logo.png

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

1 - Introduction

1.1 - Import packages & librairies

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

```
Requirement already satisfied: nlpaug in /usr/local/lib/python3.11/dist-packages (1.1.11)
Requirement already satisfied: plot_keras_history in /usr/local/lib/python3.11/dist-packages (1.1.39)
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)
Requirement already satisfied: sanitize-ml-labels>=1.0.48 in /usr/local/lib/python3.11/dist-packages (from plot_keras_history) (1.1
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.
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.
Requirement already satisfied: compress-json in /usr/local/lib/python3.11/dist-packages (from sanitize-ml-labels>=1.0.48->plot_kera
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.
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->plot_keras_history) (
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->plot_keras_history) (
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->nlp
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.
```

```
# 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]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
# 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/flipkart_images'
image_path_aug = '/content/drive/My Drive/Colab Notebooks/OCDS_P6/augmented_images'
save_path = '/content/drive/My Drive/Colab Notebooks/OCDS_P6'
image_save_path = save_path
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# 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 0x7d0ce2f58cd0>, 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)
```

```
Sat May 17 14:29:09 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   36C    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 = 13
```

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

1.2 - Define text functions

1.2.1 - Text augmentation

```
# Initialize augmenters once
synonym_aug = naw.SynonymAug(aug_src='wordnet', aug_p=0.1)  # Replace ~10% words with synonyms
random_insertion_aug = naw.RandomWordAug(action='insert', aug_p=0.1)  # Insert random words
contextual_aug = naw.ContextualWordEmbsAug(model_path='bert-base-uncased', action='insert', aug_p=0.1)
random_word_aug = naw.RandomWordAug(action='swap', aug_p=0.1)  # swap words instead of insert
random_word_aug = naw.RandomWordAug(action='delete', aug_p=0.1)  # delete words instead of insert

def augment_text(text):
    # Apply synonym replacement
    augmented_text = synonym_aug.augment(text)
    # Apply random insertion
    # augmented_text = random_insertion_aug.augment(augmented_text) # nlpaug bug - currently HS
    augmented_text = naw.RandomWordAug(action='swap', aug_p=0.1)
    augmented_text = contextual_aug.augment(augmented_text)
    return augmented_text

def augment_corpus(df, text_column, label_column):
    augmented_texts = []
    labels = []

    for idx, row in df.iterrows():
        original_text = row[text_column]
        label = row[label_column]

        # Augment the text
        augmented_text = augment_text(original_text)

        augmented_texts.append(augmented_text)
        labels.append(label)

    augmented_df = pd.DataFrame({
        'augmented_text': augmented_texts,
        'label': labels
    })
    return augmented_df
```

```
C:\Users\celin\DS Projets Python\OCDS-repos-all\OCDS-P6\venv39\lib\site-packages\huggingface_hub\file_download.py:896: FutureWarning
  warnings.warn(
```

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

1.2.2 - Text processing

```
def tokenize(word_string):
    tokenizer = RegexpTokenizer(r"[a-zA-Z]+")
    word_tokens = tokenizer.tokenize(word_string.lower())
    return word_tokens

def remove_stop_words(word_list):
    stop_words = list(set(stopwords.words('english')))
    filtered_words = [word for word in word_list if not word in stop_words]
    filtered_words_trim = [word for word in filtered_words if len(word) > 2]
    return filtered_words_trim

def lemmatize(word_list):
    lemmatizer = WordNetLemmatizer()
    lem_words = [lemmatizer.lemmatize(word) for word in word_list]
    return lem_words

def stem(word_list):
    stemmer = PorterStemmer()
    stem_words = [stemmer.stem(word) for word in word_list]
    return stem_words

def filter_words(word_list, words_to_trim):
    filtered_words = [word for word in word_list if not word in words_to_trim]
    # filtered_w2 = [w for w in filtered_w if len(w) > 2]
    return filtered_words
```

```
# Sub-function to remove specific words from a list of tokens
def trim_words_fct(tokens, words_to_trim):
    words_to_trim_set = set(words_to_trim)
    return [token for token in tokens if token not in words_to_trim_set]
```

```
# Text prep function for Bag-of-Words, Tf-idf & Word2Vec
def bag_of_words_transfo(word_string):
    word_tokens = tokenize(word_string)
    sw = remove_stop_words(word_tokens)
    trimmed = trim_words_fct(sw, words_to_trim)
    # lem_w = lemmatize(sw)
    transf_desc_text = ' '.join(trimmed)
    return transf_desc_text

# Text prep function for Bag-of-Words with lemmatization
def bag_of_words_transfo_lem(word_string):
    word_tokens = tokenize(word_string)
    sw = remove_stop_words(word_tokens)
    lem_w = lemmatize(sw)
    trimmed = trim_words_fct(lem_w, words_to_trim)
    transf_desc_text = ' '.join(trimmed)
    return transf_desc_text

# Text prep function for deep learning (BERT & USE)
def deep_lean_transfo(word_string) :
    word_tokens = tokenize(word_string)
    # sw = remove_stop_words(word_tokens)
    # trimmed = trim_words_fct(lem_w, words_to_trim)
    # lem_w = lemmatize(sw)
    transf_desc_text = ' '.join(word_tokens)
    return transf_desc_text
```

Start coding or generate with AI.

1.2.3 - Graphing functions

```
# recoloring function for wordclouds
def sunset_color_func(word, font_size, position, orientation, random_state=None, **kwargs):
    return random.choice(sunset_palette)
```

Start coding or generate with AI.

1.3 - Define text classification metrics

```
def calc_ari(features, perplexity=30, n_components=2):
    time1 = time.time()
    num_labels = len(np.unique(y_cat_num))
    random_state = 42

    # t-SNE embedding
    tsne = manifold.TSNE(
        n_components=n_components,
        perplexity=perplexity,
        n_iter=2000,
        init='random',
        learning_rate=200,
        random_state=random_state
    )
    X_tsne = tsne.fit_transform(features)

    # UMAP embedding
    reducer = umap.UMAP(
        n_components=n_components,
        n_neighbors=15,
        min_dist=0.1,
        random_state=random_state
    )
    X_umap = reducer.fit_transform(features)

    # Clustering on t-SNE embedding
    cls = cluster.KMeans(n_clusters=num_labels, n_init=100, random_state=random_state)
    cls.fit(X_tsne)

    ARI = np.round(metrics.adjusted_rand_score(y_cat_num, cls.labels_), 4)
    fit_time = np.round(time.time() - time1, 2)
    print(f"ARI: {ARI}, Time: {fit_time}s")

    return ARI, X_tsne, X_umap, cls.labels_, fit_time
```

```
def find_optimum_perplexity(features):
    perplexity_range = np.arange(5, 55, 5)
    divergence = []

    for p in perplexity_range:
        model = TSNE(n_components=2, init="pca", perplexity=p, random_state=42)
        _ = model.fit_transform(features)
        divergence.append(model.kl_divergence_)

    # Prepare data for seaborn
    data_plot = pd.DataFrame({'Perplexity': perplexity_range, 'KL Divergence': divergence})
    sns.set(rc={'figure.figsize':(6, 4), 'axes.facecolor':'white', 'figure.facecolor':'gainsboro'})

    sns.lineplot(x='Perplexity', y='KL Divergence', data=data_plot, marker="o", color=banana, linewidth=2)
    plt.title("t-SNE KL Divergence vs Perplexity")
    plt.xlabel("Perplexity Values")
    plt.ylabel("KL Divergence")
    plt.xticks(perplexity_range)

    # Generate timestamp and save plot
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"optimum_perplexity_{timestamp}.png"
    plt.savefig(save_path + '/' + filename)

    plt.show()

    # Find perplexity with minimum divergence
    min_index = np.argmin(divergence)
    optimum_perplexity = perplexity_range[min_index]

    print(f"Optimum perplexity: {optimum_perplexity} with divergence {divergence[min_index]:.4f}")
    return optimum_perplexity
```

```
def TSNE_visu_fct(X_tsne, y_cat_num, labels, ARI):
    """
    Visualize t-SNE embeddings colored by true categories and by cluster labels,
    using a custom color palette for both.

    Parameters:
    - X_tsne: np.ndarray of shape (n_samples, 2)
    - y_cat_num: array-like of true category numbers (ints)
    - labels: array-like of cluster numbers (ints)
    - ARI: float, Adjusted Rand Index
    - sunset_palette: list of color hex codes
    - products_trim: DataFrame with column 'real_clusters' (category names)
    """
    # Get unique category names in the order of their numeric encoding
    # (Assume y_cat_num is label-encoded so 0 maps to first unique, etc.)
    category_names = list(y_cat_num.unique())
    n_categories = len(category_names)
    n_clusters = len(np.unique(labels))

    # Ensure palette is long enough
    sunset_palette = ["#FFEB3B", "#FFDA44", "#FFC107", "#FFB300", "#FFA000", "#FF8F00", "#FF6F00", "#FF5722", "#FF3D00", "#FF2D00",
                  "#E53935", "#D32F2F", "#C62828", "#B71C1C", "#FF5252", "#FF1744", "#FF4081", "#F50057", "#D5006D", "#C51162"]
    palette_cat = sunset_palette * ((n_categories // len(sunset_palette)) + 1)
    palette_cluster = sunset_palette * ((n_clusters // len(sunset_palette)) + 1)

    fig = plt.figure(figsize=(15, 6))

    # Left: by true category
    ax1 = fig.add_subplot(121)
    scatter1 = ax1.scatter(
        X_tsne[:, 0], X_tsne[:, 1],
        c=y_cat_num,
        cmap=matplotlib.colors.ListedColormap(palette_cat[:n_categories]),
        s=40, alpha=0.85
    )
    # Custom legend for categories
    handles1 = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=palette_cat[i], markersize=10)
                for i in range(n_categories)]
    ax1.legend(handles1, category_names, loc="best", title="Category")
    ax1.set_title('t-SNE by True Categories')

    # Right: by cluster label
    ax2 = fig.add_subplot(122)
    scatter2 = ax2.scatter(
        X_tsne[:, 0], X_tsne[:, 1],
        c=labels,
        cmap=matplotlib.colors.ListedColormap(palette_cluster[:n_clusters]),
        s=40, alpha=0.85
    )
    # Custom legend for clusters
    cluster_labels = [f'Cluster {i}' for i in range(n_clusters)]
    handles2 = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=palette_cluster[i], markersize=10)
                for i in range(n_clusters)]
    ax2.legend(handles2, cluster_labels, loc="best", title="Cluster")
    ax2.set_title('t-SNE by Clusters')

    plt.suptitle(f"t-SNE Visualization | ARI: {ARI}", fontsize=14, fontweight='bold')
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
    print("ARI:", ARI)
```

```
def UMAP_visu_fct(X_umap, y_cat_num, labels, ARI):
    """
    Visualize UMAP embeddings colored by true categories and by cluster labels,
    using a custom color palette for both.

    Parameters:
    - X_umap: np.ndarray of shape (n_samples, 2), UMAP embedding
    - y_cat_num: array-like of true category numbers (ints)
    - labels: array-like of cluster numbers (ints)
    - ARI: float, Adjusted Rand Index
    """
    category_names = np.unique(y_cat_num)
    n_categories = len(category_names)
    n_clusters = len(np.unique(labels))

    sunset_palette = [
        "#FFEB3B", "#FFDA44", "#FFC107", "#FFB300", "#FFA000", "#FF8F00", "#FF6F00", "#FF5722", "#FF3D00", "#FF2D00",
        "#E53935", "#D32F2F", "#C62828", "#B71C1C", "#FF5252", "#FF1744", "#FF4081", "#F50057", "#D5006D", "#C51162"
    ]
    palette_cat = sunset_palette * ((n_categories // len(sunset_palette)) + 1)
    palette_cluster = sunset_palette * ((n_clusters // len(sunset_palette)) + 1)

    fig = plt.figure(figsize=(15, 6))

    # Left: by true category
    ax1 = fig.add_subplot(121)
    scatter1 = ax1.scatter(
        X_umap[:, 0], X_umap[:, 1],
        c=y_cat_num,
        cmap=matplotlib.colors.ListedColormap(palette_cat[:n_categories]),
        s=40, alpha=0.85
    )
    handles1 = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=palette_cat[i], markersize=10)
                for i in range(n_categories)]
    ax1.legend(handles1, category_names, loc="best", title="Category")
    ax1.set_title('UMAP by True Categories')

    # Right: by cluster label
    ax2 = fig.add_subplot(122)
    scatter2 = ax2.scatter(
        X_umap[:, 0], X_umap[:, 1],
        c=labels,
        cmap=matplotlib.colors.ListedColormap(palette_cluster[:n_clusters]),
        s=40, alpha=0.85
    )
    cluster_labels = [f'Cluster {i}' for i in range(n_clusters)]
    handles2 = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=palette_cluster[i], markersize=10)
                for i in range(n_clusters)]
    ax2.legend(handles2, cluster_labels, loc="best", title="Cluster")
    ax2.set_title('UMAP by Clusters')

    plt.suptitle(f"UMAP Visualization | ARI: {ARI}", fontsize=14, fontweight='bold')
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
    print("ARI:", ARI)
```

