efficientnet-v2

March 26, 2024

1 Import Necesssary Libraries

[ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import os
import time
import shutil
import pathlib
import itertools
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
```

2 Preparing Image's for Training from Directory

[ ]:
```
def define_paths(data_dir):
    filepaths = []
    labels = []

    folds = os.listdir(data_dir)
    for fold in folds:
        foldpath = os.path.join(data_dir, fold)
        filelist = os.listdir(foldpath)
```

```
        for file in filelist:
            fpath = os.path.join(foldpath, file)
            filepaths.append(fpath)
            labels.append(fold)

    return filepaths, labels


# Concatenate data paths with labels into one dataframe ( to later be fitted into the model )
def define_df(files, classes):
    Fseries = pd.Series(files, name= 'filepaths')
    Lseries = pd.Series(classes, name='labels')
    return pd.concat([Fseries, Lseries], axis= 1)

# Split dataframe to train, valid, and test
def split_data(data_dir):
    # train dataframe
    files, classes = define_paths(data_dir)
    df = define_df(files, classes)
    strat = df['labels']
    train_df, dummy_df = train_test_split(df,  train_size= 0.8, shuffle= True, random_state= 123, stratify= strat)

    # valid and test dataframe
    strat = dummy_df['labels']
    valid_df, test_df = train_test_split(dummy_df,  train_size= 0.5, shuffle= True, random_state= 123, stratify= strat)

    return train_df, valid_df, test_df
```

[ ]:
```
def create_gens (train_df, valid_df, test_df, batch_size):
    '''
    This function takes train, validation, and test dataframe and fit them into image data generator, because model takes data from image data generator.
    Image data generator converts images into tensors. '''


    # define model parameters
    img_size = (224, 224)
    channels = 3 # either BGR or Grayscale
    color = 'rgb'
    img_shape = (img_size[0], img_size[1], channels)

    # Recommended : use custom function for test data batch size, else we can use normal batch size.
    ts_length = len(test_df)
```

```
    test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length +␣
↪1) if ts_length%n == 0 and ts_length/n <= 80]))
    test_steps = ts_length // test_batch_size

    # This function which will be used in image data generator for data␣
↪augmentation, it just take the image and return it again.
    def scalar(img):
        return img

    tr_gen = ImageDataGenerator(preprocessing_function= scalar,␣
↪horizontal_flip= True)
    ts_gen = ImageDataGenerator(preprocessing_function= scalar)

    train_gen = tr_gen.flow_from_dataframe( train_df, x_col= 'filepaths',␣
↪y_col= 'labels', target_size= img_size, class_mode= 'categorical',
                                        color_mode= color, shuffle= True,␣
↪batch_size= batch_size)

    valid_gen = ts_gen.flow_from_dataframe( valid_df, x_col= 'filepaths',␣
↪y_col= 'labels', target_size= img_size, class_mode= 'categorical',
                                        color_mode= color, shuffle= True,␣
↪batch_size= batch_size)

    # Note: we will use custom test_batch_size, and make shuffle= false
    test_gen = ts_gen.flow_from_dataframe( test_df, x_col= 'filepaths', y_col=␣
↪'labels', target_size= img_size, class_mode= 'categorical',
                                        color_mode= color, shuffle= False,␣
↪batch_size= test_batch_size)

    return train_gen, valid_gen, test_gen
```

[ ]:
```
# data_dir = '/kaggle/input/v1-dataset-plant-village/
↪Plant_leave_diseases_dataset_with_augmentation'

data_dir ='/kaggle/input/plant-disease-classification-merged-dataset'

    # Get splitted data
train_df, valid_df, test_df = split_data(data_dir)

    # Get Generators
batch_size = 40
train_gen, valid_gen, test_gen = create_gens(train_df, valid_df, test_df,␣
↪batch_size)
```

