Project 1: Chest X-Ray image classification

Alireza Amiri

Part 1 - Image Classification

As the first part of the project, you should use your skills in image classification to solve this problem. You should train your image classifier on the dataset. All the design parameters including type of the network, architecture, train parameters, evaluation metrics, batch size, optimizer, the input resolution, etc. are arbitrarily and should be chosen based on your knowledge, so be careful about your design. The minimum requirement of this part is:

(a) Training a MLP network with optimized parameters (layers, neurons, . . .)

(b) Training a CNN network with optimized parameters (layers, kernel size, . . .)

(c) Train at least 2 networks with state-of-the-art architectures (VGG16, VGG19, ResNet50, Inceptionv3, Mo�bileNetv2, Xception, . . .) as the feature extractors. Freeze all the layers in the networks and train your own classifier on top of them.

(d) Train the networks from (c) again but unfreeze last 2 layers.

(e) Train the networks from (c) again but unfreeze last 6 layers.

(f) Train the best performing network from sections (a)–(e) again, with and without data augmentation.

Data preparation

Downloading the dataset and unzipping it

```
#!gdown 15KhnKTU89tZEy3CYbN-JFdhViiyiJQDS
!unzip chest-xray-pneumonia.zip -d chest-xray-pneumonia
y
```

```
Downloading...
From (original): https://drive.google.com/uc?id=15KhnKTU89tZEy3CYbN-JFdhViiyiJQDS
From (redirected): https://drive.google.com/uc?id=15KhnKTU89tZEy3CYbN-JFdhViiyiJQDS&confirm=t&uuid=919d5daf-2755-4f95-a841-07ddf452a4ce
To: /content/chest-xray-pneumonia.zip
100% 1.22G/1.22G [00:14<00:00, 81.9MB/s]
Archive:  chest-xray-pneumonia.zip
replace chest-xray-pneumonia/content/chest-xray-pneumonia/test/NORMAL/NORMAL2-IM-0206-0001.jpeg? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

Organizing the files and folders to match the requirements of the project.

according to downloaded dataset and the description of homework, we can see that images are categorised in Train,Test and Validation folders. inside each, there are 2 folders named Normal and Pneumonia.

The point to consider is that our target in this project is to classify images into 3 classes as below:

1. Normal
2. Pneumonia-bacteria
3. Pneumonia-virus

We can see that the separation of Normal and Pneumonia is already done and they are seperated in different folders, but to consider the type of dise, which is bacteria and virus, we must run another process.

In this case, we can see the name of diseases type is written in each image's file name. so in order to seperate them, we must search for words *bacteria* and *virus* in each file name, and then create a new folder with a new architecture to includ these 3 classes.

This is done using the code below:

First, downloaded files directories are saved into 3 variables, and then we create new directories and folders with the same structure but with 3 classes instead of 2.

Then, the files from original folder are copied to the recently created folders. The consideration to make here is that all Normal images are directly copied to corresponding folder, but the images of Pneumonia class are first seperated based on their disease type and then moved to the corresponding folder.

```
import os
import shutil
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.applications import VGG16
from sklearn.metrics import classification_report, confusion_matrix

# Define original directories
original_train_dir = 'C://Users//lenovo//Documents//AI and ES//Project 1//content//chest-xray-pneumonia//train'
original_val_dir = 'C://Users//lenovo//Documents//AI and ES//Project 1//content//chest-xray-pneumonia//val'
original_test_dir = 'C://Users//lenovo//Documents//AI and ES//Project 1//content//chest-xray-pneumonia//test'

# Define new directories
base_dir = 'C:\\Users\\lenovo\Documents\\AI and ES\\Project 1\\content\\chest-xray-pneumonia-preprocessed'
new_train_dir = os.path.join(base_dir, 'train')
new_val_dir = os.path.join(base_dir, 'val')
new_test_dir = os.path.join(base_dir, 'test')

