

Chapter 1

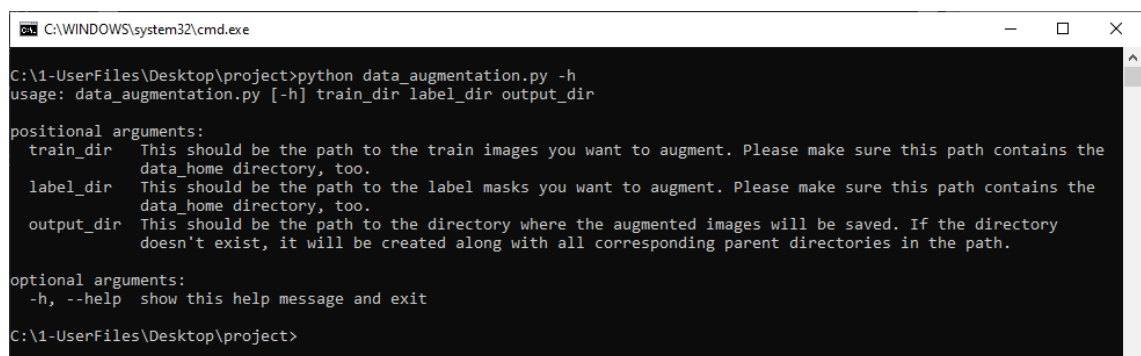
User Manual

In this chapter, detailed instructions on how to execute the source code have been provided. All explanations contain screenshots that demonstrate the inputs and outputs of the executable commands. This chapter assumes that the steps listed in the **Maintenance Manual** have been followed and the project repository has been set up.

Data augmentation

To execute this functionality, you need to open a Command Prompt window or a Terminal and navigate to the project directory. The following instructions will walk you through the process of running the example:

1. Run the command - `python data_augmentation.py -h`. It will display help information about the script. More specifically, it shows what arguments are required in order for the script to be executed. Figure 1 displays the output of this command. It is visible that the script requires 3 positional arguments - directories. That is, arguments that need to be given exactly in the order they are requested.



```
C:\WINDOWS\system32\cmd.exe

C:\1-UserFiles\Desktop\project>python data_augmentation.py -h
usage: data_augmentation.py [-h] train_dir label_dir output_dir

positional arguments:
  train_dir  This should be the path to the train images you want to augment. Please make sure this path contains the
             data_home directory, too.
  label_dir  This should be the path to the label masks you want to augment. Please make sure this path contains the
             data_home directory, too.
  output_dir This should be the path to the directory where the augmented images will be saved. If the directory
             doesn't exist, it will be created along with all corresponding parent directories in the path.

optional arguments:
  -h, --help  show this help message and exit

C:\1-UserFiles\Desktop\project>
```

Figure 1: Output of the help information for the `data_augmentation.py` script. It expects a directory with images to augment, the path to the folder containing the corresponding ground-truth masks and an output location where the newly augmented images will be saved.

2. An example of the execution of the data augmentation algorithm is shown in Figure 2. The red numbers shown on the screenshot correspond to the following 4 items:
 - (a) Shows the directories residing in a repository, **KITTI_test**, before data augmentation is applied.
 - (b) This part counts the number of available labelled masks - 384.

- (c) Here, we execute the script on the training images from the **um_road/** directory which annotated labels are in **enc_gt_image_1/**. The provided output location is a directory **augmented_images** residing in the **data_road/training** folder. Therefore, after the algorithm is completed, we will look for the augmented images there.
- (d) The second arrow (4 in the screenshot) shows the newly-created augmented images' directory. A test is also executed to check the number of labels in **enc_gt_image_1/**. The result indicates that the number of samples has increased by 95 - the exact number of **um_road** images.

```

C:\WINDOWS\system32\cmd.exe
Now we will show the directory structure of KITTI_test

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>dir /s /b /o:n /ad KITTI_test
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\enc_gt_image_1
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\training
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\training\um_road
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\training\umm_road
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\training\uu_road

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>cd KITTI_test\data_road/enc_gt_image_1

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\enc_gt_image_1>dir /a-d | find "File(s)"
384 File(s)          1,264,011 bytes

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\enc_gt_image_1>cd ../../..

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>python data_augmentation.py KITTI_test\data_road/training/um_road
KITTI_test\data_road/enc_gt_image_1 KITTI_test\data_road/training/augmented_images

First 3 image-label pairs to augment:
('um_000000.png', 'um_road_000000.png')
('um_000001.png', 'um_road_000001.png')
('um_000002.png', 'um_road_000002.png')

Augmenting samples... Depending on the number of examples this can take some time.
The average expected execution time is 1 minutes.
100% | 95/95 [00:29<00:00, 3.26it/s]

New files have been created and saved to path KITTI_test\data_road/training/augmented_images

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>dir /s /b /o:n /ad KITTI_test
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\enc_gt_image_1
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\training
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\training\augmented_images
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\training\um_road
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\training\umm_road
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\training\uu_road

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>cd KITTI_test\data_road/enc_gt_image_1

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\enc_gt_image_1>dir /a-d | find "File(s)"
479 File(s)          2,540,087 bytes

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\KITTI_test\data_road\enc_gt_image_1>

```

Figure 2: This figure shows the execution and the output of the data augmentation algorithm.

The effect of the augmentation operation is presented in Figure 3. The image on the left (3a) is the original KITTI image, while the right one (3b) is the augmented version. It is visible how the transformations affected the data - image is rotated, mirrored horizontally and zoomed/cropped.



(a) Original KITTI road image



(b) Augmentation using mirroring, rotation and zoom

Figure 3: The effect of applying data augmentation operations on an image from the KITTI data set.

Note: Currently, augmented samples for all images are included in the data set. The augmented ground-truth masks are located in the **enc_gt_image_1/** directory within **data_road** and have the word ‘*new*’ in their names. The **new_augs/** folder contains all raw images’ augmented versions within **data_road/training/**. Creating new augmented images using this method will overwrite the existing labels as they are always saved to the main label directory. The other augmentations will be stored in a directory of your choice and hence the old ones will not be deleted (unless you specify the same output directory as the current one).

Training

This section describes the process of training a model. It is important to note that this may take a long time if the training set contains a lot of images. To get a better idea of how it is performed, a simple training example will be provided below with step-by-step explanations.