```
Found 63269 validated image filenames belonging to 88 classes.
Found 7908 validated image filenames belonging to 88 classes.
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/preprocessing/image.py:1137:
UserWarning: Found 1 invalid image filename(s) in x_col="filepaths". These
filename(s) will be ignored.
  warnings.warn(
```

```
Found 7909 validated image filenames belonging to 88 classes.
```

3 Define the Model

[ ]:
```
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)
class_count = len(list(train_gen.class_indices.keys())) # to define number of␣
 ↪classes in dense layer

# create pre-trained model (you can built on pretrained model such as : ␣
 ↪efficientnet, VGG , Resnet )
# we will use efficientnetb3 from EfficientNet family.
strategy1 = tf.distribute.MirroredStrategy()
with strategy1.scope():
    base_model = tf.keras.applications.efficientnet.EfficientNetB3(include_top=␣
 ↪False, weights= "imagenet", input_shape= img_shape, pooling= 'max')
    model1 = Sequential([
            base_model,
            BatchNormalization(axis= -1, momentum= 0.99, epsilon= 0.001),
            Dense(256, kernel_regularizer= regularizers.l2(l= 0.016),␣
 ↪activity_regularizer= regularizers.l1(0.006),
                        bias_regularizer= regularizers.l1(0.006), activation=␣
 ↪'relu'),
            Dropout(rate= 0.45, seed= 123),
            Dense(class_count, activation= 'softmax')])

    model1.compile(Adamax(learning_rate= 0.001), loss=␣
 ↪'categorical_crossentropy', metrics= ['accuracy'])

    model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb3 (Functional  (None, 1536)              10783535
 )

 batch_normalization_1 (Bat  (None, 1536)              6144
 chNormalization)

 dense_2 (Dense)             (None, 256)               393472
```

```
 dropout_1 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 88)                22616

=================================================================
Total params: 11205767 (42.75 MB)
Trainable params: 11115392 (42.40 MB)
Non-trainable params: 90375 (353.03 KB)
_________________________________________________________________
```

4 Callbacks

[ ]:
```
checkpoint_callback = ModelCheckpoint(filepath='/kaggle/working/best_modelv2',
                                      monitor='val_accuracy',
                                      save_best_only=True,
                                      save_weights_only=False,
                                      mode='max',
                                      verbose=1)
```

[ ]:
```
early_stopping_callback = EarlyStopping(monitor='val_loss',
                                        patience=1,
                                        restore_best_weights=True,
                                        verbose=1)
```

5 Train the Model

[ ]:
```
history = model.fit(x= train_gen,
                    epochs= 40,
                    verbose= 1,
                    callbacks=[checkpoint_callback, early_stopping_callback],
                    validation_data= valid_gen,
                    validation_steps= None,
                    use_multiprocessing=True,  # Use dual CPUs
                    workers=3 , # Number of worker processes
                    max_queue_size=10,
                    shuffle= False)
```

```
Epoch 1/40

2024-03-18 10:35:22.228137: E
tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout failed:
INVALID_ARGUMENT: Size of values 0 does not match size of permutation 4 @ fanin
shape
insequential/efficientnetb3/block1b_drop/dropout/SelectV2-2-TransposeNHWCToNCHW-
LayoutOptimizer
```

