Deep Learning Practical Assignment 3A

May 3, 2023

Name – Chikane Aniket Ballu / Roll No. – 4123/ Importing Images &

Libraries

[5]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
 load_img, ␣ ↪img_to_array
```

[6]:
```
train_dir = r'D:\DL Practical\New Plant Diseases
Dataset(Augmented)\train' val_dir = r'D:\DL Practical\New Plant
Diseases Dataset(Augmented)\valid'
```

[7]:
```
img_size = 224
batch_size = 32
```

Preprocessing

[8]:
```
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(img_size,␣
 ↪img_size),
                                                    batch_size=batch_size,
                                                    class_mode='categorical')
```

```
Found 600 images belonging to 3 classes.
```

[9]:
```
val_datagen = ImageDataGenerator(rescale=1./255)
val_generator = val_datagen.flow_from_directory(val_dir,
                                                target_size=(img_size,␣
 ↪img_size),
                                                batch_size=batch_size,
                                                class_mode='categorical')
```

```
Found 600 images belonging to 3 classes.
```

[10]:
```
list(train_generator.class_indices)
```

[10]:
```
['Tomato___Bacterial_spot', 'Tomato___Early_blight',
'Tomato___healthy']
```

Building our Model

[11]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, BatchNormalization
```

[12]:
```python
model = Sequential()

model.add((Conv2D(32,  3,3), activation='relu', input_shape=(img_size,␣
↪img_size, 3))))
model.add(BatchNormalization())
model.add((MaxPooling2D(2,2)))
model.add((Conv2D(64,  3,3), activation='relu')))
model.add(BatchNormalization())
model.add((MaxPooling2D(2,2)))
model.add((Conv2D(64,  3,3), activation='relu')))
model.add(BatchNormalization())
model.add((MaxPooling2D(2,2)))
model.add((Conv2D(128,  3,3), activation='relu')))
model.add(BatchNormalization())
model.add((MaxPooling2D(2,2)))

model.add((Flatten()))

model.add((Dense(128, activation='relu')))
model.add((Dropout(0.2)))
model.add((Dense(64, activation='relu')))
model.add((Dense(train_generator.num_classes, activation='softmax')))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)            (None, 222, 222, 32) 896

                                                      128
 batch_normalization (BatchN (None, 222,
 222, 32) ormalization)

                                                      0
 max_pooling2d (MaxPooling2D (None, 111,
 111, 32) )

 conv2d_1 (Conv2D)          (None, 109, 109, 64) 18496

                                                      256
 batch_normalization_1 (Batc (None, 109,
 109, 64) hNormalization)
```

```
 max_pooling2d_1 (MaxPooling (None, 54,        0
 54, 64) 2D)
 conv2d_2 (Conv2D)       (None, 52, 52, 64)   36928

 batch_normalization_2 (Batc (None, 52, 52,    256
 64) hNormalization)

 max_pooling2d_2 (MaxPooling (None, 26,        0
 26, 64) 2D)

 conv2d_3 (Conv2D)       (None, 24, 24, 128)  73856

 batch_normalization_3 (Batc (None, 24, 24,    512
 128) hNormalization)

 max_pooling2d_3 (MaxPooling (None, 12,        0
 12, 128) 2D)

 flatten (Flatten)       (None, 18432)        0

 dense (Dense)           (None, 128)          2359424

 dropout (Dropout)       (None, 128)          0

 dense_1 (Dense)         (None, 64)           8256

 dense_2 (Dense)         (None, 3)            195
=================================================================
Total params: 2,499,203
Trainable params: 2,498,627
Non-trainable params: 576

_________________________________________________________________
```

[13]:
```
model.compile(optimizer='adam',
 loss='categorical_crossentropy',␣ ↪metrics=['accuracy'])
```