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

1.4 - Define image augmentation functions

1.4.1 - 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.
```

1.4.2 - 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.

1.4.3 - 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=
                    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.
```

1.5 - 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)
```

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

1.6 - 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

```
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
5    100
3    100
2    100
4    100
0    100
6    100
1    100
Name: count, dtype: int64
```

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

```
Validation class distribution:
real_clusters
6    25
2    25
1    25
3    25
4    25
0    25
5    25
Name: count, dtype: int64
```

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

```
Test class distribution:
real_clusters
2    25
5    25
0    25
4    25
3    25
1    25
6    25
Name: count, 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.
```

1.7 - Data augmentation

1.7.1 - Text data

```
# Augment only training data
augmented_train_df = augment_corpus(train_df, 'corpus_bow_lem', 'real_clusters')
print(augmented_train_df.shape)
augmented_train_df.head()
```

```
(700, 2)
```

	augmented_text	label
0	[weirdo sticker wrap design mint grey bottle p...	5
1	[bergner kadhai aluminum non stick get strong ...	5
2	[exotic india adi gautama vajrasattva showpiec...	3
3	[linkup dsl]	2
4	[aroma comfort polyester brown printed eyelet ...	4

```
augmented_train_df.to_parquet('augmented_text_train_df.gzip', compression='gzip')
```

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

1.7.2 - Images

1.7.2.1 - 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)
```

```
Found 700 images in training set to augment.
Processed 0/700 images
Processed 100/700 images
Processed 200/700 images
Processed 300/700 images
Processed 400/700 images
Processed 500/700 images
Processed 600/700 images
Augmented training images saved to: /content/drive/My Drive/Colab Notebooks/OCDS_P6/augmented_images
```

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

1.7.2.2 - Training + validation images

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

```
Found 875 images in training set to augment.
Processed 0/875 images
Processed 100/875 images
Processed 200/875 images
Processed 300/875 images
Processed 400/875 images
Processed 500/875 images
Processed 600/875 images
Processed 700/875 images
Processed 800/875 images
Augmented training images saved to: /content/drive/My Drive/Colab Notebooks/OCDS_P6/augmented_images
```

```
# 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
```

```
Found 875 images in set across 7 clusters.
Processed 0/875 images
Processed 25/875 images
Processed 50/875 images
Processed 75/875 images
Processed 100/875 images
Processed 125/875 images
Processed 150/875 images
Processed 175/875 images
Processed 200/875 images
Processed 225/875 images
Processed 250/875 images
Processed 275/875 images
Processed 300/875 images
Processed 325/875 images
Processed 350/875 images
Processed 375/875 images
Processed 400/875 images
Processed 425/875 images
Processed 450/875 images
Processed 475/875 images
Processed 500/875 images
Processed 525/875 images
Processed 550/875 images
Processed 575/875 images
Processed 600/875 images
Processed 625/875 images
Processed 650/875 images
Processed 675/875 images
Processed 700/875 images
Processed 725/875 images
Processed 750/875 images
Processed 775/875 images
Processed 800/875 images
Processed 825/875 images
Processed 850/875 images
Images saved to subfolders in: /content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets_aug
```

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

1.7.3 - Create grouped subfolders for ResNet50 re-training

```
# 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)
```

```
Merging files from '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets' and '/content/drive/My Drive/Colab Noteboo
Found subfolders in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets': ['5', '3', '2', '4', '0', '6', '1']
Processing subfolder '5'...
Found 100 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets/5'
Found 25 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/val_images_subsets/5'
Processing subfolder '3'...
Found 100 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets/3'
Found 25 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/val_images_subsets/3'
Processing subfolder '2'...
Found 100 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets/2'
Found 25 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/val_images_subsets/2'
Processing subfolder '4'...
```

```
Found 100 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets/4'
Found 25 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/val_images_subsets/4'
Processing subfolder '0'...
Found 100 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets/0'
Found 25 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/val_images_subsets/0'
Processing subfolder '6'...
Found 100 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets/6'
Found 25 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/val_images_subsets/6'
Processing subfolder '1'...
Found 100 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets/1'
Found 25 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/val_images_subsets/1'
Merging process completed.
```

```
# 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)
```

```
Merging files from '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets' and '/content/drive/My Drive/Colab Not
Found subfolders in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets': ['5', '3', '2', '4', '0', '6', '1']
Processing subfolder '5'...
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets/5'
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets_aug/5'
Processing subfolder '3'...
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets/3'
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets_aug/3'
Processing subfolder '2'...
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets/2'
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets_aug/2'
Processing subfolder '4'...
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets/4'
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets_aug/4'
Processing subfolder '0'...
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets/0'
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets_aug/0'
Processing subfolder '6'...
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets/6'
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets_aug/6'
Processing subfolder '1'...
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets/1'
Found 125 files in '/content/drive/My Drive/Colab Notebooks/OCDS_P6/train_val_images_subsets_aug/1'
Merging process completed.
```

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

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

1.8 - Generate features

1.8.1 - Text data - using Tf-idf

1.8.1.1 - Training text data

1.8.1.1-a - With data augmentation

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

```
ctf = TfidfVectorizer(stop_words='english', max_df=0.95, min_df=1)
```

```
augmented_train_df['augmented_text_str'] = augmented_train_df['augmented_text'].apply(lambda x: ' '.join(x) if isinstance(x, list) else
ctf_fit = ctf.fit(augmented_train_df['augmented_text_str'])
```

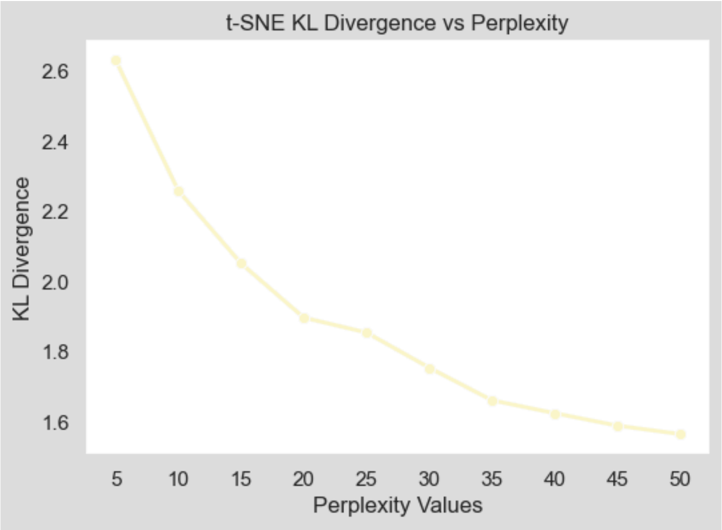
```
ctf_transform = ctf.transform(augmented_train_df['augmented_text_str'])
```

```
voc_ctf = ctf.get_feature_names_out()
voc_ctf_df = pd.DataFrame(ctf_transform.todense(), columns=voc_ctf)
print(voc_ctf_df.shape)
voc_ctf_df.head()
```

```
(700, 3559)
```

	60	79	aapno	aari	ability	abject	abkl	abode	abroad	absorbency	absorbent	absorber	absorbing	abstract	abstra
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	(
1	0.0000	0.0000	0.0000	0.0000	0.1049	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	(
2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	(
3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	(
4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	(

```
opt_perplexity_ctf = find_optimum_perplexity(voc_ctf_df)
```



```
Optimum perplexity: 50 with divergence 1.5653
```

```
y_cat_num = train_df['real_clusters']
# ARI_ctf, X_tsne_ctf, labels_ctf, fit_time_ctf  = calc_ari(ctf_transform, opt_perplexity_ctf) # ARI: 0.4532, Time: 6.02s for both rando
ARI_ctf, X_tsne_ctf, X_umap_ctf, labels_ctf, fit_time_ctf  = calc_ari(voc_ctf_df, opt_perplexity_ctf)
# model_results.append({'Model': 'Tf-idf', 'ARI': ARI_ctf, 'Fitting Time (s)': fit_time_ctf})
print('ARI :', ARI_ctf, 'Fitting Time (s):', fit_time_ctf)
```

```
ARI: 0.3963, Time: 14.03s
ARI : 0.3963 Fitting Time (s): 14.03
```

```
voc_ctf_df.to_csv(save_path + '/'+ 'text_features_train_aug.csv')
```

