```
import numpy as np
np.random.seed(345)
from tensorflow.random import set_seed
set_seed(3)
import shutil
from collections import defaultdict
import json
from pathlib import Path
import os
import numpy as np
from PIL import Image, ImageDraw, ImageFont
from numpy import asarray
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Activation
from tensorflow.keras.regularizers import l1_l2
from sklearn.metrics import precision_recall_fscore_support, accuracy_score
from sklearn import datasets
from sklearn import model_selection
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

```
from keras_preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Conv2D,MaxPooling2D, Dropout,Flatten,Dense,Activation,
BatchNormalization

from tensorflow.keras.optimizers import SGD

from tensorflow.keras import regularizers

from tensorflow.keras.models import Model, load_model, Sequential

from tensorflow.keras.optimizers import Adam, Adamax

tf.keras.backend.clear_session()

from tensorflow.keras.callbacks import CSVLogger

import time

from datetime import datetime

keras.__version__, tf.__version__

('2.6.0', '2.6.2')
```

Set pathways

```
train = '../input/vcor-vehicle-color-recognition-dataset/train/'

val = '../input/vcor-vehicle-color-recognition-dataset/val/'

test = '../input/vcor-vehicle-color-recognition-dataset/test/'

inference = '../input/fashion-product-images-small/images/'

annot_err = '../input/vcor-annot-errors/annotation-error.txt'
```

Number of images per category

The following shows the number of images across color categories.

```
def count_images(dataset):
    """
```

```
    dataset: 'train/' or 'test/'
    """
    v = []
    for i in sorted(os.listdir(dataset)):
        v.append(len(os.listdir(dataset+i)))
    df = pd.DataFrame({'color': sorted(os.listdir(dataset)), 'count': v})
    return(df)
df = count_images(train)
df
```

	color	count
0	beige	421
1	black	406
2	blue	742
3	brown	565
4	gold	210
5	green	563
6	grey	428
7	orange	534
8	pink	483
9	purple	536
10	red	637
11	silver	362
12	tan	400
13	white	403
14	yellow	577

```
df['count'].min(), df['count'].mean(), df['count'].max()
(210, 484.46666666666664, 742)
df['count'].sum()
7267
```

In the following a list is created of the 15 color categories. The list will be sorted.

```
class_subset = sorted(os.listdir(train))
class_subset
['beige',
 'black',
 'blue',
 'brown',
 'gold',
 'green',
 'grey',
 'orange',
 'pink',
 'purple',
 'red',
 'silver',
 'tan',
 'white'
'Yellow ']
isize = 224
model_name='EfficientNetB3'
```

```
base_model=tf.keras.applications.EfficientNetB3(include_top=False,
weights='imagenet',input_shape=(isize, isize, 3), pooling='max')

x=base_model.output

x=keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)

x = Dense(256, activation='relu')(x)

x=Dropout(rate=.45, seed=123)(x)

output=Dense(len(class_subset), activation='softmax')(x)

model=Model(inputs=base_model.input, outputs=output)

model.compile(Adamax(learning_rate=.0001), loss='categorical_crossentropy',
metrics=['accuracy'])

train_generator = ImageDataGenerator(
    rescale=1/255.,
    brightness_range=None,
    width_shift_range=0.5,
    rotation_range=False,
    horizontal_flip=True,
    vertical_flip=False
)


valid_generator = ImageDataGenerator(rescale=1/255.)

test_generator = ImageDataGenerator(rescale=1./255)


 def flatten(l):
    return [item for sublist in l for item in sublist]
def dataframe_keras(sourcedir, removal_files, textdir):
    """Automatically returns a DataFrame that can be used in flow_from_dataframe method.
```

```
    sourcedir: where the data folders corresponding to each class live, for ex. 'fashion-train/'

    Each class is in a different folder.

    removal_files: indicate 'yes' if specific image files need to be removed, for instance because of annotation errors.

    textdir: directory where a textfile can be found with files to be removed.