Parameter Configuration

To train a model, we need a configuration object (Python dictionary) that contains parameters like batch size, number of epochs, data set paths and many others. Such a dictionary is created using the **config.py** Python script. One dictionary is also provided with this distribution - **default_config.pickle**. This file is a serialised (*pickled*) version of the configuration dictionary object that can be loaded into any script during runtime. It contains all default values of each parameter which are also available in Figure 4. It is similar to the help information received for the data augmentation script. An example script initiation command is given in the first line of the output of the help request. Definitions for each of the optional arguments are also provided along with their default values.

```
C:\WINDOWS\system32\cmd.exe
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>python config.py -h
usage: config.py [-h] [-i IMSIZE] [-b {2,4,8,16,32}] [-c CLASSES] [-e EPOCHS] [-mv {unet,mob_net}] [-v [0.0, 1.0]]
                [-s STEPS_PER_EPOCH] [-a AUGMENTED_DATA] [-p [0.0, 1.0]] [-d DATA_HOME] [-tr TRAIN_DIR] [-l LABEL_DIR]
                [-tt TEST_DIR] [-ms MODEL_SAVE_DIR] [-cf CONFIG_FILENAME]

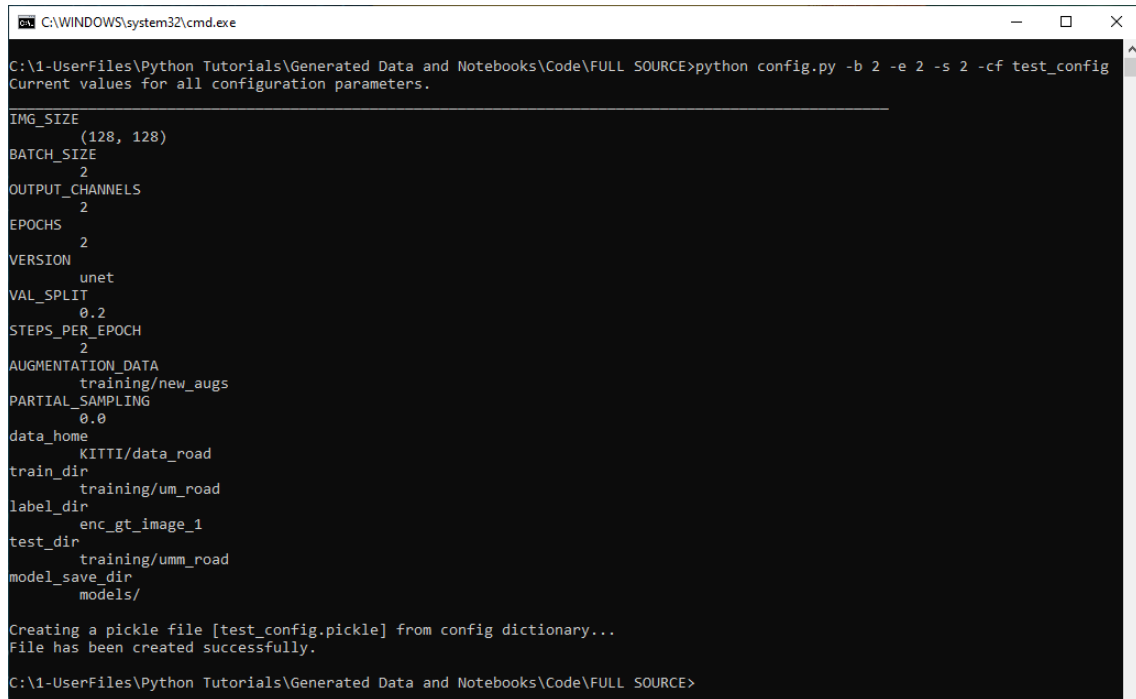
optional arguments:
  -h, --help            show this help message and exit
  -i IMSIZE, --imsize IMSIZE
                        This defines the size of the images inputted to the network. The data must be a tuple of two
                        elements (Width, Height). NOTE: Model will not work if the given size is not compatible with its
                        Input Layer. (default: (128, 128))
  -b {2,4,8,16,32}, --bsize {2,4,8,16,32}
                        The number of samples to load in each batch (the available sizes are shown in curly the brackets
                        - {2,4,...,32}). Depending on the capacity of the GPU or RAM, large "bsize" might terminate
                        execution. (default: 4)
  -c CLASSES, --classes CLASSES
                        Number of object classes the model needs to choose from for each pixel. (default: 2)
  -e EPOCHS, --epochs EPOCHS
                        Number of training epochs. (default: 20)
  -mv {unet,mob_net}, --model_version {unet,mob_net}
                        The model architecture that will be used for training. (default: unet)
  -v [0.0, 1.0], --val_split [0.0, 1.0]
                        What fraction of the data to use for validation during training. (default: 0.2)
  -s STEPS_PER_EPOCH, --steps_per_epoch STEPS_PER_EPOCH
                        Number of batches to iterate over in each epoch. (default: None)
  -a AUGMENTED_DATA, --augmented_data AUGMENTED_DATA
                        Path to the augmented images. The corresponding augmented labels are expected to be in the main
                        ground-truth folder. (default: training/new_augs)
  -p [0.0, 1.0], --partial_sampling [0.0, 1.0]
                        What fraction of the data to use for partial sampling during transfer learning. (default: 0.0)
  -d DATA_HOME, --data_home DATA_HOME
                        Path to directory containing all training and testing sample subdirectories. (default:
                        KITTI/data_road)
  -tr TRAIN_DIR, --train_dir TRAIN_DIR
                        Path to the training images within the "--data_home" directory (default: training/um_road)
  -l LABEL_DIR, --label_dir LABEL_DIR
                        Path to ground-truth labels within "--data_home". This directory must include also the augmented
                        images' labels. (default: enc_gt_image_1)
  -tt TEST_DIR, --test_dir TEST_DIR
                        Path to the testing samples within "--data_home". (default: training/umm_road)
  -ms MODEL_SAVE_DIR, --model_save_dir MODEL_SAVE_DIR
                        Name of the directory where the trained model an its weights will be saved. (default: models/)
  -cf CONFIG_FILENAME, --config_filename CONFIG_FILENAME
                        The name of the file that will be used to save the serialised (pickled) dictionary object. Do
                        not add a file extension - a .pickle one will be added anyway. (default: default_config)

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>
```

Figure 4: This figure shows the help information associated with the **config.py** script.

The generation of a configuration file is demonstrated in Fig. 5. It can be seen how the specified arguments, namely batch size, epochs, steps per epoch and configuration filename, are

different from the default values. We generate a configuration dictionary that will train a U-net model using 128×128 images as input for the neural network. Additionally, the training process will continue for 2 epochs and during each of them, 2 batches containing 2 images will be presented to the model. The resulting model will be saved in the **models/** directory within the current project folder.



```
C:\WINDOWS\system32\cmd.exe

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>python config.py -b 2 -e 2 -s 2 -cf test_config
Current values for all configuration parameters.

IMG_SIZE
(128, 128)
BATCH_SIZE
2
OUTPUT_CHANNELS
2
EPOCHS
2
VERSION
unet
VAL_SPLIT
0.2
STEPS_PER_EPOCH
2
AUGMENTATION_DATA
training/new_augs
PARTIAL_SAMPLING
0.0
data_home
KITTI/data_road
train_dir
training/um_road
label_dir
enc_gt_image_1
test_dir
training/umm_road
model_save_dir
models/

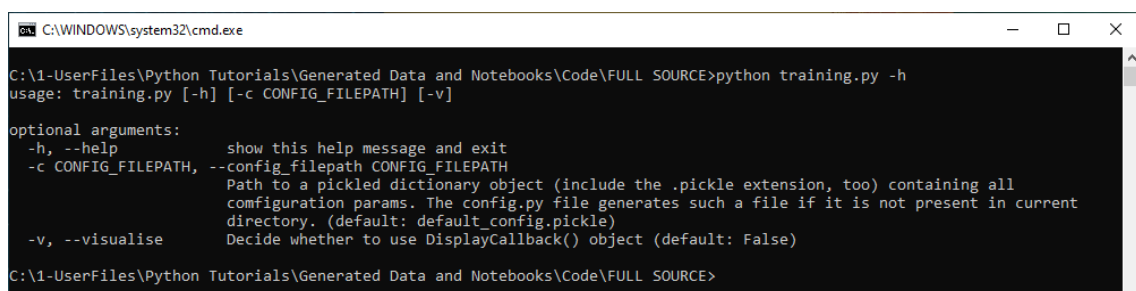
Creating a pickle file [test_config.pickle] from config dictionary...
File has been created successfully.

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>
```

Figure 5: An example execution of the **config.py** file. The different options specified in the command are reflected in given output list.

Model Training

The **training.py** script is responsible for training a model on a given data set. The help information for the script can be accessed using `python training.py -h` and the resulting output is shown in Fig. 6.



```
C:\WINDOWS\system32\cmd.exe

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>python training.py -h
usage: training.py [-h] [-c CONFIG_FILEPATH] [-v]

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG_FILEPATH, --config_filepath CONFIG_FILEPATH
                        Path to a pickled dictionary object (include the .pickle extension, too) containing all
                        configuration params. The config.py file generates such a file if it is not present in current
                        directory. (default: default_config.pickle)
  -v, --visualise        Decide whether to use DisplayCallback() object (default: False)

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>
```

Figure 6: Help information for the **training.py** script file. The ‘-v’ flag if specified, activates a specific callback object (`DisplayCallback()`). It uses the intermediate state of the model after each epoch to predict several test images. This is useful because it provides us with insight into the current development of the model during a certain epoch.

