This notebook won't run in a Windows environment - use Google Colab.

See https://www.perplexity.ai/search/this-is-the-school-project-i-a-Kq_luszJTBue69jHFhnZcw for list of alternative models

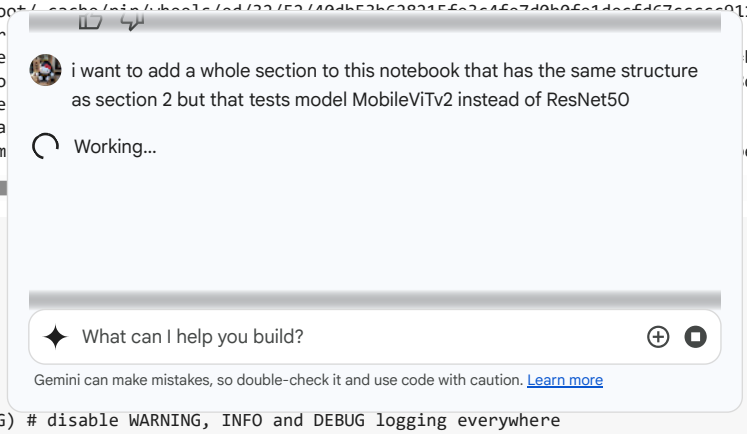
```
Start coding or generate with AI.
```

1 - Introduction

1.1 - Import packages & librairies

```
# run cell below first when restarting runtime in Google Colab
!pip install nlpaug plot_keras_history
```

```
Collecting nlpaug
  Downloading nlpaug-1.1.11-py3-none-any.whl.metadata (14 kB)
Collecting plot_keras_history
  Downloading plot_keras_history-1.1.39.tar.gz (12 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.11/dist-packages (from nlpaug) (2.0.2)
Requirement already satisfied: pandas>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from nlpaug) (2.2.2)
Requirement already satisfied: requests>=2.22.0 in /usr/local/lib/python3.11/dist-packages (from nlpaug) (2.32.3)
Requirement already satisfied: gdown>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from nlpaug) (5.2.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from plot_keras_history) (3.10.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from plot_keras_history) (1.15.3)
Collecting sanitize_ml_labels>=1.0.48 (from plot_keras_history)
  Downloading sanitize_ml_labels-1.1.4.tar.gz (324 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 324.5/324.5 kB 13.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dist-packages (from gdown>=4.0.0->nlpaug) (4.13.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from gdown>=4.0.0->nlpaug) (3.18.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from gdown>=4.0.0->nlpaug) (4.67.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2.0->nlpaug) (2.9.0
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2.0->nlpaug) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2.0->nlpaug) (2025.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.22.0->nlpaug)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.22.0->nlpaug) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.22.0->nlpaug) (2.4.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.22.0->nlpaug) (2025.4
Collecting compress-json (from sanitize_ml_labels>=1.0.48->plot_keras_history)
  Downloading compress_json-1.1.1.tar.gz (6.6 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->plot_keras_history) (1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->plot_keras_history) (0.12.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->plot_keras_history) (4
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->plot_keras_history) (1
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->plot_keras_history) (24
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->plot_keras_history) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->plot_keras_history) (3
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=1.2.0->nlpa
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4->gdown>=4.0.0->nlpaug)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4->gdown>=4.0
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.11/dist-packages (from requests[socks]->gdown>=4.0.0
Downloading nlpaug-1.1.11-py3-none-any.whl (410 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 410.5/410.5 kB 19.1 MB/s eta 0:00:00
Building wheels for collected packages: plot_keras_history, sanitize_ml_labels, compress-json
  Building wheel for plot_keras_history (setup.py) ... done
  Created wheel for plot_keras_history: filename=plot_keras_history-1.1.39-py3-none-any.whl size=10667 sha256=7c9704631994e3eddf24f6
  Stored in directory: /root/.cache/pip/wheels/56/8d/d7/bd70289b1bd192664225cd608fd08437ecc725c3f8918383d9
  Building wheel for sanitize_ml_labels (setup.py) ... done
  Created wheel for sanitize_ml_labels: filename=sanitize_ml_labels-1.1.4-py3-none-any.whl size=324285 sha256=eecb006c6a9d6ac505256
  Stored in directory: /root/.cache/pip/wheels/ed/22/53/40db52b638215fe2c45e7d0b0fe1decfdc7eeeec01118df507
  Building wheel for compr
  Created wheel for compre                                                                    bb8b8b1939342070c3225f4da42865654
  Stored in directory: /ro                                                                    525e
Successfully built plot_ke
Installing collected packa
Successfully installed com                                                                    els-1.1.4
```




```
# utilities
import sys
import datetime
from datetime import datetime
import random
import time
import logging
logging.disable(logging.WARNING) # disable WARNING, INFO and DEBUG logging everywhere
import os
import shutil
```

```
os.environ["TF_KERAS"]='1'
os.environ["TF_XLA_FLAGS"] = "--tf_xla_enable_xla_devices=false"
os.environ["OMP_NUM_THREADS"] = '1'  # needed to avoid memory leak warning with K-Means in Windows environment
from os import listdir
from glob import glob
from timeit import default_timer as timer

# data cleaning & processing
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

# dataviz
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.image import imread
import seaborn as sns
import plotly.express as px
from matplotlib.ticker import StrMethodFormatter
from matplotlib.ticker import FormatStrFormatter
from plot_keras_history import show_history, plot_history

# text processing
import re
import nltk
from nltk.tokenize import word_tokenize, RegexpTokenizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.corpus import stopwords
from collections import defaultdict
from nltk.stem import PorterStemmer, WordNetLemmatizer
from collections import Counter
from wordcloud import WordCloud

# text augmentation
import nlpaug.augmenter.word as naw

# image processing
import cv2
from PIL import Image

# image augmentation
import albumentations as A
from albumentations.pytorch import ToTensorV2

# modelisation
from sklearn import cluster, metrics, manifold, decomposition
from sklearn.cluster import MiniBatchKMeans, KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
import lightgbm
from lightgbm import LGBMClassifier
import xgboost as xgb
from xgboost import XGBClassifier
import umap
import tensorflow as tf
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, GlobalAveragePooling1D, Flatten, Dense, Dropout
from tensorflow.keras.layers import Rescaling, RandomFlip, RandomRotation, RandomZoom
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.applications import Xception, InceptionV3
from tensorflow.keras.applications.xception import preprocess_input as xception_preprocess
from tensorflow.keras.applications.inception_v3 import preprocess_input as inception_preprocess

# metrics
from sklearn import metrics
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, make_scorer, fbeta_score, precision_score, recall_score

# set dataframe display options
pd.set_option('max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.float_format', lambda x: '%.4f' % x) # Suppress scientific notation and show only 4 decimals
# pd.set_option('display.float_format', lambda x: '%.f' % x) # Suppress scientific notation and show only integer part

# silence warnings after checking
import warnings
# pd.set_option('future.no_silent_downcasting', False) # introduced in pandas 2.0.0., this notebook uses 1.4.4
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
# warnings.simplefilter(action='ignore', category=pd.errors.SettingWithCopyWarning) # introduced in pandas 2.0.0., this notebook uses 1.
# from PIL import ImageDecompressionBombWarning
warnings.simplefilter('ignore', Image.DecompressionBombWarning)

# extract colors from logo for ppt slideshow
# banana = findColor('banana.png')
# print("banana hex :", banana)
banana = '#fcf7c9'

viridis_sample = ['#481567FF','#453781FF','#39568CFF','#2D708EFF','#238A8DFF','#20A387FF','#3CBB75FF', '#73D055FF','#B8DE29FF']

viridis_palette = ['#440154', '#481e70', '#443982', '#3a528b', '#30678d', '#287b8e', '#20908c', '#20a485', '#35b778', '#5ec961',
                   '#90d643', '#c7e01f', '#fde724']

sunset_palette = ["#FFEB3B", "#FFDA44", "#FFC107", "#FFB300", "#FFA000", "#FF8F00", "#FF6F00", "#FF5722", "#FF3D00", "#FF2D00",
                  "#E53935", "#D32F2F", "#C62828", "#B71C1C", "#FF5252", "#FF1744", "#FF4081", "#F50057", "#D5006D", "#C51162"]

palette = ['#440154', '#481e70', '#443982', '#3a528b', '#30678d', '#287b8e', '#20908c', '#20a485', '#35b778', '#5ec961',
           '#90d643', '#c7e01f', '#fde724', "#FFEB3B", "#FFDA44", "#FFC107", "#FFB300", "#FFA000", "#FF8F00", "#FF6F00",
           "#FF5722", "#FF3D00", "#FF2D00", "#E53935", "#D32F2F", "#C62828", "#B71C1C", "#FF5252", "#FF1744", "#FF4081",
           "#F50057", "#D5006D", "#C51162"]
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
# run this cell in Google Colab only
from google.colab import drive
drive.mount('/content/drive')
image_path = '/content/drive/My Drive/Colab Notebooks/OCDS_P6&P8/flipkart_images'
image_path_aug = '/content/drive/My Drive/Colab Notebooks/OCDS_P6&P8/augmented_images'
save_path = '/content/drive/My Drive/Colab Notebooks/OCDS_P6&P8'
image_save_path = save_path
```

```
Mounted at /content/drive
```

```
# import custom user-defined functions
functions_path = os.path.join(save_path, 'functions.py')

# Check if the path is already in sys.path
if os.path.dirname(functions_path) not in sys.path:
    sys.path.append(os.path.dirname(functions_path))

from functions import *
```

```
# import split df with photo names, pre-processed text and product categories
train_df = pd.read_parquet(save_path + '/'+'train_df.gzip')
y_train = train_df['real_clusters']
val_df = pd.read_parquet(save_path + '/'+'val_df.gzip')
y_val = val_df['real_clusters']
test_df = pd.read_parquet(save_path + '/'+'test_df.gzip')
y_test = test_df['real_clusters']
train_val_df = pd.concat([train_df, val_df], axis=0, ignore_index=True)
y_train_val_images = pd.concat([y_train, y_val], axis=0, ignore_index=True)

# import merged text and image features
X_train_aug = pd.read_csv(save_path + '/' + 'X_train_aug.csv')
X_train_no_aug = pd.read_csv(save_path + '/' + 'X_train_no_aug.csv')
X_val_aug = pd.read_csv(save_path + '/' + 'X_val_aug.csv')
X_val_no_aug = pd.read_csv(save_path + '/' + 'X_val_no_aug.csv')
X_test_aug = pd.read_csv(save_path + '/' + 'X_test_aug.csv')
X_test_no_aug = pd.read_csv(save_path + '/' + 'X_test_no_aug.csv')
# create / import datasets for best model re-training
# X_train_val_aug = pd.concat([X_train_aug, X_val_aug], axis=0, ignore_index=True)
```

```
# print(X_train_val_aug.shape)
# X_train_val_aug.to_csv(save_path + '/' + 'X_traing_val_aug.csv', index=False)
X_train_val_aug = pd.read_csv(save_path + '/' + 'X_traing_val_aug.csv')
# X_train_val_no_aug = pd.concat([X_train_no_aug, X_val_no_aug], axis=0, ignore_index=True)
# print(X_train_val_no_aug.shape)
# X_train_val_no_aug.to_csv(save_path + '/' + 'X_traing_val_no_aug.csv', index=False)
X_train_val_no_aug = pd.read_csv(save_path + '/' + 'X_traing_val_no_aug.csv')
# y_train_val = pd.concat([y_train, y_val], axis=0, ignore_index=True)
# print(y_train_val.shape)
# y_train_val.to_csv(save_path + '/' + 'y_train_val.csv', index=False)
y_train_val = pd.read_csv(save_path + '/' + 'y_train_val.csv')

# import text features
text_features_train_aug = pd.read_csv(save_path + '/'+ 'text_features_train_aug.csv')
text_features_train_no_aug = pd.read_csv(save_path + '/'+ 'text_features_train_no_aug.csv')
text_features_val_aug = pd.read_csv(save_path + '/'+'text_features_val_aug.csv')
text_features_val_no_aug = pd.read_csv(save_path + '/'+'text_features_val_no_aug.csv')
text_features_test_aug = pd.read_csv(save_path + '/'+'text_features_test_aug.csv')
text_features_test_no_aug = pd.read_csv(save_path + '/'+'text_features_test_no_aug.csv')

# import image features
image_features_train_aug = pd.read_csv(save_path + '/' + 'image_features_train_aug.csv')
image_features_train_no_aug = pd.read_csv(save_path + '/' + 'image_features_train_no_aug.csv')
image_features_val = pd.read_csv(save_path + '/'+ 'image_features_val.csv')
image_features_test = pd.read_csv(save_path + '/'+ 'image_features_test.csv')
```

```
print(np.__version__, '\n')
print(tf.__file__, '\n')
print(tf.__version__, '\n')
print(hasattr(tf, 'keras'), '\n')
print(type(tf), '\n')
print(tf.__spec__, '\n')
print(dir(tf.keras))  # should list keras submodules
```

```
2.0.2

/usr/local/lib/python3.11/dist-packages/tensorflow/__init__.py

2.18.0

True

<class 'module'>

ModuleSpec(name='tensorflow', loader=<_frozen_importlib_external.SourceFileLoader object at 0x7d62038f2850>, origin='/usr/local/lib/

['DTypePolicy', 'FloatDTypePolicy', 'Function', 'Initializer', 'Input', 'InputSpec', 'KerasTensor', 'Layer', 'Loss', 'Metric', 'Mode
```

