Image Classification

Yash Gawande

—

Data Science

—

Navodita Infotech

INDEX	
Sr.No.	
1	Introduction
2	Code Explain
3	Conclusion

INTRODUCTION

COVID-19, short for "Coronavirus Disease 2019," is a highly contagious respiratory illness caused by the severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). The disease was first identified in December 2019 in the city of Wuhan, Hubei province, China. It quickly spread globally, leading to a pandemic declared by the World Health Organization (WHO) in March 2020.

We, have dataset of chest X-ray of patients which including the pneumonia, covid-19, Normal. We have challenge to develop a multiclass classification algorithm capable of detecting and classifying the covid-19, Normal (Healthy) and Pneumonia in Chest X-ray images.

Code Explain:

```
# Importing the libraryes
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,
MaxPool2D,Flatten,Dense
from tensorflow.keras.preprocessing.image
import ImageDataGenerator
import warnings
warnings.filterwarnings("ignore")
```

In the above cell we are Importing the sum of important modules for the image classification model such as NumPy, TensorFlow, ImageDataGenerator, etc.

```
# path to the

train_dir = "drive/MyDrive/Covid19-dataset/train"
test_dir = "drive/MyDrive/Covid19-dataset/test"
```

Giving path of the training and testing dataset. (Path to the Dataset is to be changed accordingly)

```
#image argumentation

train_generator = ImageDataGenerator(
    rotation_range=20,
    width_shift_range = 0.2,
    height_shift_range=0.2,
    horizontal_flip = True,
    shear_range=0.2,
    zoom_range=0.2


)
```

1. **train_generator**: This is an object of the **ImageDataGenerator** class, which is part of the Keras library. It is used for data augmentation during the training of neural networks.
2. **rotation_range**: Augments the training data by randomly rotating the images by a specified degree. In this case, the rotation is done within the range of [-20, 20] degrees.

3. **width_shift_range**: Augments the training data by shifting the width of the image by a fraction of its total width. Here, it is set to 0.2, which means a maximum shift of 20% of the image width.
4. **height_shift_range**: Like **width_shift_range**, but it shifts the height of the image. Also set to 0.2, allowing a maximum shift of 20% of the image height.
5. **horizontal_flip**: Augments the training data by flipping images horizontally (left-to-right).
6. **shear_range**: Applies shear transformation to the images. It shears the image by a specified intensity (20% in this case).
7. **zoom_range**: Augments the training data by applying zoom to the images. It randomly zooms in or out by a specified percentage (20% in this case).

These augmentations are applied randomly to the input images during the training process, helping the model generalize better and handle variations in the input data.

```
#Test Generator
test_generator = ImageDataGenerator(rescale=1.0/255)
```

1. **test_generator**: This is an object of the **ImageDataGenerator** class.
2. **rescale=1.0/255**: This parameter is used to rescale the pixel values of the images. Pixel values are typically in the range of [0, 255]. By setting **rescale=1.0/255**, it scales the pixel values down to the range [0, 1]. This normalization is common in deep learning models as it helps in better convergence during training.

```
width, height =512,512

batch_size=32
```

Setting the Width, height and batch_size for the image.

```
x_train = train_generator.flow_from_directory(

    train_dir,
    target_size=(width,height),
    batch_size=batch_size,
    class_mode="categorical"
)
```

1. **train_generator.flow_from_directory**: This method generates batches of augmented data from image files in a directory. It is a convenient way to load large datasets of images and apply real-time data augmentation during training.