The output produced by the training algorithm is illustrated in Fig. 7 and Fig. 8. The output is separated into two screenshots for clearer visualisation. In the first image, we execute the Python

file from the *Command Prompt* by specifying the test configuration file created during the previous task.

```
C:\WINDOWS\system32\cmd.exe
```

```
C:\Users\UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>python training.py -c test_config.pickle
```

```
Getting image and label filenames and adding them to DataGen's data attribute...
100%|██████████████████████████████████████████████████████████████████████████████| 95/95 [00:00<?, ?it/s]
```

```
Testing for repetitions (not wanted images) in test and train separation from the same directory...
```

```
Generating testing data...
100%|██████████████████████████████████████████████████████████████████████████████| 96/96 [00:00<00:00, 96513.23it/s]
```

```
Result of the test: Passed
```

```
Printing summary for model...
Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 128, 128, 3)]	0	
conv2d (Conv2D)	(None, 128, 128, 64)	1792	input_1[0][0]
conv2d_1 (Conv2D)	(None, 128, 128, 64)	36928	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 64, 64, 128)	73856	max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 128)	147584	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 128)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 32, 32, 256)	295168	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 32, 32, 256)	590080	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 256)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 16, 16, 512)	1180160	max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, 16, 16, 512)	2359808	conv2d_6[0][0]
dropout (Dropout)	(None, 16, 16, 512)	0	conv2d_7[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 512)	0	dropout[0][0]
conv2d_8 (Conv2D)	(None, 8, 8, 1024)	4719616	max_pooling2d_3[0][0]
conv2d_9 (Conv2D)	(None, 8, 8, 1024)	9438208	conv2d_8[0][0]
dropout_1 (Dropout)	(None, 8, 8, 1024)	0	conv2d_9[0][0]
up_sampling2d (UpSampling2D)	(None, 16, 16, 1024)	0	dropout_1[0][0]
conv2d_10 (Conv2D)	(None, 16, 16, 512)	2097664	up_sampling2d[0][0]
concatenate (Concatenate)	(None, 16, 16, 1024)	0	dropout[0][0] conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 16, 16, 512)	4719104	concatenate[0][0]
conv2d_12 (Conv2D)	(None, 16, 16, 512)	2359808	conv2d_11[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 32, 32, 512)	0	conv2d_12[0][0]

Figure 7: This figure presents a screenshot of the first part of **training.py** script's output. Different messages within the output text describe the executed processes and the resulting output.

The different stages of the execution will be briefly described below.

1. The training process begins with the initialisation of a *DataGen* object that will iterate over the images in the directories specified by the paths in the config file. This data generator provides image-label pairs for the training process, creates validation data and testing data generators, normalises images, performs encoding, etc.
2. Tests are executed to check the integrity of the data.
3. The specified model architecture is created and its internal structure defined in terms of layers, output shapes and parameter counts is printed.
4. A prompt requests a name for the model to be inputted. Make sure that the name starts with

‘model’ because the evaluation algorithm looks for it.

5. The training starts and epoch-by-epoch the model is trained. After each iteration over the data, a validation stage is initiated.
6. At the end of the training, the model is saved to the directory from the configuration file with the name specified by us. Additional log information like current date and time, metric values, etc. are also appended to the name to allow uniqueness and prevent overriding.

```
C:\WINDOWS\system32\cmd.exe
conv2d_12 (Conv2D)          (None, 16, 16, 512) 2359808 conv2d_11[0][0]
up_sampling2d_1 (UpSampling2D) (None, 32, 32, 512) 0 conv2d_12[0][0]
conv2d_13 (Conv2D)          (None, 32, 32, 256) 524544 up_sampling2d_1[0][0]
concatenate_1 (Concatenate) (None, 32, 32, 512) 0 conv2d_5[0][0]
conv2d_14 (Conv2D)          (None, 32, 32, 256) 1179904 concatenate_1[0][0]
conv2d_15 (Conv2D)          (None, 32, 32, 256) 590080 conv2d_14[0][0]
up_sampling2d_2 (UpSampling2D) (None, 64, 64, 256) 0 conv2d_15[0][0]
conv2d_16 (Conv2D)          (None, 64, 64, 128) 131200 up_sampling2d_2[0][0]
concatenate_2 (Concatenate) (None, 64, 64, 256) 0 conv2d_3[0][0]
conv2d_17 (Conv2D)          (None, 64, 64, 128) 295040 concatenate_2[0][0]
conv2d_18 (Conv2D)          (None, 64, 64, 128) 147584 conv2d_17[0][0]
up_sampling2d_3 (UpSampling2D) (None, 128, 128, 128) 0 conv2d_18[0][0]
conv2d_19 (Conv2D)          (None, 128, 128, 64) 32832 up_sampling2d_3[0][0]
concatenate_3 (Concatenate) (None, 128, 128, 128) 0 conv2d_1[0][0]
conv2d_20 (Conv2D)          (None, 128, 128, 64) 73792 concatenate_3[0][0]
conv2d_21 (Conv2D)          (None, 128, 128, 64) 36928 conv2d_20[0][0]
conv2d_22 (Conv2D)          (None, 128, 128, 2) 1154 conv2d_21[0][0]
conv2d_23 (Conv2D)          (None, 128, 128, 1) 3 conv2d_22[0][0]
=====
Total params: 31,032,837
Trainable params: 31,032,837
Non-trainable params: 0

Enter a name for the name your trained model will be saved with. (Example: 'model_AUG_um_umm_unet_kitti')
--> model_test

Epoch 1/2
2/2 [=====] - ETA: 0s - loss: 0.4537 - accuracy: 0.8195
Preparing validation generator for 38 validation samples and starting validation...
Finished loading the last batches.
2/2 [=====] - 13s 10s/step - loss: 0.4807 - accuracy: 0.8198 - val_loss: 0.3610 - val_accuracy: 0.8260

Epoch 00001: loss improved from inf to 0.53457, saving model to models\model_test.hdf5
Epoch 2/2
Finished loading the last batches. - ETA: 0s - loss: 0.3541 - accuracy: 0.8210Loading batch 20..
2/2 [=====] - 11s 9s/step - loss: 0.3507 - accuracy: 0.8239 - val_loss: 0.3435 - val_accuracy: 0.8260

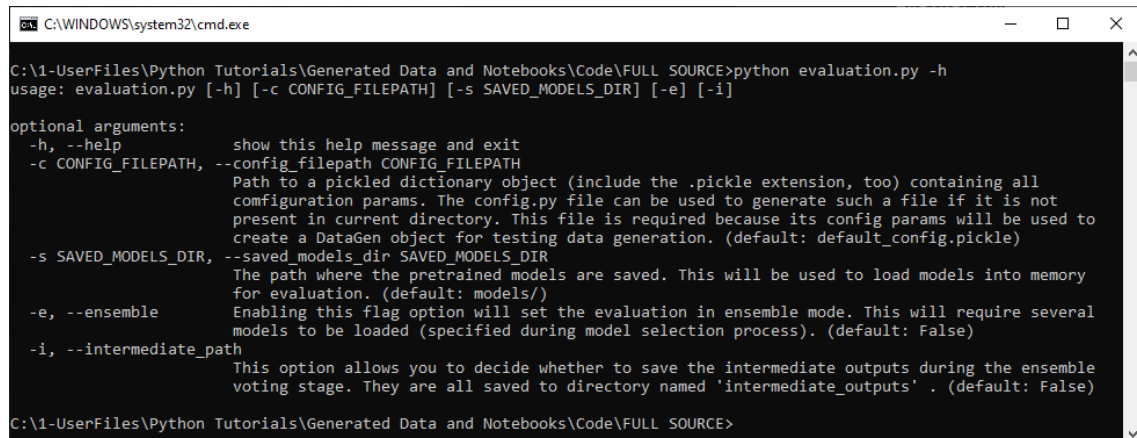
Epoch 00002: loss improved from 0.53457 to 0.34404, saving model to models\model_test.hdf5
Model saved successfully!

Model training finished. You can access the trained model from:
-> models/models/model_test_May-17-2021_13-05-58_imgsize_(128, 128)_epochs_2_val_acc_0.8260_val_loss_0.3435
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>
```

Figure 8: This figure presents a screenshot of the second part of **training.py** script’s output. Different messages within the output text describe the executed processes and the resulting output.

Evaluation

After successfully training and saving a model, it is time to explain how we can evaluate its performance on the testing data. The **evaluation.py** script loads a trained model from a specified directory and generates predictions for the testing data using that model. This file can also perform inference time measurement. Help information is available in Fig. 9.