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

1.8.1.1-b - Without data augmentation

```
ctf_no_aug = TfidfVectorizer(stop_words='english', max_df=0.95, min_df=1)
```

```
train_df['text_str'] = train_df['corpus_bow_lem'].apply(lambda x: ' '.join(x) if isinstance(x, list) else str(x))
ctf_fit_no_aug = ctf_no_aug.fit(train_df['text_str'])
```

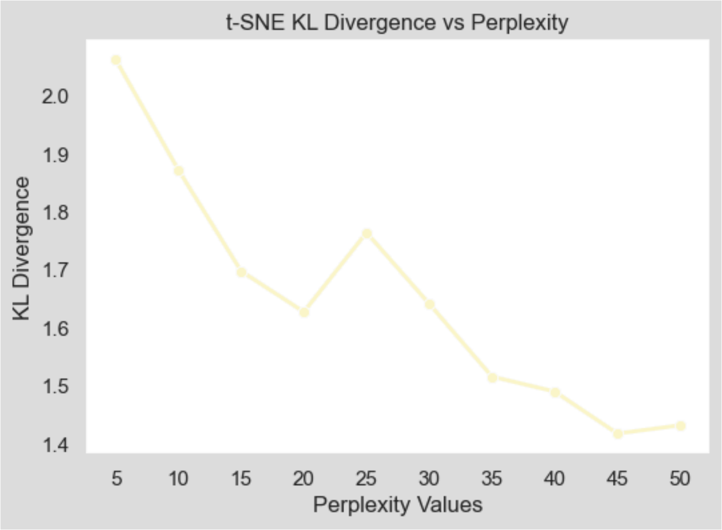
```
ctf_transform_no_aug = ctf_no_aug.transform(train_df['text_str'])
```

```
voc_ctf_no_aug = ctf_no_aug.get_feature_names_out()
voc_ctf_df_no_aug = pd.DataFrame(ctf_transform_no_aug.todense(), columns=voc_ctf_no_aug)
print(voc_ctf_df_no_aug.shape)
voc_ctf_df_no_aug.head()
```

```
(700, 2631)
```

	aapno	aari	ability	abkl	abode	abroad	absorbency	absorbent	absorbing	abstract	accent	access	accessory	accident	ac
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
1	0.0000	0.0000	0.2389	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

```
opt_perplexity_ctf_no_aug = find_optimum_perplexity(voc_ctf_df_no_aug)
```



```
Optimum perplexity: 45 with divergence 1.4182
```

```
y_cat_num = train_df['real_clusters']
# ARI_ctf, X_tsne_ctf, labels_ctf, fit_time_ctf  = calc_ari(ctf_transform, opt_perplexity_ctf) # ARI: 0.4532, Time: 6.02s for both rando
ARI_ctf_no_aug, X_tsne_ctf_no_aug, X_umap_ctf_no_aug, labels_ctf_no_aug, fit_time_ctf_no_aug  = calc_ari(voc_ctf_df_no_aug, opt_perplexi
# model_results.append({'Model': 'Tf-idf', 'ARI': ARI_ctf, 'Fitting Time (s)': fit_time_ctf})
print('ARI :', ARI_ctf_no_aug, 'Fitting Time (s):', fit_time_ctf_no_aug)
```

```
ARI: 0.3777, Time: 5.93s
ARI : 0.3777 Fitting Time (s): 5.93
```

```
voc_ctf_df_no_aug.to_csv(save_path + '/'+ 'text_features_train_no_aug.csv')
```

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

1.8.1.2 - Validation text data

1.8.1.2-a - With data augmentation

```
# use vectorizer fitted on augmented training data to ensure same number of features
ctf_val_transform = ctf_fit.transform(val_df['corpus_bow_lem'])
voc_ctf_val = ctf_fit.get_feature_names_out() # Use ctf_fit, not just ctf
voc_ctf_val_df = pd.DataFrame(ctf_val_transform.todense(), columns=voc_ctf_val, index=val_df.index)
print(voc_ctf_val_df.shape)
voc_ctf_val_df.head()
```

```
(175, 3559)
```

	60	79	aapno	aari	ability	abject	abkl	abode	abroad	absorbency	absorbent	absorber	absorbing	abstract	abst
502	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
343	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
688	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
263	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
756	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

```
voc_ctf_val_df.to_csv(save_path + '/'+'text_features_val_aug.csv')
```

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

1.8.1.2-b - Without data augmentation

```
# use vectorizer fitted on training data to ensure same number of features
ctf_val_transform_no_aug = ctf_fit_no_aug.transform(val_df['corpus_bow_lem'])
voc_ctf_val_no_aug = ctf_fit_no_aug.get_feature_names_out() # Use ctf_fit, not just ctf
voc_ctf_val_df_no_aug = pd.DataFrame(ctf_val_transform_no_aug.todense(), columns=voc_ctf_val_no_aug, index=val_df.index)
print(voc_ctf_val_df_no_aug.shape)
voc_ctf_val_df_no_aug.head()
```

```
(175, 2631)
```

	aapno	aari	ability	abkl	abode	abroad	absorbency	absorbent	absorbing	abstract	accent	access	accessory	accident
502	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
343	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
688	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
263	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
756	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

```
voc_ctf_val_df_no_aug.to_csv(save_path + '/'+'text_features_val_no_aug.csv')
```

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

1.8.1.3 - Test text data

1.8.1.3-a - With data augmentation

```
# use vectorizer fitted on augmented training data to ensure same number of features
ctf_test_transform = ctf_fit.transform(test_df['corpus_bow_lem'])
voc_ctf_test = ctf_fit.get_feature_names_out()
voc_ctf_test_df = pd.DataFrame(ctf_test_transform.todense(), columns=voc_ctf_test, index=test_df.index)
print(voc_ctf_test_df.shape)
voc_ctf_test_df.head()
```

```
(175, 3559)
```

	60	79	aapno	aari	ability	abject	abkl	abode	abroad	absorbency	absorbent	absorber	absorbing	abstract	abs
1028	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
628	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
465	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
1022	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
650	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

```
voc_ctf_test_df.to_csv(save_path + '/'+'text_features_test_aug.csv')
```

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

1.8.1.3-a - Without data augmentation

```
# use vectorizer fitted on augmented training data to ensure same number of features
ctf_test_transform_no_aug = ctf_fit_no_aug.transform(test_df['corpus_bow_lem'])
voc_ctf_test_no_aug = ctf_fit_no_aug.get_feature_names_out()
voc_ctf_test_df_no_aug = pd.DataFrame(ctf_test_transform_no_aug.todense(), columns=voc_ctf_test_no_aug, index=test_df.index)
print(voc_ctf_test_df_no_aug.shape)
voc_ctf_test_df_no_aug.head()
```

```
(175, 2631)
```

	aapno	aari	ability	abkl	abode	abroad	absorbency	absorbent	absorbing	abstract	accent	access	accessory	accident
1028	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
628	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
465	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
1022	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
650	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

```
voc_ctf_test_df_no_aug.to_csv(save_path + '/'+'text_features_test_no_aug.csv')
```

Start coding or generate with AI.

1.8.2 - Images - using ResNet50

1.8.2.1 - Training image data

1.8.2.1-a - Training image data with augmentation

```
# Generate subsets only once - remove folder before re-generating
# save_image_subsets(image_path_aug, image_save_path + '/' + 'train_images_subsets_aug', val_test_df=train_df)
```

```
Found 700 images in set across 7 clusters.
Processed 450/700 images
Processed 225/700 images
Processed 175/700 images
Processed 800/700 images
Processed 1025/700 images
Processed 775/700 images
Processed 350/700 images
Processed 975/700 images
Processed 875/700 images
Processed 375/700 images
Processed 600/700 images
Processed 425/700 images
Processed 925/700 images
Processed 125/700 images
Processed 250/700 images
Processed 200/700 images
Processed 550/700 images
Processed 500/700 images
Processed 1000/700 images
Processed 950/700 images
Processed 100/700 images
Processed 50/700 images
Processed 750/700 images
Processed 700/700 images
Processed 900/700 images
Processed 725/700 images
Processed 825/700 images
Processed 75/700 images
Images saved to subfolders in: /content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets_aug
```

```
list_photos = [file for file in listdir(image_path_aug)]
print(len(list_photos))
```

```
700
```

```
# Load ResNet50 pretrained on ImageNet, exclude top classification layer, use global average pooling
base_model_50 = ResNet50(weights='imagenet', include_top=False, pooling='avg')

# The model outputs a 2048-dimensional feature vector per image
model_50 = base_model_50

# print(model_50.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kerne
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
temps_3 = time.time()

images_features_50 = []
```

```
i=0
for image_num in range(len(list_photos)) :
    if i%100 == 0 :
      print(f'progress : {i / len(list_photos) * 100:.2f} %')
    i +=1
    image = load_img(image_path_aug+'/'+list_photos[image_num], target_size=(224, 224))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)

    image = preprocess_input(image)
    images_features_50.append(model_50.predict(image, verbose=0)[0])

    duration_3 = time.time() - temps_3
print("Features creation time with RN50 : ", "%15.2f" % duration_3, "secs")

images_features_50 = np.asarray(images_features_50)
images_features_50.shape
```

```
progress : 0.00 %
progress : 14.29 %
progress : 28.57 %
progress : 42.86 %
progress : 57.14 %
progress : 71.43 %
progress : 85.71 %
Features creation time with RN50 :             66.05 secs
(700, 2048)
```

```
images_features_50 = pd.DataFrame(images_features_50)
images_features_50.to_csv(save_path +'/'+'image_features_train_aug.csv', index=False)
```

