


Mounting Google Drive This cell correctly mounts Google Drive to access datasets stored in Google Colab. Simple and essential step for data loading.

Double-click (or enter) to edit

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

 Mounted at /content/drive

Double-click (or enter) to edit


Double-click (or enter) to edit

Extracting Dataset from ZIP Efficiently extracts the dataset from a ZIP file into the local directory – a common practice for managing large datasets in Colab.

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


Verifying Dataset Structure Lists the contents of the base directory to confirm the dataset has been extracted correctly and is organized into train, validation, and test folders.

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

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

Installing TensorFlow Installs TensorFlow, which is essential for building and training deep learning models. This is necessary if it's not pre-installed.

```
pip install tensorflow
```

 Collecting tensorflow
 Downloading tensorflow-2.19.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (456.5 MB)
 Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.19.0)
 Collecting astunparse>=1.6.0 (from tensorflow==2.19.0)
 Downloading astunparse-1.6.3-py2.py3-none-any.whl.metadata (4.4 kB)
 Collecting flatbuffers>=24.3.25 (from tensorflow==2.19.0)

```

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/
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-pa
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: protobuf!=4.21.0,!4.21.1,!4.21.2,!4.21.3,!4.21.4,!
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-
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-package
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.11/c
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.n
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
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: markdown>=2.6.8 in /usr/lib/python3/dist-packages (fro
Collecting tensorboard-data-server<0.8.0,>=0.7.0 (from tensorboard~2.19.0->tensorflo
Downloading tensorboard_data_server-0.7.2-py3-none-manylinux_2_31_x86_64.whl.metada
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-pa
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/c
Requirement already satisfied: mdurl~0.1 in /usr/local/lib/python3.11/dist-packages
Downloading tensorflow-2.19.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.
_____ 644.9/644.9 MB 1.1 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 4.0 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 47.1 MB/s eta 0:00:00

```

Importing Libraries and Setting Paths Well-organized import block and dataset paths setup; this prepares the environment for model development and data loading.

```
import os
import pathlib
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, applications
```

```
# Set the correct path to your dataset
training_path = '/content/cats_vs_dogs_small/train'
validation_path = '/content/cats_vs_dogs_small/validation'
testing_path = '/content/cats_vs_dogs_small/test'
```

Loading the Datasets This block uses TensorFlow's `image_dataset_from_directory` to load and preprocess the train, validation, and test datasets efficiently. Specifying image size and batch size ensures consistent input shape and manageable computation during training.

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

ds_val = keras.preprocessing.image_dataset_from_directory(
    validation_path,
    image_size=(180, 180),
    batch_size=32)

ds_test = keras.preprocessing.image_dataset_from_directory(
    testing_path,
    image_size=(180, 180),
    batch_size=32)
```

```
➡ Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

Displaying Sample Images This function is a great way to visually inspect a batch of training images and verify that labels are correctly assigned. The use of `plt.subplot` provides a clean 3x3 grid layout, which helps in getting a quick feel for the dataset.

```
# Display sample images
def show_images(dataset):
    plt.figure(figsize=(10, 10))
    for images, model_labels in dataset.take(1):
        for i in range(9):
            ax = plt.subplot(3, 3, i + 1)
```

```
plt.imshow(images[i].numpy().astype("uint8"))  
plt.title("Cat" if model_labels[i] == 0 else "Dog")  
plt.axis("off")  
plt.show()
```

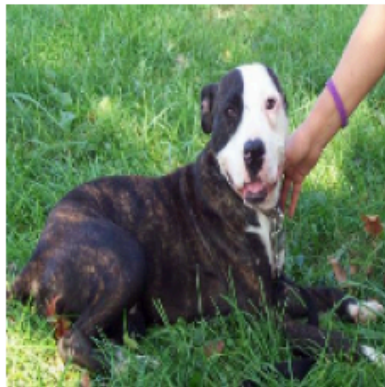
```
# Call the function to display images  
show_images(ds_train)
```



Dog



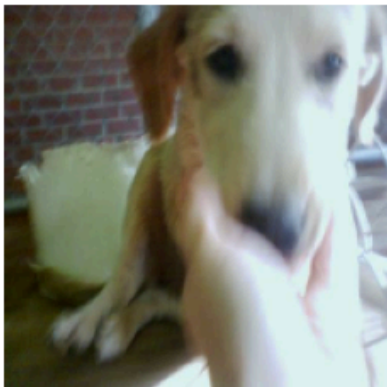
Dog



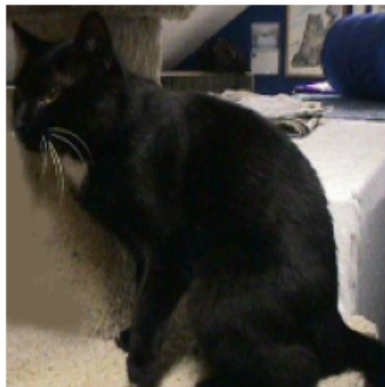
Dog



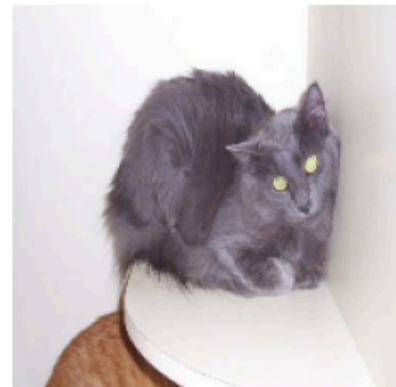
Dog



Cat



Cat



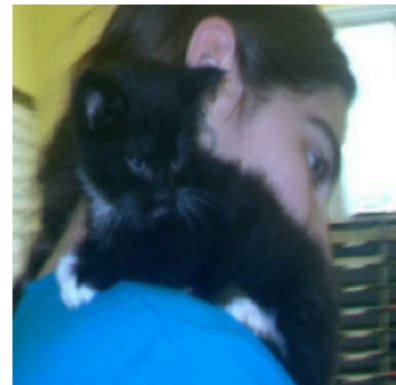
Dog



Cat



Cat



Custom CNN Model Definition This function builds a straightforward but effective CNN architecture for binary classification. The increasing number of filters and use of max pooling layers help extract spatial features, while the final dense layers classify the images. Using sigmoid activation and binary_crossentropy is appropriate for the two-class (cat vs. dog) problem.

```
# Function to create a CNN model from scratch
def build_custom_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 Generator Setup This function enhances the training dataset using real-time data augmentation, which helps improve model generalization. It uses ImageDataGenerator to apply transformations like rotation, zoom, and flipping, which are especially useful for image classification tasks. The validation generator is appropriately left unaugmented to ensure consistent evaluation.