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    print(f"TensorFlow detected {len(gpus)} GPU(s):")
    for gpu in gpus:
        print(f" - {gpu}")
else:
    print("TensorFlow did NOT detect any GPUs.")

print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))
print(tf.test.is_built_with_cuda())
```

```
TensorFlow detected 1 GPU(s):
 - PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')
Num GPUs Available:  1
True
```

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Thu Jun  5 07:58:47 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   41C    P8              9W /   70W |       2MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
```

```
+---------------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                             GPU Memory |
|        ID   ID                                                              Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

```
from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')
```

```
Your runtime has 54.8 gigabytes of available RAM

You are using a high-RAM runtime!
```

```
# initialise random state for all models and transformers
rs_list = [8, 13, 42]
rs = rs_list[random.randrange(len(rs_list))]
print("Random state =", rs)
```

```
Random state = 8
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

1.2 - Define image augmentation functions

1.2.1 - Image augmentation

```
# Define augmentation pipeline
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=15, p=0.5),
    # A.RandomResizedCrop(height=224, width=224, scale=(0.8, 1.0), p=0.5), # outdated syntax
    A.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0), p=0.5), # updated syntax
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1, p=0.5),
    A.GaussianBlur(blur_limit=(3, 7), p=0.3),
    A.CoarseDropout(max_holes=1, max_height=32, max_width=32, p=0.3),
], p=1.0)

def augment_and_save_train_images(image_path, image_save_path, train_df):
    # Create output directory if it doesn't exist
    save_dir = os.path.join(image_save_path, "augmented_images")
    os.makedirs(save_dir, exist_ok=True)

    # Get the list of image filenames from train_df
    train_image_files = set(train_df['image'].tolist())

    print(f"Found {len(train_image_files)} images in training set to augment.")

    for idx, filename in enumerate(train_image_files):
        img_path = os.path.join(image_path, filename)
        image = cv2.imread(img_path)
        if image is None:
            print(f"Warning: Could not load image {img_path}. Skipping.")
            continue

        # Convert BGR to RGB for Albumentations
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Apply augmentation
        augmented = transform(image=image_rgb)
        aug_image = augmented['image']

        # Convert back RGB to BGR for saving with OpenCV
        aug_image_bgr = cv2.cvtColor(aug_image, cv2.COLOR_RGB2BGR)

        # Add '_aug' suffix before the file extension
```

```
        name, ext = os.path.splitext(filename)
        save_filename = f"{name}{ext}"
        save_path = os.path.join(save_dir, save_filename)

        cv2.imwrite(save_path, aug_image_bgr)

        if idx % 100 == 0:
            print(f"Processed {idx}/{len(train_image_files)} images")

    print(f"Augmented training images saved to: {save_dir}")
```

```
Start coding or generate with AI.
```