Training our Model

[14]:
```
model.fit(train_generator, epochs=50, validation_data=val_generator)
```

```
Epoch 1/50
19/19 [==============================] - 75s 4s/step - loss: 1.8025 -
accuracy:
0.6917 - val_loss: 1.2368 - val_accuracy: 0.3850
Epoch 2/50
19/19 [==============================] - 75s 4s/step - loss: 0.3386 -
accuracy:
0.9083 - val_loss: 1.8735 - val_accuracy: 0.5233
```

```
Epoch 3/50
19/19 [==============================] - 75s 4s/step - loss: 0.3262 -
accuracy:
0.9333 - val_loss: 3.6816 - val_accuracy: 0.3317
Epoch 4/50
19/19 [==============================] - 75s 4s/step - loss: 0.2124 -
accuracy:
0.9383 - val_loss: 4.7265 - val_accuracy: 0.3333
Epoch 5/50
19/19 [==============================] - 75s 4s/step - loss: 0.2041 -
accuracy:
0.9267 - val_loss: 5.9973 - val_accuracy: 0.3567
Epoch 6/50
19/19 [==============================] - 75s 4s/step - loss: 0.2451 -
accuracy:
0.9267 - val_loss: 7.1449 - val_accuracy: 0.3367
Epoch 7/50
19/19 [==============================] - 75s 4s/step - loss: 0.1496 -
accuracy:
0.9583 - val_loss: 5.6165 - val_accuracy: 0.4767
Epoch 8/50
19/19 [==============================] - 75s 4s/step - loss: 0.1024 -
accuracy:
0.9667 - val_loss: 6.4472 - val_accuracy: 0.3333
Epoch 9/50
19/19 [==============================] - 75s 4s/step - loss: 0.2211 -
accuracy:
0.9500 - val_loss: 14.6802 - val_accuracy: 0.3333
Epoch 10/50
19/19 [==============================] - 74s 4s/step - loss: 0.1686 -
accuracy:
0.9617 - val_loss: 6.3950 - val_accuracy: 0.3833
Epoch 11/50
19/19 [==============================] - 75s 4s/step - loss: 0.1618 -
accuracy:
0.9533 - val_loss: 9.0740 - val_accuracy: 0.3333
Epoch 12/50
19/19 [==============================] - 77s 4s/step - loss: 0.0924 -
accuracy:
0.9683 - val_loss: 10.3126 - val_accuracy: 0.4150
Epoch 13/50
19/19 [==============================] - 75s 4s/step - loss: 0.0735 -
accuracy:
0.9683 - val_loss: 13.9305 - val_accuracy: 0.3800
Epoch 14/50
```

```
19/19 [==============================] - 75s 4s/step - loss: 0.0385 -
accuracy:
0.9867 - val_loss: 24.5295 - val_accuracy: 0.3333
Epoch 15/50
19/19 [==============================] - 75s 4s/step - loss: 0.0462 -
accuracy:
0.9850 - val_loss: 16.4559 - val_accuracy: 0.3367
Epoch 16/50
19/19 [==============================] - 75s 4s/step - loss: 0.0802 -
accuracy:
0.9850 - val_loss: 11.3096 - val_accuracy: 0.5317
Epoch 17/50
19/19 [==============================] - 76s 4s/step - loss: 0.0701 -
accuracy:
0.9833 - val_loss: 22.0208 - val_accuracy: 0.3400
Epoch 18/50
19/19 [==============================] - 75s 4s/step - loss: 0.1175 -
accuracy:
0.9800 - val_loss: 11.4855 - val_accuracy: 0.4783
Epoch 19/50
19/19 [==============================] - 79s 4s/step - loss: 0.0455 -
accuracy:
0.9900 - val_loss: 15.0479 - val_accuracy: 0.3750
Epoch 20/50
19/19 [==============================] - 79s 4s/step - loss: 0.0583 -
accuracy:
0.9817 - val_loss: 5.7751 - val_accuracy: 0.6450
Epoch 21/50
19/19 [==============================] - 75s 4s/step - loss: 0.2200 -
accuracy:
0.9733 - val_loss: 5.5756 - val_accuracy: 0.6417
Epoch 22/50
19/19 [==============================] - 75s 4s/step - loss: 0.0589 -
accuracy:
0.9867 - val_loss: 3.3282 - val_accuracy: 0.6817
Epoch 23/50
19/19 [==============================] - 76s 4s/step - loss: 0.1534 -
accuracy:
0.9683 - val_loss: 6.6120 - val_accuracy: 0.5117
Epoch 24/50
19/19 [==============================] - 75s 4s/step - loss: 0.1722 -
accuracy:
0.9633 - val_loss: 3.1207 - val_accuracy: 0.7050
Epoch 25/50
19/19 [==============================] - 74s 4s/step - loss: 0.0495 -
accuracy:
```

```
0.9817 - val_loss: 4.8652 - val_accuracy: 0.6983
Epoch 26/50
19/19 [==============================] - 74s 4s/step - loss: 0.1204 -
accuracy:
0.9867 - val_loss: 7.3578 - val_accuracy: 0.5617
Epoch 27/50
19/19 [==============================] - 75s 4s/step - loss: 0.3199 -
accuracy:
0.9650 - val_loss: 2.3957 - val_accuracy: 0.7933
Epoch 28/50
19/19 [==============================] - 74s 4s/step - loss: 0.1606 -
accuracy:
0.9717 - val_loss: 11.0629 - val_accuracy: 0.4150
Epoch 29/50
19/19 [==============================] - 80s 4s/step - loss: 0.1567 -
accuracy:
0.9717 - val_loss: 10.4260 - val_accuracy: 0.5350
Epoch 30/50
19/19 [==============================] - 76s 4s/step - loss: 0.2967 -
accuracy:
0.9633 - val_loss: 9.0304 - val_accuracy: 0.5933
Epoch 31/50
19/19 [==============================] - 82s 4s/step - loss: 0.1021 -
accuracy:
0.9800 - val_loss: 1.3196 - val_accuracy: 0.7900
Epoch 32/50
19/19 [==============================] - 83s 4s/step - loss: 0.0476 -
accuracy:
0.9900 - val_loss: 1.7618 - val_accuracy: 0.7567
Epoch 33/50
19/19 [==============================] - 84s 4s/step - loss: 0.0965 -
accuracy:
0.9950 - val_loss: 0.3019 - val_accuracy: 0.9383
Epoch 34/50
19/19 [==============================] - 85s 5s/step - loss: 0.0176 -
accuracy:
0.9933 - val_loss: 0.6969 - val_accuracy: 0.9033
Epoch 35/50
19/19 [==============================] - 85s 5s/step - loss: 0.0419 -
accuracy:
0.9917 - val_loss: 2.9759 - val_accuracy: 0.7417
Epoch 36/50
19/19 [==============================] - 84s 4s/step - loss: 0.0315 -
accuracy:
0.9933 - val_loss: 1.1280 - val_accuracy: 0.8667
Epoch 37/50
```

```
19/19 [==============================] - 84s 4s/step - loss: 0.0178 -
accuracy:
0.9967 - val_loss: 1.0208 - val_accuracy: 0.8733
Epoch 38/50
19/19 [==============================] - 84s 4s/step - loss: 0.0983 -
accuracy:
0.9883 - val_loss: 0.3770 - val_accuracy: 0.9317
Epoch 39/50
19/19 [==============================] - 85s 5s/step - loss: 0.0321 -
accuracy:
0.9867 - val_loss: 0.3187 - val_accuracy: 0.9400
Epoch 40/50
19/19 [==============================] - 84s 4s/step - loss: 0.0758 -
accuracy:
0.9933 - val_loss: 0.6674 - val_accuracy: 0.9000
Epoch 41/50
19/19 [==============================] - 85s 5s/step - loss: 0.0510 -
accuracy:
0.9833 - val_loss: 0.7602 - val_accuracy: 0.8867
Epoch 42/50
19/19 [==============================] - 90s 5s/step - loss: 0.0989 -
accuracy:
0.9717 - val_loss: 2.0547 - val_accuracy: 0.7933
Epoch 43/50
19/19 [==============================] - 86s 5s/step - loss: 0.0494 -
accuracy:
0.9917 - val_loss: 1.0881 - val_accuracy: 0.8817
Epoch 44/50
19/19 [==============================] - 85s 5s/step - loss: 0.0121 -
accuracy:
0.9967 - val_loss: 5.7480 - val_accuracy: 0.6733
Epoch 45/50
19/19 [==============================] - 96s 5s/step - loss: 0.0136 -
accuracy:
0.9950 - val_loss: 1.2025 - val_accuracy: 0.8667
Epoch 46/50
19/19 [==============================] - 91s 5s/step - loss: 0.0122 -
accuracy:
0.9950 - val_loss: 0.4112 - val_accuracy: 0.9333
Epoch 47/50
19/19 [==============================] - 89s 5s/step - loss: 0.0039 -
accuracy:
1.0000 - val_loss: 0.4226 - val_accuracy: 0.9333
Epoch 48/50
19/19 [==============================] - 86s 5s/step - loss: 0.0011 -
accuracy:
```

```
1.0000 - val_loss: 0.5084 - val_accuracy: 0.9317
Epoch 49/50
19/19 [==============================] - 94s 5s/step - loss: 0.0331 -
accuracy:
0.9933 - val_loss: 0.4111 - val_accuracy: 0.9367
Epoch 50/50
19/19 [==============================] - 88s 5s/step - loss: 0.0301 -
accuracy:
0.9900 - val_loss: 1.4860 - val_accuracy: 0.8583
```

```
[14]: <keras.callbacks.History at 0x22526437af0>
```