```
# Data augmentation and preprocessing with sample size parameter
def prepare_data_flows(training_path, validation_path, batch_size, num_samples=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)

    gen_train = train_datagen.flow_from_directory(
        training_path,
        target_size=(180, 180),
```

```

        batch_size=batch_size,
        class_mode='binary'
    )

    val_generator = val_datagen.flow_from_directory(
        validation_path,
        target_size=(180, 180),
        batch_size=batch_size,
        class_mode='binary'
    )

    return gen_train, val_generator

```

Model Training Function This function handles the training loop using the fit method and tracks performance over epochs. Returning the history object is useful for later analysis or visualization of training and validation metrics.

```

# Train the model
def fit_model(model, gen_train, gen_val, epochs=30):
    history = model.fit(gen_train,
                        validation_data=gen_val,
                        epochs=epochs)

    return history

```

Training the Baseline Model (Model A) This step initiates training of the custom CNN using a smaller subset of 1000 training samples, which is useful for benchmarking and observing the impact of limited data. The modular approach with reusable functions keeps the workflow clean and flexible.

```

# Step 1: Train model from scratch with 1000 samples
train_generator_1, validation_generator_1 = prepare_data_flows(training_path, validation_path)

model_A = build_custom_model()

history_A = fit_model(model_A, train_generator_1, validation_generator_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:1
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:1
    self._warn_if_super_not_called()
Epoch 1/30

```



```

63/63 ————— 27s 394ms/step - accuracy: 0.5164 - loss: 0.7758 - val_acc
Epoch 2/30
63/63 ————— 25s 391ms/step - accuracy: 0.5536 - loss: 0.6817 - val_acc
Epoch 3/30
63/63 ————— 24s 384ms/step - accuracy: 0.5283 - loss: 0.6808 - val_acc
Epoch 4/30
63/63 ————— 25s 388ms/step - accuracy: 0.5976 - loss: 0.6639 - val_acc
Epoch 5/30
63/63 ————— 25s 387ms/step - accuracy: 0.6259 - loss: 0.6361 - val_acc
Epoch 6/30
63/63 ————— 25s 392ms/step - accuracy: 0.6591 - loss: 0.6311 - val_acc
Epoch 7/30
63/63 ————— 25s 391ms/step - accuracy: 0.6733 - loss: 0.6000 - val_acc
Epoch 8/30
63/63 ————— 25s 388ms/step - accuracy: 0.6611 - loss: 0.6142 - val_acc
Epoch 9/30
63/63 ————— 24s 386ms/step - accuracy: 0.6791 - loss: 0.6120 - val_acc
Epoch 10/30
63/63 ————— 25s 390ms/step - accuracy: 0.7142 - loss: 0.5630 - val_acc
Epoch 11/30
63/63 ————— 25s 391ms/step - accuracy: 0.6650 - loss: 0.6097 - val_acc
Epoch 12/30
63/63 ————— 25s 391ms/step - accuracy: 0.6731 - loss: 0.5900 - val_acc
Epoch 13/30
63/63 ————— 25s 389ms/step - accuracy: 0.6975 - loss: 0.5841 - val_acc
Epoch 14/30
63/63 ————— 25s 386ms/step - accuracy: 0.6953 - loss: 0.5736 - val_acc
Epoch 15/30
63/63 ————— 24s 383ms/step - accuracy: 0.7186 - loss: 0.5510 - val_acc
Epoch 16/30
63/63 ————— 25s 387ms/step - accuracy: 0.7171 - loss: 0.5562 - val_acc
Epoch 17/30
63/63 ————— 24s 385ms/step - accuracy: 0.6927 - loss: 0.5711 - val_acc
Epoch 18/30
63/63 ————— 25s 388ms/step - accuracy: 0.7343 - loss: 0.5226 - val_acc
Epoch 19/30
63/63 ————— 25s 388ms/step - accuracy: 0.7213 - loss: 0.5427 - val_acc
Epoch 20/30
63/63 ————— 25s 388ms/step - accuracy: 0.7536 - loss: 0.5024 - val_acc
Epoch 21/30
63/63 ————— 24s 382ms/step - accuracy: 0.7310 - loss: 0.5393 - val_acc
Epoch 22/30
63/63 ————— 24s 376ms/step - accuracy: 0.7530 - loss: 0.5115 - val_acc
Epoch 23/30
63/63 ————— 25s 386ms/step - accuracy: 0.7332 - loss: 0.5278 - val_acc
Epoch 24/30
63/63 ————— 24s 378ms/step - accuracy: 0.7233 - loss: 0.5385 - val_acc
Epoch 25/30
63/63 ————— 24s 379ms/step - accuracy: 0.7436 - loss: 0.5152 - val_acc

```

Training with More Data (Model B) This step increases the sample size to 1500, which allows you to observe how model performance improves with more training data. Keeping the model architecture consistent helps isolate the effect of dataset size.

```
# Step 2: Increase training samples to 1500
```

```
train_generator_2, validation_generator_2 = prepare_data_flows(training_path, validation_path)
```

```
model_B = build_custom_model()
```

```
history_B = fit_model(model_B, train_generator_2, validation_generator_2)
```



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

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