1.2.2 - Image processing

```
def extract_features_from_list(model, preprocess_func, list_photos, path, target_size=(299, 299)):
    features = []
    start_time = time.time()
    for i, photo in enumerate(list_photos):
        if i % 100 == 0:
            print(f"Processing image {i}/{len(list_photos)}")
        img_path = path + '/' + photo  # full path to image
        image = load_img(img_path, target_size=target_size)
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)
        image = preprocess_func(image)
        feat = model.predict(image, verbose=0)
        features.append(feat[0])
    duration = time.time() - start_time
    print(f"Features creation time: {duration:.2f} secs")
    return np.array(features)
```

```
def extract_sift_descriptors(list_photos, path, sift):
    sift_keypoints = []

    for image_num, filename in enumerate(list_photos):
        if image_num % 50 == 0:
           print(f'progress : {image_num / len(list_photos) * 100:.2f} %')

        image_path = os.path.join(path, filename)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Load grayscale

        if image is None:
            print(f"Could not load image: {image_path}")
            sift_keypoints.append(None)  # Keep alignment with input list
            continue

        # Histogram equalization
        equalized = cv2.equalizeHist(image)

        # Detect SIFT keypoints and descriptors
        kp, des = sift.detectAndCompute(equalized, None)

        sift_keypoints.append(des)

    return sift_keypoints
```

```
def list_pix(name) :
    list_image_name = [list_photos[i] for i in range(len(data)) if photo_data["product_category"][i]==name]
    return list_image_name
```

```
def save_image_set(image_path, image_save_path, val_test_df):
    # Create output directory if it doesn't exist
    save_dir = os.path.join(image_save_path)
    os.makedirs(save_dir, exist_ok=True)

    # Get the list of image filenames from train_df
    image_files = set(val_test_df['image'].tolist())

    print(f"Found {len(image_files)} images in set.")

    for idx, filename in enumerate(image_files):
        img_path = os.path.join(image_path, filename)
        image = cv2.imread(img_path)
        if image is None:
            print(f"Warning: Could not load image {img_path}. Skipping.")
            continue
```

```
        # Build the full save path for each image:
        save_file_path = os.path.join(image_save_path, filename)

        cv2.imwrite(save_file_path, image)  # Corrected this line

        if idx % 25 == 0:
          print(f"Processed {idx}/{len(image_files)} images")

    print(f"Images saved to: {save_dir}")
```

```
def save_image_subsets(image_path, image_save_path, val_test_df):
    # Ensure the main save directory exists
    os.makedirs(image_save_path, exist_ok=True)

    # Get unique cluster labels
    unique_clusters = val_test_df['real_clusters'].unique()
    print(f"Found {len(val_test_df)} images in set across {len(unique_clusters)} clusters.")

    # Create subdirectories for each cluster
    for cluster in unique_clusters:
        cluster_dir = os.path.join(image_save_path, str(cluster))
        os.makedirs(cluster_dir, exist_ok=True)

    # Iterate over rows and copy images to the appropriate cluster subfolder
    for idx, row in val_test_df.iterrows():
        filename = row['image']
        cluster = row['real_clusters']
        img_path = os.path.join(image_path, filename)
        save_dir = os.path.join(image_save_path, str(cluster))
        save_file_path = os.path.join(save_dir, filename)

        image = cv2.imread(img_path)
        if image is None:
            print(f"Warning: Could not load image {img_path}. Skipping.")
            continue

        cv2.imwrite(save_file_path, image)

        if idx % 25 == 0:
          print(f"Processed {idx}/{len(val_test_df)} images")

    print(f"Images saved to subfolders in: {image_save_path}")
```

```
def merge_subfolders(folder_a_path, folder_b_path, folder_c_path):
    print(f"Merging files from '{folder_a_path}' and '{folder_b_path}' into '{folder_c_path}'...")

    # Ensure the destination parent directory exists
    os.makedirs(folder_c_path, exist_ok=True)

    # Get list of subfolders from folder A (assuming B has the same)
    try:
        subfolders_a = [d for d in os.listdir(folder_a_path) if os.path.isdir(os.path.join(folder_a_path, d))]
        print(f"Found subfolders in '{folder_a_path}': {subfolders_a}")
    except FileNotFoundError:
        print(f"Error: Source folder A not found at '{folder_a_path}'")
        return
    except Exception as e:
        print(f"An error occurred while listing subfolders in '{folder_a_path}': {e}")
        return

    # Iterate through each subfolder found in folder A
    for subfolder_name in subfolders_a:
        subfolder_a_full_path = os.path.join(folder_a_path, subfolder_name)
        subfolder_b_full_path = os.path.join(folder_b_path, subfolder_name)
        subfolder_c_full_path = os.path.join(folder_c_path, subfolder_name)

        # Create the corresponding subfolder in the destination folder C
        os.makedirs(subfolder_c_full_path, exist_ok=True)

        print(f"Processing subfolder '{subfolder_name}'...")

        # Copy files from subfolder A to subfolder C
        if os.path.isdir(subfolder_a_full_path):
            try:
                files_in_a = os.listdir(subfolder_a_full_path)
                print(f"Found {len(files_in_a)} files in '{subfolder_a_full_path}'")
                for item_name in files_in_a:
                    source_item_path = os.path.join(subfolder_a_full_path, item_name)
                    destination_item_path = os.path.join(subfolder_c_full_path, item_name)
                    # Only copy if it's a file and doesn't already exist in the destination
```

```
                    if os.path.isfile(source_item_path):
                         # Avoid copying if the file name already exists (in case of duplicates)
                        if not os.path.exists(destination_item_path):
                            shutil.copy2(source_item_path, destination_item_path) # copy2 preserves metadata
            except FileNotFoundError:
                 print(f"Warning: Subfolder A '{subfolder_name}' not found at '{subfolder_a_full_path}'. Skipping.")
                 pass # Skip if the subfolder is missing in A
            except Exception as e:
                print(f"An error occurred while copying files from '{subfolder_a_full_path}': {e}")


        # Copy files from subfolder B to subfolder C
        if os.path.isdir(subfolder_b_full_path):
             try:
                files_in_b = os.listdir(subfolder_b_full_path)
                print(f"Found {len(files_in_b)} files in '{subfolder_b_full_path}'")
                for item_name in files_in_b:
                    source_item_path = os.path.join(subfolder_b_full_path, item_name)
                    destination_item_path = os.path.join(subfolder_c_full_path, item_name)
                     # Only copy if it's a file and doesn't already exist in the destination
                    if os.path.isfile(source_item_path):
                        # Avoid copying if the file name already exists (in case of duplicates)
                        if not os.path.exists(destination_item_path):
                            shutil.copy2(source_item_path, destination_item_path) # copy2 preserves metadata
             except FileNotFoundError:
                 print(f"Warning: Subfolder B '{subfolder_name}' not found at '{subfolder_b_full_path}'. Skipping.")
                 pass # Skip if the subfolder is missing in B
             except Exception as e:
                print(f"An error occurred while copying files from '{subfolder_b_full_path}': {e}")

    print("Merging process completed.")
```

```
def merge_subfolders_w_dups(folder_a_path, folder_b_path, folder_c_path):
    print(f"Merging files from '{folder_a_path}' and '{folder_b_path}' into '{folder_c_path}'...")

    # Ensure the destination parent directory exists
    os.makedirs(folder_c_path, exist_ok=True)

    # Get list of subfolders from folder A (assuming B has the same)
    try:
        subfolders_a = [d for d in os.listdir(folder_a_path) if os.path.isdir(os.path.join(folder_a_path, d))]
        print(f"Found subfolders in '{folder_a_path}': {subfolders_a}")
    except FileNotFoundError:
        print(f"Error: Source folder A not found at '{folder_a_path}'")
        return
    except Exception as e:
        print(f"An error occurred while listing subfolders in '{folder_a_path}': {e}")
        return

    # Iterate through each subfolder found in folder A
    for subfolder_name in subfolders_a:
        subfolder_a_full_path = os.path.join(folder_a_path, subfolder_name)
        subfolder_b_full_path = os.path.join(folder_b_path, subfolder_name)
        subfolder_c_full_path = os.path.join(folder_c_path, subfolder_name)

        # Create the corresponding subfolder in the destination folder C
        os.makedirs(subfolder_c_full_path, exist_ok=True)

        print(f"Processing subfolder '{subfolder_name}'...")

        # Copy files from subfolder A to subfolder C
        if os.path.isdir(subfolder_a_full_path):
            try:
                files_in_a = os.listdir(subfolder_a_full_path)
                print(f"Found {len(files_in_a)} files in '{subfolder_a_full_path}'")
                for item_name in files_in_a:
                    source_item_path = os.path.join(subfolder_a_full_path, item_name)
                    destination_item_path = os.path.join(subfolder_c_full_path, item_name)
                    # Only copy if it's a file
                    if os.path.isfile(source_item_path):
                        # Check if the file name already exists in the destination
                        base, ext = os.path.splitext(item_name)
                        counter = 1
                        while os.path.exists(destination_item_path):
                            destination_item_path = os.path.join(subfolder_c_full_path, f"{base}_{counter}{ext}")
                            counter += 1
                        shutil.copy2(source_item_path, destination_item_path) # copy2 preserves metadata
            except FileNotFoundError:
                 print(f"Warning: Subfolder A '{subfolder_name}' not found at '{subfolder_a_full_path}'. Skipping.")
                 pass # Skip if the subfolder is missing in A
            except Exception as e:
                print(f"An error occurred while copying files from '{subfolder_a_full_path}': {e}")
```

```
        # Copy files from subfolder B to subfolder C
        if os.path.isdir(subfolder_b_full_path):
             try:
                files_in_b = os.listdir(subfolder_b_full_path)
                print(f"Found {len(files_in_b)} files in '{subfolder_b_full_path}'")
                for item_name in files_in_b:
                    source_item_path = os.path.join(subfolder_b_full_path, item_name)
                    destination_item_path = os.path.join(subfolder_c_full_path, item_name)
                     # Only copy if it's a file
                    if os.path.isfile(source_item_path):
                        # Check if the file name already exists in the destination
                        base, ext = os.path.splitext(item_name)
                        counter = 1
                        while os.path.exists(destination_item_path):
                            destination_item_path = os.path.join(subfolder_c_full_path, f"{base}_{counter}{ext}")
                            counter += 1
                        shutil.copy2(source_item_path, destination_item_path) # copy2 preserves metadata
             except FileNotFoundError:
                 print(f"Warning: Subfolder B '{subfolder_name}' not found at '{subfolder_b_full_path}'. Skipping.")
                 pass # Skip if the subfolder is missing in B
             except Exception as e:
                print(f"An error occurred while copying files from '{subfolder_b_full_path}': {e}")

    print("Merging process completed.")
```

```
Start coding or generate with AI.
```