# Create new directory structure
for dir_path in [new_train_dir, new_val_dir, new_test_dir]:
    os.makedirs(os.path.join(dir_path, 'healthy'), exist_ok=True)
    os.makedirs(os.path.join(dir_path, 'pneumonia-bacteria'), exist_ok=True)
    os.makedirs(os.path.join(dir_path, 'pneumonia-virus'), exist_ok=True)

def move_files(original_dir, new_dir):
    for category in ['NORMAL', 'PNEUMONIA']:
        category_path = os.path.join(original_dir, category)
        if category == 'NORMAL':
            for filename in os.listdir(category_path):
                src = os.path.join(category_path, filename)
                dst = os.path.join(new_dir, 'healthy', filename)
                shutil.copy(src, dst)
        else:
            for filename in os.listdir(category_path):
                if 'bacteria' in filename:
                    dst_category = 'pneumonia-bacteria'
                elif 'virus' in filename:
                    dst_category = 'pneumonia-virus'
                else:
                    continue
                src = os.path.join(category_path, filename)
                dst = os.path.join(new_dir, dst_category, filename)
                shutil.copy(src, dst)

# Move files to the new directory structure
move_files(original_train_dir, new_train_dir)
move_files(original_val_dir, new_val_dir)
move_files(original_test_dir, new_test_dir)
```

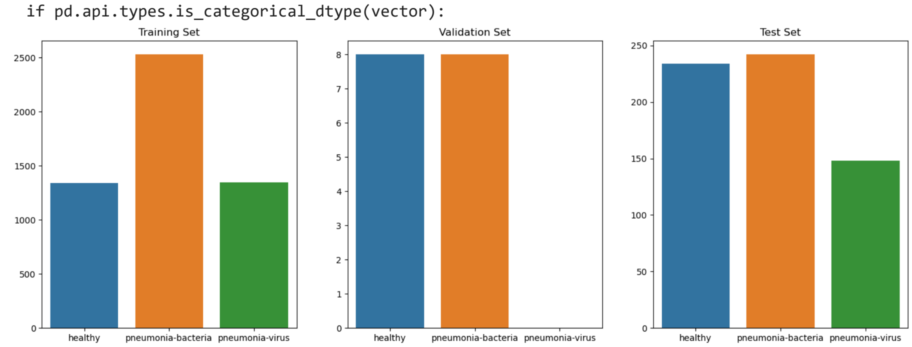
Data Analysis

Having the files organised properly, we can get some analytical information of the given dataset.

This is done by counting the number of images in each folder and also counting the number of each label present in each folder. at the end, bar charts are plotted.

```
# Count the number of images in each class
def count_images(directory):
    count_dict = {}
    for subdir in os.listdir(directory):
        subdir_path = os.path.join(directory, subdir)
        count_dict[subdir] = len(os.listdir(subdir_path))
    return count_dict

train_counts = count_images(new_train_dir)
val_counts = count_images(new_val_dir)
test_counts = count_images(new_test_dir)

# Plot the distribution of the dataset
fig, axs = plt.subplots(1, 3, figsize=(18, 6))
sns.barplot(x=list(train_counts.keys()), y=list(train_counts.values()), ax=axs[0]).set_title('Training Set')
sns.barplot(x=list(val_counts.keys()), y=list(val_counts.values()), ax=axs[1]).set_title('Validation Set')
sns.barplot(x=list(test_counts.keys()), y=list(test_counts.values()), ax=axs[2]).set_title('Test Set')
plt.show()
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categoric
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categoric
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1765: FutureWarning: unique with
  order = pd.unique(vector)
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categoric
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categoric
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categoric
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1765: FutureWarning: unique with
  order = pd.unique(vector)
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categoric
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categoric
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categoric
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1765: FutureWarning: unique with
  order = pd.unique(vector)
C:\Users\lenovo\anaconda3\lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categoric
  if pd.api.types.is_categorical_dtype(vector):
```



Preprocessing

Next, we apply the preprocessings required on the resulting dataset.

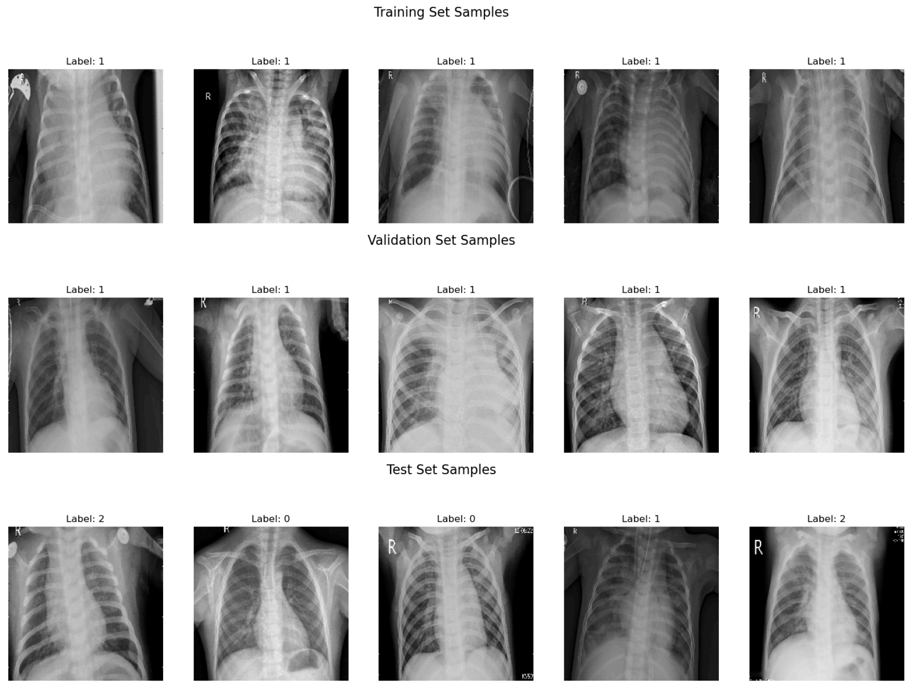
The process includes data augmentation in which crops all the images to an specific size and then add rotation and zoom to images to enable the model to learn patterns in data. also, the value of pixels are normalized by dividing their values by 255.

At the end, sample files are presented.