```
1582/1582 [==============================] - ETA: 0s - loss: 2.8743 - accuracy:
0.8315
Epoch 1: val_accuracy improved from -inf to 0.93146, saving model to
/kaggle/working/best_modelv2
1582/1582 [==============================] - 903s 501ms/step - loss: 2.8743 -
accuracy: 0.8315 - val_loss: 0.8218 - val_accuracy: 0.9315
Epoch 2/40
1582/1582 [==============================] - ETA: 0s - loss: 0.7772 - accuracy:
0.9272
Epoch 2: val_accuracy improved from 0.93146 to 0.94600, saving model to
/kaggle/working/best_modelv2
1582/1582 [==============================] - 752s 474ms/step - loss: 0.7772 -
accuracy: 0.9272 - val_loss: 0.5974 - val_accuracy: 0.9460
Epoch 3/40
1582/1582 [==============================] - ETA: 0s - loss: 0.5887 - accuracy:
0.9494
Epoch 3: val_accuracy improved from 0.94600 to 0.95435, saving model to
/kaggle/working/best_modelv2
1582/1582 [==============================] - 764s 482ms/step - loss: 0.5887 -
accuracy: 0.9494 - val_loss: 0.5218 - val_accuracy: 0.9544
Epoch 4/40
1582/1582 [==============================] - ETA: 0s - loss: 0.4876 - accuracy:
0.9606
Epoch 4: val_accuracy improved from 0.95435 to 0.96232, saving model to
/kaggle/working/best_modelv2
1582/1582 [==============================] - 754s 475ms/step - loss: 0.4876 -
accuracy: 0.9606 - val_loss: 0.4386 - val_accuracy: 0.9623
Epoch 5/40
1582/1582 [==============================] - ETA: 0s - loss: 0.4262 - accuracy:
0.9675
Epoch 5: val_accuracy did not improve from 0.96232
1582/1582 [==============================] - 689s 435ms/step - loss: 0.4262 -
accuracy: 0.9675 - val_loss: 0.4325 - val_accuracy: 0.9616
Epoch 6/40
1582/1582 [==============================] - ETA: 0s - loss: 0.3745 - accuracy:
0.9740
Epoch 6: val_accuracy improved from 0.96232 to 0.96244, saving model to
/kaggle/working/best_modelv2
1582/1582 [==============================] - 750s 473ms/step - loss: 0.3745 -
accuracy: 0.9740 - val_loss: 0.3904 - val_accuracy: 0.9624
Epoch 7/40
1582/1582 [==============================] - ETA: 0s - loss: 0.3411 - accuracy:
0.9773
Epoch 7: val_accuracy improved from 0.96244 to 0.96510, saving model to
/kaggle/working/best_modelv2
1582/1582 [==============================] - 758s 478ms/step - loss: 0.3411 -
accuracy: 0.9773 - val_loss: 0.3725 - val_accuracy: 0.9651
Epoch 8/40
```

```
1582/1582 [==============================] - ETA: 0s - loss: 0.3117 - accuracy:
0.9807
Epoch 8: val_accuracy improved from 0.96510 to 0.96573, saving model to
/kaggle/working/best_modelv2
1582/1582 [==============================] - 758s 478ms/step - loss: 0.3117 -
accuracy: 0.9807 - val_loss: 0.3636 - val_accuracy: 0.9657
Epoch 9/40
1582/1582 [==============================] - ETA: 0s - loss: 0.2856 - accuracy:
0.9841
Epoch 9: val_accuracy improved from 0.96573 to 0.96737, saving model to
/kaggle/working/best_modelv2
1582/1582 [==============================] - 752s 474ms/step - loss: 0.2856 -
accuracy: 0.9841 - val_loss: 0.3355 - val_accuracy: 0.9674
Epoch 10/40
1582/1582 [==============================] - ETA: 0s - loss: 0.2657 - accuracy:
0.9861
Epoch 10: val_accuracy did not improve from 0.96737
Restoring model weights from the end of the best epoch: 9.
1582/1582 [==============================] - 685s 432ms/step - loss: 0.2657 -
accuracy: 0.9861 - val_loss: 0.3490 - val_accuracy: 0.9674
Epoch 10: early stopping
```

[ ]:
```
model1.save('/kaggle/working/efficient-v2.keras')
```

6 Do a sample prediction

[ ]:
```
model = load_model('/kaggle/working/efficient-v2.keras')
```

[ ]:
```
def preprocess_image(image_path, img_size=(224, 224)):
    img = image.load_img(image_path, target_size=img_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array
```

[ ]:
```
def classify_images(model, image_paths):
    classes = []
    for path in image_paths:
        preprocessed_img = preprocess_image(path)
        prediction = model.predict(preprocessed_img)
        predicted_class = np.argmax(prediction)
        classes.append(predicted_class)
    return classes
```

[ ]:
```
folder_names = [
    'Apple__black_rot',
    'Apple__healthy',
    'Apple__rust',
```