1.2.3 - Graphing functions

```
def plot_pca_scree(explained_variance_ratio, cumulative_variance, components,
                   num_components_99, num_component_threshold,
                   num_component_kaiser, cumulative_variance_kaiser,
                   max_ticks=10,
                   save_path=None):

    sns.set(rc={'figure.figsize': (7, 4), 'axes.facecolor': 'white', 'figure.facecolor': 'gainsboro'})

    step = max(1, len(components) // max_ticks)
    xticks_to_show = components[::step]

    # Define colors
    individual_color = 'cornflowerblue'
    cumulative_color = banana
    line_99_color = 'limegreen'
    line_kaiser_color = 'fuchsia'

    plt.plot(components, explained_variance_ratio * 100,
             linewidth=2, color=individual_color, label='Individual Explained Variance')
    plt.plot(components, cumulative_variance * 100,
             linewidth=2, color=cumulative_color, label='Cumulative Explained Variance')

    plt.axhline(y=99, color=line_99_color, linestyle='--',
                label='99% Variance Threshold', linewidth=0.5)

    plt.axhline(y=cumulative_variance_kaiser, color=line_kaiser_color, linestyle='--',
                label=f'Kaiser Variance {cumulative_variance_kaiser}% ', linewidth=0.5)

    plt.axvline(x=num_components_99, color=line_99_color, linestyle='-.',
                label=f'{num_components_99} Components for 99% Variance', linewidth=0.5)

    plt.axvline(x=num_component_threshold, color=line_kaiser_color, linestyle='-.',
                label=(f"Kaiser's rule ({num_component_kaiser} components)\n"
                       f"{cumulative_variance_kaiser} % explained variance"),
                       linewidth=0.5)

    plt.xlabel('Principal Component', fontsize=10, fontweight='bold')
    plt.ylabel('Explained Variance (%)', fontsize=10, fontweight='bold')
    plt.title('Scree Plot with Variance Thresholds', fontsize=16, fontweight='bold')
    plt.xticks(xticks_to_show)
    plt.legend()
    plt.grid(axis='y', color='gainsboro')
    plt.tight_layout()

    if save_path:
        plt.savefig(save_path)
    plt.show()
```

```
def plot_tsne_clusters(df_tsne, save_path=None):
    sns.set(rc={'figure.figsize': (8, 5), 'axes.facecolor': 'white', 'figure.facecolor': 'gainsboro'})
    sns.scatterplot(x="tsne1", y="tsne2", hue="cat_clusters", data=df_tsne, legend="brief", palette=sns.color_palette('Set2', n_colors=7
                    s=50, alpha=0.5)
    plt.title('t-SNE - product categories', fontsize=14, fontweight='bold')
    plt.xlabel('t-SNE 1', fontsize=10, fontweight='bold')
    plt.ylabel('t-SNE 2', fontsize=10, fontweight='bold')
    plt.axhline(y=0, color='gainsboro', linewidth=1)
    plt.axvline(x=0, color='gainsboro', linewidth=1)
    plt.legend(prop={'size': 10})
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path)
    plt.show()
```

```
def plot_tsne_kmeans_clusters(df_tsne, save_path=None):
    plt.figure(figsize=(8,5))
    sns.scatterplot(x="tsne1", y="tsne2", hue='cluster_kmeans', palette=sns.color_palette('Set2', n_colors=7), s=50, alpha=0.5,
                    data=df_tsne, legend="brief")
    plt.axhline(y=0, color='gainsboro', linewidth=1)
    plt.axvline(x=0, color='gainsboro', linewidth=1)
    plt.title('t-SNE - K-Means clusters ', fontsize=14, fontweight='bold')
    plt.xlabel('t-SNE 1', fontsize=10, fontweight='bold')
    plt.ylabel('t-SNE 2', fontsize=10, fontweight='bold')
    plt.legend(prop={'size': 9})
    plt.tight_layout()

    if save_path:
        plt.savefig(save_path)
    plt.show()
```

```
def compute_tsne(feat_pca, photo_data, random_state=None):
    tsne = manifold.TSNE(n_components=2, perplexity=30, n_iter=2000,
                         init='random', random_state=random_state)
    X_tsne = tsne.fit_transform(feat_pca)

    df_tsne = pd.DataFrame(X_tsne[:, :2], columns=['tsne1', 'tsne2'])
    df_tsne["real_clusters"] = photo_data['real_clusters'].values
    df_tsne["cat_clusters"] = photo_data['real_clusters'].astype(str) + " - " + photo_data['product_category']

    print(f"t-SNE DataFrame shape: {df_tsne.shape}")
    return df_tsne
```

```
Start coding or generate with AI.
```

1.3 - Define image classification metrics

```
def pca_analysis(im_features, random_state=None):
    pca = PCA(random_state=random_state)
    pca.fit(im_features)

    explained_variance_ratio = pca.explained_variance_ratio_
    cumulative_variance = np.cumsum(explained_variance_ratio)
    components = np.arange(1, len(explained_variance_ratio) + 1)

    num_components_99 = np.argmax(cumulative_variance >= 0.99) + 1
    threshold = 1 / len(explained_variance_ratio)
    num_component_threshold = np.argmax(explained_variance_ratio < threshold) + 1

    # Kaiser criterion: components with eigenvalue > average eigenvalue (threshold)
    num_component_kaiser = np.argmax(explained_variance_ratio < threshold) + 1
    cumulative_variance_kaiser = round(np.sum(explained_variance_ratio[:num_component_kaiser]) * 100, 2)

    return (explained_variance_ratio, cumulative_variance, components,
            num_components_99, threshold, num_component_threshold,
            num_component_kaiser, cumulative_variance_kaiser)
```

```
# Create results DataFrame
models_data_P8 = []
histories = {}
```

```
Start coding or generate with AI.
```

1.4 - Import & split data

```
category_mapping = pd.read_csv(save_path + '/' +'category_mapping.csv')
category_mapping
```

	product_category	category_numeric
0	Baby Care	0
1	Beauty and Personal Care	1
2	Computers	2
3	Home Decor & Festive Needs	3
4	Home Furnishing	4
5	Kitchen & Dining	5
6	Watches	6

Next steps: Generate code with category_mapping | View recommended plots | New interactive sheet

```
products_trim_final = pd.read_parquet(save_path + '/' +'products_trim_final.parquet.gzip')
# products_trim_final.head(1)
```

```
feat = products_trim_final['corpus_deep_learn']
# y_cat_num = products_trim_final['real_clusters']
```

```
# First split: train (70%) and temp (30%)
train_df, temp_df = train_test_split(products_trim_final, test_size=(350/1050), stratify=products_trim_final['real_clusters'],
                                     random_state=rs)

# Second split: validation (15%) and test (15%) from temp (30%)
val_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df['real_clusters'], random_state=rs)
```

```
# Check the sizes
print(f"Train size: {len(train_df)}")
print(f"Validation size: {len(val_df)}")
print(f"Test size: {len(test_df)}")
```

```
Train size: 700
Validation size: 175
Test size: 175
```

```
print("Train class distribution:")
train_df['real_clusters'].value_counts()
```

```
Train class distribution:
```

real_clusters	count
0	100
2	100
5	100
6	100
1	100
4	100
3	100

dtype: int64

```
print("Validation class distribution:")
val_df['real_clusters'].value_counts()
```

```
Validation class distribution:
```

real_clusters	count
4	25
5	25
3	25
1	25
2	25
6	25
0	25

dtype: int64

```
print("Test class distribution:")
test_df['real_clusters'].value_counts()
```

```
Test class distribution:
```

real_clusters	count
6	25
2	25
1	25
0	25
4	25
5	25
3	25

dtype: int64

```
train_df.to_parquet('train_df.gzip', compression='gzip')
val_df.to_parquet('val_df.gzip', compression='gzip')
test_df.to_parquet('test_df.gzip', compression='gzip')
```

```
Start coding or generate with AI.
```

1.5 - Data augmentation

```
Start coding or generate with AI.
```

1.5.1 - Training images

```
# train_df = pd.read_parquet(save_path + '/'+'train_df.gzip')
```

```
# !!!Empty directory before regenerating!!!
# augment_and_save_train_images(image_path, image_save_path, train_df)
```

```
Start coding or generate with AI.
```

1.5.2 - Training + validation images

```
# !!!Empty directory before regenerating!!!
# augment_and_save_train_images(image_path, image_save_path, train_val_df)
```

```
# Generate subsets only once - remove folder before re-generating
# save_image_subsets(image_save_path + '/' +'augmented_images_train_val', image_save_path + '/' + 'train_val_images_subsets_aug', val_te
```

```
Start coding or generate with AI.
```

1.5.3 - Create grouped image subfolders

```
# Create grouped subfolders for train + val images
# source_folder_1 = image_save_path + '/' + 'train_images_subsets'
# source_folder_2 = image_save_path + '/' + 'val_images_subsets'
# destination_folder_1 = image_save_path + '/' + 'train_val_images_subsets'
# merge_subfolders(source_folder_1, source_folder_2, destination_folder_1)
```

```
# Create grouped subfolders for (train + val images) + (train + val images) augmented
# source_folder_3 = image_save_path + '/' + 'train_val_images_subsets'
# source_folder_4 = image_save_path + '/' + 'train_val_images_subsets_aug'
# destination_folder_2 = image_save_path + '/' + 'train_val_images_subsets_ALL'
# merge_subfolders_w_dups(source_folder_3, source_folder_4, destination_folder_2)
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

2 - Supervised image classification - Transfer Learning with ResNet50

2.1 - Instanciate model

