Guide of the code created for smoke plume segmentation

File name	Format file	Usage	Variables to modify	File to execute
experiment_name	.json	Configuration parametes	all	
utils	.py	Loads configuration parameter (cfg)	-	
main	.py	Main file to execute	*EXPERIMENT* *WandB API key*	**X**
prediction_output	.py	'main' file to predict new random images (without masks)	*EXPERIMENT* and *Images* (path and extension of images to read and predict)	**X**
dataset	.py	Creates the training, validation and test datasets	-	
unet_model	.py	UNet model architecture	-	
unet_model_original	.py	Original UNet model architecture	-	
train	.py		-	
evaluation	.py		-	
predict	.py	Obtain predictions for different datasets	-	
video_segmentation	.json	Configuration parameters for video	all	
video_segmentation	.py	'main' file for video segmentation	video_params EXPERIMENT	**X**
video_prediction	.py	Functions for video segmenting	-	

experiment_name.json

This json file is where the different parameters are defined for each test performed of the model training. This json file are saved the experiment folder.

- "train_path": "/home /ALBA/unet/smoke_dataset/train/"
- "test_path": "/home /ALBA/unet/smoke_dataset/test/"
- "num_augmentations": 4
- "pretrained_encoder": "MobileNetV2" or "ResNet50"
- "batch_size": 16
- "learning_rate": 0.0001
- "lr_scheduler": "None"
- "num_epochs": 200
- "img_width": 256
- "img_height": 256
- "img_channels": 3
- "n_classes": 1
- "test_size": 0.1
- "weight_path": "/home /ALBA/unet/output/weights/"
- "model_path": "/home /ALBA/unet/output/model/"
- "best_model_path": "/home /ALBA/unet/output/best_model/"
- "save_every_epoch": 100
- "CUDA_VISIBLE_DEVICES": "1"
- "dropout": 0

utils.py

This python file save and loads the configuration parameters of the json file created (experiment_name.json).

main.py

Main Python script, this is the only file needed to execute in order to proceed with the model training process. The different functions needed to proceed are called and executed in this script. The variables to be changed in the functions called are marked in bold.

The datasets name variables created are the following (they can be changed): TrainDataset, ValDataset and TestDataset.

EXPERIMENT variable must be changed with the name of the test to perform.

os.environ["WANDB_API_KEY"] = **'change for your WandB key number'**

ShowTable(model, **dataset name**, **'Title'**)

ConfusionMatrix(model, **dataset name**, cfg, **path to save it**)

SavePredictions(model, **dataset name**, cfg, **path to save it**)

prediction_output.py

It's like the main script but designed for the prediction of images that are not in the original dataset and they don't have a ground truth mask.

EXPERIMENT variable must be changed with the name of the model to use for the prediction.

Images variable must be changed with the path of the images folder to predict and its extension (jpg or png).

save_path variable must be changed with the path of the folder to save the predicted images.

dataset.py

This Python script aims to create and adapt the dataset to the format needed in order to be applied in the training, validating and testing process. The functions created are the following:

- ***LoadImages:*** this function reads the jpg images and png masks from the path defined in the cfg json file, for both training or testing datasets (this should be specified when calling the function, as a function variable mode='test'). The images read are converted to RGB and the masks to gray scale image. Afterwards both are normalized and saved on a image and mask dictionary variable type.

- ***LoadForPrediction:*** This function is used to read and load new images to obtain its prediction (images that don't have ground truth mask). It reads the images from the path and extension defined (when calling this function in the main or prediction_output files). The images read are converted to RGB, normalized and saved on an image dictionary variable type.

- ***ResizeDataset:*** this function resizes both, images and masks to the image height and with specified in the cfg json file. It converts it to a (256, 256, 3) size for images and (256, 256, 1) for masks.

- ***DatasetToFloat:*** this function takes the dataset created and converts it to a float variable (to obtain the correct format in order to make the posterior tasks).

- ***DataAugmentation:*** this function objective is to apply data augmentation to create new training samples from the existing data (train dataset). Data augmentation is done using the *Albumentations* library. It returns a dataset with the images and its corresponding masks after data augmentation pre-processing is applied.

- ***CrearDataset:*** The aim of this function is to create the dataset for training the model. The dataset output obtained after applying this function consists on the *AugmentedDataset* variable output which is the training data and *ValDataset*, which is the validation data. First, the training dataset is split into train and validation data and saved into two different dictionary variables, data augmentation is applied only on the train data, this new dataset is saved. Both datasets are pre-processed (resized and converted to float).