```
# Display some sample images with their labels
def display_sample_images(generator, title):
```

```
def display_sample_images(generator, title):
    x, y = next(generator)
    fig, axes = plt.subplots(1, 5, figsize=(20, 5))
    fig.suptitle(title, fontsize=16)
    for i in range(5):
        axes[i].imshow(x[i])
        axes[i].set_title(f"Label: {np.argmax(y[i])}")
        axes[i].axis('off')
    plt.show()

# Define new paths
train_dir = new_train_dir
val_dir = new_val_dir
test_dir = new_test_dir

# Define parameters
IMG_SIZE = (224, 224)
BATCH_SIZE = 32

# Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Load data
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    train_dir,  # Using train_dir with validation_split
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

# Display sample images
# Display sample images
display_sample_images(train_generator, 'Training Set Samples')
display_sample_images(val_generator, 'Validation Set Samples')
display_sample_images(test_generator, 'Test Set Samples')

display
```

```
Found 4173 images belonging to 3 classes.
Found 1043 images belonging to 3 classes.
Found 624 images belonging to 3 classes.
```




```
<function IPython.core.display_functions.display(*objs, include=None, exclude=None,
metadata=None, transient=None, display_id=None, raw=False, clear=False, **kwargs)>
```

Part (a): MLP model training:

The first step before training an MLP model is importing necessary packages.

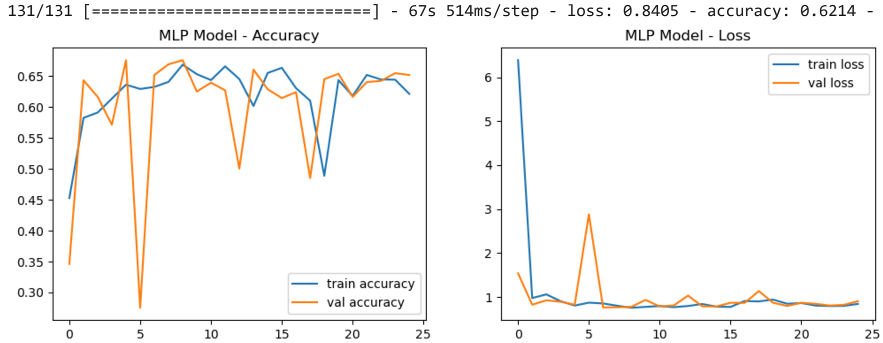
Next, we must define a class for plotting required charts (such as accuracy and loss) for evaluating each of the trained models

MLP architecture: A 4 layer MLP is first designed with input shape of (224,224,1), 2 Dense layers with 128 neuros, one layer with 64 neurons and a final layer with 3 neurons acting as output layer.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.applications import VGG16
from sklearn.metrics import classification_report, confusion_matrix



# Plot training history
def plot_training_history(history, title):
    fig, axs = plt.subplots(1, 2, figsize=(12, 4))
    axs[0].plot(history.history['accuracy'], label='train accuracy')
    axs[0].plot(history.history['val_accuracy'], label='val accuracy')
    axs[0].set_title(f'{title} - Accuracy')
    axs[0].legend()
    axs[1].plot(history.history['loss'], label='train loss')
    axs[1].plot(history.history['val_loss'], label='val loss')
    axs[1].set_title(f'{title} - Loss')
    axs[1].legend()
    plt.show()
```

```
# MLP Model
mlp_model = Sequential([
    Flatten(input_shape=(224, 224, 3)),