```
Epoch 1/30
```

```
63/63 ————— 26s 383ms/step - accuracy: 0.4924 - loss: 1.1050 - val_acc
```

```
Epoch 2/30
```

```
63/63 ————— 24s 380ms/step - accuracy: 0.5044 - loss: 0.6931 - val_acc
```

```
Epoch 3/30
```

```
63/63 ————— 24s 374ms/step - accuracy: 0.5144 - loss: 0.6913 - val_acc
```

```
Epoch 4/30
```

```
63/63 ————— 24s 381ms/step - accuracy: 0.5932 - loss: 0.6710 - val_acc
```

```
Epoch 5/30
```

```
63/63 ————— 24s 377ms/step - accuracy: 0.6033 - loss: 0.6640 - val_acc
```

```
Epoch 6/30
```

```
63/63 ————— 24s 377ms/step - accuracy: 0.5853 - loss: 0.6789 - val_acc
```

```
Epoch 7/30
```

```
63/63 ————— 24s 382ms/step - accuracy: 0.6355 - loss: 0.6576 - val_acc
```

```
Epoch 8/30
```

```
63/63 ————— 24s 382ms/step - accuracy: 0.6330 - loss: 0.6495 - val_acc
```

```
Epoch 9/30
```

```
63/63 ————— 24s 379ms/step - accuracy: 0.6295 - loss: 0.6301 - val_acc
```

```
Epoch 10/30
```

```
63/63 ————— 24s 379ms/step - accuracy: 0.6549 - loss: 0.6227 - val_acc
```

```
Epoch 11/30
```

```
63/63 ————— 24s 375ms/step - accuracy: 0.6800 - loss: 0.5876 - val_acc
```

```
Epoch 12/30
```

```
63/63 ————— 24s 379ms/step - accuracy: 0.6701 - loss: 0.6095 - val_acc
```

```
Epoch 13/30
```

```
63/63 ————— 24s 384ms/step - accuracy: 0.6954 - loss: 0.5867 - val_acc
```

```
Epoch 14/30
```

```
63/63 ————— 24s 378ms/step - accuracy: 0.7184 - loss: 0.5573 - val_acc
```

```
Epoch 15/30
```

```
63/63 ————— 24s 381ms/step - accuracy: 0.7145 - loss: 0.5608 - val_acc
```

```
Epoch 16/30
```

```
63/63 ————— 24s 378ms/step - accuracy: 0.7219 - loss: 0.5314 - val_acc
```

```
Epoch 17/30
```

```
63/63 ————— 24s 379ms/step - accuracy: 0.7246 - loss: 0.5347 - val_acc
```

```
Epoch 18/30
```

```
63/63 ————— 24s 377ms/step - accuracy: 0.7242 - loss: 0.5307 - val_acc
```

```
Epoch 19/30
```

```
63/63 ————— 24s 379ms/step - accuracy: 0.7453 - loss: 0.5252 - val_acc
```

```
Epoch 20/30
```

```
63/63 ————— 24s 379ms/step - accuracy: 0.7191 - loss: 0.5544 - val_acc
```

```
Epoch 21/30
```

```
63/63 ————— 24s 378ms/step - accuracy: 0.7281 - loss: 0.5383 - val_acc
```

```
Epoch 22/30
```

```
63/63 ————— 24s 379ms/step - accuracy: 0.7401 - loss: 0.5145 - val_acc
```

```
Epoch 23/30
```

```
63/63 ————— 24s 378ms/step - accuracy: 0.7306 - loss: 0.5254 - val_acc
```

```
Epoch 24/30
```



```

63/63 ————— 24s 378ms/step - accuracy: 0.7666 - loss: 0.5008 - val_acc
Epoch 25/30
63/63 ————— 24s 378ms/step - accuracy: 0.7490 - loss: 0.5140 - val_acc
Epoch 26/30
63/63 ————— 24s 382ms/step - accuracy: 0.7324 - loss: 0.5290 - val_acc
Epoch 27/30
63/63 ————— 24s 376ms/step - accuracy: 0.7525 - loss: 0.4840 - val_acc

```

Training with Full Dataset (Model C) This final training phase uses all 2000 samples, completing the comparison across dataset sizes. This progressive approach effectively demonstrates the impact of training data volume on model performance.

Step 3: Use the full 2000 samples

```

train_generator_3, validation_generator_3 = prepare_data_flows(training_path, validation_path)
model_C = build_custom_model()
history_C = fit_model(model_C, train_generator_3, validation_generator_3)

```

```

⇒ 63/63 ————— 26s 387ms/step - accuracy: 0.4883 - loss: 0.9694 - val_acc
Epoch 2/30
63/63 ————— 24s 380ms/step - accuracy: 0.5606 - loss: 0.6848 - val_acc
Epoch 3/30
63/63 ————— 24s 382ms/step - accuracy: 0.5791 - loss: 0.6809 - val_acc
Epoch 4/30
63/63 ————— 24s 383ms/step - accuracy: 0.5613 - loss: 0.6781 - val_acc
Epoch 5/30
63/63 ————— 24s 383ms/step - accuracy: 0.6077 - loss: 0.6500 - val_acc
Epoch 6/30
63/63 ————— 24s 382ms/step - accuracy: 0.5848 - loss: 0.6564 - val_acc
Epoch 7/30
63/63 ————— 24s 384ms/step - accuracy: 0.6444 - loss: 0.6413 - val_acc
Epoch 8/30
63/63 ————— 24s 383ms/step - accuracy: 0.6406 - loss: 0.6415 - val_acc
Epoch 9/30
63/63 ————— 24s 380ms/step - accuracy: 0.6804 - loss: 0.6059 - val_acc
Epoch 10/30
63/63 ————— 24s 379ms/step - accuracy: 0.6835 - loss: 0.6153 - val_acc
Epoch 11/30
63/63 ————— 24s 381ms/step - accuracy: 0.6771 - loss: 0.5975 - val_acc
Epoch 12/30
63/63 ————— 24s 377ms/step - accuracy: 0.6854 - loss: 0.5813 - val_acc
Epoch 13/30
63/63 ————— 24s 382ms/step - accuracy: 0.6835 - loss: 0.5895 - val_acc
Epoch 14/30
63/63 ————— 24s 379ms/step - accuracy: 0.6803 - loss: 0.6077 - val_acc
Epoch 15/30
63/63 ————— 24s 380ms/step - accuracy: 0.7114 - loss: 0.5598 - val_acc
Epoch 16/30
63/63 ————— 24s 378ms/step - accuracy: 0.7155 - loss: 0.5506 - val_acc

```

```

63/63 ————— 24s 379ms/step - accuracy: 0.7240 - loss: 0.5647 - val_acc
Epoch 19/30
63/63 ————— 24s 384ms/step - accuracy: 0.7473 - loss: 0.5252 - val_acc
Epoch 20/30
63/63 ————— 24s 384ms/step - accuracy: 0.7110 - loss: 0.5642 - val_acc
Epoch 21/30
63/63 ————— 24s 382ms/step - accuracy: 0.7092 - loss: 0.5603 - val_acc
Epoch 22/30
63/63 ————— 24s 378ms/step - accuracy: 0.7464 - loss: 0.5148 - val_acc
Epoch 23/30
63/63 ————— 24s 377ms/step - accuracy: 0.7649 - loss: 0.4979 - val_acc
Epoch 24/30
63/63 ————— 24s 381ms/step - accuracy: 0.7569 - loss: 0.5003 - val_acc
Epoch 25/30
63/63 ————— 24s 379ms/step - accuracy: 0.7486 - loss: 0.5091 - val_acc
Epoch 26/30
63/63 ————— 24s 383ms/step - accuracy: 0.7269 - loss: 0.5371 - val_acc
Epoch 27/30
63/63 ————— 24s 380ms/step - accuracy: 0.7407 - loss: 0.5275 - val_acc
Epoch 28/30
63/63 ————— 24s 386ms/step - accuracy: 0.7633 - loss: 0.5073 - val_acc
Epoch 29/30
63/63 ————— 24s 383ms/step - accuracy: 0.7500 - loss: 0.5105 - val_acc

```

Transfer Learning with VGG16 (Model D) This function leverages a pretrained VGG16 model as a fixed feature extractor, which is ideal for small datasets. Freezing the base model and adding dense layers on top helps speed up training and often boosts accuracy due to learned image features.

```

# Step 4: Use a pretrained model (e.g., VGG16)
def build_pretrained_model():
    base_model = applications.VGG16(include_top=False, weights='imagenet', input_shape=(180,
    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

```

Training with Pretrained Model (VGG16) on Varying Sample Sizes This chunk mirrors the earlier experiments but now applies transfer learning with VGG16 across all sample sizes. This parallel comparison will help demonstrate how pretrained models perform better than models trained from scratch, especially on small datasets.

