracy: 0.5654 - loss: 0.6849 - val_accuracy: 0.6200 - val_loss: 0.6495
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 366ms/step - accuracy: 0.5758 - loss: 0.6712 - val_accuracy: 0.5350 - val_loss: 0.6886
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 366ms/step - accuracy: 0.5544 - loss: 0.6898 - val_accuracy: 0.6300 - val_loss: 0.6520
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.6361 - loss: 0.6462 - val_accuracy: 0.6240 - val_loss: 0.6362
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.6205 - loss: 0.6582 - val_accuracy: 0.6170 - val_loss: 0.6437
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 377ms/step - accuracy: 0.6502 - loss: 0.6258 - val_accuracy: 0.6670 - val_loss: 0.6004
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.6763 - loss: 0.6051 - val_accuracy: 0.6860 - val_loss: 0.5825
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.6830 - loss: 0.5860 - val_accuracy: 0.6730 - val_loss: 0.6065
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 366ms/step - accuracy: 0.6725 - loss: 0.6163 - val_accuracy: 0.6900 - val_loss: 0.5733
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.6789 - loss: 0.5913 - val_accuracy: 0.7170 - val_loss: 0.5410
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 373ms/step - accuracy: 0.7087 - loss: 0.5611 - val_accuracy: 0.7090 - val_loss: 0.5519
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.7328 - loss: 0.5530 - val_accuracy: 0.7120 - val_loss: 0.5496
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 373ms/step - accuracy: 0.6851 - loss: 0.5720 - val_accuracy: 0.7230 - val_loss: 0.5456
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.7054 - loss: 0.5720 - val_accuracy: 0.7090 - val_loss: 0.5675
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 371ms/step - accuracy: 0.7262 - loss: 0.5310 - val_accuracy: 0.6940 - val_loss: 0.5822
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 375ms/step - accuracy: 0.7035 - loss: 0.5606 - val_accuracy: 0.7470 - val_loss: 0.4933
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 373ms/step - accuracy: 0.7179 - loss: 0.5534 - val_accuracy: 0.7450 - val_loss: 0.5147
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 373ms/step - accuracy: 0.7573 - loss: 0.5110 - val_accuracy: 0.7320 - val_loss: 0.5412
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 367ms/step - accuracy: 0.7346 - loss: 0.5462 - val_accuracy: 0.7490 - val_loss: 0.5244
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 366ms/step - accuracy: 0.7514 - loss: 0.5082 - val_accuracy: 0.7280 - val_loss: 0.5192
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 364ms/step - accuracy: 0.7440 - loss: 0.5065 - val_accuracy: 0.7490 - val_loss: 0.4998
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.7320 - loss: 0.5315 - val_accuracy: 0.7560 - val_loss: 0.5048
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 372ms/step - accuracy: 0.7342 - loss: 0.5167 - val_accuracy: 0.7510 - val_loss: 0.5154
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.7583 - loss: 0.5031 - val_accuracy: 0.7350 - val_loss: 0.5464
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 368ms/step - accuracy: 0.7619 - loss: 0.4981 - val_accuracy: 0.7200 - val_loss: 0.5355
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.7568 - loss: 0.4925 - val_accuracy: 0.7330 - val_loss: 0.5195
```

```
# Step 4: Use a pretrained CNN model (e.g., VGG16)
def cnn_pretrained_model():
    base_model = applications.VGG16(include_top=False, weights='imagenet', input_shape=(180, 180, 3))
    base_model.trainable = False  # Freeze the base model

    model = models.Sequential([
        base_model,
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model



# Repeat Steps 1-3 with the pretrained CNN model
model_P1 = cnn_pretrained_model()
history_P1 = train_model(model_P1, train_gen_1, val_gen_1)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_not
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 42s 636ms/step - accuracy: 0.6812 - loss: 1.1828 - val_accuracy: 0.8770 - val_loss: 0.3046
```