    Dense(128, activation='relu'),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(3, activation='softmax')  # Ensure this matches the number of classes
])

mlp_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

mlp_history = mlp_model.fit(
    train_generator,
    epochs=25,
    validation_data=val_generator
)

plot_training_history(mlp_history, 'MLP Model')
```

```
Epoch 6/25
131/131 [==============================] - 68s 522ms/step - loss: 0.8693 - accuracy: 0.6293 - v
Epoch 7/25
131/131 [==============================] - 67s 514ms/step - loss: 0.8517 - accuracy: 0.6326 - v
Epoch 8/25
131/131 [==============================] - 68s 518ms/step - loss: 0.7996 - accuracy: 0.6408 - v
Epoch 9/25
131/131 [==============================] - 68s 522ms/step - loss: 0.7527 - accuracy: 0.6686 - v
Epoch 10/25
131/131 [==============================] - 68s 521ms/step - loss: 0.7740 - accuracy: 0.6532 - v
Epoch 11/25
131/131 [==============================] - 67s 510ms/step - loss: 0.7934 - accuracy: 0.6437 - v
Epoch 12/25
131/131 [==============================] - 67s 515ms/step - loss: 0.7673 - accuracy: 0.6659 - v
Epoch 13/25
131/131 [==============================] - 68s 520ms/step - loss: 0.7938 - accuracy: 0.6453 - v
Epoch 14/25
131/131 [==============================] - 68s 523ms/step - loss: 0.8388 - accuracy: 0.6017 - v
Epoch 15/25
131/131 [==============================] - 67s 513ms/step - loss: 0.7790 - accuracy: 0.6552 - v
Epoch 16/25
131/131 [==============================] - 68s 518ms/step - loss: 0.7724 - accuracy: 0.6636 - v
Epoch 17/25
131/131 [==============================] - 68s 517ms/step - loss: 0.9064 - accuracy: 0.6307 - v
Epoch 18/25
131/131 [==============================] - 68s 519ms/step - loss: 0.8974 - accuracy: 0.6104 - v
Epoch 19/25
131/131 [==============================] - 67s 510ms/step - loss: 0.9398 - accuracy: 0.4889 - v
Epoch 20/25
131/131 [==============================] - 68s 518ms/step - loss: 0.8448 - accuracy: 0.6437 - v
Epoch 21/25
131/131 [==============================] - 67s 512ms/step - loss: 0.8603 - accuracy: 0.6180 - v
Epoch 22/25
131/131 [==============================] - 68s 516ms/step - loss: 0.8044 - accuracy: 0.6520 - v
Epoch 23/25
131/131 [==============================] - 67s 512ms/step - loss: 0.7933 - accuracy: 0.6446 - v
Epoch 24/25
131/131 [==============================] - 67s 515ms/step - loss: 0.7968 - accuracy: 0.6444 - v
Epoch 25/25
131/131 [==============================] - 67s 514ms/step - loss: 0.8405 - accuracy: 0.6214 - v
```



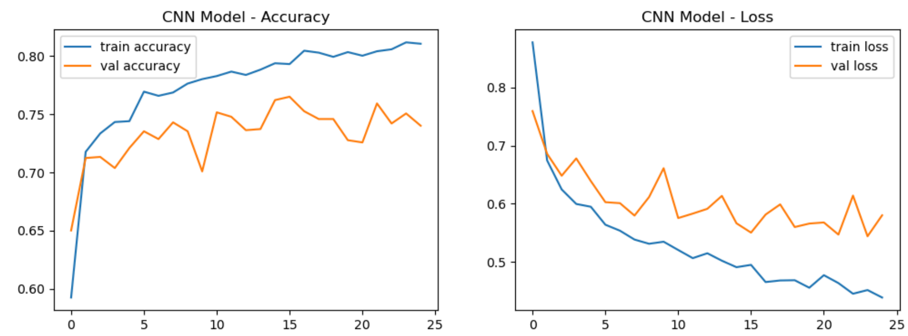
Part (b): CNN model training:

In this part, a CNN model is trained with 4 layers of Convolution and Max pooling layers. At the end, the output of these layers are flattened and using another dense layer with only 3 neurons and a softmax activation function, the labels are defined.

```
# CNN Model
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(3, activation='softmax')  # Ensure this matches the number of classes
])

cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

cnn_history = cnn_model.fit(
    train_generator,
    epochs=25,
    validation_data=val_generator
)

plot_training_history(cnn_history, 'CNN Model')
```

```
Epoch 1/25
131/131 [==============================] - 137s 1s/step - loss: 0.8778 - accuracy: 0.5924 - val
Epoch 2/25
131/131 [==============================] - 100s 762ms/step - loss: 0.6745 - accuracy: 0.7177 -
Epoch 3/25
131/131 [==============================] - 102s 780ms/step - loss: 0.6245 - accuracy: 0.7335 -
Epoch 4/25
131/131 [==============================] - 98s 752ms/step - loss: 0.5995 - accuracy: 0.7434 - v
Epoch 5/25
131/131 [==============================] - 106s 811ms/step - loss: 0.5947 - accuracy: 0.7441 -
Epoch 6/25
131/131 [==============================] - 99s 757ms/step - loss: 0.5637 - accuracy: 0.7695 - v
Epoch 7/25
131/131 [==============================] - 100s 760ms/step - loss: 0.5535 - accuracy: 0.7659 -
Epoch 8/25
131/131 [==============================] - 98s 751ms/step - loss: 0.5382 - accuracy: 0.7688 - v
Epoch 9/25
131/131 [==============================] - 99s 759ms/step - loss: 0.5309 - accuracy: 0.7764 - v
Epoch 10/25
131/131 [==============================] - 99s 755ms/step - loss: 0.5345 - accuracy: 0.7803 - v
Epoch 11/25
131/131 [==============================] - 103s 785ms/step - loss: 0.5202 - accuracy: 0.7829 -
Epoch 12/25
131/131 [==============================] - 101s 770ms/step - loss: 0.5062 - accuracy: 0.7867 -
Epoch 13/25
131/131 [==============================] - 101s 767ms/step - loss: 0.5146 - accuracy: 0.7838 -
Epoch 14/25
131/131 [==============================] - 96s 735ms/step - loss: 0.5018 - accuracy: 0.7884 - v
Epoch 15/25
131/131 [==============================] - 96s 733ms/step - loss: 0.4906 - accuracy: 0.7939 - v
Epoch 16/25
131/131 [==============================] - 96s 735ms/step - loss: 0.4946 - accuracy: 0.7932 - v
Epoch 17/25
131/131 [==============================] - 95s 726ms/step - loss: 0.4650 - accuracy: 0.8047 - v
Epoch 18/25
131/131 [==============================] - 94s 713ms/step - loss: 0.4677 - accuracy: 0.8030 - v
Epoch 19/25
131/131 [==============================] - 94s 721ms/step - loss: 0.4681 - accuracy: 0.7994 - v
Epoch 20/25
131/131 [==============================] - 96s 733ms/step - loss: 0.4552 - accuracy: 0.8035 - v
Epoch 21/25
131/131 [==============================] - 95s 724ms/step - loss: 0.4768 - accuracy: 0.8004 - v
Epoch 22/25
131/131 [==============================] - 93s 712ms/step - loss: 0.4632 - accuracy: 0.8042 - v
Epoch 23/25
131/131 [==============================] - 93s 713ms/step - loss: 0.4449 - accuracy: 0.8059 - v
Epoch 24/25
131/131 [==============================] - 93s 707ms/step - loss: 0.4512 - accuracy: 0.8119 - v
Epoch 25/25
131/131 [==============================] - 96s 729ms/step - loss: 0.4384 - accuracy: 0.8107 - v
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[13], line 24
     16 cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])
     18 cnn_history = cnn_model.fit(
```

```
# Plot training history
def plot_training_history(history, title):
    fig, axs = plt.subplots(1, 2, figsize=(12, 4))
    axs[0].plot(history.history['accuracy'], label='train accuracy')
    axs[0].plot(history.history['val_accuracy'], label='val accuracy')
    axs[0].set_title(f'{title} - Accuracy')
    axs[0].legend()
    axs[1].plot(history.history['loss'], label='train loss')
    axs[1].plot(history.history['val_loss'], label='val loss')
    axs[1].set_title(f'{title} - Loss')
    axs[1].legend()
    plt.show()

plot_training_history(cnn_history, 'CNN Model')
```



Part (c): State-Of-The-Art model training:

For subsequent sections of this project, we need to utilize pretrained models and add the desired classifiers on top of them.