- ***LoadTestDataset:*** this function loads and pre-processes the images and masks of the test dataset. The input data of this function is the configuration file (cfg). It returns the *TestDataset* variable.
    - LoadImage
    - ResizeDataset
    - DatasetToFloat

- ***ShowDatasetExample:*** this function aims to output a plot with different random augmentation images of the dataset with its original image.

unet_model.py

This python script is where the UNet model is created, and all the functions created to evaluate the training process (TPR, TNR, IOU, Dice and Dice loss).

Two variants of the backbone (encoder) used for in the model are created, ResNet50 or MobileNetV2. The encoder used has to be defined in the configuration file (experiment_name.json).

unet_model_original.py

This script contains the original UNet model (Ronneberger et al., 2015) and all the functions created to evaluate the training process (TPR, TNR, IOU, Dice and Dice loss).

train.py

- ***ModelGeneration:*** read the model to be trained and define the metrics, callbacks and loss function that are gonna be used in the training process of this model.

- ***TrainModel:*** read all the callbacks and execute the model training process.

evaluation.py

- ***wb_mask:*** with this function the ground truth vs the segmented output (prediction) are shown in WandB platform. The classes (or labels) created are 0 (background or non-smoke) and 1 (smoke).

- ***ShowPredictions:*** function created to visualize in WandB the predictions output of each epoch trained of the images in the dataset defined where the function is called in main.py script. An interactive visualization is shown, where the background and the smoke can be visualized for both predicted output and ground truth mask.

- ***ShowTable:*** function created to visualize in WandB the predictions output plot of the dataset defined where the function is called in main.py script. Showing the original image, original mask (ground truth), and predicted mask.

- ***ConfusionMatrix:*** obtain the confusion matrix of the predicted data.

predict.py

This script is created to obtain the predictions of the trained model applied to image datasets with groundtruth (SavePredictions) or to image datasets without groundtruth (SavePrediction_new).

- ***LoadModel:*** this function is created for loading the model already trained.

- ***SavePredictions:*** this function is used for obtaining the output of the prediction of test dataset, it consists of a plot of 4 output images (original image, ground truth, predicted mask and the predicted mask overlaid on the original image).

- ***SavePredictions_new:*** this function is used for obtaining the output of the prediction for new images (images that don't have a ground truth mask). It consists of a plot of 3 output images (original image, predicted mask, and the predicted mask overlaid on the original image).

video_segmentation.json

This json file is where the different parameters for segmenting video sequences are defined for each video. This json file are saved the experiment folder. The different variables that can be modified are the following, including the name and format of the video and it's posterior image frames, which trained model use to predict the smoke and the paths where to read and save files.

- "video_name": "Napi1"
- "form_video": ".mp4"
- "form_img": ".jpg"
- "model_name": "final_unet_model.h5"
- "output": "/home/ALBA/unet/output/video"
- "video_dataset": "/home/ALBA/unet/video_dataset"
- "model_path": "/home/ALBA/unet/output/model/"
- "CUDA_VISIBLE_DEVICES": "0"

video_segmentation.py

This Python script is like the main file for video segmentation, this is the only file needed to execute in order to proceed with the segmentation of a video sequence. The different functions needed to proceed are called and executed in this script.

video_params and EXPERIMENT variables must be changed with the name of the test to perform. These are the configuration json files with all the parameters needed.

The video_segmentation.json file is the specific configuration parameters file created to define the name of the video to segment, the extension format, and the path where it will be saved. This json file can be found in the experiment folder.

video_prediction.py

Python script file where the functions needed for segmenting video sequences are created.

- ***ExtractFrames:*** this function divides the video sequence into different frames (for the posterior segmentation of each one).

- ***get_img_from_fig:*** This function aims to convert a matplotlib figure object into an image. It takes a matplotlib figure, saves it as a PNG image with a specified resolution, and then converts the image into a format that can be processed using OpenCV

- ***visualize_pred:*** in this function the visualization output of the predictions is executed, where a plot with the original image, the predicted mask and the predicted mask overlaid on the original image is shown in the same plot (for each frame of the video).

- ***VideoPred:*** this function predicts the frames cropped from the video and afterwards it reconstructs the video with the segmented output.