```
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 616ms/step - accuracy: 0.8312 - loss: 0.3694 - val_accuracy: 0.8940 - val_loss: 0.2800
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 616ms/step - accuracy: 0.8561 - loss: 0.3440 - val_accuracy: 0.8970 - val_loss: 0.2460
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 616ms/step - accuracy: 0.8576 - loss: 0.3212 - val_accuracy: 0.9010 - val_loss: 0.2360
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 613ms/step - accuracy: 0.8712 - loss: 0.3152 - val_accuracy: 0.8810 - val_loss: 0.3048
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 614ms/step - accuracy: 0.8344 - loss: 0.3711 - val_accuracy: 0.9020 - val_loss: 0.2301
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 613ms/step - accuracy: 0.8824 - loss: 0.3020 - val_accuracy: 0.9120 - val_loss: 0.2083
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 618ms/step - accuracy: 0.8775 - loss: 0.2855 - val_accuracy: 0.9000 - val_loss: 0.2793
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 618ms/step - accuracy: 0.8779 - loss: 0.2876 - val_accuracy: 0.9140 - val_loss: 0.2165
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 611ms/step - accuracy: 0.8893 - loss: 0.2457 - val_accuracy: 0.9110 - val_loss: 0.2151
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 615ms/step - accuracy: 0.8967 - loss: 0.2535 - val_accuracy: 0.9170 - val_loss: 0.2235
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.9234 - loss: 0.2007 - val_accuracy: 0.9160 - val_loss: 0.2220
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 612ms/step - accuracy: 0.8859 - loss: 0.2840 - val_accuracy: 0.9090 - val_loss: 0.2167
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 612ms/step - accuracy: 0.8913 - loss: 0.2398 - val_accuracy: 0.9070 - val_loss: 0.2222
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 609ms/step - accuracy: 0.9031 - loss: 0.2241 - val_accuracy: 0.9130 - val_loss: 0.2128
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 611ms/step - accuracy: 0.8901 - loss: 0.2533 - val_accuracy: 0.9030 - val_loss: 0.2249
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 612ms/step - accuracy: 0.8995 - loss: 0.2342 - val_accuracy: 0.9020 - val_loss: 0.2231
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 612ms/step - accuracy: 0.9170 - loss: 0.2141 - val_accuracy: 0.9050 - val_loss: 0.2147
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 609ms/step - accuracy: 0.9134 - loss: 0.2069 - val_accuracy: 0.9120 - val_loss: 0.2174
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 603ms/step - accuracy: 0.8962 - loss: 0.2305 - val_accuracy: 0.9090 - val_loss: 0.2224
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 614ms/step - accuracy: 0.9014 - loss: 0.2209 - val_accuracy: 0.9050 - val_loss: 0.2406
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 613ms/step - accuracy: 0.8944 - loss: 0.2440 - val_accuracy: 0.9120 - val_loss: 0.2179
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 612ms/step - accuracy: 0.9115 - loss: 0.2092 - val_accuracy: 0.9060 - val_loss: 0.2111
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 621ms/step - accuracy: 0.9019 - loss: 0.2321 - val_accuracy: 0.9150 - val_loss: 0.2152
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 610ms/step - accuracy: 0.9237 - loss: 0.1884 - val_accuracy: 0.9000 - val_loss: 0.2258
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 612ms/step - accuracy: 0.8972 - loss: 0.2194 - val_accuracy: 0.9090 - val_loss: 0.2279
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 615ms/step - accuracy: 0.9154 - loss: 0.2068 - val_accuracy: 0.9120 - val_loss: 0.2146
```