```
# Repeat Steps 1-3 with the pretrained model
model_P1 = build_pretrained_model()
history_P1 = fit_model(model_P1, train_generator_1, validation_generator_1)

model_P2 = build_pretrained_model()
history_P2 = fit_model(model_P2, train_generator_2, validation_generator_2)

model_P3 = build_pretrained_model()
history_P3 = fit_model(model_P3, train_generator_3, validation_generator_3)
```



```

Epoch 9/30
63/63 ————— 39s 613ms/step - accuracy: 0.8791 - loss: 0.2756 - val_acc
Epoch 10/30
63/63 ————— 38s 605ms/step - accuracy: 0.8926 - loss: 0.2633 - val_acc
Epoch 11/30
63/63 ————— 39s 611ms/step - accuracy: 0.8719 - loss: 0.2964 - val_acc
Epoch 12/30
63/63 ————— 39s 621ms/step - accuracy: 0.8900 - loss: 0.2561 - val_acc
Epoch 13/30
63/63 ————— 38s 610ms/step - accuracy: 0.8916 - loss: 0.2486 - val_acc
Epoch 14/30
63/63 ————— 39s 613ms/step - accuracy: 0.8822 - loss: 0.2568 - val_acc

```

Performance Visualization This function provides clear and insightful plots of accuracy and loss across training epochs, allowing for an easy comparison of how different models perform with various dataset sizes. Plotting both training and validation metrics helps identify underfitting, overfitting, and the benefits of transfer learning.

```

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

    epochs = range(len(acc))

    plt.figure()
    plt.plot(epochs, acc, 'b', label='Training acc')
    plt.plot(epochs, val_acc, 'r', label='Validation acc')
    plt.title(title + ' Accuracy')
    plt.legend()

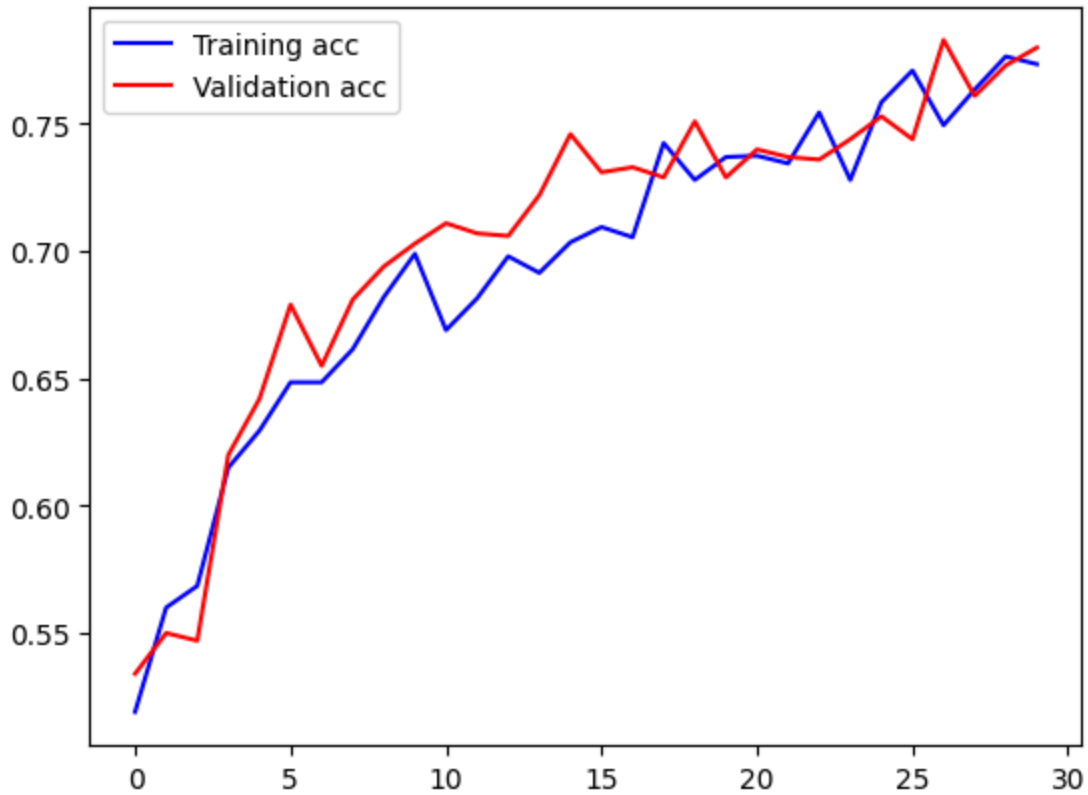
    plt.figure()
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')
    plt.title(title + ' Loss')
    plt.legend()
    plt.show()

# Plotting performance for each model
visualize_performance(history_A, 'Model from Scratch - 1000 Samples')
visualize_performance(history_B, 'Model from Scratch - 1500 Samples')
visualize_performance(history_C, 'Model from Scratch - 2000 Samples')
visualize_performance(history_P1, 'Pretrained Model - 1000 Samples')
visualize_performance(history_P2, 'Pretrained Model - 1500 Samples')
visualize_performance(history_P3, 'Pretrained Model - 2000 Samples')