```
def init_model():
    # Initialize ResNet50 base model
    model_init = ResNet50(include_top=False,
                          weights="imagenet",
                          input_shape=(224, 224, 3))  # Changed to ResNet50

    # Freeze all layers in base model
    for layer in model_init.layers:
        layer.trainable = False

    # Add custom top layers
    x = model_init.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    predictions = Dense(7, activation='softmax')(x)  # Keep final layer

    # Create full model
    model = Model(inputs=model_init.input, outputs=predictions)

    # Compile model (keeping original optimizer/loss)
    model.compile(loss="categorical_crossentropy",
                  optimizer='rmsprop',
                  metrics=['accuracy', tf.keras.metrics.F1Score(name='f1_macro', average='macro')])

    # print(model.summary())
    return model
```

```
def init_model_aug():
    # Data augmentation
    data_augmentation = Sequential([
        RandomFlip("horizontal", input_shape=(224, 224, 3)),
        RandomRotation(0.1),
        RandomZoom(0.1),
        ])

    # Initialize ResNet50 base model
    model_init = ResNet50(include_top=False,
                          weights="imagenet",
                          input_shape=(224, 224, 3))

    # Freeze all layers in base model
    for layer in model_init.layers:
        layer.trainable = False

    # Add custom top layers, including data augmentation
    x = model_init.input
    x = data_augmentation(x)
```

```
    x = model_init(x)
    x = GlobalAveragePooling2D()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    predictions = Dense(7, activation='softmax')(x)

    # Create full model
    model = Model(inputs=model_init.input, outputs=predictions)

    # Compile model
    model.compile(loss="categorical_crossentropy",
                  optimizer='rmsprop',
                  metrics=['accuracy', tf.keras.metrics.F1Score(name='f1_macro', average='macro')])

    # print(model.summary())
    return model
```

```
def train_and_evaluate_model(model_name, model, epochs, dataset_train, dataset_val, callbacks=None, verbose=0):
    if callbacks is None:
        callbacks = []  # Default is empty list

    if not any(isinstance(cb, EarlyStopping) for cb in callbacks):
        early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
        callbacks.append(early_stopping)
    else:
        early_stopping = [cb for cb in callbacks if isinstance(cb, EarlyStopping)][0]

    start_time = time.time()
    history = model.fit(dataset_train, epochs=epochs, validation_data=dataset_val, callbacks=callbacks, verbose=verbose)
    end_time = time.time()
    fit_time = end_time - start_time

    early_stopping_epoch = early_stopping.stopped_epoch + 1 if early_stopping.stopped_epoch is not None else len(history.history['loss'])

    y_true_val = []
    for _, labels in dataset_val:
        y_true_val.extend(np.argmax(labels.numpy(), axis=1))

    y_true_val = np.array(y_true_val)


    y_pred_val_probs = model.predict(dataset_val, verbose=verbose)

    y_pred_val = np.argmax(y_pred_val_probs, axis=1)


    val_accuracy = accuracy_score(y_true_val, y_pred_val)
    val_precision_macro = precision_score(y_true_val, y_pred_val, average='macro', zero_division=0)
    val_recall_macro = recall_score(y_true_val, y_pred_val, average='macro', zero_division=0)
    val_f1_macro_sklearn = f1_score(y_true_val, y_pred_val, average='macro', zero_division=0) # Recalculate with sklearn for consistency
    val_f2_macro = fbeta_score(y_true_val, y_pred_val, beta=2, average='macro', zero_division=0)


    train_acc_hist = history.history['accuracy'][-1]
    val_acc_hist = history.history['val_accuracy'][-1]
    train_f1_hist = history.history['f1_macro'][-1]
    val_f1_hist = history.history['val_f1_macro'][-1]


    model_data = {
        'model_name': model_name,
        'epochs_run': len(history.history['loss']),
        'early_stopping_epoch': early_stopping_epoch,
        'fit_time': fit_time,


        'Train Accuracy (Hist)': train_acc_hist,
        'Val Accuracy (Hist)': val_acc_hist,
        'Train f1_macro (Hist)': train_f1_hist,
        'Val f1_macro (Hist)': val_f1_hist,

        'Val Accuracy (Sklearn)': val_accuracy,
        'Val Precision (Sklearn)': val_precision_macro,
        'Val Recall (Sklearn)': val_recall_macro,
        'Val f1_macro (Sklearn)': val_f1_macro_sklearn,
        'Val f2_macro (Sklearn)': val_f2_macro
    }

    return model_data, history
```

Start coding or generate with AI.

2.2 - Import image sets

```
batch_size = 32

def dataset_fct(path, validation_split=0, data_type=None) :
    dataset = tf.keras.utils.image_dataset_from_directory(
                    path, labels='inferred', label_mode='categorical',
                    class_names=None, batch_size=batch_size, image_size=(224, 224), shuffle=False,
                    seed=rs, validation_split=validation_split, subset=data_type
                    )
    return dataset
```

```
dataset_train_aug = dataset_fct(path=save_path + '/' + 'train_images_subsets_aug', validation_split=0, data_type=None)
```

```
Found 700 files belonging to 7 classes.
```

```
dataset_train = dataset_fct(path=save_path + '/' + 'train_images_subsets', validation_split=0, data_type=None)
```

```
Found 700 files belonging to 7 classes.
```

```
dataset_val = dataset_fct(path=save_path + '/' + 'val_images_subsets', validation_split=0, data_type=None)
```

```
Found 175 files belonging to 7 classes.
```

```
dataset_test = dataset_fct(path=save_path + '/' + 'test_images_subsets', validation_split=0, data_type=None)
```

```
Found 175 files belonging to 7 classes.
```

```
dataset_train_val = dataset_fct(path=save_path + '/' + 'train_val_images_subsets', validation_split=0, data_type=None)
```

```
Found 875 files belonging to 7 classes.
```

```
dataset_train_val_aug = dataset_fct(path=save_path + '/' + 'train_val_images_subsets_aug', validation_split=0, data_type=None)
```

```
Found 875 files belonging to 7 classes.
```

```
dataset_train_val_ALL = dataset_fct(path=save_path + '/' + 'train_val_images_subsets_ALL', validation_split=0, data_type=None)
```

```
Found 1750 files belonging to 7 classes.
```

Start coding or generate with AI.

2.3 - Modelisation with external image augmentation

```
# Create model
with tf.device('/gpu:0'):
    model_1 = init_model()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernel
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

```
# Create callbacks
model_1_save_path = "./model_1_best_weights.h5"
checkpoint_1 = ModelCheckpoint(model_1_save_path, monitor='val_loss', verbose=1,
                               save_best_only=True, mode='min')
es_1 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
callbacks_list_1 = [checkpoint_1, es_1]
```

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

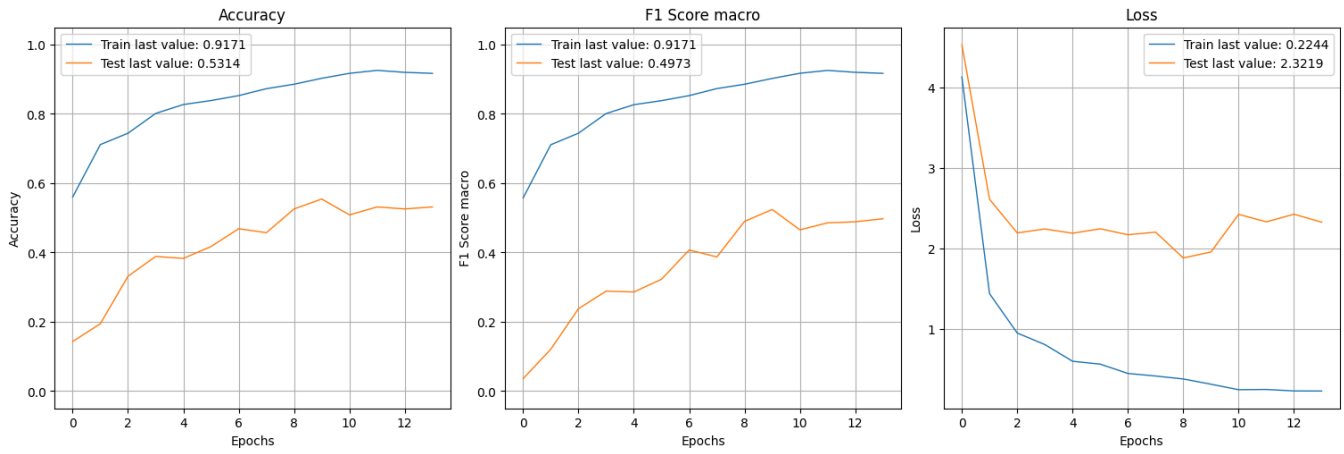
```
Num GPUs Available:  1
```

```
with tf.device('/gpu:0'):
    # Train model on augmented images & evaluate
    model_1_data, history_1 = train_and_evaluate_model('ResNet50 External Data Augmentation', model_1, epochs=50,
                                                       dataset_train=dataset_train_aug, dataset_val=dataset_val,
                                                       callbacks=callbacks_list_1)
    models_data_P8.append(model_1_data)
    histories['model_1'] = history_1
```

```
histories['model_1'] = history_1
```

```
Epoch 1: val_loss improved from inf to 4.52755, saving model to ./model_1_best_weights.h5

Epoch 2: val_loss improved from 4.52755 to 2.60655, saving model to ./model_1_best_weights.h5

Epoch 3: val_loss improved from 2.60655 to 2.18852, saving model to ./model_1_best_weights.h5

Epoch 4: val_loss did not improve from 2.18852

Epoch 5: val_loss improved from 2.18852 to 2.18464, saving model to ./model_1_best_weights.h5

Epoch 6: val_loss did not improve from 2.18464

Epoch 7: val_loss improved from 2.18464 to 2.16598, saving model to ./model_1_best_weights.h5

Epoch 8: val_loss did not improve from 2.16598

Epoch 9: val_loss improved from 2.16598 to 1.87759, saving model to ./model_1_best_weights.h5

Epoch 10: val_loss did not improve from 1.87759

Epoch 11: val_loss did not improve from 1.87759

Epoch 12: val_loss did not improve from 1.87759

Epoch 13: val_loss did not improve from 1.87759

Epoch 14: val_loss did not improve from 1.87759
Epoch 14: early stopping
```

```
show_history(histories['model_1'])
plot_history(histories['model_1'], path="save_path + '/' + ResNet50_augmented_data.png")
plt.close()
```




```
Start coding or generate with AI.
```