```
model_P2 = cnn_pretrained_model()
history_P2 = train_model(model_P2, train_gen_2, val_gen_2)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 611ms/step - accuracy: 0.6175 - loss: 1.6180 - val_accuracy: 0.8390 - val_loss: 0.3447
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.8254 - loss: 0.3782 - val_accuracy: 0.9040 - val_loss: 0.2432
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 609ms/step - accuracy: 0.8433 - loss: 0.3454 - val_accuracy: 0.9150 - val_loss: 0.2322
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 610ms/step - accuracy: 0.8677 - loss: 0.3214 - val_accuracy: 0.9070 - val_loss: 0.2356
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 615ms/step - accuracy: 0.8702 - loss: 0.2850 - val_accuracy: 0.9040 - val_loss: 0.2448
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.8427 - loss: 0.3216 - val_accuracy: 0.8940 - val_loss: 0.2364
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 612ms/step - accuracy: 0.8837 - loss: 0.2707 - val_accuracy: 0.9110 - val_loss: 0.2162
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 623ms/step - accuracy: 0.8652 - loss: 0.3129 - val_accuracy: 0.9160 - val_loss: 0.2199
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.8823 - loss: 0.2638 - val_accuracy: 0.9140 - val_loss: 0.2173
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 605ms/step - accuracy: 0.8825 - loss: 0.2814 - val_accuracy: 0.9040 - val_loss: 0.2542
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 604ms/step - accuracy: 0.8911 - loss: 0.2526 - val_accuracy: 0.9140 - val_loss: 0.2209
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 609ms/step - accuracy: 0.8966 - loss: 0.2433 - val_accuracy: 0.9190 - val_loss: 0.2161
Epoch 13/30
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 612ms/step - accuracy: 0.8963 - loss: 0.2428 - val_accuracy: 0.9180 - val_loss: 0.2245
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 609ms/step - accuracy: 0.8733 - loss: 0.2654 - val_accuracy: 0.9160 - val_loss: 0.2179
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 611ms/step - accuracy: 0.8941 - loss: 0.2442 - val_accuracy: 0.9100 - val_loss: 0.2256
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 614ms/step - accuracy: 0.9027 - loss: 0.2316 - val_accuracy: 0.9070 - val_loss: 0.2212
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 616ms/step - accuracy: 0.8905 - loss: 0.2330 - val_accuracy: 0.8920 - val_loss: 0.2747
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 609ms/step - accuracy: 0.8898 - loss: 0.2601 - val_accuracy: 0.9040 - val_loss: 0.2513
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 609ms/step - accuracy: 0.8924 - loss: 0.2279 - val_accuracy: 0.9040 - val_loss: 0.2563
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 610ms/step - accuracy: 0.8874 - loss: 0.2390 - val_accuracy: 0.9060 - val_loss: 0.2237
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 605ms/step - accuracy: 0.9088 - loss: 0.2234 - val_accuracy: 0.9110 - val_loss: 0.2439
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 607ms/step - accuracy: 0.8906 - loss: 0.2635 - val_accuracy: 0.9000 - val_loss: 0.2502
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 605ms/step - accuracy: 0.8740 - loss: 0.2758 - val_accuracy: 0.9160 - val_loss: 0.2122
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 609ms/step - accuracy: 0.9016 - loss: 0.2277 - val_accuracy: 0.9110 - val_loss: 0.2120
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 607ms/step - accuracy: 0.8794 - loss: 0.2538 - val_accuracy: 0.8970 - val_loss: 0.2531
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 610ms/step - accuracy: 0.9154 - loss: 0.2165 - val_accuracy: 0.9160 - val_loss: 0.2199