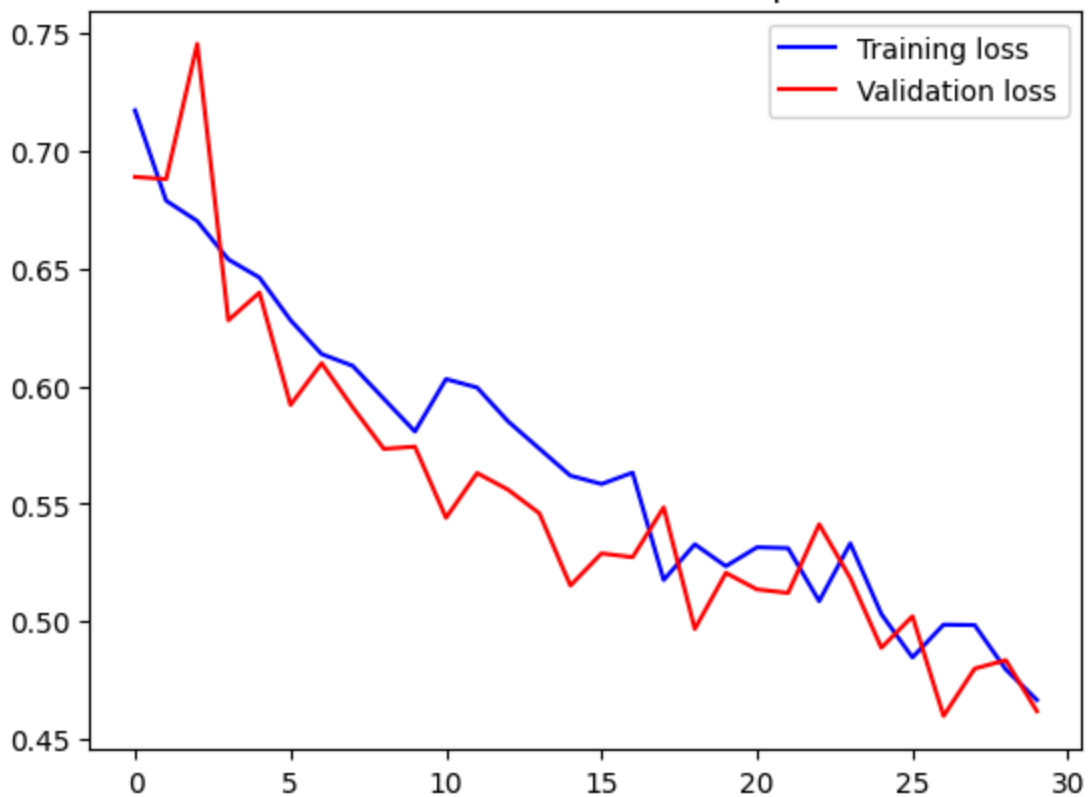
```



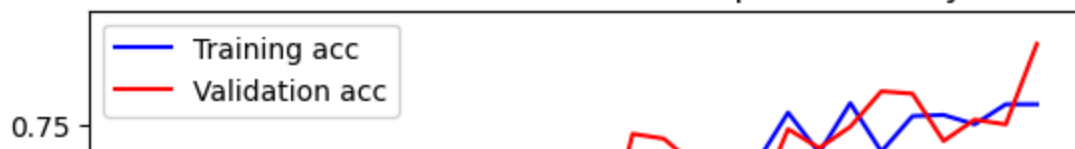
Model from Scratch - 1000 Samples Accuracy

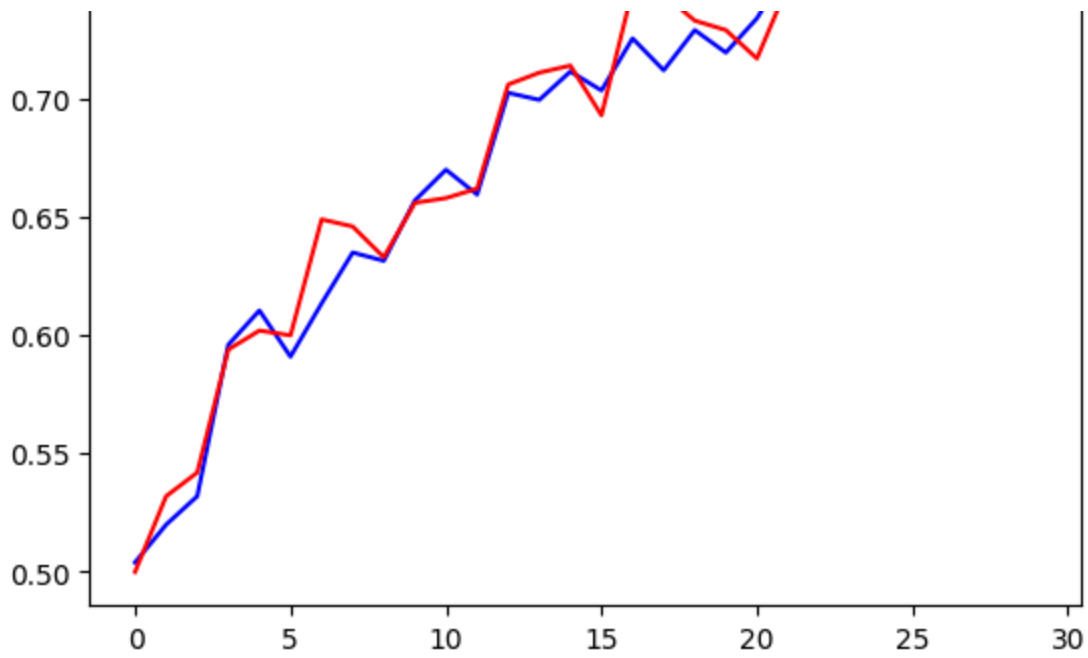


Model from Scratch - 1000 Samples Loss

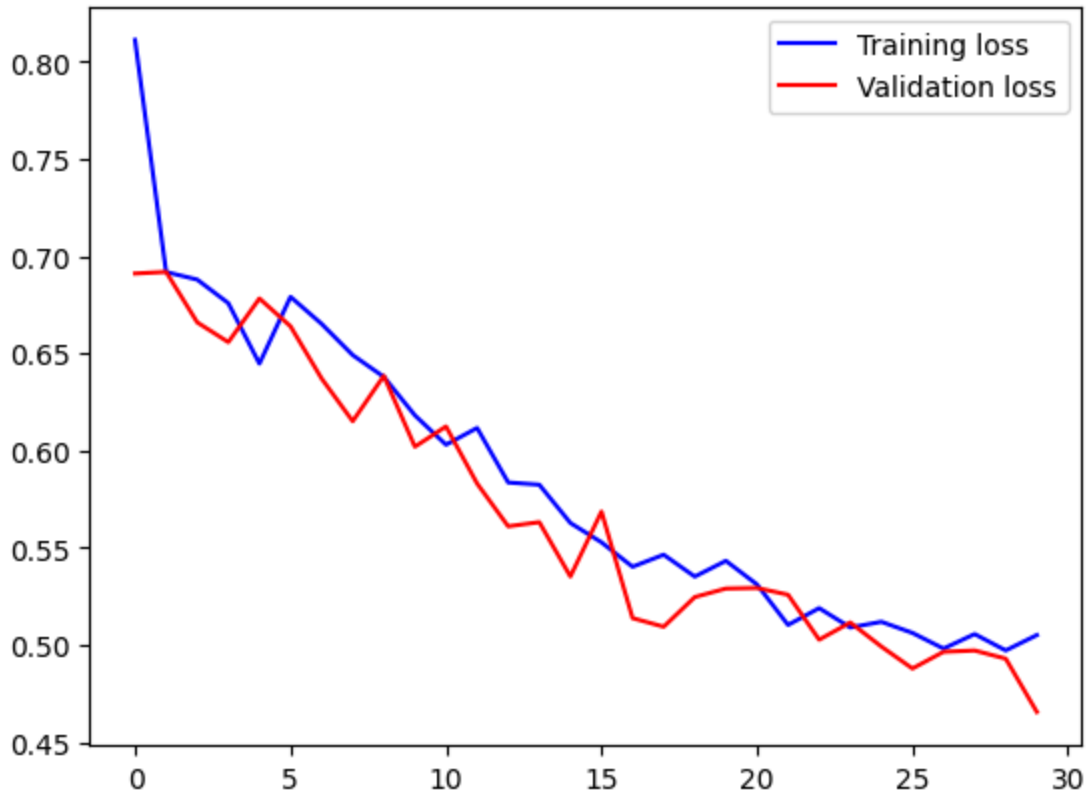


Model from Scratch - 1500 Samples Accuracy

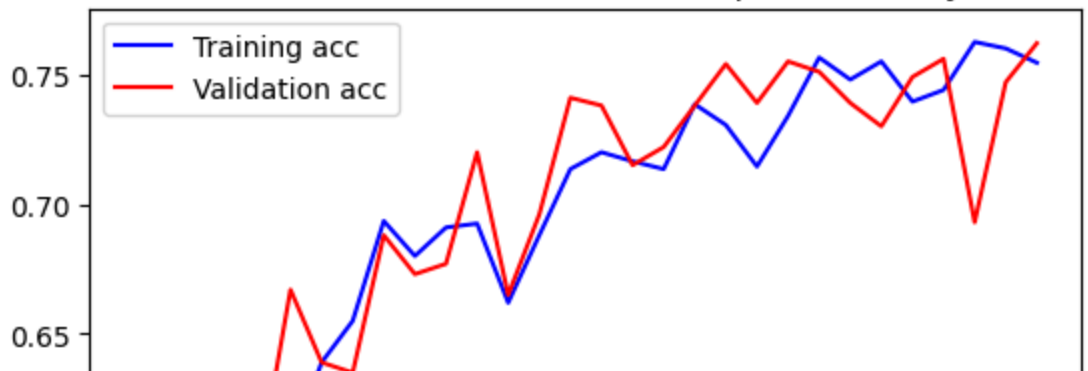


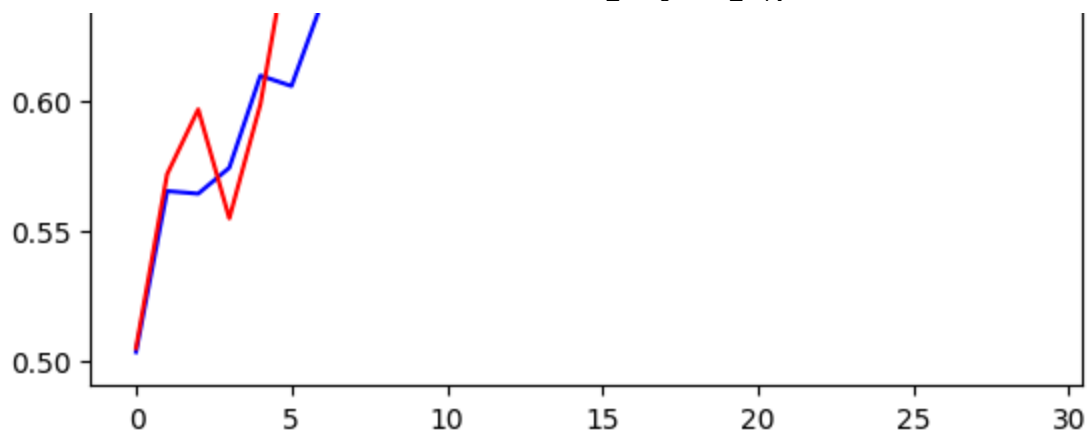


Model from Scratch - 1500 Samples Loss



Model from Scratch - 2000 Samples Accuracy