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

1.8.2.1-b - Training image data without augmentation

```
# Generate subsets only once - remove folder before re-generating
# save_image_subsets(image_path, image_save_path + '/' + 'train_images_subsets', val_test_df=train_df)
```

```
Found 700 images in set across 7 clusters.
Processed 450/700 images
Processed 225/700 images
Processed 175/700 images
Processed 800/700 images
Processed 1025/700 images
Processed 775/700 images
Processed 350/700 images
Processed 975/700 images
Processed 875/700 images
Processed 375/700 images
Processed 600/700 images
Processed 425/700 images
Processed 925/700 images
Processed 125/700 images
Processed 250/700 images
Processed 200/700 images
Processed 550/700 images
Processed 500/700 images
Processed 1000/700 images
Processed 950/700 images
Processed 100/700 images
Processed 50/700 images
Processed 750/700 images
Processed 700/700 images
Processed 900/700 images
Processed 725/700 images
Processed 825/700 images
Processed 75/700 images
Images saved to subfolders in: /content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images_subsets
```

```
train_df = pd.read_parquet(save_path + '/'+ 'train_df.gzip')
save_image_set(image_path, image_save_path + '/' + 'train_images', val_test_df=train_df)
```

```
Found 700 images in set.
Processed 0/700 images
Processed 25/700 images
Processed 50/700 images
Processed 75/700 images
Processed 100/700 images
Processed 125/700 images
Processed 150/700 images
Processed 175/700 images
Processed 200/700 images
Processed 225/700 images
Processed 250/700 images
Processed 275/700 images
Processed 300/700 images
Processed 325/700 images
Processed 350/700 images
Processed 375/700 images
Processed 400/700 images
Processed 425/700 images
Processed 450/700 images
Processed 475/700 images
Processed 500/700 images
Processed 525/700 images
Processed 550/700 images
Processed 575/700 images
Processed 600/700 images
Processed 625/700 images
Processed 650/700 images
Processed 675/700 images
Images saved to: /content/drive/My Drive/Colab Notebooks/OCDS_P6/train_images
```

```
train_list_photos = [file for file in listdir(image_save_path + '/' + 'train_images')]
print(len(train_list_photos))
```

```
700
```

```
# Load ResNet50 pretrained on ImageNet, exclude top classification layer, use global average pooling
base_model_50_train = ResNet50(weights='imagenet', include_top=False, pooling='avg')

# The model outputs a 2048-dimensional feature vector per image
model_50_train = base_model_50_train

# print(model_50_train.summary())
```

```
temps_9 = time.time()

images_features_50_train = []
i=0
for image_num in range(len(train_list_photos)) :
    if i%25 == 0 :
      print(f'progress : {i / len(train_list_photos) * 100:.2f} %')
    i +=1
    image_train = load_img(image_save_path + '/' + 'train_images'+'/'+train_list_photos[image_num], target_size=(224, 224))
    image_train = img_to_array(image_train)
    image_train = np.expand_dims(image_train, axis=0)

    image_train = preprocess_input(image_train)
    images_features_50_train.append(model_50_train.predict(image_train, verbose=0)[0])

    duration_9 = time.time() - temps_9
print("train features creation time with RN50 : ", "%15.2f" % duration_9, "secs")

images_features_50_train = np.asarray(images_features_50_train)
images_features_50_train.shape
```

```
progress : 0.00 %
progress : 3.57 %
progress : 7.14 %
progress : 10.71 %
progress : 14.29 %
progress : 17.86 %
progress : 21.43 %
progress : 25.00 %
progress : 28.57 %
progress : 32.14 %
progress : 35.71 %
progress : 39.29 %
progress : 42.86 %
progress : 46.43 %
progress : 50.00 %
progress : 53.57 %
progress : 57.14 %
progress : 60.71 %
progress : 64.29 %
progress : 67.86 %
progress : 71.43 %
progress : 75.00 %
progress : 78.57 %
progress : 82.14 %
progress : 85.71 %
progress : 89.29 %
progress : 92.86 %
progress : 96.43 %
train features creation time with RN50 :            66.63 secs
(700, 2048)
```

```
image_features_train = pd.DataFrame(images_features_50_train)
image_features_train.to_csv(save_path +'/'+'image_features_train_no_aug.csv', index=False)
```

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

1.8.2.2 - Validation image data

```
# Generate subsets only once - remove folder before re-generating
# save_image_subsets(save_path +'/' + 'val_images', save_path + '/' + 'val_images_subsets', val_test_df=val_df)
```

```
Found 175 images in set across 7 clusters.
Processed 275/175 images
Processed 475/175 images
Processed 575/175 images
Processed 400/175 images
Processed 300/175 images
Processed 25/175 images
Processed 675/175 images
Processed 850/175 images
Images saved to subfolders in: /content/drive/My Drive/Colab Notebooks/OCDS_P6/val_images_subsets
```

```
# val_df = pd.read_parquet(save_path + '/'+'val_df.gzip')
save_image_set(image_path, image_save_path + '/' + 'val_images', val_test_df=val_df)
```

```
Found 175 images in set.
Processed 0/175 images
Processed 25/175 images
Processed 50/175 images
Processed 75/175 images
Processed 100/175 images
Processed 125/175 images
Processed 150/175 images
Images saved to: /content/drive/My Drive/Colab Notebooks/OCDS_P6/val_images
```

```
val_list_photos = [file for file in listdir(image_save_path + '/' + 'val_images')]
print(len(val_list_photos))
```

```
175
```

```
# Load ResNet50 pretrained on ImageNet, exclude top classification layer, use global average pooling
base_model_50_val = ResNet50(weights='imagenet', include_top=False, pooling='avg')

# The model outputs a 2048-dimensional feature vector per image
model_50_val = base_model_50_val

# print(model_50.summary())
```

```
temps_5 = time.time()

images_features_50_val = []
i=0
for image_num in range(len(val_list_photos)) :
    if i%25 == 0 :
      print(f'progress : {i / len(val_list_photos) * 100:.2f} %')
    i +=1
    image_val = load_img(image_save_path + '/' + 'val_images'+'/'+val_list_photos[image_num], target_size=(224, 224))
    image_val = img_to_array(image_val)
    image_val = np.expand_dims(image_val, axis=0)

    image_val = preprocess_input(image_val)
    images_features_50_val.append(model_50_val.predict(image_val, verbose=0)[0])

    duration_5 = time.time() - temps_5
print("Val features creation time with RN50 : ", "%15.2f" % duration_5, "secs")

images_features_50_val = np.asarray(images_features_50_val)
images_features_50_val.shape
```

```
progress : 0.00 %
progress : 14.29 %
progress : 28.57 %
progress : 42.86 %
progress : 57.14 %
progress : 71.43 %
progress : 85.71 %
Val features creation time with RN50 :            17.95 secs
(175, 2048)
```

```
image_features_val = pd.DataFrame(images_features_50_val)
image_features_val.to_csv(save_path +'/'+'image_features_val.csv', index=False)
```

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

1.8.2.3 - Test image data

```
# Generate subsets only once - remove folder before re-generating
# save_image_subsets(save_path +'/' + 'test_images', save_path + '/' + 'test_images_subsets', val_test_df=test_df)
```

```
Found 175 images in set across 7 clusters.
Processed 650/175 images
Processed 150/175 images
Processed 625/175 images
Processed 525/175 images
Processed 325/175 images
Processed 0/175 images
Images saved to subfolders in: /content/drive/My Drive/Colab Notebooks/OCDS_P6/test_images_subsets
```

```
test_df = pd.read_parquet(save_path + '/'+'test_df.gzip')
save_image_set(image_path, image_save_path + '/' + 'test_images', val_test_df=test_df)
```

```
Found 175 images in set.
Processed 0/175 images
Processed 25/175 images
Processed 50/175 images
Processed 75/175 images
Processed 100/175 images
Processed 125/175 images
Processed 150/175 images
Images saved to: /content/drive/My Drive/Colab Notebooks/OCDS_P6/test_images
```

```
test_list_photos = [file for file in listdir(image_save_path + '/' + 'test_images')]
print(len(test_list_photos))
```

```
175
```

```
# Load ResNet50 pretrained on ImageNet, exclude top classification layer, use global average pooling
base_model_50_test = ResNet50(weights='imagenet', include_top=False, pooling='avg')

# The model outputs a 2048-dimensional feature vector per image
model_50_test = base_model_50_test

# print(model_50.summary())
```

```
temps_7 = time.time()

images_features_50_test = []
i=0
for image_num in range(len(test_list_photos)) :
    if i%25 == 0 :
      print(f'progress : {i / len(test_list_photos) * 100:.2f} %')
    i +=1
    image_test = load_img(image_save_path + '/' + 'test_images'+'/'+test_list_photos[image_num], target_size=(224, 224))
    image_test = img_to_array(image_test)
    image_test = np.expand_dims(image_test, axis=0)

    image_test = preprocess_input(image_test)
    images_features_50_test.append(model_50_test.predict(image_test, verbose=0)[0])

    duration_7 = time.time() - temps_7
print("Test features creation time with RN50 : ", "%15.2f" % duration_7, "secs")

images_features_50_test = np.asarray(images_features_50_test)
images_features_50_test.shape
```

```
progress : 0.00 %
progress : 14.29 %
progress : 28.57 %
progress : 42.86 %
progress : 57.14 %
progress : 71.43 %
progress : 85.71 %
Test features creation time with RN50 :             17.95 secs
(175, 2048)
```

```
image_features_test = pd.DataFrame(images_features_50_test)
image_features_test.to_csv(save_path +'/'+'image_features_test.csv', index=False)
```

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

1.8.3 - Merge text and image features

1.8.3.1 - Training data

1.8.3.1-a - With data augmentation

```
train_df = pd.read_parquet(save_path + '/'+'train_df.gzip')
y_train = train_df['real_clusters']
```

```
text_features_aug = pd.read_csv(save_path +'/' + 'text_features_train_aug.csv').drop(columns=['Unnamed: 0'])
print(text_features_aug.shape)
text_features_aug.head(1)
```

```
(700, 3559)
      60      79   aapno    aari  ability  abject    abkl   abode  abroad  absorbency  absorbent  absorber  absorbing  abstract  abstra
0 0.0000  0.0000  0.0000  0.0000   0.0000  0.0000  0.0000  0.0000  0.0000      0.0000     0.0000    0.0000     0.0000    0.0000
```

```
image_features_aug = pd.read_csv(save_path +'/'+'image_features_train_aug.csv')
print(image_features_aug.shape)
image_features_aug.head(1)
```

```
(700, 2048)
       0       1       2       3       4       5       6       7       8       9      10      11      12      13      14      15      16
0 0.2641  0.1605  0.2063  1.7017  0.0418  0.4177  0.0745  0.5710  0.7634  0.0897  0.9480  0.2685  0.0173  0.1969  0.6553  1.1963  0.0063  0.4
```

```
text_features_aug = text_features_aug.reset_index(drop=True)
image_features_aug = image_features_aug.reset_index(drop=True)

X_train_aug = pd.concat([text_features_aug, image_features_aug], axis=1)
print(X_train_aug.shape)
X_train_aug.to_csv(save_path +'/' + 'X_train_aug.csv', index=False)
X_train_aug.head(1)
```

```
(700, 5607)
      60      79   aapno    aari  ability  abject    abkl   abode  abroad  absorbency  absorbent  absorber  absorbing  abstract  abstra
0 0.0000  0.0000  0.0000  0.0000   0.0000  0.0000  0.0000  0.0000  0.0000      0.0000     0.0000    0.0000     0.0000    0.0000
```