Evaluating our Model

```
[15]: loss, accuracy = model.evaluate(val_generator)
print("Loss :",loss)
print("Accuracy (Test Data) :",accuracy*100)
```

```
19/19 [==============================] - 19s 969ms/step - loss: 1.4860
accuracy: 0.8583
Loss : 1.4859689474105835
Accuracy (Test Data) : 85.83333492279053
```

Testing our Model

```
[19]: import numpy as np img_path =r'D:\DL
Practical\New Plant Diseases␣
         ↪Dataset(Augmented)\valid\Tomato___Early_blight\5b86ab6a-3823-4886-85fd
                                                  02190898563c___RS_Er
 ↪B 8452.JPG' img = load_img(img_path,
target_size=(224, 224)) img_array =
img_to_array(img) img_array =
np.expand_dims(img_array, axis=0)
img_array /= 255.
```

```
[20]: prediction = model.predict(img_array)
class_names=['Tomato___Bacterial_spot', 'Tomato___Early_blight',␣
↪'Tomato___healthy']
```

```
1/1 [==============================] - 0s 38ms/step
```

```
[21]: predicted_class = np.argmax(prediction)
print(prediction)
print(predicted_class)
print('Predicted class:', class_names[predicted_class])
```

```
[[3.7160314e-07 9.9999964e-01 1.8681075e-10]]
1
Predicted class: Tomato___Early_blight
```