2.4 - Modelisation with integrated image augmentation

```
# Create model
with tf.device('/gpu:0'):
    model_2 = init_model_aug()
```

```
# Create callbacks
model_2_save_path = "./model_2_best_weights.h5"
checkpoint_2 = ModelCheckpoint(model_2_save_path, monitor='val_loss', verbose=1,
                               save_best_only=True, mode='min')
es_2 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
callbacks_list_2 = [checkpoint_2, es_2]
```

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

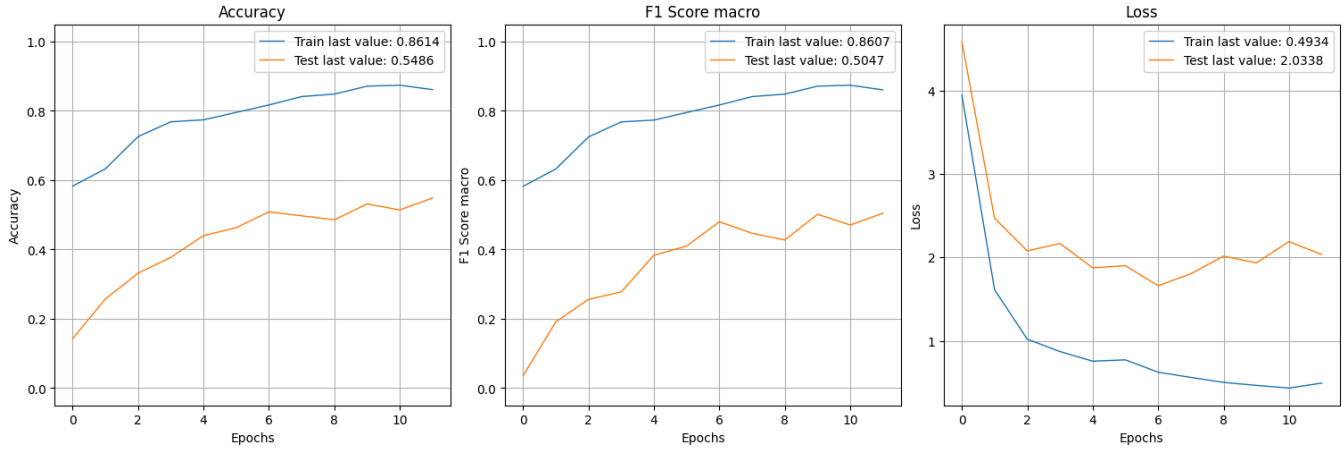
```
Num GPUs Available:  1
```

```
with tf.device('/gpu:0'):
    # Train model on original images & evaluate
    model_2_data, history_2 = train_and_evaluate_model('ResNet50 Integrated Data Augmentation', model_2, epochs=50,
```

```
                                                dataset_train=dataset_train, dataset_val=dataset_val,
                                                callbacks=callbacks_list_2)
    models_data_P8.append(model_2_data)
    histories['model_2'] = history_2
```

```
Epoch 1: val_loss improved from inf to 4.58384, saving model to ./model_2_best_weights.h5

Epoch 2: val_loss improved from 4.58384 to 2.46773, saving model to ./model_2_best_weights.h5

Epoch 3: val_loss improved from 2.46773 to 2.07727, saving model to ./model_2_best_weights.h5

Epoch 4: val_loss did not improve from 2.07727

Epoch 5: val_loss improved from 2.07727 to 1.87417, saving model to ./model_2_best_weights.h5

Epoch 6: val_loss did not improve from 1.87417

Epoch 7: val_loss improved from 1.87417 to 1.65936, saving model to ./model_2_best_weights.h5

Epoch 8: val_loss did not improve from 1.65936

Epoch 9: val_loss did not improve from 1.65936

Epoch 10: val_loss did not improve from 1.65936

Epoch 11: val_loss did not improve from 1.65936

Epoch 12: val_loss did not improve from 1.65936
Epoch 12: early stopping
```

```
show_history(histories['model_2'])
plot_history(histories['model_2'], path="save_path + '/' + ResNet50_integrated_augmentation.png")
plt.close()
```




```
Start coding or generate with AI.
```

2.5 - Modelisation without image augmentation

```
# Create model
with tf.device('/gpu:0'):
    model_3 = init_model()
```

```
model_3_save_path = "./model_3_best_weights.h5"
checkpoint_3 = ModelCheckpoint(model_3_save_path, monitor='val_loss', verbose=1,
                               save_best_only=True, mode='min')
es_3 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
callbacks_list_3 = [checkpoint_3, es_3]
```

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

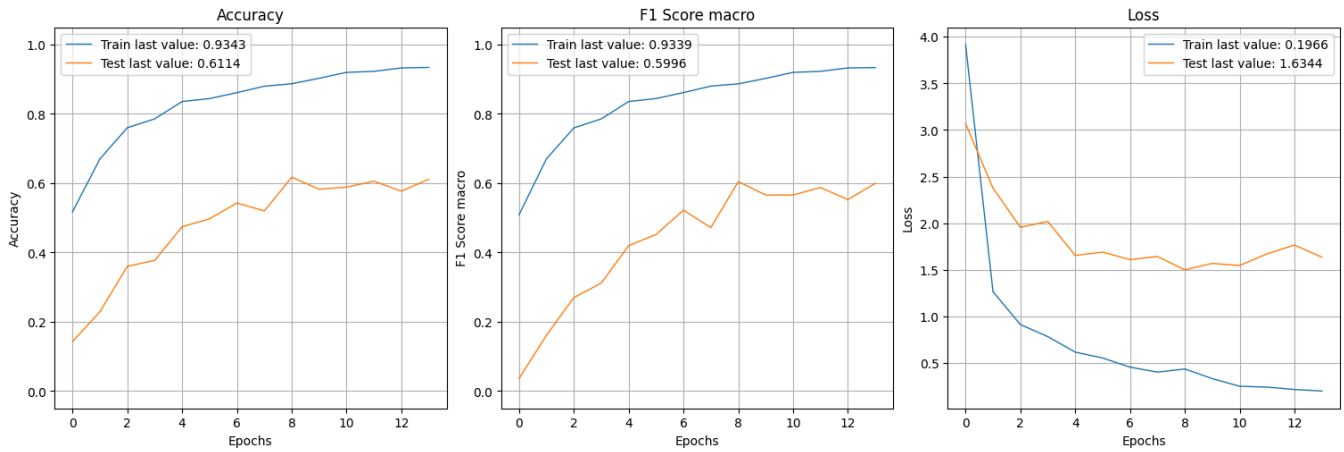
```
Num GPUs Available:  1
```

```
with tf.device('/gpu:0'):
    # Train model on original images & evaluate
    model_3_data, history_3 = train_and_evaluate_model('ResNet50 Original Data', model_3, epochs=50,
                                                dataset_train=dataset_train, dataset_val=dataset_val,
                                                callbacks=callbacks_list_3)
```

```
    models_data_P8.append(model_3_data)
    histories['model_3'] = history_3
```

```
Epoch 1: val_loss improved from inf to 3.06803, saving model to ./model_3_best_weights.h5

Epoch 2: val_loss improved from 3.06803 to 2.37519, saving model to ./model_3_best_weights.h5

Epoch 3: val_loss improved from 2.37519 to 1.95488, saving model to ./model_3_best_weights.h5

Epoch 4: val_loss did not improve from 1.95488

Epoch 5: val_loss improved from 1.95488 to 1.65243, saving model to ./model_3_best_weights.h5

Epoch 6: val_loss did not improve from 1.65243

Epoch 7: val_loss improved from 1.65243 to 1.60755, saving model to ./model_3_best_weights.h5

Epoch 8: val_loss did not improve from 1.60755

Epoch 9: val_loss improved from 1.60755 to 1.49789, saving model to ./model_3_best_weights.h5

Epoch 10: val_loss did not improve from 1.49789

Epoch 11: val_loss did not improve from 1.49789

Epoch 12: val_loss did not improve from 1.49789

Epoch 13: val_loss did not improve from 1.49789

Epoch 14: val_loss did not improve from 1.49789
Epoch 14: early stopping
```

```
show_history(histories['model_3'])
plot_history(histories['model_3'], path="save_path + '/' + ResNet50_original_data.png")
plt.close()
```




```
Start coding or generate with AI.
```