    """


    coll_labelnames = []

    coll_idfiles = []


    for i in os.listdir(sourcedir):

        idfiles = os.listdir(os.path.join(sourcedir, i))


        labelnames = len(idfiles) * [str(i)]




        coll_labelnames.append(labelnames)

        coll_idfiles.append(idfiles)


    df = pd.DataFrame({'label': flatten(coll_labelnames), 'idfiles': flatten(coll_idfiles)})

    df['id'] = str(sourcedir) + df['label'] + '/' + df['idfiles']


    print('Raw data before removal has shape:', df.shape)


    # removes image files that are mentioned in textfile.
```

```
    if removal_files == 'yes':
        with open(textdir) as f:
            files_to_remove = f.read().splitlines()
        files_to_remove = list(filter(lambda x: 'jpg' in x, files_to_remove))
        print('Number of image files to be removed:', len(files_to_remove))
        df = df[~df['idfiles'].isin(files_to_remove)]
        print('Raw data after removal has shape:', df.shape)
    else:
        print('No images removed')


    return(df)
traindf = dataframe_keras(train, 'yes', annot_err)
validdf = dataframe_keras(val, 'yes', annot_err)
Raw data before removal has shape: (7267, 3)
Number of image files to be removed: 64
Raw data after removal has shape: (7202, 3)
Raw data before removal has shape: (1550, 3)
Number of image files to be removed: 64
Raw data after removal has shape: (1550, 3)
traindf.head()
label    idfiles  id
0        orange 2cc754bf26.jpg        ../input/vcor-vehicle-color-recognition-datase...
1        orange a48677b1fe.jpg        ../input/vcor-vehicle-color-recognition-datase...
2        orange 8b0bd6dd65.jpg        ../input/vcor-vehicle-color-recognition-datase...
3        orange bfe246fdb1.jpg        ../input/vcor-vehicle-color-recognition-datase...
```

4 orange f25641efb1.jpg ../input/vcor-vehicle-color-recognition-datase...

```
BATCH_SIZE = 32
traingen=train_generator.flow_from_dataframe(
    dataframe=traindf,
    directory=None,
    x_col="id",
    y_col="label",
    subset="training",
    batch_size=BATCH_SIZE,
    shuffle=True,
    class_mode="categorical",
    target_size=(isize, isize))

validgen=valid_generator.flow_from_dataframe(
    dataframe=validdf,
    directory=None,
    x_col="id",
    y_col="label",
    subset="training",
    batch_size=BATCH_SIZE,
    shuffle=True,
    class_mode="categorical",
    target_size=(isize, isize))
model.save_weights("model_weights.h5")
```

The following displays performance for train and validation data.

```
plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper left')

plt.show()

class_subset = sorted(os.listdir(test))

test_generator = ImageDataGenerator(rescale=1./255)

testgen = test_generator.flow_from_directory(test,

                                             target_size=(isize, isize),

                                             batch_size=1,

                                             class_mode=None,

                                             classes=class_subset,

                                             shuffle=False

                                             )
```

Found 1556 images belonging to 15 classes.

```
model_predict = model.predict(testgen)

model_predict.shape
```

(1556, 15)

```
df_testprediction = pd.DataFrame(model_predict, index=testgen.filenames,
columns=class_subset)

pd.set_option('display.float_format', lambda x: '%.4f' % x)