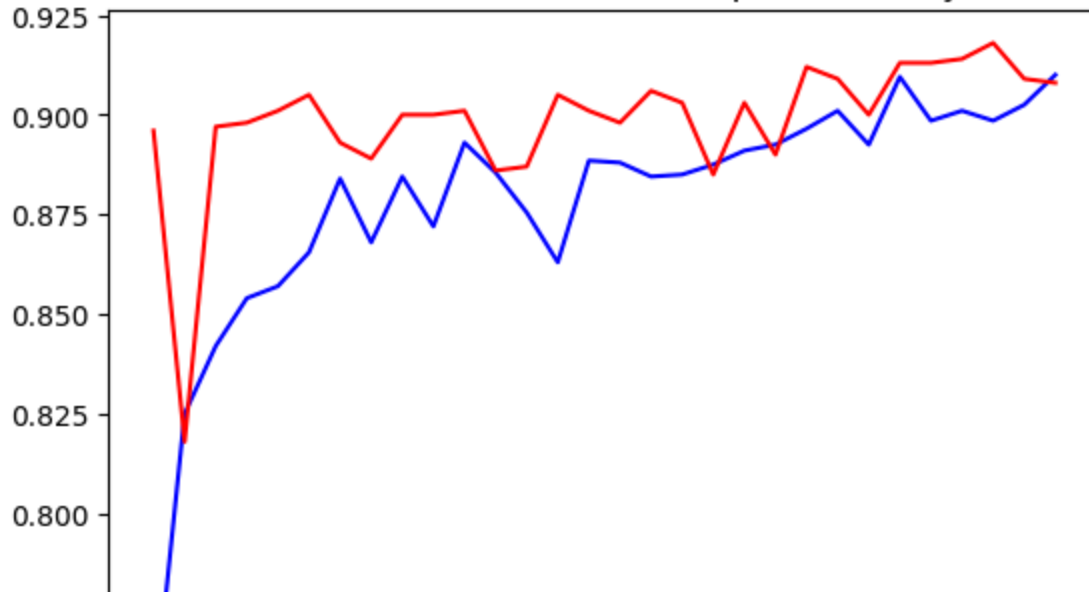


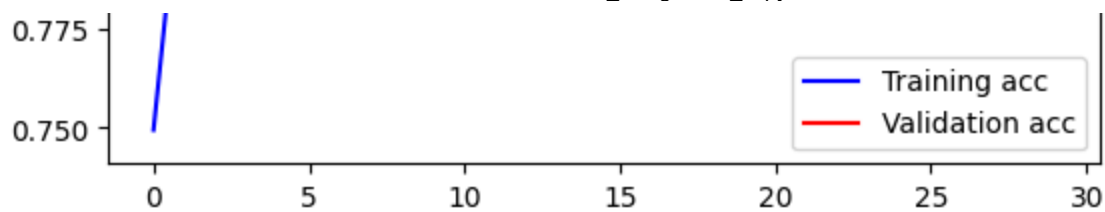


Model from Scratch - 2000 Samples Loss

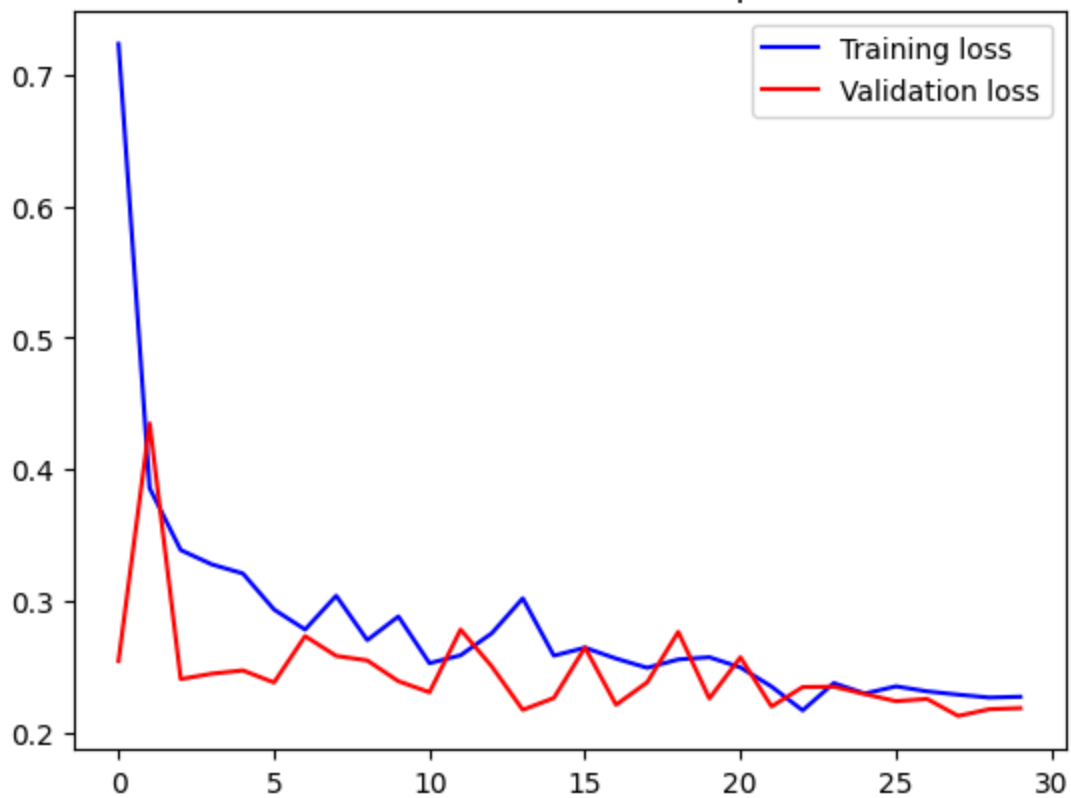


Pretrained Model - 1000 Samples Accuracy

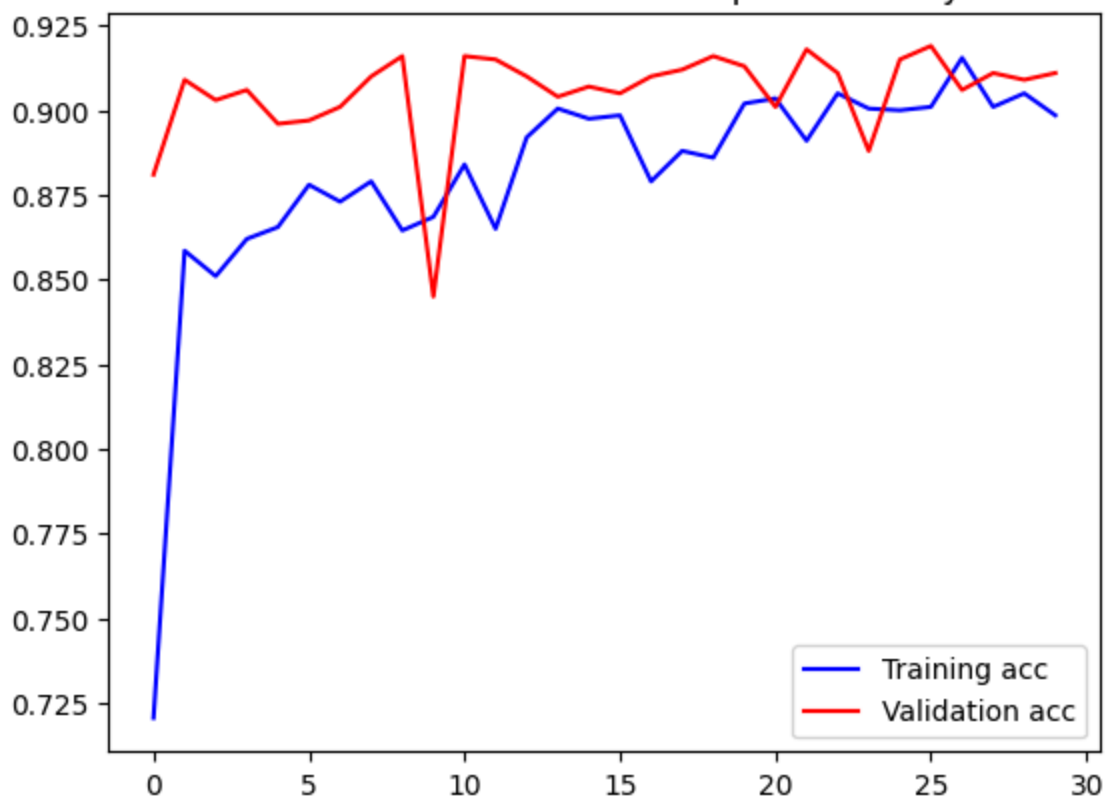




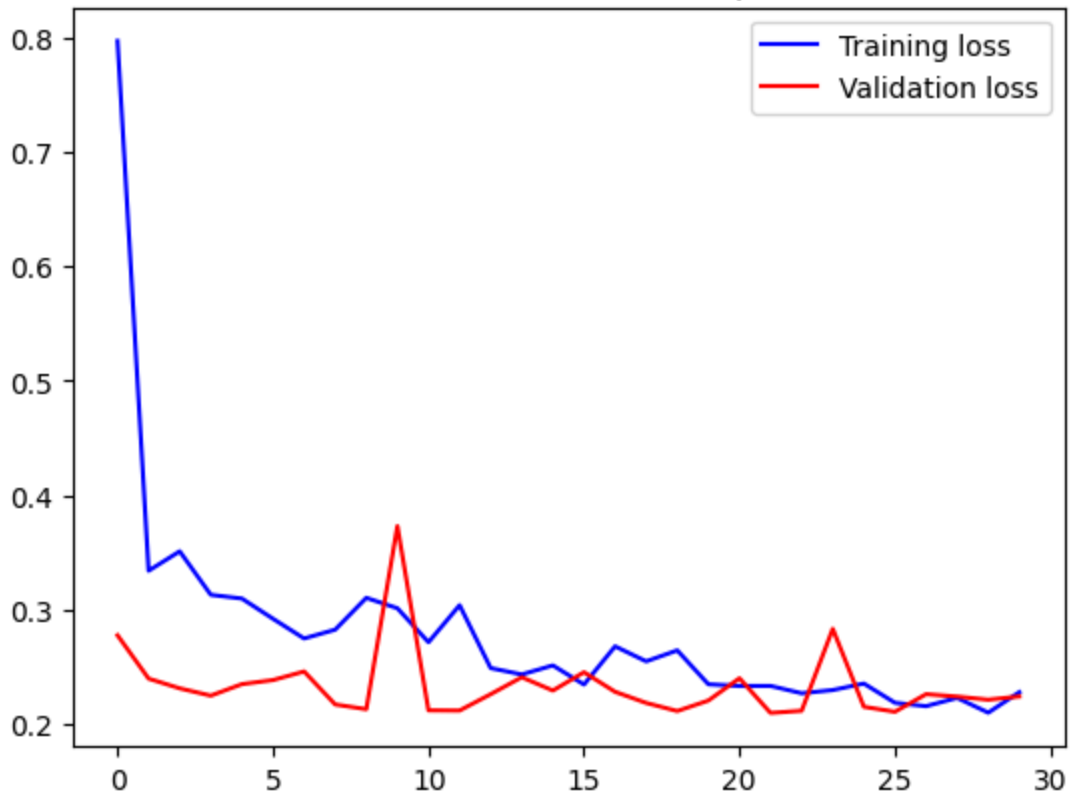
Pretrained Model - 1000 Samples Loss



Pretrained Model - 1500 Samples Accuracy



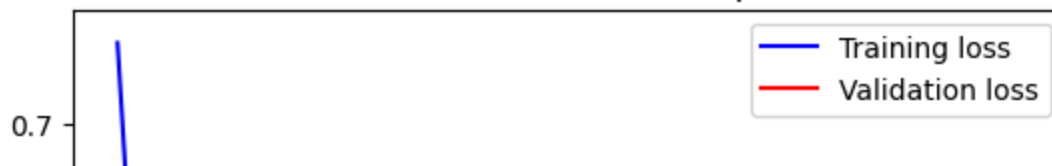