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 612ms/step - accuracy: 0.9176 - loss: 0.1931 - val_accuracy: 0.9180 - val_loss: 0.2233
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 606ms/step - accuracy: 0.9152 - loss: 0.2108 - val_accuracy: 0.8980 - val_loss: 0.2552
Epoch 29/30
```

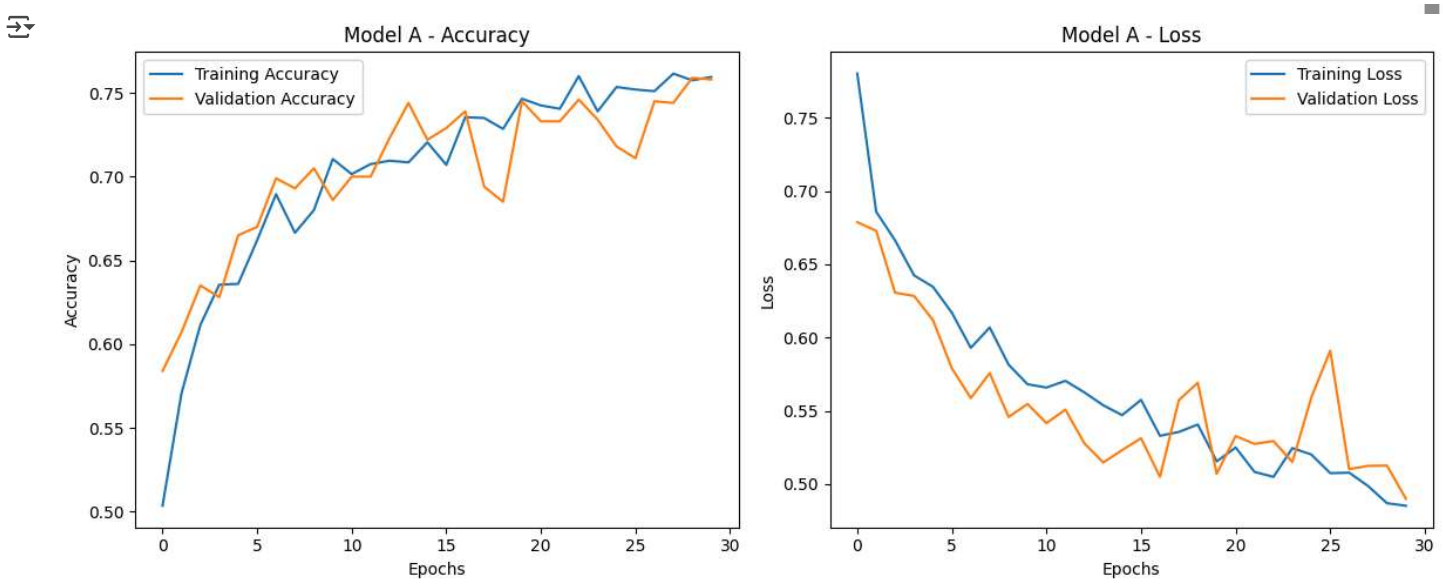
```
model_P3 = cnn_pretrained_model()
history_P3 = train_model(model_P3, train_gen_3, val_gen_3)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 41s 623ms/step - accuracy: 0.5813 - loss: 1.6053 - val_accuracy: 0.8860 - val_loss: 0.2871
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 611ms/step - accuracy: 0.8265 - loss: 0.3853 - val_accuracy: 0.8920 - val_loss: 0.2453
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 614ms/step - accuracy: 0.8446 - loss: 0.3377 - val_accuracy: 0.9020 - val_loss: 0.2489
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 612ms/step - accuracy: 0.8647 - loss: 0.3002 - val_accuracy: 0.9090 - val_loss: 0.2152
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.8700 - loss: 0.2921 - val_accuracy: 0.8960 - val_loss: 0.2376
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.8587 - loss: 0.3223 - val_accuracy: 0.9080 - val_loss: 0.2289
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 603ms/step - accuracy: 0.8574 - loss: 0.3244 - val_accuracy: 0.9000 - val_loss: 0.2264
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.8729 - loss: 0.2880 - val_accuracy: 0.9030 - val_loss: 0.2229
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.8939 - loss: 0.2539 - val_accuracy: 0.9120 - val_loss: 0.2254
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 613ms/step - accuracy: 0.8852 - loss: 0.2588 - val_accuracy: 0.9060 - val_loss: 0.2222
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 605ms/step - accuracy: 0.8909 - loss: 0.2558 - val_accuracy: 0.8940 - val_loss: 0.2408
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 610ms/step - accuracy: 0.8770 - loss: 0.2638 - val_accuracy: 0.8940 - val_loss: 0.2415
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 604ms/step - accuracy: 0.9056 - loss: 0.2336 - val_accuracy: 0.9060 - val_loss: 0.2346
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 602ms/step - accuracy: 0.8940 - loss: 0.2442 - val_accuracy: 0.9010 - val_loss: 0.2249
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 606ms/step - accuracy: 0.8997 - loss: 0.2381 - val_accuracy: 0.9060 - val_loss: 0.2200
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 609ms/step - accuracy: 0.9063 - loss: 0.2230 - val_accuracy: 0.8900 - val_loss: 0.2629
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 39s 611ms/step - accuracy: 0.9066 - loss: 0.2284 - val_accuracy: 0.9020 - val_loss: 0.2557
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 606ms/step - accuracy: 0.8952 - loss: 0.2458 - val_accuracy: 0.9120 - val_loss: 0.2186
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 603ms/step - accuracy: 0.9018 - loss: 0.2317 - val_accuracy: 0.9100 - val_loss: 0.2392
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.9134 - loss: 0.2156 - val_accuracy: 0.8990 - val_loss: 0.2342
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 606ms/step - accuracy: 0.8907 - loss: 0.2529 - val_accuracy: 0.8890 - val_loss: 0.2651
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 607ms/step - accuracy: 0.9067 - loss: 0.2262 - val_accuracy: 0.8950 - val_loss: 0.2365
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 606ms/step - accuracy: 0.8794 - loss: 0.2628 - val_accuracy: 0.8990 - val_loss: 0.2134
```