```
C:\WINDOWS\system32\cmd.exe
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>python evaluation.py -h
usage: evaluation.py [-h] [-c CONFIG_FILEPATH] [-s SAVED_MODELS_DIR] [-e] [-i]

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG_FILEPATH, --config_filepath CONFIG_FILEPATH
                        Path to a pickled dictionary object (include the .pickle extension, too) containing all
                        configuration params. The config.py file can be used to generate such a file if it is not
                        present in current directory. This file is required because its config params will be used to
                        create a DataGen object for testing data generation. (default: default_config.pickle)
  -s SAVED_MODELS_DIR, --saved_models_dir SAVED_MODELS_DIR
                        The path where the pretrained models are saved. This will be used to load models into memory
                        for evaluation. (default: models/)
  -e, --ensemble         Enabling this flag option will set the evaluation in ensemble mode. This will require several
                        models to be loaded (specified during model selection process). (default: False)
  -i, --intermediate_path
                        This option allows you to decide whether to save the intermediate outputs during the ensemble
                        voting stage. They are all saved to directory named 'intermediate_outputs' . (default: False)

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>
```

Figure 9: Help information for the **evaluation.py** script file. The ‘-e’ flag, if specified, indicates that several models will be used as a joint ensemble to generate segmentation masks for the test images. The ‘-i’ flag defines whether the script should save the intermediate outputs of the ensemble algorithm - these include each ensemble model’s individual prediction, their pixel-wise summed predictions and the final average predicted mask.

The process of evaluation will be described similarly to the training pipeline. The output of the execution is provided in Fig. 10, Fig. 11, Fig. 12, respectively. Three screenshots of different sections of the output are presented in these figures.

The first one (10) shows the execution command and the choice of a model to use for inference. The program lists all available models and waits for user input. The selected model (the pretrained ‘v3’ model) is loaded into memory and a summary of its structure is printed for reference.

In the second part (11), the user is prompted to decide whether they want to measure accurate inference time. If this option is selected, the model will not produce any visual elements as output (as in real-time applications this is not needed). After the user skips the timing option, they are asked to choose how many testing images should be predicted. We choose 1 here for the sake of this example. The mask is predicted and overlaid on top of the real image along with the corresponding ground-truth label. The large overlap is visible (pink colour). The red pixels indicate missed road pixels, while the blue region means that the model incorrectly assumed that this is road. The result is shown on the screen using the *matplotlib* Python package. The prediction of the model trained in this example is also shown in Fig. 13.

In the final part of the output (12), the user is prompted whether they would like to save all the overlays and the associated predicted and true masks to a directory. Fig. 14 shows the contents of the save directory after the predictions have been generated and exported as PNG files.


```

C:\WINDOWS\system32\cmd.exe - python evaluation.py -c test_config.pickle -s models/
C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>python evaluation.py -c test_config.pickle -s models/
1 - model_AUG_um_umm_unet_v3_kitti_Mar-25-2021_13-54-36_imgsize_(128, 128)_epochs_20_val_acc_0.9590_val_loss_0.1268
2 - model_test_May-17-2021_13-05-58_imgsize_(128, 128)_epochs_2_val_acc_0.8260_val_loss_0.3435
Choose a model from here and type its index in the input space below:
Type choice here -> 1

Model loaded successfully. Printing model summary...

Model: "model_2"

```

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[None, 128, 128, 3]	0	
conv2d_48 (Conv2D)	(None, 128, 128, 64)	1792	input_3[0][0]
conv2d_49 (Conv2D)	(None, 128, 128, 64)	36928	conv2d_48[0][0]
max_pooling2d_8 (MaxPooling2D)	(None, 64, 64, 64)	0	conv2d_49[0][0]
conv2d_50 (Conv2D)	(None, 64, 64, 128)	73856	max_pooling2d_8[0][0]
conv2d_51 (Conv2D)	(None, 64, 64, 128)	147584	conv2d_50[0][0]
max_pooling2d_9 (MaxPooling2D)	(None, 32, 32, 128)	0	conv2d_51[0][0]
conv2d_52 (Conv2D)	(None, 32, 32, 256)	295168	max_pooling2d_9[0][0]
conv2d_53 (Conv2D)	(None, 32, 32, 256)	590080	conv2d_52[0][0]
max_pooling2d_10 (MaxPooling2D)	(None, 16, 16, 256)	0	conv2d_53[0][0]
conv2d_54 (Conv2D)	(None, 16, 16, 512)	1180160	max_pooling2d_10[0][0]
conv2d_55 (Conv2D)	(None, 16, 16, 512)	2359808	conv2d_54[0][0]
dropout_4 (Dropout)	(None, 16, 16, 512)	0	conv2d_55[0][0]
max_pooling2d_11 (MaxPooling2D)	(None, 8, 8, 512)	0	dropout_4[0][0]
conv2d_56 (Conv2D)	(None, 8, 8, 1024)	4719616	max_pooling2d_11[0][0]
conv2d_57 (Conv2D)	(None, 8, 8, 1024)	9438208	conv2d_56[0][0]
dropout_5 (Dropout)	(None, 8, 8, 1024)	0	conv2d_57[0][0]
up_sampling2d_8 (UpSampling2D)	(None, 16, 16, 1024)	0	dropout_5[0][0]
conv2d_58 (Conv2D)	(None, 16, 16, 512)	2097664	up_sampling2d_8[0][0]
concatenate_8 (Concatenate)	(None, 16, 16, 1024)	0	dropout_4[0][0] conv2d_58[0][0]
conv2d_59 (Conv2D)	(None, 16, 16, 512)	4719104	concatenate_8[0][0]
conv2d_60 (Conv2D)	(None, 16, 16, 512)	2359808	conv2d_59[0][0]
up_sampling2d_9 (UpSampling2D)	(None, 32, 32, 512)	0	conv2d_60[0][0]
conv2d_61 (Conv2D)	(None, 32, 32, 256)	524544	up_sampling2d_9[0][0]
concatenate_9 (Concatenate)	(None, 32, 32, 512)	0	conv2d_53[0][0]

Figure 10: This figure presents a screenshot of the first part of **evaluation.py** script's output. Different messages within the output text describe the executed processes and the resulting output.

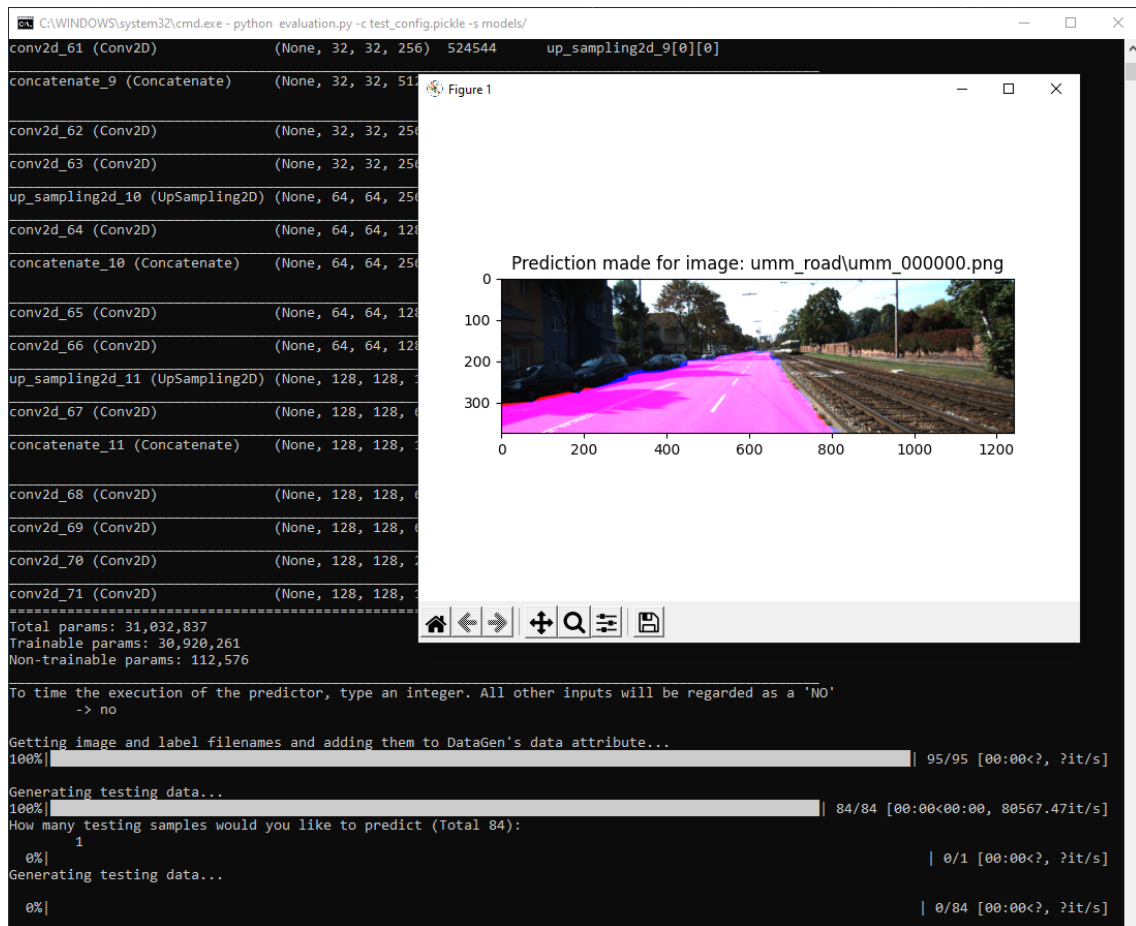


Figure 11: This figure presents a screenshot of the second part of `evaluation.py` script's output. Different messages within the output text describe the executed processes and the resulting output.