```
    'Apple__scab',
    'Cassava__bacterial_blight',
    'Cassava__brown_streak_disease',
    'Cassava__green_mottle',
    'Cassava__healthy',
    'Cassava__mosaic_disease',
    'Cherry__healthy',
    'Cherry__powdery_mildew',
    'Chili__healthy',
    'Chili__leaf curl',
    'Chili__leaf spot',
    'Chili__whitefly',
    'Chili__yellowish',
    'Coffee__cercospora_leaf_spot',
    'Coffee__healthy',
    'Coffee__red_spider_mite',
    'Coffee__rust',
    'Corn__common_rust',
    'Corn__gray_leaf_spot',
    'Corn__healthy',
    'Corn__northern_leaf_blight',
    'Cucumber__diseased',
    'Cucumber__healthy',
    'Gauva__diseased',
    'Gauva__healthy',
    'Grape__black_measles',
    'Grape__black_rot',
    'Grape__healthy',
    'Grape__leaf_blight_(isariopsis_leaf_spot)',
    'Jamun__diseased',
    'Jamun__healthy',
    'Lemon__diseased',
    'Lemon__healthy',
    'Mango__diseased',
    'Mango__healthy',
    'Peach__bacterial_spot',
    'Peach__healthy',
    'Pepper_bell__bacterial_spot',
    'Pepper_bell__healthy',
    'Pomegranate__diseased',
    'Pomegranate__healthy',
    'Potato__early_blight',
    'Potato__healthy',
    'Potato__late_blight',
    'Rice__brown_spot',
    'Rice__healthy',
    'Rice__hispa',
```

```
    'Rice__leaf_blast',
    'Rice__neck_blast',
    'Soybean__bacterial_blight',
    'Soybean__caterpillar',
    'Soybean__diabrotica_speciosa',
    'Soybean__downy_mildew',
    'Soybean__healthy',
    'Soybean__mosaic_virus',
    'Soybean__powdery_mildew',
    'Soybean__rust',
    'Soybean__southern_blight',
    'Strawberry___leaf_scorch',
    'Strawberry__healthy',
    'Sugarcane__bacterial_blight',
    'Sugarcane__healthy',
    'Sugarcane__red_rot',
    'Sugarcane__red_stripe',
    'Sugarcane__rust',
    'Tea__algal_leaf',
    'Tea__anthracnose',
    'Tea__bird_eye_spot',
    'Tea__brown_blight',
    'Tea__healthy',
    'Tea__red_leaf_spot',
    'Tomato__bacterial_spot',
    'Tomato__early_blight',
    'Tomato__healthy',
    'Tomato__late_blight',
    'Tomato__leaf_mold',
    'Tomato__mosaic_virus',
    'Tomato__septoria_leaf_spot',
    'Tomato__spider_mites_(two_spotted_spider_mite)',
    'Tomato__target_spot',
    'Tomato__yellow_leaf_curl_virus',
    'Wheat__brown_rust',
    'Wheat__healthy',
    'Wheat__septoria',
    'Wheat__yellow_rust'
]
```

[ ]:
```
test_image_paths = ['/kaggle/input/plant-disease-classification-merged-dataset/
 ↪Soybean__mosaic_virus/DSC_0133.jpg']

# Classify test images
predicted_classes = classify_images(model, test_image_paths)

# Display predicted classes
```

```
print(folder_names[predicted_classes[0]])
```

```
1/1 [==============================] - 0s 140ms/step
Soybean__mosaic_virus
```