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

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

1.8.3.1-b - Without data augmentation

```
train_df = pd.read_parquet(save_path + '/'+'train_df.gzip')
y_train = train_df['real_clusters']
```

```
text_features_no_aug = pd.read_csv(save_path +'/' + 'text_features_train_no_aug.csv').drop(columns=['Unnamed: 0'])
print(text_features_no_aug.shape)
text_features_no_aug.head(1)
```

```
(700, 2631)
   aapno    aari  ability    abkl   abode  abroad  absorbency  absorbent  absorbing  abstract  accent  access  accessory  accident  ac
0 0.0000  0.0000   0.0000  0.0000  0.0000  0.0000      0.0000     0.0000     0.0000    0.0000  0.0000  0.0000     0.0000    0.0000
```

```
image_features_train_no_aug = pd.read_csv(save_path +'/'+'image_features_train_no_aug.csv')
print(image_features_train_no_aug.shape)
image_features_train_no_aug.head(1)
```

```
(700, 2048)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	0.1864	2.0764	0.0318	0.0000	0.0106	0.0352	0.1957	0.0066	0.0285	0.5973	0.0000	0.0100	0.5650	0.7558	0.1504	0.0000	0.0000	0.2

```
text_features_no_aug = text_features_no_aug.reset_index(drop=True)
image_features_no_aug = image_features_train_no_aug.reset_index(drop=True)

X_train_no_aug = pd.concat([text_features_no_aug, image_features_no_aug], axis=1)
print(X_train_no_aug.shape)
X_train_no_aug.to_csv(save_path +'/' + 'X_train_no_aug.csv', index=False)
X_train_no_aug.head(1)
```

```
(700, 4679)
```

	aapno	aari	ability	abkl	abode	abroad	absorbency	absorbent	absorbing	abstract	accent	access	accessory	accident	ac
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

```
train_df = pd.read_parquet(save_path + '/'+'train_df.gzip')
y_train = train_df['real_clusters']
```

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

1.8.3.2 - Validation data

1.8.3.2-a - With data augmentation

```
val_df=pd.read_parquet(save_path + '/'+'val_df.gzip')
y_val = val_df['real_clusters']
```

```
val_text_features_aug = pd.read_csv(save_path +'/' + 'text_features_val_aug.csv').drop(columns=['Unnamed: 0'])
print(val_text_features_aug.shape)
val_text_features_aug.head(1)
```

```
(175, 3559)
```

	60	79	aapno	aari	ability	abject	abkl	abode	abroad	absorbency	absorbent	absorber	absorbing	abstract	abstra
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

```
image_features_val_aug = pd.read_csv(save_path +'/' + 'image_features_val.csv')
print(image_features_val.shape)
image_features_val.head(1)
```

```
(175, 2048)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	0.0961	0.3338	0.4152	0.0780	0.4182	0.0214	0.0364	0.1787	0.3924	0.2206	0.0061	0.1573	0.4481	0.5972	0.1128	0.6123	0.0000	0.1

```
X_val_aug = pd.concat([val_text_features_aug, image_features_val], axis=1)
print(X_val_aug.shape)
X_val_aug.to_csv(save_path +'/' + 'X_val_aug.csv', index=False)
```

```
(175, 5607)
```

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

1.8.3.2-b - Without data augmentation

```
val_text_features_no_aug = pd.read_csv(save_path +'/' + 'text_features_val_no_aug.csv').drop(columns=['Unnamed: 0'])
print(val_text_features_no_aug.shape)
val_text_features_no_aug.head(1)
```

```
(175, 2631)
```

	aapno	aari	ability	abkl	abode	abroad	absorbency	absorbent	absorbing	abstract	accent	access	accessory	accident	ac
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

```
image_features_val_aug = pd.read_csv(save_path +'/' + 'image_features_val.csv')
print(image_features_val.shape)
image_features_val.head(1)
```

```
(175, 2048)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	0.0961	0.3338	0.4152	0.0780	0.4182	0.0214	0.0364	0.1787	0.3924	0.2206	0.0061	0.1573	0.4481	0.5972	0.1128	0.6123	0.0000	0.1

```
X_val_no_aug = pd.concat([val_text_features_no_aug, image_features_val], axis=1)
print(X_val_no_aug.shape)
X_val_no_aug.to_csv(save_path +'/' + 'X_val_no_aug.csv', index=False)
```

```
(175, 4679)
```

Start coding or generate with AI.

1.8.3.3 - Test data

1.8.3.3-a - With data augmentation

```
test_df=pd.read_parquet(save_path + '/'+'test_df.gzip')
y_test = test_df['real_clusters']
```

```
test_text_features_aug = pd.read_csv(save_path +'/' + 'text_features_test_aug.csv').drop(columns=['Unnamed: 0'])
print(test_text_features_aug.shape)
test_text_features_aug.head(1)
```

```
(175, 3559)
```

	60	79	aapno	aari	ability	abject	abkl	abode	abroad	absorbency	absorbent	absorber	absorbing	abstract	abstra
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	(

```
image_features_test = pd.read_csv(save_path +'/' + 'image_features_test.csv')
print(image_features_test.shape)
image_features_test.head(1)
```

```
(175, 2048)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	0.1865	1.0133	0.4206	0.0726	0.0000	0.0000	0.1678	1.2262	0.7542	0.2218	0.8193	0.6511	1.5894	0.4037	0.0181	1.9568	0.0000	0.0

```
X_test_aug = pd.concat([test_text_features_aug, image_features_test], axis=1)
print(X_test_aug.shape)
X_test_aug.to_csv(save_path +'/' + 'X_test_aug.csv', index=False)
```

```
(175, 5607)
```

Start coding or generate with AI.

1.8.3.3-b - Without data augmentation

```
test_text_features_no_aug = pd.read_csv(save_path +'/' + 'text_features_test_no_aug.csv').drop(columns=['Unnamed: 0'])
print(test_text_features_no_aug.shape)
test_text_features_no_aug.head(1)
```

```
(175, 2631)
```

	aapno	aari	ability	abkl	abode	abroad	absorbency	absorbent	absorbing	abstract	accent	access	accessory	accident	ac
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

```
image_features_test = pd.read_csv(save_path +'/' + 'image_features_test.csv')
print(image_features_test.shape)
image_features_test.head(1)
```

```
(175, 2048)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	0.1865	1.0133	0.4206	0.0726	0.0000	0.0000	0.1678	1.2262	0.7542	0.2218	0.8193	0.6511	1.5894	0.4037	0.0181	1.9568	0.0000	0.0

```
X_test_no_aug = pd.concat([test_text_features_no_aug, image_features_test], axis=1)
print(X_test_no_aug.shape)
X_test_no_aug.to_csv(save_path +'/' + 'X_test_no_aug.csv', index=False)
```

```
(175, 4679)
```

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

2 - Supervised classification - image data only

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_kerne
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 4s 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]
```

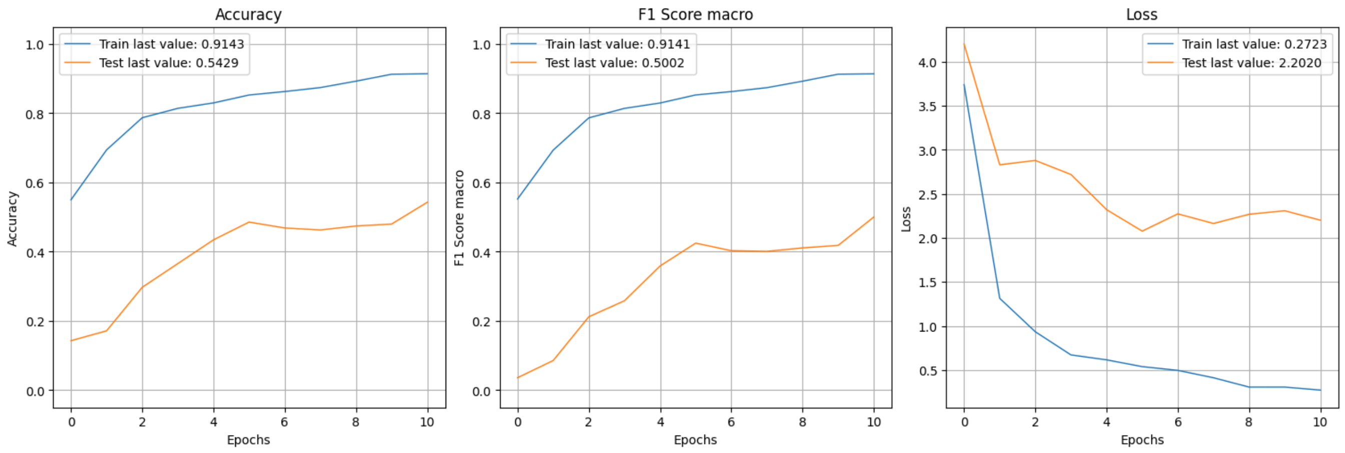
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
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.append(model_1_data)
    histories['model_1'] = history_1
```

```

Epoch 1: val_loss improved from inf to 4.20142, saving model to ./model_1_best_weights.h5

Epoch 2: val_loss improved from 4.20142 to 2.83131, saving model to ./model_1_best_weights.h5
Epoch 3: val_loss did not improve from 2.83131

Epoch 4: val_loss improved from 2.83131 to 2.72084, saving model to ./model_1_best_weights.h5

Epoch 5: val_loss improved from 2.72084 to 2.32152, saving model to ./model_1_best_weights.h5

Epoch 6: val_loss improved from 2.32152 to 2.07804, saving model to ./model_1_best_weights.h5