Pretrained Model - 1500 Samples Loss

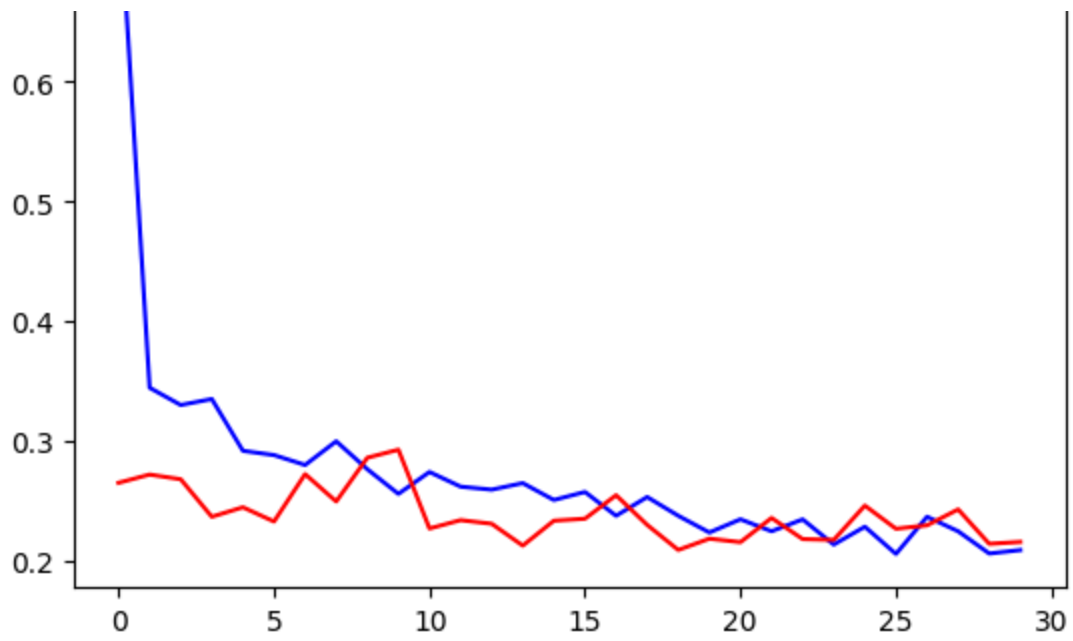


Pretrained Model - 2000 Samples Accuracy



Pretrained Model - 2000 Samples Loss





Result Aggregation for Summary Metrics This function neatly summarizes final training and validation performance for all models, making it easy to compare and report outcomes. The structured output is ideal for drawing conclusions about the impact of training size and the use of pretrained models.