2.6 - Modelisation on Train + Val sets

```
# Create model
with tf.device('/gpu:0'):
    model_4 = init_model()
```

```
model_4_save_path = "./model_4_best_weights.h5"
checkpoint_4 = ModelCheckpoint(model_4_save_path, monitor='val_loss', verbose=1,
                               save_best_only=True, mode='min')
es_4 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
callbacks_list_4 = [checkpoint_4, es_4]
```

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

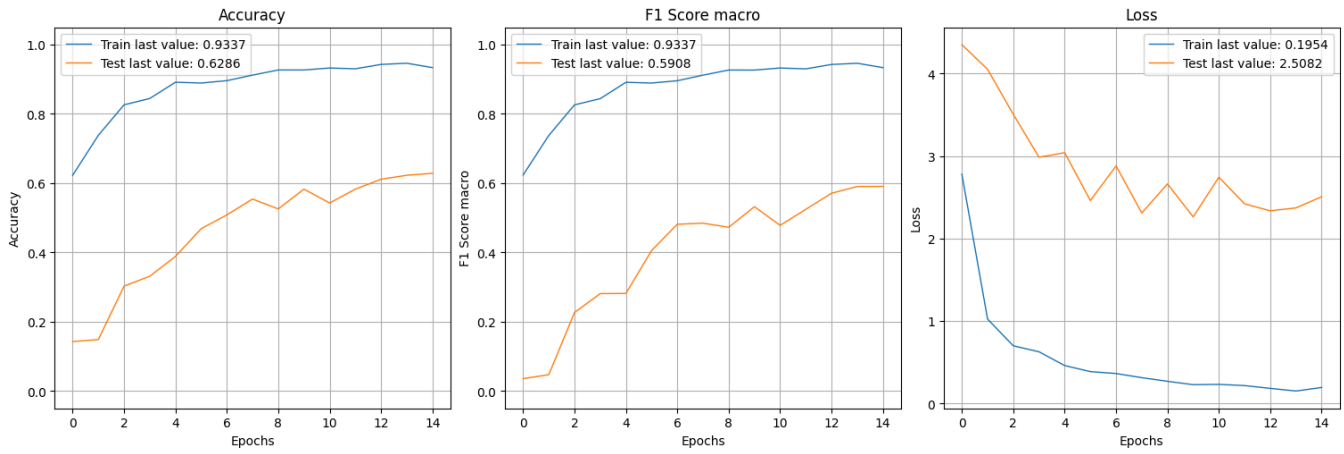
```
Num GPUs Available:  1
```

```
with tf.device('/gpu:0'):
    # Train model on original images & evaluate
    model_4_data, history_4 = train_and_evaluate_model('ResNet50 Original Data Train + Val', model_4, epochs=50,
```

```
                                                dataset_train=dataset_train_val, dataset_val=dataset_test,
                                                callbacks=callbacks_list_4)
    models_data_P8.append(model_4_data)
    histories['model_4'] = history_4
```

```
Epoch 1: val_loss improved from inf to 4.35136, saving model to ./model_4_best_weights.h5

Epoch 2: val_loss improved from 4.35136 to 4.05204, saving model to ./model_4_best_weights.h5

Epoch 3: val_loss improved from 4.05204 to 3.50656, saving model to ./model_4_best_weights.h5

Epoch 4: val_loss improved from 3.50656 to 2.98646, saving model to ./model_4_best_weights.h5

Epoch 5: val_loss did not improve from 2.98646

Epoch 6: val_loss improved from 2.98646 to 2.46139, saving model to ./model_4_best_weights.h5

Epoch 7: val_loss did not improve from 2.46139

Epoch 8: val_loss improved from 2.46139 to 2.31014, saving model to ./model_4_best_weights.h5

Epoch 9: val_loss did not improve from 2.31014

Epoch 10: val_loss improved from 2.31014 to 2.26304, saving model to ./model_4_best_weights.h5

Epoch 11: val_loss did not improve from 2.26304

Epoch 12: val_loss did not improve from 2.26304

Epoch 13: val_loss did not improve from 2.26304

Epoch 14: val_loss did not improve from 2.26304

Epoch 15: val_loss did not improve from 2.26304
Epoch 15: early stopping
```

```
show_history(histories['model_4'])
plot_history(histories['model_4'], path="save_path + '/' + ResNet50_original_train&val_data.png")
plt.close()
```




```
Start coding or generate with AI.
```

2.7 - Modelisation on train + val sets augmented

```
# Create model
with tf.device('/gpu:0'):
    model_5 = init_model()
```

```
model_5_save_path = "./model_5_best_weights.h5"
checkpoint_5 = ModelCheckpoint(model_5_save_path, monitor='val_loss', verbose=1,
                               save_best_only=True, mode='min')
es_5 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
callbacks_list_5 = [checkpoint_5, es_5]
```

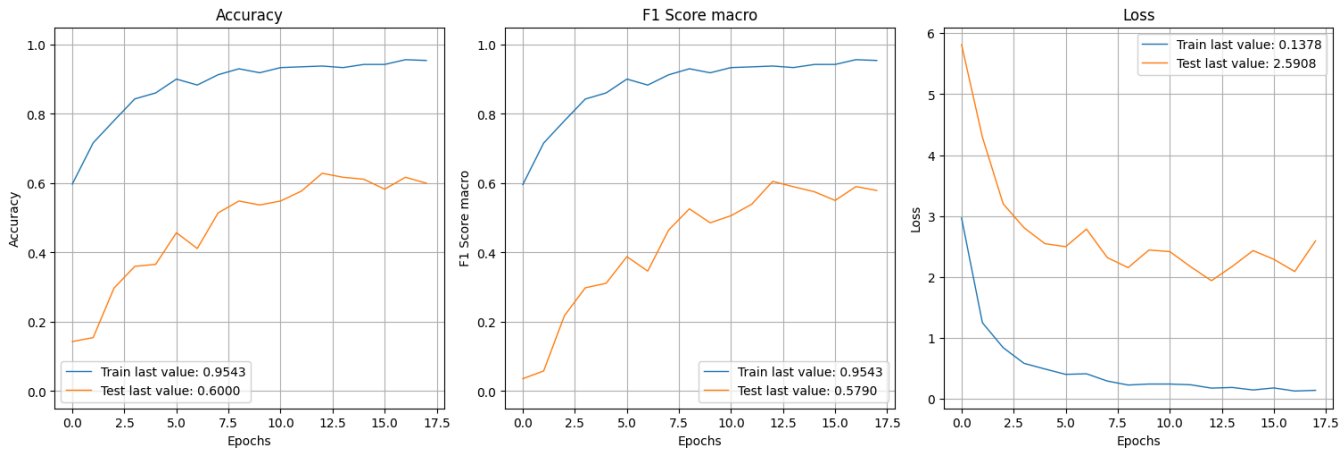
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

```
with tf.device('/gpu:0'):
    # Train model on original images & evaluate
    model_5_data, history_5 = train_and_evaluate_model('ResNet50 Augmented Data Train + Val', model_5, epochs=50,
                                                       dataset_train=dataset_train_val_aug, dataset_val=dataset_test,
                                                       callbacks=callbacks_list_5)
    models_data_P8.append(model_5_data)
    histories['model_5'] = history_5
```

```
Epoch 1: val_loss improved from inf to 5.81271, saving model to ./model_5_best_weights.h5

Epoch 2: val_loss improved from 5.81271 to 4.29450, saving model to ./model_5_best_weights.h5

Epoch 3: val_loss improved from 4.29450 to 3.19842, saving model to ./model_5_best_weights.h5

Epoch 4: val_loss improved from 3.19842 to 2.80715, saving model to ./model_5_best_weights.h5

Epoch 5: val_loss improved from 2.80715 to 2.54601, saving model to ./model_5_best_weights.h5

Epoch 6: val_loss improved from 2.54601 to 2.49634, saving model to ./model_5_best_weights.h5

Epoch 7: val_loss did not improve from 2.49634

Epoch 8: val_loss improved from 2.49634 to 2.31691, saving model to ./model_5_best_weights.h5

Epoch 9: val_loss improved from 2.31691 to 2.15221, saving model to ./model_5_best_weights.h5

Epoch 10: val_loss did not improve from 2.15221

Epoch 11: val_loss did not improve from 2.15221

Epoch 12: val_loss did not improve from 2.15221

Epoch 13: val_loss improved from 2.15221 to 1.93771, saving model to ./model_5_best_weights.h5

Epoch 14: val_loss did not improve from 1.93771

Epoch 15: val_loss did not improve from 1.93771

Epoch 16: val_loss did not improve from 1.93771

Epoch 17: val_loss did not improve from 1.93771

Epoch 18: val_loss did not improve from 1.93771
Epoch 18: early stopping
```

```
show_history(histories['model_5'])
plot_history(histories['model_5'], path="save_path + '/' + ResNet50_aug_train&val_data.png")
plt.close()
```




```
Start coding or generate with AI.
```