df_testprediction
```

	beige	black	blue	brown	gold	green	grey	orange	pink	purple	red	silver	tan

white yellow

beige/02e37c0e56.jpg 0.3216 0.0013 0.0005 0.0290 0.0009 0.0083 0.0629 0.0003 0.0008 0.0003 0.0081 0.1611 0.4017 0.0012 0.0021

beige/05aeb6ddec.jpg 0.4453 0.0010 0.0026 0.4808 0.0078 0.0016 0.0129 0.0004 0.0007 0.0005 0.0007 0.0045 0.0367 0.0040 0.0005

beige/0b01008bee.jpg 0.6594 0.0005 0.0004 0.0215 0.0043 0.0010 0.0020 0.0004 0.0006 0.0002 0.0005 0.0056 0.2966 0.0060 0.0010

beige/0c3bb456ee.jpg 0.2540 0.0016 0.0005 0.0074 0.0081 0.0016 0.0058 0.0028 0.0017 0.0016 0.0013 0.0268 0.6491 0.0328 0.0049

beige/0e491569c8.jpg 0.2490 0.0010 0.0003 0.1141 0.0047 0.0004 0.0081 0.0017 0.0067 0.0008 0.0004 0.0026 0.6062 0.0039 0.0002

... ...

yellow/fd1da2f797.jpg 0.0002 0.0000 0.0001 0.0001 0.1420 0.0000 0.0000 0.0004 0.0000 0.0001 0.0000 0.0000 0.0000 0.0001 0.8569

yellow/fd3d2525fb.jpg 0.0000 0.0000 0.0000 0.0000 0.0001 0.0000 0.0000 0.0002 0.0001 0.0000 0.0000 0.0000 0.0000 0.0002 0.9992

yellow/feacf53e9e.jpg 0.0006 0.0001 0.0002 0.0000 0.1752 0.0029 0.0001 0.0001 0.0000 0.0001 0.0000 0.0002 0.0007 0.0001 0.8196

yellow/feeded3e6e.jpg 0.0000 0.0000 0.0000 0.0000 0.9355 0.0004 0.0000 0.0000 0.0000 0.0002 0.0000 0.0000 0.0000 0.0000 0.0637

yellow/ff029a0e58.jpg 0.0000 0.0000 0.0000 0.0000 0.0200 0.0000 0.0000 0.0007 0.0000 0.0001 0.0000 0.0000 0.0001 0.0001 0.9789

1556 rows × 15 columns

```
predicted_classes = np.argmax(model.predict(testgen), axis=-1)
#model.predict_classes(testgen). Argmax returns indices of max values

class_indices = traingen.class_indices

class_indices = dict((v,k) for k,v in class_indices.items())

true_classes = testgen.classes
```

```
len(predicted_classes)
```

```
1556
```

```
def display_results(y_true, y_preds, class_labels):

    results = pd.DataFrame(precision_recall_fscore_support(y_true, y_preds),
                           columns=class_labels).T

    results.rename(columns={0: 'Precision', 1: 'Recall',
                            2: 'F-Score', 3: 'Support'}, inplace=True)

    results.sort_values(by='F-Score', ascending=False, inplace=True)
    global_acc = accuracy_score(y_true, y_preds)

    print("Overall Categorical Accuracy: {:.2f}%".format(global_acc*100))
    return results

def plot_predictions(y_true, y_preds, test_generator, class_indices):

    fig = plt.figure(figsize=(20, 10))
    for i, idx in enumerate(np.random.choice(test_generator.samples, size=20, replace=False)):
        ax = fig.add_subplot(4, 5, i + 1, xticks=[], yticks=[])
        ax.imshow(np.squeeze(test_generator[idx]))
        pred_idx = y_preds[idx]
        true_idx = y_true[idx]
```

```
        plt.tight_layout()

        ax.set_title("{}\n({})".format(class_indices[pred_idx], class_indices[true_idx]), color=("green"
if pred_idx == true_idx else "red"))

    fig.savefig('testgallery.png')

display_results(true_classes, predicted_classes, class_indices.values())
```

Overall Categorical Accuracy: 83.55%

	Precision	Recall	F-Score	Support
blue	0.9744	0.9560	0.9651	159.0000
green	0.9590	0.9669	0.9630	121.0000
purple	0.9732	0.9478	0.9604	115.0000
yellow	0.9587	0.9355	0.9469	124.0000
orange	0.9123	0.9123	0.9123	114.0000
pink	0.9381	0.8835	0.9100	103.0000
red	0.8819	0.9338	0.9071	136.0000
white	0.8571	0.9070	0.8814	86.0000
black	0.8721	0.8621	0.8671	87.0000
brown	0.8211	0.8347	0.8279	121.0000
grey	0.7143	0.7609	0.7368	92.0000
silver	0.7143	0.6494	0.6803	77.0000
gold	0.7500	0.6000	0.6667	45.0000
tan	0.4364	0.5581	0.4898	86.0000
beige	0.4605	0.3889	0.4217	90.0000