```

C:\WINDOWS\system32\cmd.exe

conv2d_63 (Conv2D)          (None, 32, 32, 256) 590080 conv2d_62[0][0]
up_sampling2d_10 (UpSampling2D) (None, 64, 64, 256) 0 conv2d_63[0][0]
conv2d_64 (Conv2D)          (None, 64, 64, 128) 131200 up_sampling2d_10[0][0]
concatenate_10 (Concatenate) (None, 64, 64, 256) 0 conv2d_51[0][0]
conv2d_65 (Conv2D)          (None, 64, 64, 128) 295040 concatenate_10[0][0]
conv2d_66 (Conv2D)          (None, 64, 64, 128) 147584 conv2d_65[0][0]
up_sampling2d_11 (UpSampling2D) (None, 128, 128, 128) 0 conv2d_66[0][0]
conv2d_67 (Conv2D)          (None, 128, 128, 64) 32832 up_sampling2d_11[0][0]
concatenate_11 (Concatenate) (None, 128, 128, 128) 0 conv2d_49[0][0]
conv2d_68 (Conv2D)          (None, 128, 128, 64) 73792 concatenate_11[0][0]
conv2d_69 (Conv2D)          (None, 128, 128, 64) 36928 conv2d_68[0][0]
conv2d_70 (Conv2D)          (None, 128, 128, 2) 1154 conv2d_69[0][0]
conv2d_71 (Conv2D)          (None, 128, 128, 1) 3 conv2d_70[0][0]
=====
Total params: 31,032,837
Trainable params: 30,920,261
Non-trainable params: 112,576

To time the execution of the predictor, type an integer. All other inputs will be regarded as a 'NO'
-> no

Getting image and label filenames and adding them to DataGen's data attribute...
100%| 95/95 [00:00<?, ?it/s]

Generating testing data...
100%| 84/84 [00:00<00:00, 80567.47it/s]
How many testing samples would you like to predict (Total 84):
1
0%| 0/1 [00:00<?, ?it/s]
Generating testing data...
100%| 1/1 [00:52<00:00, 52.82s/it]

Would you like to save the predicted images in a directory? (Y\n)
Typing anything else will abort this functionality.
y
Now, input a valid directory name to save the images in.
test predictions
Saving all images to "test predictions" now...
100%| 1/1 [00:00<00:00, 6.15it/s]

Average time taken for prediction in seconds: 52.82
The given time includes various transforms and plotting operations.
If an accurate result is sought that measures only the inference time
and its corresponding image pre-processing, consider requesting timing at the beginning.

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE>

```

Figure 12: This figure presents a screenshot of the third part of **evaluation.py** script's output. Different messages within the output text describe the executed processes and the resulting output.

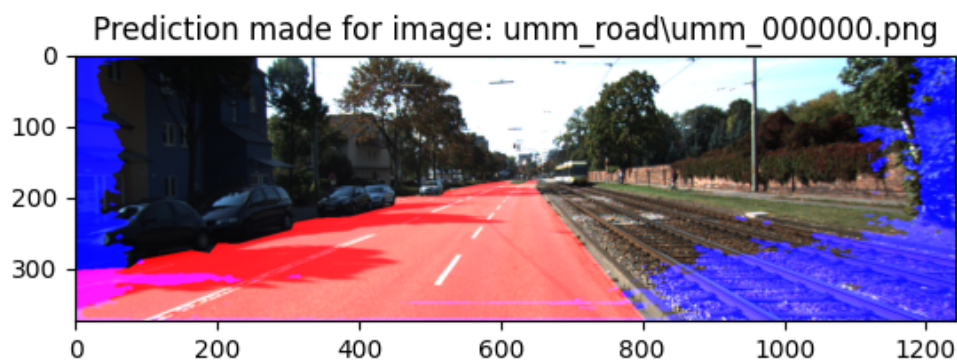


Figure 13: The result of predicting a test image (*umm_000000.png*) using the test model trained in this example.

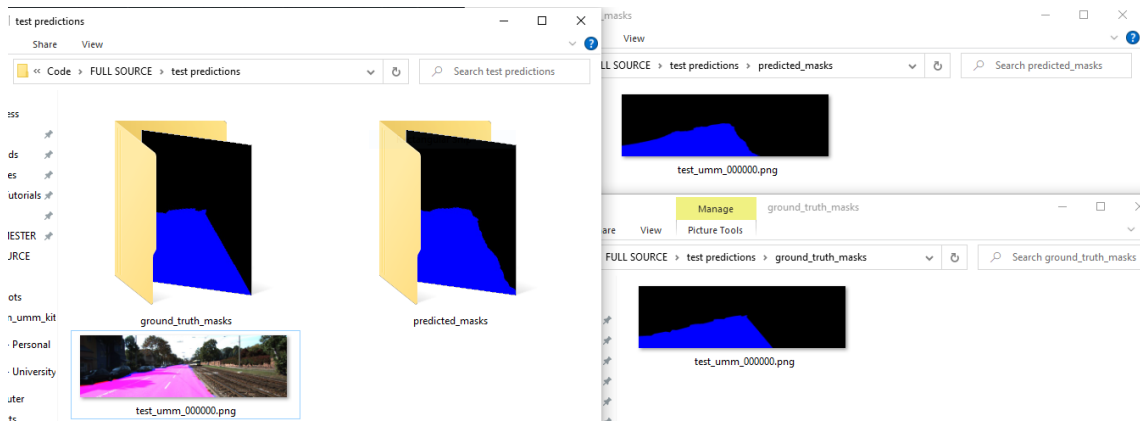


Figure 14: The contents of the directory where the predictions of a model are saved. The overlay and the corresponding true and predicted segmentation masks for each predicted image are saved accordingly.

Metric Calculation and Model Comparison

In order to evaluate the models' performance and their differences. We need to generate overlays and predicted masks using the **evaluation.py** script. At least two images per model are required so that the statistical operations and tests (e.g., mean, standard deviation, t-test) can be executed.

For this example, the predictions of the trained model for 3 test images were generated and saved in a directory called **test predictions_pred_imgs/**. The name of each directory that contains samples to evaluate must include the sequence 'pred_imgs'. Before running the metric calculation script, we have to copy the **test predictions_pred_imgs/** directory to the corresponding location. The specific folder is called 'ALL PREDICTED IMAGES' and it resides in the **metric_calculation** directory within the project repository. This is demonstrated in Fig. 15.

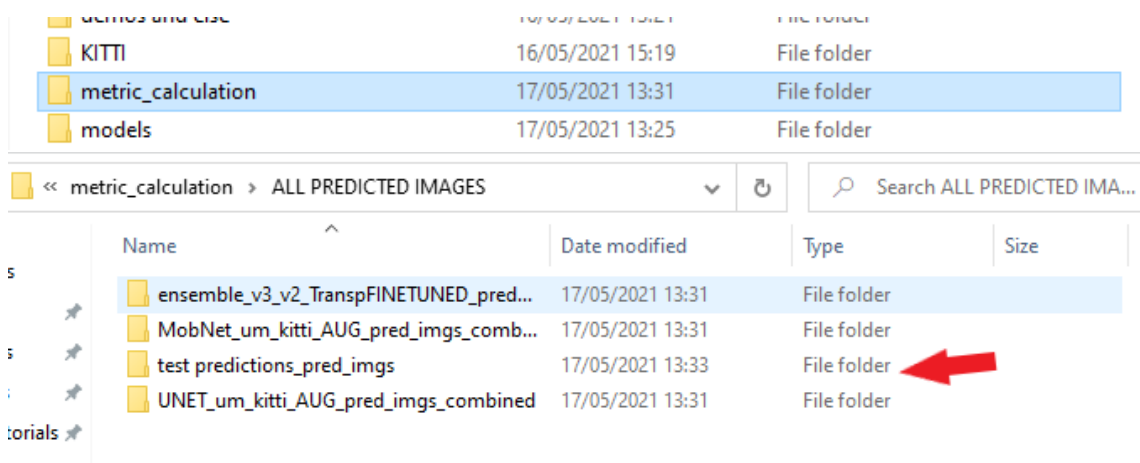
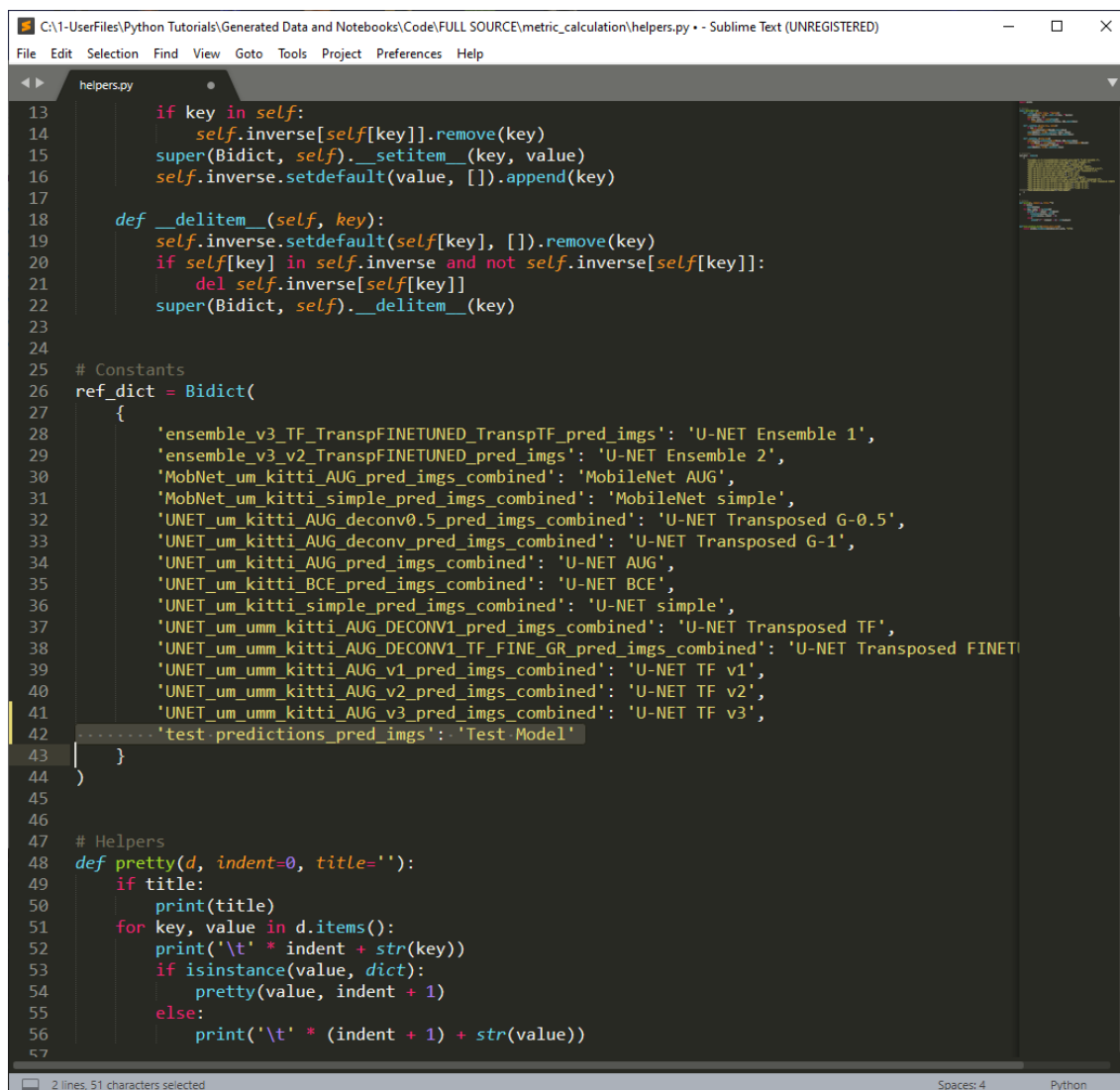


Figure 15: This figure shows the metric calculation directory to which the predicted masks have to be copied prior to score calculation and comparison. The red arrow on the second image points to the directory containing our test model's predictions.

The last preparation step before executing the algorithms is shown in Fig. 16. We have to specify a name for each model whose predictions are contained in the 'ALL PREDICTED IMAGES' directory. That is why we open the **helpers.py** file with any text editor and type this at

the end of the dictionary definition as shown in the figure.