```
# Function to summarize final results
def aggregate_results(all_histories, model_labels):
    summary = {}

    for i, history in enumerate(all_histories):
        final_train_acc = history.history['accuracy'][-1]
        final_val_acc = history.history['val_accuracy'][-1]
        final_train_loss = history.history['loss'][-1]
        final_val_loss = history.history['val_loss'][-1]

        summary[model_labels[i]] = {
            'Final Training Accuracy': final_train_acc,
            'Final Validation Accuracy': final_val_acc,
            'Final Training Loss': final_train_loss,
            'Final Validation Loss': final_val_loss,
        }

    return summary

# Example of using the aggregate_results function
all_histories = [history_A, history_B, history_C,
                 history_P1, history_P2, history_P3]

model_labels = [
    'Model from Scratch - 1000 Samples',
    'Model from Scratch - 1500 Samples',
    'Model from Scratch - 2000 Samples',
    'Pretrained Model - 1000 Samples',
    'Pretrained Model - 1500 Samples',
    'Pretrained Model - 2000 Samples'
]

# Get the summary of final values only
```

Visual Comparison of Final Model Performances This final visualization function gives a concise side-by-side comparison of training and validation metrics across all models. The use of line plots with markers helps emphasize performance trends, and rotating the labels improves readability—excellent for wrapping up your experiment.

```
# Function to compare model performances visually
def compare_models(final_scores):
```

```
model_labels = list(final_scores.keys())
train_acc = [final_scores[label]['Final Training Accuracy'] for label in model_labels]
val_acc = [final_scores[label]['Final Validation Accuracy'] for label in model_labels]
train_loss = [final_scores[label]['Final Training Loss'] for label in model_labels]
val_loss = [final_scores[label]['Final Validation Loss'] for label in model_labels]

# Accuracy comparison plot
plt.figure(figsize=(10, 5))
plt.plot(model_labels, train_acc, label='Training Accuracy', marker='o')
plt.plot(model_labels, val_acc, label='Validation Accuracy', marker='o')
plt.xticks(rotation=45)
plt.title('Final Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Loss comparison plot
plt.figure(figsize=(10, 5))
plt.plot(model_labels, train_loss, label='Training Loss', marker='o')
plt.plot(model_labels, val_loss, label='Validation Loss', marker='o')
plt.xticks(rotation=45)
plt.title('Final Loss Comparison')
plt.xlabel('Model')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Call aggregate_results to get the final scores
final_scores = aggregate_results(all_histories, model_labels) # This line was added

# Plot the comparisons
compare_models(final_scores)
```