```
plot_predictions(true_classes, predicted_classes, testgen, class_indices)

pd.DataFrame(confusion_matrix(true_classes, predicted_classes),
index=list(class_indices.values()), columns=list(class_indices.values()) )
```

	beige	black	blue	brown	gold	green	grey	orange	pink	purple	red	silver	tan	white	yellow

beige	35	0	0	6	0	0	1	0	0	0	0	2
44	1	1										
black	0	75	1	2	0	0	9	0	0	0	0	0
0	0	0										
blue	1	1	152	0	0	1	1	0	0	2	0	0
1	0	0										
brown	4	2	0	101	4	0	2	2	0	0	0	0
5	1	0										
gold	3	2	0	6	27	0	0	1	0	0	0	0
4	0	2										
green	0	1	1	0	0	117	1	0	0	0	0	0
1	0	0										
grey	1	5	0	2	0	2	70	0	0	0	0	11
1	0	0										
orange	0	0	0	1	0	0	0	104	1	0	6	0
0	0	2										
pink	0	0	0	0	0	0	0	0	91	1	11	0
0	0	0										
purple	0	0	2	1	0	0	1	0	2	109	0	0
0	0	0										
red	0	0	0	0	0	0	1	5	3	0	127	0
0	0	0										
silver	2	0	0	1	0	1	8	0	0	0	0	50
4	11	0										
tan	30	0	0	3	1	0	2	0	0	0	0	2
48	0	0										
white	0	0	0	0	0	0	1	0	0	0	0	5
2	78	0										
yellow	0	0	0	0	4	1	1					

```
def color_tag(sourcedir, imin, isize, thr):
    """
```

```
    Displays dominant colors beyond a given threshold.

    sourcedir: directory where image can be found, for ex.:
'/home/administrateur/Documents/RR/colors/fashion-train/purple'

    imin : image input, for ex 'blue-car.jpg'

    isize: input image size, for ex. 224

    thr: chosen threshold value

    """


    image = tf.keras.preprocessing.image.load_img(os.path.join(sourcedir, imin),
target_size=(isize, isize))

    data = asarray(image)

    ndata = np.expand_dims(data, axis=0)

    y_prob = model.predict(ndata/255)

    y_prob.argmax(axis=-1)


    print('color', [sorted(os.listdir(train))[i] for i in np.where(np.ravel(y_prob)>thr)[0]])

    print('values', [np.ravel(y_prob)[i] for i in list(np.where(np.ravel(y_prob)>thr)[0])])

tf.keras.preprocessing.image.load_img(os.path.join(inference, '10000.jpg'), target_size=(227,
227))


color_tag(inference, '10000.jpg', isize, 0.1)

color ['black', 'brown', 'grey', 'purple']

values [0.26599875, 0.106961556, 0.12884375, 0.25004336]

Example 2


tf.keras.preprocessing.image.load_img(os.path.join(inference, '10001.jpg'), target_size=(227,
227))
```

color_tag(inference, '10001.jpg', isize, 0.1)

color ['blue']

values [0.89126843]

Example 3

tf.keras.preprocessing.image.load_img(os.path.join(inference, '10004.jpg'), target_size=(227, 227))

color_tag(inference, '10004.jpg', isize, 0.1)

color ['black']

values [0.9450714]

Example 4

tf.keras.preprocessing.image.load_img(os.path.join(inference, '10025.jpg'), target_size=(227, 227))

color_tag(inference, '10025.jpg', isize, 0.1)

color ['pink', 'red']

values [0.5042953, 0.3663747]

Example 5

tf.keras.preprocessing.image.load_img(os.path.join(inference, '10081.jpg'), target_size=(227, 227))

color_tag(inference, '10081.jpg', isize, 0.1)

color ['beige', 'purple', 'white']

values [0.25849727, 0.1886504, 0.14310381]