Epoch 7: val_loss did not improve from 2.07804

Epoch 8: val_loss did not improve from 2.07804

Epoch 9: val_loss did not improve from 2.07804

Epoch 10: val_loss did not improve from 2.07804

Epoch 11: val_loss did not improve from 2.07804
Epoch 11: 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]
```

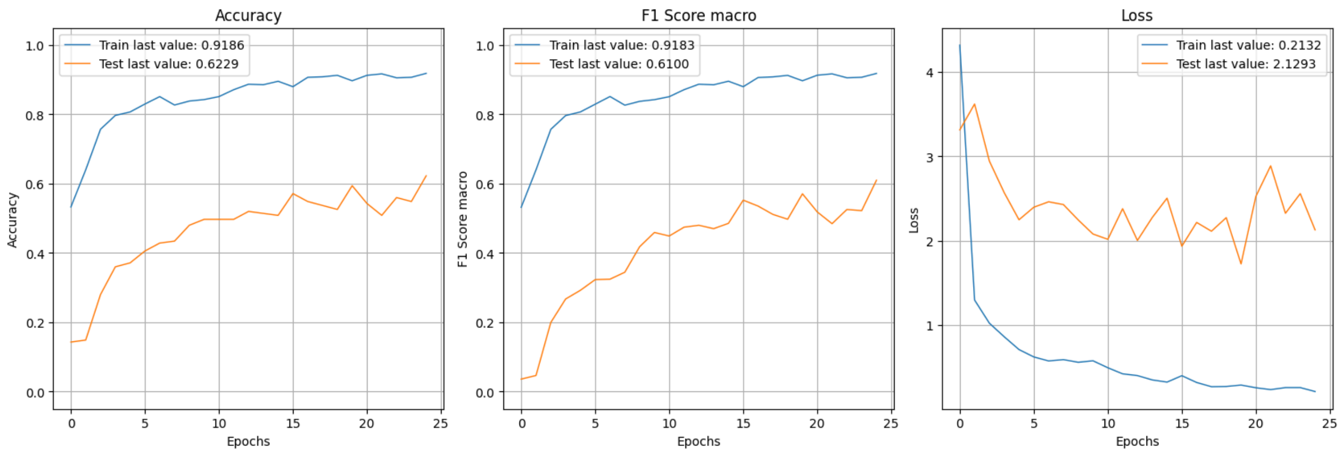
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
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.append(model_2_data)
    histories['model_2'] = history_2
```

```
Epoch 1: val_loss improved from inf to 3.31462, saving model to ./model_2_best_weights.h5

Epoch 2: val_loss did not improve from 3.31462

Epoch 3: val_loss improved from 3.31462 to 2.94449, saving model to ./model_2_best_weights.h5

Epoch 4: val_loss improved from 2.94449 to 2.57075, saving model to ./model_2_best_weights.h5

Epoch 5: val_loss improved from 2.57075 to 2.24926, saving model to ./model_2_best_weights.h5

Epoch 6: val_loss did not improve from 2.24926

Epoch 7: val_loss did not improve from 2.24926

Epoch 8: val_loss did not improve from 2.24926

Epoch 9: val_loss improved from 2.24926 to 2.24704, saving model to ./model_2_best_weights.h5

Epoch 10: val_loss improved from 2.24704 to 2.07996, saving model to ./model_2_best_weights.h5

Epoch 11: val_loss improved from 2.07996 to 2.01744, saving model to ./model_2_best_weights.h5

Epoch 12: val_loss did not improve from 2.01744

Epoch 13: val_loss improved from 2.01744 to 2.00408, saving model to ./model_2_best_weights.h5

Epoch 14: val_loss did not improve from 2.00408

Epoch 15: val_loss did not improve from 2.00408

Epoch 16: val_loss improved from 2.00408 to 1.93603, saving model to ./model_2_best_weights.h5

Epoch 17: val_loss did not improve from 1.93603

Epoch 18: val_loss did not improve from 1.93603

Epoch 19: val_loss did not improve from 1.93603

Epoch 20: val_loss improved from 1.93603 to 1.72577, saving model to ./model_2_best_weights.h5

Epoch 21: val_loss did not improve from 1.72577

Epoch 22: val_loss did not improve from 1.72577

Epoch 23: val_loss did not improve from 1.72577

Epoch 24: val_loss did not improve from 1.72577

Epoch 25: val_loss did not improve from 1.72577
Epoch 25: 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]
```

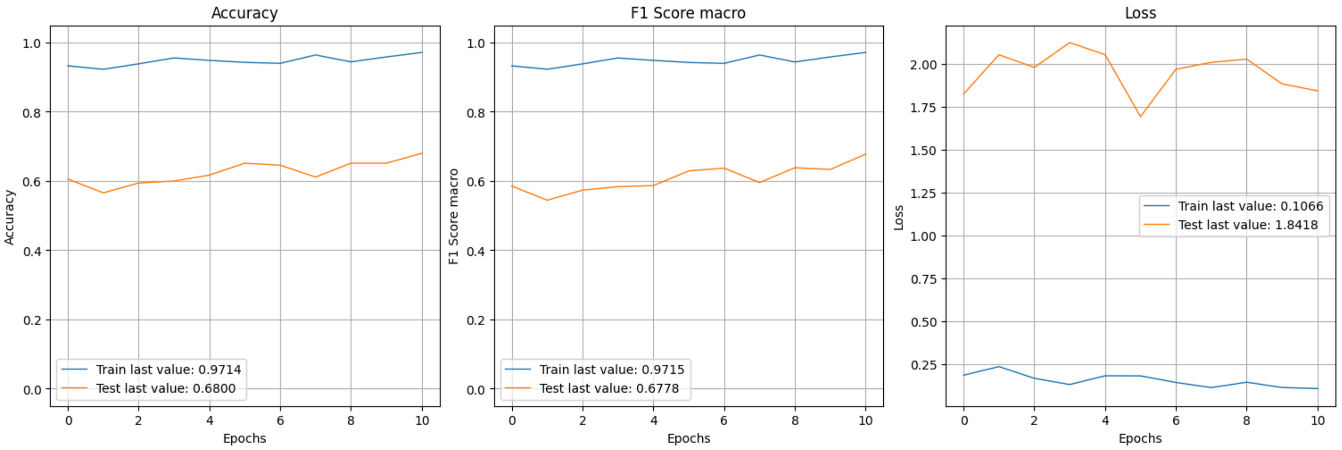
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

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

```

Epoch 1: val_loss did not improve from 1.68565

Epoch 2: val_loss did not improve from 1.68565

Epoch 3: val_loss did not improve from 1.68565

Epoch 4: val_loss did not improve from 1.68565

Epoch 5: val_loss did not improve from 1.68565

Epoch 6: val_loss did not improve from 1.68565

Epoch 7: val_loss did not improve from 1.68565

Epoch 8: val_loss did not improve from 1.68565

Epoch 9: val_loss did not improve from 1.68565

Epoch 10: val_loss did not improve from 1.68565

Epoch 11: val_loss did not improve from 1.68565
Epoch 11: 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]
```

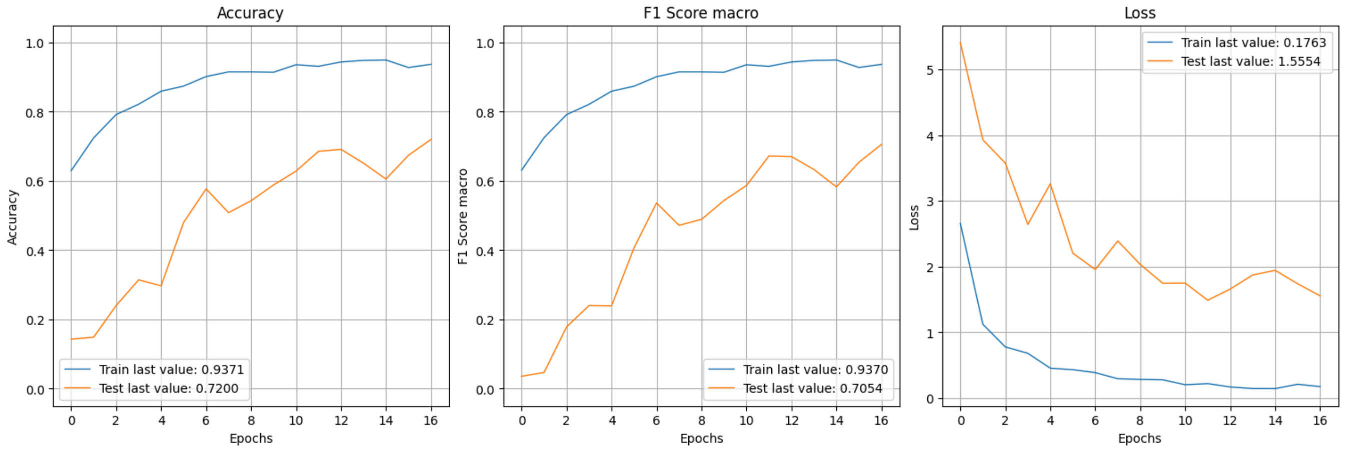
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
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.append(model_4_data)
    histories['model_4'] = history_4
```

```
Epoch 1: val_loss improved from inf to 5.40883, saving model to ./model_4_best_weights.h5

Epoch 2: val_loss improved from 5.40883 to 3.93021, saving model to ./model_4_best_weights.h5

Epoch 3: val_loss improved from 3.93021 to 3.57572, saving model to ./model_4_best_weights.h5

Epoch 4: val_loss improved from 3.57572 to 2.64239, saving model to ./model_4_best_weights.h5

Epoch 5: val_loss did not improve from 2.64239

Epoch 6: val_loss improved from 2.64239 to 2.20580, saving model to ./model_4_best_weights.h5

Epoch 7: val_loss improved from 2.20580 to 1.95765, saving model to ./model_4_best_weights.h5

Epoch 8: val_loss did not improve from 1.95765

Epoch 9: val_loss did not improve from 1.95765

Epoch 10: val_loss improved from 1.95765 to 1.74811, saving model to ./model_4_best_weights.h5

Epoch 11: val_loss did not improve from 1.74811

Epoch 12: val_loss improved from 1.74811 to 1.48972, saving model to ./model_4_best_weights.h5