```
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 599ms/step - accuracy: 0.9077 - loss: 0.2199 - val_accuracy: 0.9090 - val_loss: 0.2288
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 596ms/step - accuracy: 0.8992 - loss: 0.2191 - val_accuracy: 0.9130 - val_loss: 0.2196
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 604ms/step - accuracy: 0.9022 - loss: 0.2284 - val_accuracy: 0.8850 - val_loss: 0.2840
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.8900 - loss: 0.2784 - val_accuracy: 0.9110 - val_loss: 0.2118
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 608ms/step - accuracy: 0.9099 - loss: 0.2193 - val_accuracy: 0.8960 - val_loss: 0.2756
Epoch 29/30
```

```
# Performance visualization function
def visualize_performance(history, title):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    train_loss = history.history['loss']
    val_loss = history.history['val_loss']

    plot.figure(figsize=(12, 5))

    # Accuracy plot
    plot.subplot(1, 2, 1)
    plot.plot(acc, label='Training Accuracy')
    plot.plot(val_acc, label='Validation Accuracy')
    plot.title(f'{title} - Accuracy')
    plot.xlabel('Epochs')
    plot.ylabel('Accuracy')
    plot.legend()

    # Loss plot
    plot.subplot(1, 2, 2)
    plot.plot(train_loss, label='Training Loss')
    plot.plot(val_loss, label='Validation Loss')
    plot.title(f'{title} - Loss')
    plot.xlabel('Epochs')
    plot.ylabel('Loss')
    plot.legend()

    plot.tight_layout()
    plot.show()
```

```
visualize_performance(history_A, "Model A")
```




```
# Function to summarize final results
def aggregate_results(all_histories):
    summary = {}
    for label, history in all_histories:
```

```
            final_train_acc = history.history['accuracy'][-1]
            final_val_acc = history.history['val_accuracy'][-1]
            summary[label] = {
                'Final Training Accuracy': round(final_train_acc, 4),
                'Final Validation Accuracy': round(final_val_acc, 4)
            }
        return summary


    # Example of using the aggregate_results function
    all_histories = [
        ("Scratch Model A", history_A),
        ("Scratch Model B", history_B),
        ("Scratch Model C", history_C),
        ("Pretrained Model P1", history_P1),
        ("Pretrained Model P2", history_P2),
        ("Pretrained Model P3", history_P3)
    ]

    results = aggregate_results(all_histories)

    # Print results
    for model_name, metrics in results.items():
        print(f"{model_name}: {metrics}")
```

```
Scratch Model A: {'Final Training Accuracy': 0.7595, 'Final Validation Accuracy': 0.758}
Scratch Model B: {'Final Training Accuracy': 0.759, 'Final Validation Accuracy': 0.745}
Scratch Model C: {'Final Training Accuracy': 0.748, 'Final Validation Accuracy': 0.74}
Pretrained Model P1: {'Final Training Accuracy': 0.904, 'Final Validation Accuracy': 0.904}
Pretrained Model P2: {'Final Training Accuracy': 0.9085, 'Final Validation Accuracy': 0.905}
Pretrained Model P3: {'Final Training Accuracy': 0.908, 'Final Validation Accuracy': 0.914}
```

```
    # Function to compare model performances visually
    def compare_models(final_scores):
        labels = list(final_scores.keys())
        train_acc = [final_scores[label]['Final Training Accuracy'] for label in labels]
        val_acc = [final_scores[label]['Final Validation Accuracy'] for label in labels]

        x = range(len(labels))
        plot.figure(figsize=(10, 5))

        plot.bar(x, train_acc, width=0.4, label='Training Accuracy', align='center')
        plot.bar([p + 0.4 for p in x], val_acc, width=0.4, label='Validation Accuracy', align='center')

        plot.xticks([p + 0.2 for p in x], labels, rotation=30)
        plot.ylim(0, 1.0)
        plot.title("Final Training vs Validation Accuracy")
        plot.xlabel("Model")
        plot.ylabel("Accuracy")
        plot.legend()
        plot.tight_layout()
        plot.show()
```

Start coding or generate with AI.