```

13     if key in self:
14         self.inverse[self[key]].remove(key)
15     super(Bidict, self).__setitem__(key, value)
16     self.inverse.setdefault(value, []).append(key)
17
18     def __delitem__(self, key):
19         self.inverse.setdefault(self[key], []).remove(key)
20         if self[key] in self.inverse and not self.inverse[self[key]]:
21             del self.inverse[self[key]]
22         super(Bidict, self).__delitem__(key)
23
24
25 # Constants
26 ref_dict = Bidict(
27     {
28         'ensemble_v3_TF_TranspFINETUNED_TranspTF_pred_imgs': 'U-NET Ensemble 1',
29         'ensemble_v3_v2_TranspFINETUNED_pred_imgs': 'U-NET Ensemble 2',
30         'MobNet_um_kitti_AUG_pred_imgs_combined': 'MobileNet AUG',
31         'MobNet_um_kitti_simple_pred_imgs_combined': 'MobileNet simple',
32         'UNET_um_kitti_AUG_deconv0.5_pred_imgs_combined': 'U-NET Transposed G-0.5',
33         'UNET_um_kitti_AUG_deconv_pred_imgs_combined': 'U-NET Transposed G-1',
34         'UNET_um_kitti_AUG_pred_imgs_combined': 'U-NET AUG',
35         'UNET_um_kitti_BCE_pred_imgs_combined': 'U-NET BCE',
36         'UNET_um_kitti_simple_pred_imgs_combined': 'U-NET simple',
37         'UNET_um_umm_kitti_AUG_DECONV1_pred_imgs_combined': 'U-NET Transposed TF',
38         'UNET_um_umm_kitti_AUG_DECONV1_TF_FINE_GR_pred_imgs_combined': 'U-NET Transposed FINET',
39         'UNET_um_umm_kitti_AUG_v1_pred_imgs_combined': 'U-NET TF v1',
40         'UNET_um_umm_kitti_AUG_v2_pred_imgs_combined': 'U-NET TF v2',
41         'UNET_um_umm_kitti_AUG_v3_pred_imgs_combined': 'U-NET TF v3',
42         ..... 'test-predictions_pred_imgs': 'Test-Model'
43     }
44 )
45
46
47 # Helpers
48 def pretty(d, indent=0, title=''):
49     if title:
50         print(title)
51     for key, value in d.items():
52         print('\t' * indent + str(key))
53         if isinstance(value, dict):
54             pretty(value, indent + 1)
55         else:
56             print('\t' * (indent + 1) + str(value))
57

```

Figure 16: This is a very important step that specifies to the metric calculation algorithm what name to assign to the scores calculated for the given predictions directory.

The score calculation process compares all segmentation masks and generates various tables. The printed output (Fig. 17, Fig. 18) shows the computed scores for the given images. At the end of the execution, all the scores are added to a dictionary object which is serialised to a *'pickle'* file. This file will be used in the next stage, where the scores will be compared and plotted to graphs. This algorithm produces several Excel tables and text files, all containing the scores in different formats. Some contain t-test results and p-values, while others percentage differences in performance between models across all metrics. Figure 19 shows the contents of the produced MS Excel files and the rest of the elements in the output directory.

```

C:\WINDOWS\system32\cmd.exe

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\metric_calculation>python score_calc_and_save_to_pickle.py
1. Iterating over directory: 'ensemble_v3_v2_TranspFINETUNED_pred_imgs'
2. Iterating over directory: 'MobNet_um_kitti_AUG_pred_imgs_combined'
3. Iterating over directory: 'test_predictions_pred_imgs'
4. Iterating over directory: 'UNET_um_kitti_AUG_pred_imgs_combined'
All calculations have finished.
Mean scores for each metric and model
TP
[('U-NET Ensemble 2', 104183.73958333333), ('MobileNet AUG', 88960.94791666667), ('Test Model', 102543.33333333333), ('U-NET AUG', 96355.55208333333)]
FP
[('U-NET Ensemble 2', 344942.19791666667), ('MobileNet AUG', 348885.33333333333), ('Test Model', 347137.66666666667), ('U-NET AUG', 347516.72916666667)]
FN
[('U-NET Ensemble 2', 11289), ('MobileNet AUG', 7346), ('Test Model', 8708), ('U-NET AUG', 8715)]
Recall
[('U-NET Ensemble 2', 4493), ('MobileNet AUG', 19716), ('Test Model', 7360), ('U-NET AUG', 12321)]
Precision
[('U-NET Ensemble 2', 0.9038924779300219), ('MobileNet AUG', 0.9234782758237681), ('Test Model', 0.9182558443460728), ('U-NET AUG', 0.9177957887828921)]
PA
[('U-NET Ensemble 2', 0.9625286647886338), ('MobileNet AUG', 0.8221113840727676), ('Test Model', 0.9288512388769478), ('U-NET AUG', 0.8917329237158665)]
F1
[('U-NET Ensemble 2', 0.9660751019863338), ('MobileNet AUG', 0.9417850581985295), ('Test Model', 0.9654986580783682), ('U-NET AUG', 0.9547538380215052)]
IoU
[('U-NET Ensemble 2', 0.6121367715228694), ('MobileNet AUG', 0.5879546872178224), ('Test Model', 0.6143708920621159), ('U-NET AUG', 0.6028321952696778)]
[('U-NET Ensemble 2', 0.8721931997316855), ('MobileNet AUG', 0.7675202214868547), ('Test Model', 0.8650984969856654), ('U-NET AUG', 0.8241015673141534)]