Epoch 13: val_loss did not improve from 1.48972

Epoch 14: val_loss did not improve from 1.48972

Epoch 15: val_loss did not improve from 1.48972

Epoch 16: val_loss did not improve from 1.48972

Epoch 17: val_loss did not improve from 1.48972
Epoch 17: 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]
```

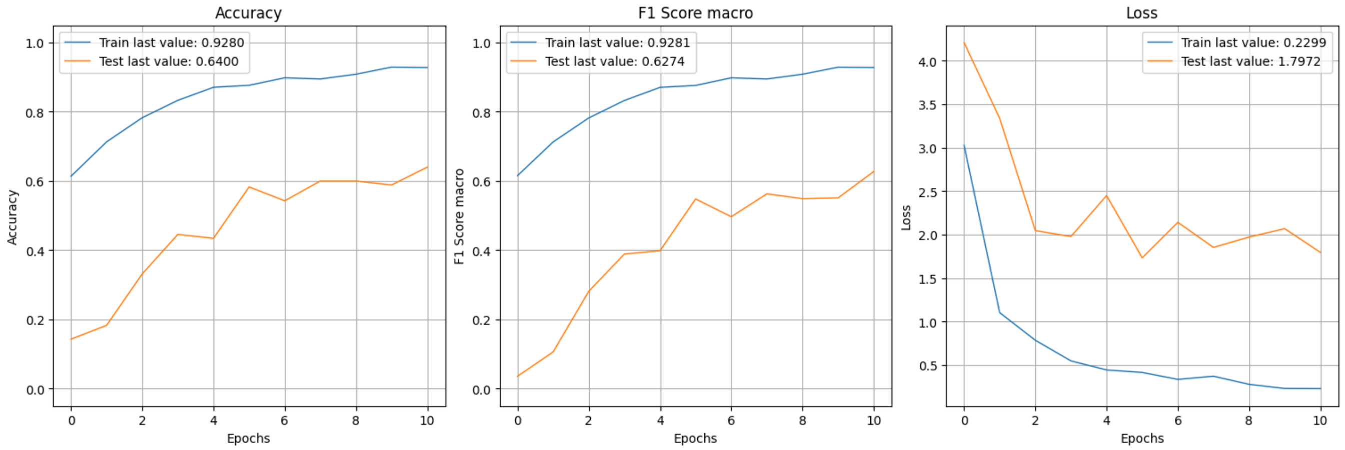
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
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.append(model_5_data)
    histories['model_5'] = history_5
```

```
Epoch 1: val_loss improved from inf to 4.21019, saving model to ./model_5_best_weights.h5

Epoch 2: val_loss improved from 4.21019 to 3.33827, saving model to ./model_5_best_weights.h5

Epoch 3: val_loss improved from 3.33827 to 2.04769, saving model to ./model_5_best_weights.h5

Epoch 4: val_loss improved from 2.04769 to 1.97927, saving model to ./model_5_best_weights.h5

Epoch 5: val_loss did not improve from 1.97927

Epoch 6: val_loss improved from 1.97927 to 1.73417, saving model to ./model_5_best_weights.h5

Epoch 7: val_loss did not improve from 1.73417

Epoch 8: val_loss did not improve from 1.73417

Epoch 9: val_loss did not improve from 1.73417

Epoch 10: val_loss did not improve from 1.73417

Epoch 11: val_loss did not improve from 1.73417
Epoch 11: 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]
```

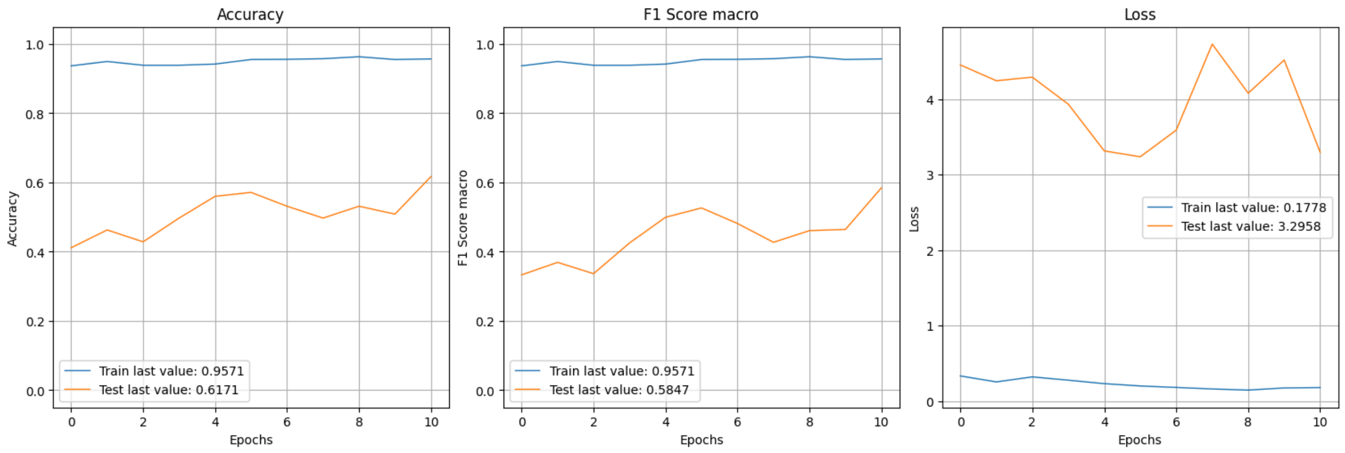
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
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.append(model_6_data)
    histories['model_6'] = history_6
```

```
Epoch 1: val_loss did not improve from 4.01921

Epoch 2: val_loss did not improve from 4.01921

Epoch 3: val_loss did not improve from 4.01921

Epoch 4: val_loss improved from 4.01921 to 3.93385, saving model to ./model_6_best_weights.h5

Epoch 5: val_loss improved from 3.93385 to 3.31444, saving model to ./model_6_best_weights.h5

Epoch 6: val_loss improved from 3.31444 to 3.23670, saving model to ./model_6_best_weights.h5

Epoch 7: val_loss did not improve from 3.23670

Epoch 8: val_loss did not improve from 3.23670

Epoch 9: val_loss did not improve from 3.23670

Epoch 10: val_loss did not improve from 3.23670

Epoch 11: val_loss did not improve from 3.23670
Epoch 11: 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

```
# Create the DataFrame
models_data = pd.DataFrame(models_data)
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

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

5 Augmented 11 11 65.0602 0.9571 0.6171 0.9571 0.5847 0.6171 0.8000 0.6171

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

3 - Supervised classification using text and image features

3.1 - Define metrics, pipelines, pre-processing steps & hyperparameter grids

```
# Define performance metrics
scorers = {'f1_macro':  make_scorer(f1_score, average='macro'),
           'f2_macro': make_scorer(fbeta_score, beta=2, average='macro'),
           'accuracy': make_scorer(accuracy_score),
           'precision': make_scorer(precision_score, average='macro'),
           'recall': make_scorer(recall_score, average='macro')
           }
```

```
# Split numerical and categorical columns and list column names
num_X_no_aug = X_train_no_aug.select_dtypes(include='number').columns.tolist()
# check
print(len(num_X_no_aug))

num_X_aug = X_train_aug.select_dtypes(include='number').columns.tolist()

# check
print(len(num_X_aug))
```

```
4679
5607
```

```
num_transfo_ens = Pipeline(steps=[('scaler', StandardScaler())])
preproc_ens_no_aug = ColumnTransformer(transformers=[('num', num_transfo_ens, num_X_no_aug)], remainder='passthrough')
preproc_ens_aug = ColumnTransformer(transformers=[('num', num_transfo_ens, num_X_aug)], remainder='passthrough')
```

```
# initialise models
models = [
    ('Random Forest Regression', RandomForestClassifier(verbose=0, n_jobs=-1, random_state=rs)), # Baseline model
    ('LightGBM', LGBMClassifier(verbose=-1, n_jobs=-1, random_state=rs)),
    ('XGBoost', XGBClassifier(verbose=0, n_jobs=-1, random_state=rs))
]

# create list of ensemble models
ens_models = ['Random Forest Regression', 'LightGBM', 'XGBoost']

# define hyperparameter grids
param_grids = [
    # for Random Forest Regression
    {'model__n_estimators': [400, 800],
      'model__max_depth' : [10, 20],
      'model__max_features' : ["sqrt"],
      'model__min_samples_split' : [2]},
    # for LightGBM
    {'model__max_depth': [6, 8], # decrease for faster convergence
     'model__num_leaves':[60, 200], # should be smaller than 2^(max_depth) - decrease for faster convergence
     'model__min_data_in_leaf':[200, 400], # very important parameter to prevent over-fitting in a leaf-wise tree, hundreds or thousands
     'model__learning_rate': [0.1, 0.2], # increase if decreasing num_iterations
     'model__min_gain_to_split': [0.001]}, # default is 0 (no gain too small) - increase to reduce training time
    # for XGBoost
    {'model__booster':['gbtree'], # gblinear also available
     'model__device':['cuda'], # use cuda for GPU acceleration, otherwise use 'cpu'
     'model__learning_rate': [0.1, 0.2], # see also eta - default is 0.3
     'model__min_split_loss': [0.1], # see also gamma - default is 0
     'model__min_child_weight':[5],
     'model__max_depth': [6, 8], # default is 6
     'model__max_delta_step' :[5, 8],
     'model__n_estimators':[400, 800],
     'model__tree_method':['gpu_hist'], # default is 'auto'
     'model__subsample': [0.75], # set >=0.5 for good results with uniform sampling method
     'model__colsample_bytree' : [0.75], # default is 1 - decrease to control overfitting
     'model__colsample_bylevel' : [0.75], # default is 1 - decrease to control overfitting
     'model__colsample_bynode' : [0.75], # default is 1 - decrease to control overfitting
     'model__sampling_method':['gradient_based'] # gradient-based sampling works only with tree_method=gpu_hist and device=cuda
    }]
```

Start coding or generate with AI.

3.2 - Without data augmentation of training set

```
# fit models using GridSearchCV

# initialize empty list to store models evaluation metrics
model_metrics_no_aug = []

# iterate over models and respective parameter grid
for (name, model), param_grid in zip(models, param_grids):
    # create pipeline with relevant preprocessor and model
    if name in ens_models:
        pipe = Pipeline(steps=[('preprocessor', preproc_ens_no_aug), ('model', model)])
    else:
        # pipe = Pipeline(steps=[('preprocessor', preproc_lin), ('model', model)])
        print('Model not in list - check ensemble_models')

    k_values = [5]

    for k in k_values:
        skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=rs)
        spl = skf.split(X_train_no_aug, y_train)

        # Perform GridSearchCV on the pipeline with the current hyperparameter grid
        grid_search = GridSearchCV(pipe, param_grid, cv=spl, scoring=scorers, refit='f1_macro', error_score='raise', verbose=True)

        # Fit the GridSearchCV object on the training data
        train_start = timer()
        X_train_no_aug.columns = X_train_no_aug.columns.astype(str)
        X_val_no_aug.columns = X_val_no_aug.columns.astype(str)
        grid_search.fit(X_train_no_aug, y_train.values)
        train_end = timer()
        train_time = train_end - train_start

        # Get the best hyperparameters and best training score
        best_params = grid_search.best_params_
        best_score = grid_search.best_score_

        # evaluate model on validation set & get score
        val_start = timer()
        val_score = grid_search.score(X_val_no_aug, y_val) # uses score defined by scoring or best_estimator_.score
        y_pred_no_aug = grid_search.predict(X_val_no_aug)
        val_end = timer()
        val_time = val_end - val_start

        # Extract coefficients from the best estimator
        best_model = grid_search.best_estimator_

    # append results to results list
    model_metrics_no_aug.append({
        'Model': model,
        'Training set':'X_train_no_aug',
        'Best parameters': best_params,
        'Train f1_macro': best_score,
        'Train fit time': train_time,
        'Val f1_macro': val_score,
        'Val f2_macro': fbeta_score(y_val, y_pred_no_aug, beta=2, average='macro'),
        'Val Accuracy': accuracy_score(y_val, y_pred_no_aug),
        'Val Precision': precision_score(y_val, y_pred_no_aug, average='macro'),
        'Val Recall': recall_score(y_val, y_pred_no_aug, average='macro'),
        'Strat. K-Fold' : skf
    })
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

```
# store model metrics & coefs in dataframes
model_no_aug_compare = pd.DataFrame(model_metrics_no_aug)

# display dataframe
model_no_aug_compare
```