To do so, we first define a function which have base model and number of trainable layers as input. then the desired classifier which consists of a GlobalAveragePooling2D, a dense layer with 128 neurons and an output layer is added to the end of the model.

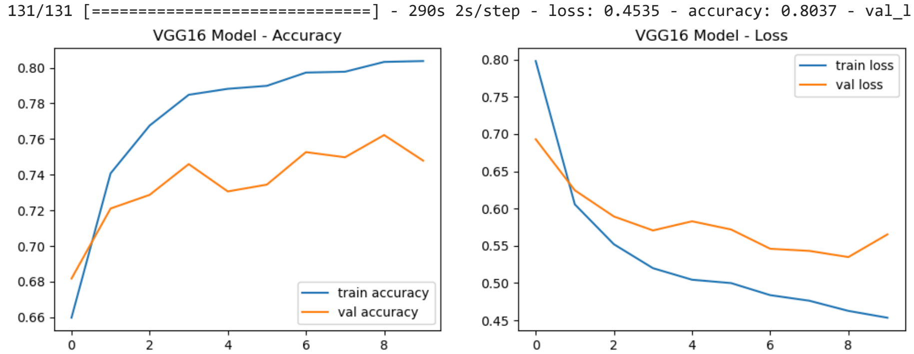
Having this function in hand, we can easily change the SOTA model and number of trainable layers and perform the actions asked in each part of the project.

```
def create_pretrained_model(base_model, trainable_layers=0):
    base_model.trainable = False
    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dense(3, activation='softmax')  # Ensure this matches the number of classes
    ])

    if trainable_layers > 0:
        for layer in base_model.layers[-trainable_layers:]:
            layer.trainable = True

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

VGG16 model (all layers freezed)

```
# VGG16 as feature extractor
vgg16_base = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
vgg16_model = create_pretrained_model(vgg16_base)

vgg16_history = vgg16_model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator
)

plot_training_history(vgg16_history, 'VGG16 Model')
```

```
Epoch 1/10
131/131 [==============================] - 285s 2s/step - loss: 0.7979 - accuracy: 0.6597 - val_l
Epoch 2/10
131/131 [==============================] - 287s 2s/step - loss: 0.6056 - accuracy: 0.7407 - val_l
Epoch 3/10
131/131 [==============================] - 287s 2s/step - loss: 0.5521 - accuracy: 0.7676 - val_l
Epoch 4/10
131/131 [==============================] - 283s 2s/step - loss: 0.5200 - accuracy: 0.7848 - val_l
Epoch 5/10
131/131 [==============================] - 283s 2s/step - loss: 0.5045 - accuracy: 0.7882 - val_l
Epoch 6/10
131/131 [==============================] - 286s 2s/step - loss: 0.5000 - accuracy: 0.7898 - val_l
Epoch 7/10
131/131 [==============================] - 287s 2s/step - loss: 0.4838 - accuracy: 0.7973 - val_l
Epoch 8/10
131/131 [==============================] - 286s 2s/step - loss: 0.4764 - accuracy: 0.7977 - val_l
Epoch 9/10
131/131 [==============================] - 288s 2s/step - loss: 0.4627 - accuracy: 0.8033 - val_l
Epoch 10/10
131/131 [==============================] - 290s 2s/step - loss: 0.4535 - accuracy: 0.8037 - val_l
```



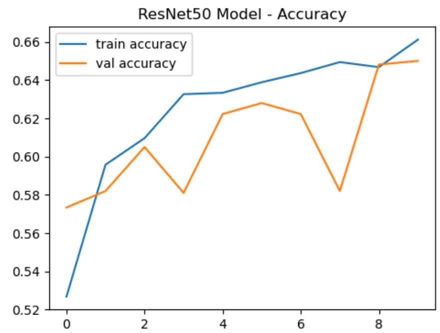
ResNet50 model (all layers freezed)

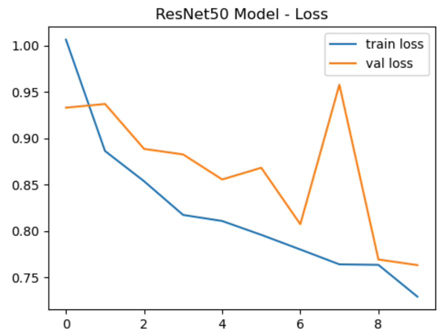
```
from tensorflow.keras.applications import ResNet50
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet5
94765736/94765736 [==============================] - 1010s 11us/step
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 5
      1 from tensorflow.keras.applications import ResNet50
      4 resnet50_base = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224,
3))
----> 5 resnet50_model = create_pretrained_model(resnet50_base)
      7 resnet50_history = resnet50_model.fit(
      8     train_generator,
      9     epochs=25,
     10     validation_data=val_generator
     11 )
     13 plot_training_history(resnet50_history, 'ResNet50 Model')