Calculating percentage difference between best model for metric and all others for all metrics
[('U-NET Ensemble 2', 0.0), ('Test Model', 1.5997280370575942), ('U-NET AUG', 0.124272375326928), ('MobileNet AUG', 17.11176872904554)]
[('MobileNet AUG', 0.0), ('U-NET AUG', 0.3938239663881776), ('Test Model', 0.5934506002901715), ('U-NET Ensemble 2', 1.1431293244148795)]
[('U-NET Ensemble 2', 0.0), ('U-NET AUG', 29.535283993115318), ('Test Model', 29.63941203491043), ('MobileNet AUG', 53.67546964334332)]
[('MobileNet AUG', 0.0), ('U-NET AUG', 68.01947893839785), ('Test Model', 167.8804347826087), ('U-NET Ensemble 2', 338.81593590028933)]
[('MobileNet AUG', 0.0), ('Test Model', 0.5807338131144655), ('U-NET AUG', 0.6391450331681725), ('U-NET Ensemble 2', 2.2573322657350703)]
[('U-NET Ensemble 2', 0.0), ('Test Model', 3.6257071554811766), ('U-NET AUG', 7.939119338488493), ('MobileNet AUG', 17.080079823282002)]
[('U-NET Ensemble 2', 0.0), ('Test Model', 0.85970426816676527), ('U-NET AUG', 1.1857783141557463), ('MobileNet AUG', 2.5791494116785967)]
[('Test Model', 0.0), ('U-NET Ensemble 2', 0.36497081096574446), ('U-NET AUG', 1.9148810465964375), ('MobileNet AUG', 4.492898078471646)]
[('U-NET Ensemble 2', 0.0), ('Test Model', 0.8201034640844803), ('U-NET AUG', 5.835043848399482), ('MobileNet AUG', 13.637813742816615)]
The results from the paired t-test for models U-NET Ensemble 2 and MobileNet AUG for TP metric are:
Ttest_relResult(statistic=-10.299797615951467, pvalue=3.7855244760007076e-17)
The results from the paired t-test for models U-NET Ensemble 2 and MobileNet AUG for FN metric are:
Ttest_relResult(statistic=-4.12854691651339, pvalue=7.842005537110486e-05)
The results from the paired t-test for models U-NET Ensemble 2 and MobileNet AUG for FP metric are:
Ttest_relResult(statistic=4.12854691651339, pvalue=7.842005537110486e-05)
The results from the paired t-test for models U-NET Ensemble 2 and MobileNet AUG for FN metric are:

```

Figure 17: Output of the metric calculation algorithm. The various scores for each model are printed out, as well as, the model-wise comparison scores.

```

C:\WINDOWS\system32\cmd.exe

[17.080079823282002, 3.6257071554811766, 7.939119338408493, -11.491598475256035, -7.807442990020733, 4.162492398199439]
PA
[2.5791494116785967, 0.05970426816676527, 1.1857783141557463, -2.45609868863369, -1.3583375427964, 1.1254021328816108]
F1
[4.112916323445879, -0.3636436179044534, 1.5434769951245177, -4.299716211428636, -2.4679352178925877, 1.9140810465964375]
IoU
[13.637813742816615, 0.8201034640844063, 5.835643848399482, -11.279441108591772, -6.865821892767918, 4.9747423494322405]

Creating pickle...
Pickle saved to [OUTPUT FILES\score_dict_with_percentage_diffs_and_ttest_results.pickle] successfully

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\metric_calculation>

```

Figure 18: The end of the output of the metric calculation algorithm. All scores are saved to a .pickle file.

Model Names	TP	TN	FP	FN	Recall	Precision	PA	F1	IoU
U-NET Ensemble 2: MobileNet AUG	3.79E-17	7.84E-05	7.84E-05	3.79E-17	0.005404	66-18	6.12E-10	5.2E-06	2.74E-12
U-NET Ensemble 2: Test Model	0.773354	0.237461	0.957455	0.966718	0.996899	0.973081	0.963635	0.990047	0.982857
U-NET Ensemble 2: U-NET AUG	9.98E-10	0.000631	0.000631	9.98E-10	0.004042	2.49E-10	3.38E-05	0.00959	3.43E-06
MobileNet AUG: Test Model	0.276229	0.762798	0.026117	0.139405	0.097763	0.189808	0.111631	0.129425	0.145832
MobileNet AUG: U-NET AUG	7.62E-09	0.083966	0.082066	7.62E-09	0.393214	1.06E-08	1.61E-06	0.000128	1.34E-07
Test Model: U-NET AUG	0.237064	0.160616	0.663238	0.096902	0.85927	0.107579	0.332323	0.465801	0.273007

Figure 19: The files produced by the score calculation script. There are two MS Excel tables that contain statistical differences and t-test results for all 2-model combinations.

After these files have been produced, we can use the second script in the **metric_calculation** directory - **score_compare.py**. It is responsible for loading the saved pickled dictionary and creating graphs from the scores where the models are plotted against two metrics. Each time a plot is created, it is shown on the screen and then saved to the 'Figures and Graphs' directory. This file also produces an Excel table where the mean scores of all compared models for all metrics are

shown. The sequence of screenshots in Figure 20, Figure 21 and Figure 22 show the discussed outputs and created files.

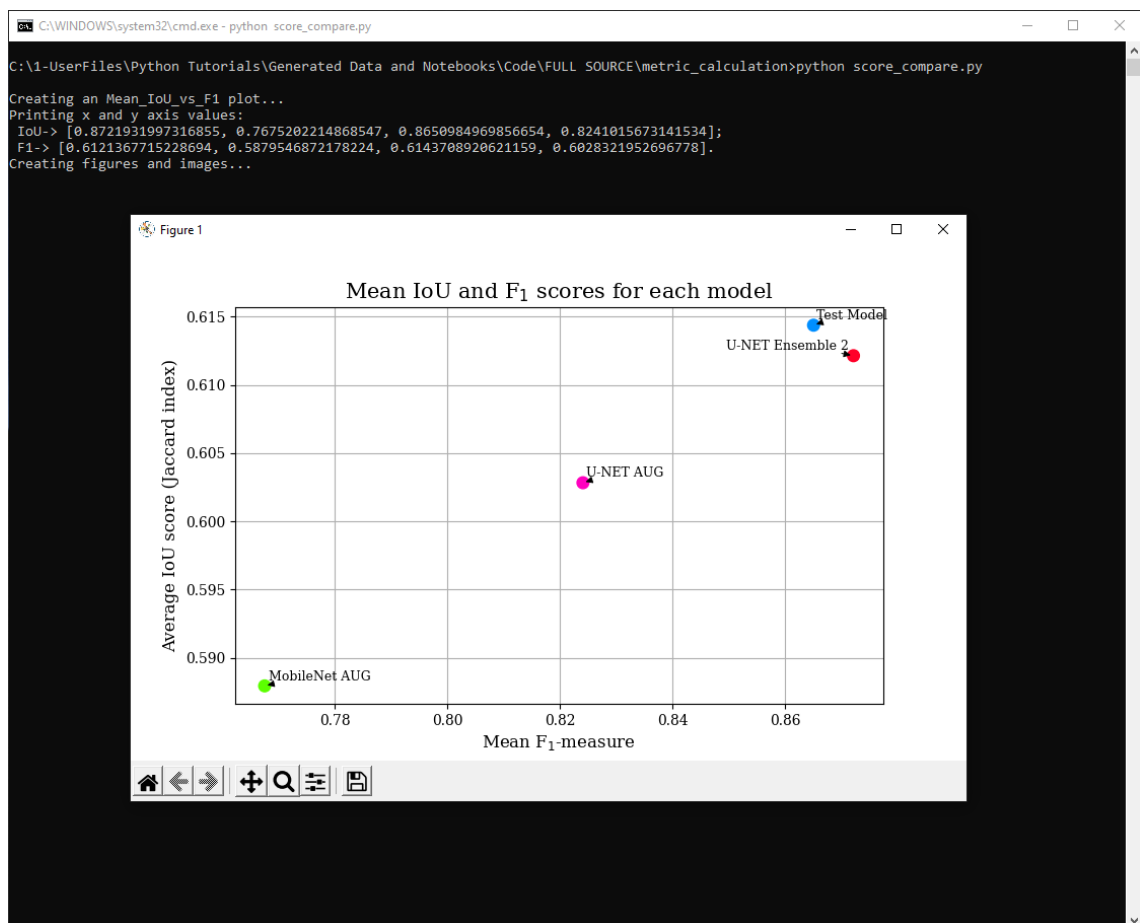


Figure 20: This image shows the first graph that is produced by the `score_compare.py` script. It compares the provided models based on their IoU and F_1 scores.