2. **train_dir**: The path to the directory containing the training images. The method expects subdirectories within **train_dir**, each containing images for a specific class.
3. **target_size=(width, height)**: Resizes all images to the specified width and height. This ensures that all input images have the same dimensions, which is necessary for feeding them into a neural network.
4. **batch_size**: The number of samples in each batch. During training, the model updates its weights based on the error calculated for each batch.
5. **class_mode="categorical"**: Specifies the type of label assignment. In this case, it indicates that the labels are in categorical format, meaning that the data consists of categories (classes) and each sample can belong to one or more classes.

```
x_test = train_generator.flow_from_directory(

    test_dir,
    target_size=(width,height),
    batch_size=batch_size,
    class_mode="categorical"
)
```

Same as X_train, but this cell is for X_test

```
model = Sequential([

    Conv2D(32,(3,3),activation="relu",input_shape=(width,height,3)),
    MaxPool2D((2,2)),
    Conv2D(64,(3,3),activation="relu"),
    MaxPool2D((2,2)),
    Conv2D(128,(3,3),activation="relu"),
    MaxPool2D((2,2)),
    Conv2D(128,(3,3),activation="relu"),
    MaxPool2D((2,2)),
    Flatten(),
    Dense(256,activation="relu"),
    Dense(128,activation="relu"),
    Dense(3,activation="softmax")


])
```

Here, in this cell we are developing the Sequential model for Convolutional Neural Network, we created **model** the instance of the **Sequential Class.**

1. **Sequential**: This initializes a Sequential model, which is a linear stack of layers.
2. **Conv2D**: Convolutional layer with specified parameters. It performs convolutional operations on the input data, extracting features using filters/kernels.

3. **MaxPool2D**: MaxPooling layer reduces the spatial dimensions of the input volume. It helps in reducing computational complexity and controlling overfitting.
4. **Flatten**: This layer flattens the input, converting 3D feature maps to a 1D feature vector.
5. **Dense**: Dense (fully connected) layers. The first two dense layers have ReLU activation functions, which introduce non-linearity. The last dense layer has softmax activation, suitable for multi-class classification tasks.

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

This line sets up the model for training using the **Adam optimizer**, **categorical Crossentropy** as the **loss function**, and monitoring the **accuracy metric**. After this line, the model is ready to be trained using the **fit** method.

```
history =
model.fit(x_train,epochs=30,batch_size=batch_size,
validation_data=x_test)
```

This line initiates the training process for the specified number of epochs, using the training data (**x_train**). The model's performance on the validation data (**x_test**) is also evaluated during training, and the training history is stored in the **history** variable. The training history includes information about the **training loss, training accuracy, validation loss, and validation accuracy** for each epoch. This information can be useful for analyzing the model's performance and identifying potential issues like overfitting.

```
plt.plot(history.history["accuracy"],label="Trainning loss")

plt.plot(history.history["val_accuracy"],label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy")
plt.legend()
```

1. **plt.plot(history.history["accuracy"], label="Training Accuracy")**: This line plots the training accuracy for each epoch. **history.history["accuracy"]** retrieves the training accuracy values recorded during training, and the **label** parameter is set to "Training Accuracy" for legend identification.
2. **plt.plot(history.history["val_accuracy"], label="Validation Accuracy")**: This line plots the validation accuracy for each epoch. **history.history["val_accuracy"]** retrieves the validation accuracy values recorded during training, and the **label** parameter is set to "Validation Accuracy" for legend identification.
3. **plt.xlabel("Epochs")**: Sets the label for the x-axis to "Epochs".

4. **plt.ylabel("Accuracy")**: Sets the label for the y-axis to "Accuracy".
5. **plt.title("Accuracy")**: Sets the title of the plot to "Accuracy".
6. **plt.legend()**: Displays a legend on the plot, which identifies the lines corresponding to training and validation accuracy.