NameError: name 'create_pretrained_model' is not defined
```

```
resnet50_base = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
resnet50_model = create_pretrained_model(resnet50_base)

resnet50_history = resnet50_model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator
)

plot_training_history(resnet50_history, 'ResNet50 Model')
```

```
Epoch 1/10
131/131 [==============================] - 129s 974ms/step - loss: 1.0062 - accuracy: 0.5267 - va
Epoch 2/10
131/131 [==============================] - 119s 906ms/step - loss: 0.8862 - accuracy: 0.5957 - va
Epoch 3/10
131/131 [==============================] - 116s 885ms/step - loss: 0.8536 - accuracy: 0.6096 - va
Epoch 4/10
131/131 [==============================] - 110s 838ms/step - loss: 0.8172 - accuracy: 0.6326 - va
Epoch 5/10
131/131 [==============================] - 115s 875ms/step - loss: 0.8107 - accuracy: 0.6334 - va
Epoch 6/10
131/131 [==============================] - 115s 874ms/step - loss: 0.7958 - accuracy: 0.6389 - va
Epoch 7/10
131/131 [==============================] - 119s 910ms/step - loss: 0.7800 - accuracy: 0.6437 - va
Epoch 8/10
131/131 [==============================] - 118s 897ms/step - loss: 0.7640 - accuracy: 0.6494 - va
Epoch 9/10
131/131 [==============================] - 120s 916ms/step - loss: 0.7634 - accuracy: 0.6468 - va
Epoch 10/10
131/131 [==============================] - 114s 872ms/step - loss: 0.7291 - accuracy: 0.6612 - va
```



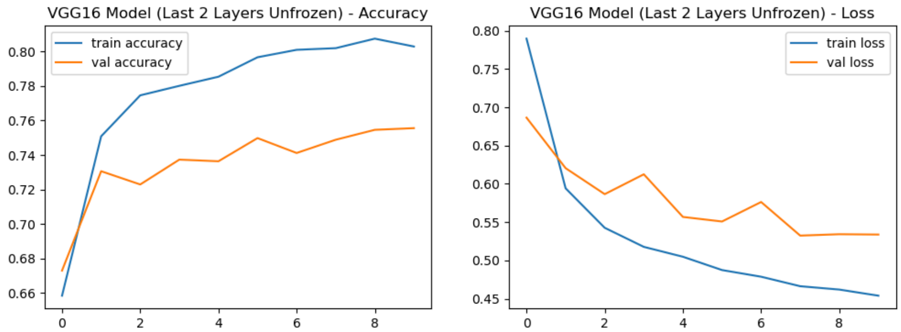


Part (d): freezing last 2 layers of SOTA models:

VGG16 model (last 2 layers unfreezed)