```

C:\WINDOWS\system32\cmd.exe

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\metric_calculation>python score_compare.py

Creating an Mean_IoU_vs_F1 plot...
Printing x and y axis values:
IoU-> [0.8721931997316855, 0.7675202214868547, 0.8650984969856654, 0.8241015673141534];
F1-> [0.6121367715228694, 0.5879546872178224, 0.6143708920621159, 0.6028321952696778].
Creating figures and images...

Creating an Mean_TP_vs_TN plot...
Printing x and y axis values:
TP-> [104183.73958333333, 88960.94791666667, 102543.33333333333, 96355.55208333333];
TN-> [344942.19791666667, 348885.33333333333, 347137.6666666667, 347516.72916666667].
Creating figures and images...

Creating an Mean_FP_vs_FN plot...
Printing x and y axis values:
FP-> [11289, 7346, 8708, 8715];
FN-> [4493, 19716, 7360, 12321].
Creating figures and images...

Creating an Excel table from all scores...
Model Names
Recall ['U-NET Ensemble 2', 'MobileNet AUG', 'Test Model', 'U-NET AUG']
Precision [0.9030924779300219, 0.9234782758237681, 0.9182558443460728, 0.9177957887828921]
PA [0.9625286647086848, 0.8221113840727676, 0.9288512388769478, 0.8917329237150665]
F1 [0.9660751019863338, 0.9417850581985295, 0.9654986580783682, 0.9547538380215052]
IoU [0.6121367715228694, 0.5879546872178224, 0.6143708920621159, 0.6028321952696778]
File already exists. Change passed filename to method or move the existing file away

Printing difference significance between
Test Model and U-NET AUG accross all metrics...
TP Ttest_relResult(statistic=1.668982594720064, pvalue=0.23706416234017685)
TN Ttest_relResult(statistic=-2.183962237910505, pvalue=0.16061597394138244)
FP Ttest_relResult(statistic=-0.5057969834536353, pvalue=0.6632380665506703)
FN Ttest_relResult(statistic=-2.9740793807978934, pvalue=0.09690214399449176)
Recall Ttest_relResult(statistic=-0.2010232130179597, pvalue=0.8592697519479033)
Precision Ttest_relResult(statistic=-2.7971314362754565, pvalue=0.10757869632837663)
PA Ttest_relResult(statistic=-1.2683683967043293, pvalue=0.33232285684928664)
F1 Ttest_relResult(statistic=-0.8941386307137615, pvalue=0.46560071017478744)
IoU Ttest_relResult(statistic=-1.4973125035313637, pvalue=0.2730074702499615)

C:\1-UserFiles\Python Tutorials\Generated Data and Notebooks\Code\FULL SOURCE\metric_calculation>

```

Figure 21: This is the rest of the output of the `score_compare.py` script, which illustrates the generation of two more figures and a table of all mean scores.

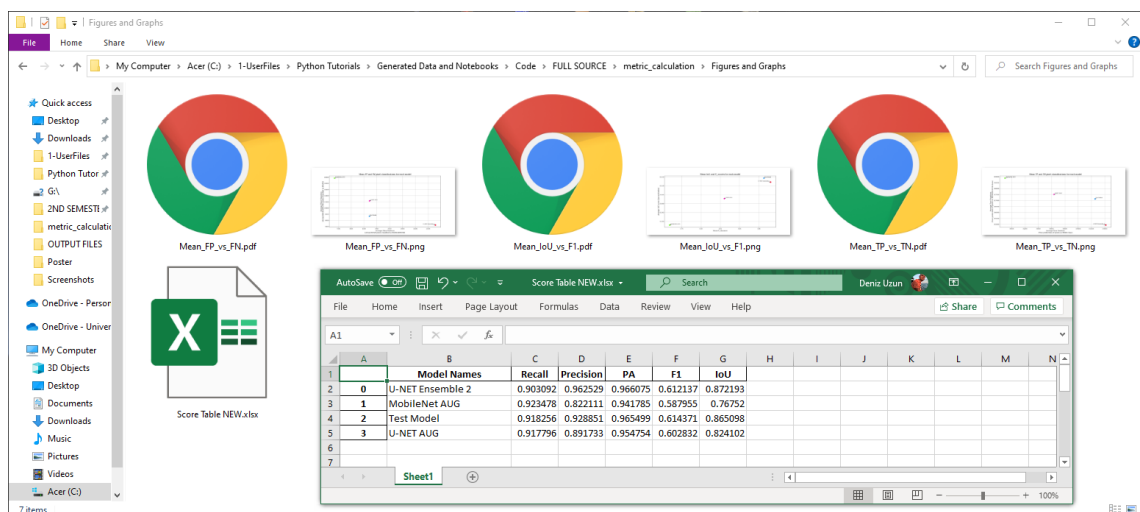


Figure 22: Here, the saved figures (in PDF and PNG format) by the `score_compare.py` script are shown in the directory along with the results summary table.)

Finally, this example will demonstrate the use of the ensemble functionality of the **evaluation.py** script. Refer to Fig. 9 for a refresher of the requirements for executing this program. The example displayed in Fig. 23 shows the interface of the ensemble evaluation algorithm. If the ‘-i’ option was included in the execution command, the model would have produced a directory called **‘intermediate_outputs’** which would contain each model’s individual prediction, the sum of all predictions and the averaged voted instance.

```
C:\WINDOWS\system32\cmd.exe
C:\1-UserFiles\Desktop\test\FULL SOURCE>python evaluation.py -e
    1 - model_AUG_um_umm_unet_v3_kitti_Mar-25-2021_13-54-36_imgsize_(128, 128)_epochs_20_val_acc_0.9590_val_loss_0.1268
    2 - model_test_May-17-2021_13-05-58_imgsize_(128, 128)_epochs_2_val_acc_0.8260_val_loss_0.3435
Choose models from here that will be loaded for ensemble.
Type their indices separated by spaces here:
-> 1 2

Loading all models...
100%|██████████████████████████████████████████████████████████████████████████████| 2/2 [00:09<00:00, 4.99s/it]
All models have been loaded successfully.

To time the execution of the predictor, type an integer. All other inputs will be regarded as a 'NO'
-> n

Getting image and label filenames and adding them to DataGen's data attribute...
100%|██████████████████████████████████████████████████████████████████████████████| 95/95 [00:00<?, ?it/s]

Generating testing data...
100%|██████████████████████████████████████████████████████████████████████████████| 96/96 [00:00<?, ?it/s]
How many testing samples would you like to predict (Total 96):
1
0%|██████████████████████████████████████████████████████████████████████████████| 0/1 [00:00<?, ?it/s]

Generating testing data...
0%|██████████████████████████████████████████████████████████████████████████████| 0/96 [00:00<?, ?it/s] C
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
100%|██████████████████████████████████████████████████████████████████████████████| 1/1 [01:39<00:00, 99.78s/it]

Would you like to save the predicted images in a directory? (Y\N)
Typing anything else will abort this functionality.
n

Average time taken for prediction in seconds: 86.92
The given time includes various transforms and plotting operations.
If an accurate result is sought that measures only the inference time
and its corresponding image pre-processing, consider requesting timing at the begining.

C:\1-UserFiles\Desktop\test\FULL SOURCE>
```

Figure 23: In this image, the evaluation script is executed for an ensemble of models. The default directory, **models/**, is checked for available models. The user is prompted to select the trained models they want to use for the ensemble scheme. The rest of the process is identical to that of normal evaluation for a single network.