The resulting plot shows how the training accuracy and validation accuracy change over the epochs. It helps in assessing the model's learning progress and potential overfitting or underfitting. If the training accuracy increases while the validation accuracy plateaus or decreases, it might indicate overfitting. Conversely, if both training and validation accuracy are low, it might suggest underfitting.

```
plt.plot(history.history["loss"])

plt.plot(history.history["val_loss"])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Model Loss")
plt.legend()
```

Same code as above cell. The resulting plot visually represents how the model's training loss and validation loss change over the course of training. It helps in assessing the convergence of the model and identifying potential issues such as overfitting or underfitting.

```
class_name = x_test.class_indices

classes = {value:key for key, value in
class_name.items()}
print("Class Names",classes)
```

1. **class_name = x_test.class_indices**: **class_indices** is an attribute of the **x_test** generator that provides a dictionary mapping class names to their corresponding indices. This line extracts this dictionary and assigns it to the variable **class_name**.
2. **classes = {value: key for key, value in class_name.items()}**: This line creates a new dictionary, **classes**, by reversing the key-value pairs of the **class_name** dictionary. Now, the indices become keys, and the corresponding class names become values.
3. **print("Class Names", classes)**: This line prints the resulting dictionary, displaying the mapping between indices and class names.

```
img =
image.load_img("0111.jpg",target_size=(width,height))
```

1. **image**: This refers to the **Image** module from the Keras preprocessing library. It's used for image processing tasks.
2. **load_img("0111.jpg", target_size=(width, height))**: This function is used to load an image file. The first argument is the file path ("0111.jpg" in this case), and the **target_size** parameter is used to resize the image to the specified width and height.
3. **"0111.jpg"**: This is the file path of the image you want to load.
4. **target_size=(width, height)**: This parameter specifies the dimensions to which the loaded image should be resized. It's a tuple containing the width and height. The variables **width** and **height** should be defined earlier in your code.

After executing this line, the variable **img** will hold the loaded image, and it can be further processed or used as input to a neural network model.

```
img_arr =
image.img_to_array(img)
```

- **image**: This refers to the **Image** module from the Keras preprocessing library.
- **img_to_array(img)**: This function is applied to the loaded image (**img**). It converts the image data into a NumPy array.

After executing this line, the variable **img_arr** will contain the NumPy array representation of the image. Each element of the array represents the intensity of a pixel in the image. The shape of the array will depend on the dimensions of the image and the number of color channels.

This NumPy array (**img_arr**) is a common format for feeding images into neural network models, as it can be easily processed and normalized as needed for the model input.

```
img_arr =
np.expand_dims(img_arr,axis=0)
```

- **np**: This is the commonly used abbreviation for the NumPy library.
- **expand_dims(img_arr, axis=0)**: The **expand_dims** function is used to add a new axis at the specified position (**axis**). In this case, **axis=0** indicates that the new axis is added at the first position (index 0).

After executing this line, the variable **img_arr** will have one additional dimension compared to its previous shape. This is often done to match the expected input shape of a neural network model. Many deep learning models expect the input to have a batch dimension, even if you're

processing a single image. The resulting shape might look something like **(1, height, width, channels)**.

```
img_arr = img_arr/255.

pred = model.predict(img_arr)
pred
```

- **img_arr**: This is the NumPy array representing the image.
- **/255.**: This operation divides each element of the array by 255.0.

Normalization is a common preprocessing step in machine learning, particularly for image data. By dividing the pixel values by 255, you're scaling them to a range between 0 and 1. This is beneficial for training neural networks, as it helps stabilize and speed up the training process.

After normalization, the variable **img_arr** contains the image data with pixel values scaled to the range [0, 1].

The next line, **pred = model.predict(img_arr)**, is used to make predictions on the normalized image using the trained neural network model (**model**). The variable **pred** will contain the model's predicted probabilities for each class.

Last line **pred** print the prediction of the image such as 0: for covid, 1: for normal, 2: for Pneumonia

From the above all cell, I have explained the entire **Convolutional Neural Network** code for the **Image Classification**.

Conclusion:

We have successfully developed the Convolutional Neural Network Model for the image classification on the Covid-19 X-ray Dataset which is classified into three categories such as X-Ray images of Covid infected people, Normal (Healthy) people and people having the Pneumonia