```
# Unfreeze last 2 layers and train
vgg16_model_2 = create_pretrained_model(vgg16_base, trainable_layers=2)

vgg16_history_2 = vgg16_model_2.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator
)

plot_training_history(vgg16_history_2, 'VGG16 Model (Last 2 Layers Unfrozen)')
```

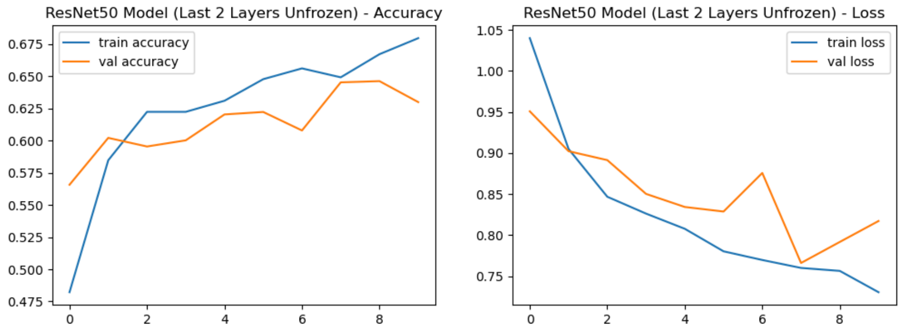
```
Epoch 1/10
131/131 [==============================] - 285s 2s/step - loss: 0.7895 - accuracy: 0.6585 - val_l
Epoch 2/10
131/131 [==============================] - 294s 2s/step - loss: 0.5941 - accuracy: 0.7508 - val_l
Epoch 3/10
131/131 [==============================] - 291s 2s/step - loss: 0.5426 - accuracy: 0.7745 - val_l
Epoch 4/10
131/131 [==============================] - 295s 2s/step - loss: 0.5177 - accuracy: 0.7800 - val_l
Epoch 5/10
131/131 [==============================] - 297s 2s/step - loss: 0.5048 - accuracy: 0.7853 - val_l
Epoch 6/10
131/131 [==============================] - 297s 2s/step - loss: 0.4875 - accuracy: 0.7965 - val_l
Epoch 7/10
131/131 [==============================] - 295s 2s/step - loss: 0.4788 - accuracy: 0.8009 - val_l
Epoch 8/10
131/131 [==============================] - 301s 2s/step - loss: 0.4664 - accuracy: 0.8018 - val_l
Epoch 9/10
131/131 [==============================] - 299s 2s/step - loss: 0.4620 - accuracy: 0.8073 - val_l
Epoch 10/10
131/131 [==============================] - 293s 2s/step - loss: 0.4540 - accuracy: 0.8028 - val_l
```



ResNet50 model (last 2 layers unfreezed)

```
resnet50_base = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
resnet50_model_2 = create_pretrained_model(resnet50_base,trainable_layers=2)

resnet50_history_2 = resnet50_model_2.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator
)

plot_training_history(resnet50_history_2, 'ResNet50 Model (Last 2 Layers Unfrozen)')
```

```
Epoch 1/10
131/131 [==============================] - 113s 849ms/step - loss: 1.0400 - accuracy: 0.4821 - va
Epoch 2/10
131/131 [==============================] - 116s 882ms/step - loss: 0.9050 - accuracy: 0.5847 - va
Epoch 3/10
131/131 [==============================] - 119s 906ms/step - loss: 0.8468 - accuracy: 0.6223 - va
Epoch 4/10
131/131 [==============================] - 117s 890ms/step - loss: 0.8262 - accuracy: 0.6223 - va
Epoch 5/10
131/131 [==============================] - 113s 861ms/step - loss: 0.8077 - accuracy: 0.6310 - va
Epoch 6/10
131/131 [==============================] - 112s 853ms/step - loss: 0.7803 - accuracy: 0.6477 - va
Epoch 7/10
131/131 [==============================] - 115s 875ms/step - loss: 0.7697 - accuracy: 0.6561 - va
Epoch 8/10
131/131 [==============================] - 112s 855ms/step - loss: 0.7600 - accuracy: 0.6492 - va
Epoch 9/10
131/131 [==============================] - 113s 859ms/step - loss: 0.7564 - accuracy: 0.6671 - va
Epoch 10/10
131/131 [==============================] - 115s 875ms/step - loss: 0.7304 - accuracy: 0.6796 - va
```



Part (e): freezing last 6 layers of SOTA models:

VGG16 model (last 6 layers unfreezed)

```
# Unfreeze last 6 layers and train
vgg16_model_6 = create_pretrained_model(vgg16_base, trainable_layers=6)

vgg16_history_6 = vgg16_model_6.fit(
    train_generator,
    epochs=5,
    validation_data=val_generator
)

plot_training_history(vgg16_history_6, 'VGG16 Model (Last 6 Layers Unfrozen)')
```