2.8 - Modelisation on train + val sets & their augmentations

```
# Create model
with tf.device('/gpu:0'):
    model_6 = init_model()
```

```
model_6_save_path = "./model_6_best_weights.h5"
checkpoint_6 = ModelCheckpoint(model_6_save_path, monitor='val_loss', verbose=1,
```

```
                            save_best_only=True, mode='min')
es_6 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
callbacks_list_6 = [checkpoint_6, es_6]
```

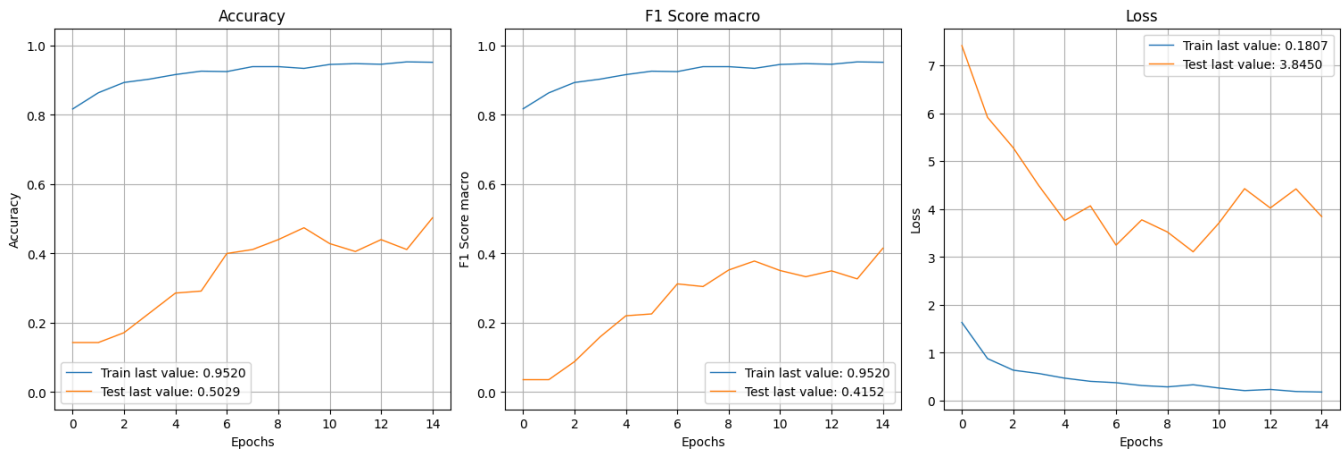
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

```
with tf.device('/gpu:0'):
    # Train model on original images & evaluate
    model_6_data, history_6 = train_and_evaluate_model('ResNet50 Original + Augmented Data Train & Val', model_6, epochs=50,
                                                       dataset_train=dataset_train_val_ALL, dataset_val=dataset_test,
                                                       callbacks=callbacks_list_6)
    models_data_P8.append(model_6_data)
    histories['model_6'] = history_6
```

```
Epoch 1: val_loss improved from inf to 7.41335, saving model to ./model_6_best_weights.h5

Epoch 2: val_loss improved from 7.41335 to 5.91449, saving model to ./model_6_best_weights.h5

Epoch 3: val_loss improved from 5.91449 to 5.28016, saving model to ./model_6_best_weights.h5

Epoch 4: val_loss improved from 5.28016 to 4.48607, saving model to ./model_6_best_weights.h5

Epoch 5: val_loss improved from 4.48607 to 3.76151, saving model to ./model_6_best_weights.h5

Epoch 6: val_loss did not improve from 3.76151

Epoch 7: val_loss improved from 3.76151 to 3.24852, saving model to ./model_6_best_weights.h5

Epoch 8: val_loss did not improve from 3.24852

Epoch 9: val_loss did not improve from 3.24852

Epoch 10: val_loss improved from 3.24852 to 3.10903, saving model to ./model_6_best_weights.h5

Epoch 11: val_loss did not improve from 3.10903

Epoch 12: val_loss did not improve from 3.10903

Epoch 13: val_loss did not improve from 3.10903

Epoch 14: val_loss did not improve from 3.10903

Epoch 15: val_loss did not improve from 3.10903
Epoch 15: early stopping
```

```
show_history(histories['model_6'])
plot_history(histories['model_6'], path="save_path + '/' + ResNet50_aug&orig_train&val_data.png")
plt.close()
```




```
Start coding or generate with AI.
```

2.9 - Store model comparison metrics

```
models_data_P8_df = pd.DataFrame(models_data_P8)
```

```
models_data_P8_df.to_csv(save_path + '/' + 'models_data_P8.csv', index=False)
```

```
models_data_P8_df
```

	model_name	epochs_run	early_stopping_epoch	fit_time	Train Accuracy (Hist)	Val Accuracy (Hist)	Train f1_macro (Hist)	Val f1_macro (Hist)	Val Accuracy (Sklearn)	Val Precision (Sklearn)	Val Recall (Sklearn)
0	ResNet50 External Data Augmentation	14	14	120.7904	0.9171	0.5314	0.9171	0.4973	0.5314	0.7499	0.5314
1	ResNet50 Integrated Data Augmentation	12	12	104.9553	0.8614	0.5486	0.8607	0.5047	0.5486	0.7575	0.5486
2	ResNet50 Original Data	14	14	66.0198	0.9343	0.6114	0.9339	0.5996	0.6114	0.7693	0.6114
3	ResNet50 Original Data Train + Val	15	15	139.7862	0.9337	0.6286	0.9337	0.5908	0.6286	0.7943	0.6286
4	ResNet50 Augmented Data Train + Val	18	18	132.1619	0.9543	0.6000	0.9543	0.5790	0.6000	0.7948	0.6000
5	ResNet50 Original + Augmented Data Train & Val	15	15	206.6450	0.9520	0.5029	0.9520	0.4152	0.5029	0.7661	0.5029

Next steps: Generate code with models_data_P8_df | View recommended plots | New interactive sheet

```
models_data = pd.read_csv(save_path + '/models_data.csv')
models_data
```

	model_name	epochs_run	early_stopping_epoch	fit_time	Train Accuracy (Hist)	Val Accuracy (Hist)	Train f1_macro (Hist)	Val f1_macro (Hist)	Val Accuracy (Sklearn)	Val Precision (Sklearn)	Val Recall (Sklearn)
0	ResNet50 External Data Augmentation	11	11	102.6436	0.9143	0.5429	0.9141	0.5002	0.5429	0.7546	0.5429
1	ResNet50 Integrated Data Augmentation	25	25	117.1815	0.9186	0.6229	0.9183	0.6100	0.6229	0.7802	0.6229
2	ResNet50 Original Data	12	11	58.6104	0.9071	0.5943	0.9069	0.5749	0.5943	0.7227	0.5943
3	ResNet50 Original Data Train + Val	17	17	101.3423	0.9371	0.7200	0.9370	0.7054	0.7200	0.8141	0.7200
4	ResNet50 Augmented Data Train + Val	11	11	55.4925	0.9280	0.6400	0.9281	0.6274	0.6400	0.8066	0.6400
5	ResNet50 Original + Augmented Data Train & Val	11	11	65.0602	0.9571	0.6171	0.9571	0.5847	0.6171	0.8000	0.6171

Next steps: Generate code with models_data | View recommended plots | New interactive sheet

```
models_data_clean = pd.read_csv(save_path + '/models_data_clean.csv')
models_data_clean
```

	model_name	epochs_run	early_stopping_epoch	fit_time	Train Accuracy (Hist)	Val Accuracy (Hist)	Train f1_macro (Hist)	Val f1_macro (Hist)	Val Accuracy (Sklearn)	Val Precision (Sklearn)	Val Recall (Sklearn)
0	ResNet50 External Data Augmentation	11	11	102.6436	0.9143	0.5429	0.9141	0.5002	0.5429	0.7546	0.5429
1	ResNet50 Integrated Data Augmentation	25	25	117.1815	0.9186	0.6229	0.9183	0.6100	0.6229	0.7802	0.6229
2	ResNet50 Original Data	12	11	58.6104	0.9071	0.5943	0.9069	0.5749	0.5943	0.7227	0.5943
3	ResNet50 Original Data Train + Val	17	17	101.3423	0.9371	0.7200	0.9370	0.7054	0.7200	0.8141	0.7200
4	ResNet50 Augmented Data Train + Val	11	11	55.4925	0.9280	0.6400	0.9281	0.6274	0.6400	0.8066	0.6400
5	ResNet50 Original + Augmented	11	11	65.0602	0.9571	0.6171	0.9571	0.5847	0.6171	0.8000	0.6171