	Model	Training set	Best parameters	Train f1_macro	Train fit time	Val f1_macro	Val f2_macro	Val Accuracy	Va Precisio
0	RandomForestClassifier(n_jobs=-1, random_state=42)	X_train_no_aug	{'model__max_depth': 10, 'model__max_features': 'sqrt', 'model__min_samples_split': 2, 'model__n_estimators': 800}	0.8020	48.7345	0.8383	0.8372	0.8400	0.856
1	LGBMClassifier(n_jobs=-1, random_state=42, verbose=-1)	X_train_no_aug	{'model__learning_rate': 0.2, 'model__max_depth': 6, 'model__min_data_in_leaf': 200, 'model__min_gain_to_split': 0.001, 'model__num_leaves': 60}	0.2039	74.9751	0.2454	0.2481	0.2514	0.247
2	XGBClassifier(base_score=None, booster=None, callbacks=None,\n colsample_bylevel=None, colsample_bynode=None,\n colsample_bytree=None, device=None, early_stopping_rounds=None,\n enable_categorical=False, eval_metric=None, feature_types=None,\n gamma=None, grow_policy=None, importance_type=None,\n interaction_constraints=None, learning_rate=None, max_bin=None,\n max_cat_threshold=None, max_cat_to_onehot=None,\n max_delta_step=None, max_depth=None, max_leaves=None,\n min_child_weight=None, missing=nan, monotone_constraints=None,\n multi_strategy=None, n_estimators=None, n_jobs=-1,\n num_parallel_tree=None, random_state=42, ...)	X_train_no_aug	{'model__booster': 'gbtree', 'model__colsample_bylevel': 0.75, 'model__colsample_bynode': 0.75, 'model__colsample_bytree': 0.75, 'model__device': 'cuda', 'model__learning_rate': 0.1, 'model__max_delta_step': 5, 'model__max_depth': 6, 'model__min_child_weight': 5, 'model__min_split_loss': 0.1, 'model__n_estimators': 400, 'model__sampling_method': 'gradient_based', 'model__subsample': 0.75, 'model__tree_method': 'gpu_hist'}	0.7382	1174.7171	0.8062	0.8017	0.8057	0.841

```
# Generate timestamp and save df
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
filename = f"model_no_aug_compare_{timestamp}.csv"
model_no_aug_compare.to_csv(save_path + '/' + filename, index=False)
```

```
pd.read_csv(save_path + '/' + 'model_no_aug_compare_20250515_205900.csv')
```

	Model	Training set	Best parameters	Train F1_macro	Train fit time	Val F1_macro	Strat. K-Fold
0	RandomForestClassifier()	X_train_no_aug	{'model__max_depth': 10, 'model__max_features': 'sqrt', 'model__min_samples_split': 2, 'model__n_estimators': 800}	0.8304	1025.7301	0.8172	StratifiedKFold(n_splits=13, random_state=13, shuffle=True)
	LGBMClassifier(n_jobs=-1,		{'model__is_unbalance': True, 'model__learning_rate': 0.2, 'model__max_depth': 6, … 'model num leaves': 60}				StratifiedKFold(n_splits=13,

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

3.3 - With data augmentation of training set

```
# fit models using GridSearchCV

# initialize empty list to store models evaluation metrics
model_metrics_aug = []

# iterate over models and respective parameter grid
for (name, model), param_grid in zip(models, param_grids):
    # create pipeline with relevant preprocessor and model
    if name in ens_models:
        pipe = Pipeline(steps=[('preprocessor', preproc_ens_aug), ('model', model)])
    else:
        # pipe = Pipeline(steps=[('preprocessor', preproc_lin), ('model', model)])
        print('Model not in list - check ensemble_models')
    k_values = [5]

    for k in k_values:
        skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=rs)
        spl = skf.split(X_train_aug, y_train)

        # Perform GridSearchCV on the pipeline with the current hyperparameter grid
        grid_search = GridSearchCV(pipe, param_grid, cv=spl, scoring=scorers, refit='f1_macro', error_score='raise', verbose=True)

        # Fit the GridSearchCV object on the training data
        train_start = timer()
        X_train_aug.columns = X_train_aug.columns.astype(str)
        X_val_aug.columns = X_val_aug.columns.astype(str)
        grid_search.fit(X_train_aug, y_train.values)
        train_end = timer()
        train_time = train_end - train_start

        # Get the best hyperparameters and best training score
        best_params = grid_search.best_params_
        best_score = grid_search.best_score_

        # evaluate model on validation set & get score
        val_start = timer()
        val_score = grid_search.score(X_val_aug, y_val) # uses score defined by scoring or best_estimator_.score
        y_pred_aug = grid_search.predict(X_val_aug)
        val_end = timer()
        val_time = val_end - val_start

        # Extract coefficients from the best estimator
        best_model = grid_search.best_estimator_

    # append results to results list
    model_metrics_aug.append({
        'Model': model,
        'Training set':'X_train_aug',
        'Best parameters': best_params,
        'Train f1_macro': best_score,
        'Train fit time': train_time,
        'Val f1_macro': val_score,
        'Val f2_macro': fbeta_score(y_val, y_pred_aug, beta=2, average='macro'),
        'Val Accuracy': accuracy_score(y_val, y_pred_aug),
        'Val Precision': precision_score(y_val, y_pred_aug, average='macro'),
        'Val Recall': recall_score(y_val, y_pred_aug, average='macro'),
        'Strat. K-Fold' : skf
    })
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

```
# store model metrics & coefs in dataframes
model_aug_compare = pd.DataFrame(model_metrics_aug)

# display dataframe
model_aug_compare
```

	Model	Training set	Best parameters	Train f1_macro	Train fit time	Val f1_macro	Val f2_macro	Val Accuracy	Val Precision
0	RandomForestClassifier(n_jobs=-1, random_state=42)	X_train_aug	{'model__max_depth': 10, 'model__max_features': 'sqrt', 'model__min_samples_split': 2, 'model__n_estimators': 800}	0.7792	50.6849	0.8184	0.8163	0.8171	0.8302
1	LGBMClassifier(n_jobs=-1, random_state=42, verbose=-1)	X_train_aug	{'model__learning_rate': 0.1, 'model__max_depth': 6, 'model__min_data_in_leaf': 200, 'model__min_gain_to_split': 0.001, 'model__num_leaves': 60}	0.1736	73.5406	0.2348	0.2435	0.2514	0.2282
2	XGBClassifier(base_score=None, booster=None, callbacks=None,\n colsample_bylevel=None, colsample_bynode=None,\n colsample_bytree=None, device=None, early_stopping_rounds=None,\n enable_categorical=False, eval_metric=None, feature_types=None,\n gamma=None, grow_policy=None, importance_type=None,\n interaction_constraints=None, learning_rate=None, max_bin=None,\n max_cat_threshold=None, max_cat_to_onehot=None,\n max_delta_step=None, max_depth=None, max_leaves=None,\n min_child_weight=None, missing=nan, monotone_constraints=None,\n multi_strategy=None, n_estimators=None, n_jobs=-1,\n num_parallel_tree=None, random_state=42, ...)	X_train_aug	{'model__booster': 'gbtree', 'model__colsample_bylevel': 0.75, 'model__colsample_bynode': 0.75, 'model__colsample_bytree': 0.75, 'model__device': 'cuda', 'model__learning_rate': 0.1, 'model__max_delta_step': 5, 'model__max_depth': 8, 'model__min_child_weight': 5, 'model__min_split_loss': 0.1, 'model__n_estimators': 800, 'model__sampling_method': 'gradient_based', 'model__subsample': 0.75, 'model__tree_method': 'gpu_hist'}	0.7261	1301.9277	0.7469	0.7456	0.7486	0.7658

```
# Generate timestamp and save df
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
filename = f"model_aug_compare_{timestamp}.csv"
model_aug_compare.to_csv(save_path + '/' + filename, index=False)
```

```
model_compare_all = pd.DataFrame(model_metrics_no_aug)
timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
filename = f"model_compare_all_{timestamp}.csv"
model_compare_all.to_csv(save_path + '/' + filename, index=False)
```

```
pd.read_csv(save_path + '/' + 'model_compare_all_20250515_222941.csv')
```

	Model	Training set	Best parameters	Train f1_macro	Train fit time	Val f1_macro	Strat. K-Fold
0	RandomForestClassifier()	X_train_no_aug	{'model__max_depth': 10, 'model__max_features': 'sqrt', 'model__min_samples_split': 2, 'model__n_estimators': 800}	0.8032	176.3609	0.8456	StratifiedKFold(n_splits=5, random_state=8, shuffle=True)
1	LGBMClassifier(n_jobs=-1, random_state=8, verbose=-1)	X_train_no_aug	{'model__learning_rate': 0.2, 'model__max_depth': 6, 'model__min_data_in_leaf': 200, 'model__min_gain_to_split': 0.001, 'model__num_leaves': 60}	0.2052	66.4446	0.2454	StratifiedKFold(n_splits=5, random_state=8, shuffle=True)
2	XGBClassifier(base_score=None, booster=None, callbacks=None,\n colsample_bylevel=None, colsample_bynode=None,\n colsample_bytree=None, device=None, early_stopping_rounds=None,\n eval_metric=None, feature_types=None,\n gamma=None, grow_policy=None, importance_type=None,\n interaction_constraints=None, learning_rate=None multi_strategy=None,	X_train_no_aug	{'model__booster': 'gbtree', 'model__colsample_bylevel': 0.75, 'model__colsample_bytree': 0.75, 'model__device': 'cuda', 'model__learning_rate': 0.1, 'model__max_delta_step': 5,	0.7567	1244.2592	0.7691	StratifiedKFold(n_splits=5, random_state=8

Start coding or generate with AI.

3.4 - Retrain & evaluate best model on test data

```
# params & pipe from best model run
best_params = {'max_depth': 10,
               'max_features': 'sqrt',
               'min_samples_split': 2,
               'n_estimators': 800
              }

best_model = RandomForestClassifier(**best_params, verbose=0, random_state=rs)
```

```
timestamp_11 = datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
best_model.fit(X_train_val_no_aug, y_train_val)
```

```
▾ RandomForestClassifier
```

), ',

StratifiedKFold(n_splits=5,