Configuration Manual

Practicum
M.Sc. in Artificial Intelligence

Utsav Pataskar
Student ID: 23195398

School of Computing
National College of Ireland

Supervisor: Kislay Raj

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Utsav Pataskar
Student ID:	23195398
Programme:	M.Sc. in Artificial Intelligence
Year:	2024
Module:	Practicum
Supervisor:	Kislay Raj
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	1340
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Utsav Pataskar
Date:	12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Utsav Pataskar
23195398

1 Introduction

This study highlights the dataset description, hardware and software specifications and deployment steps in Jupyter Notebook and related python files executions in order to recreate the results put forth done in the research of "Depth Estimation for indoor environments using Augmented and Regularized Data through Knowledge Distillation"

2 Project Overview

The project uses teacher student knowledge distillation framework to utilize the capabilities of outdoor depth estimation model - monodepth2 (Kumar et al.; 2024) to instill it's learning into the student model (DenseNet) when worked with monocular indoor images. Various implementation phases like pre-processing, feature extraction using auto-encoders, modeling DenseNet with auto-encoder and evaluating depth prediction output put forth my the student model.

3 System Requirements

3.1 Hardware Specification

The following is minimum hardware requirement for code execution:

- Processor: 11^{th} Gen Intel(R) - i7 11800H @ 2.30GHz, 2304Mhz
- Cores: 8
- Logical Processors: 16
- RAM: 16GB or more
- GPU Specs: 6GB - NVIDIA GeForce RTX 3060, Driver Version - 566.03 with CUDA 12.7 support
- Machine Harddisk: 1TB SSD

3.2 Software Specification

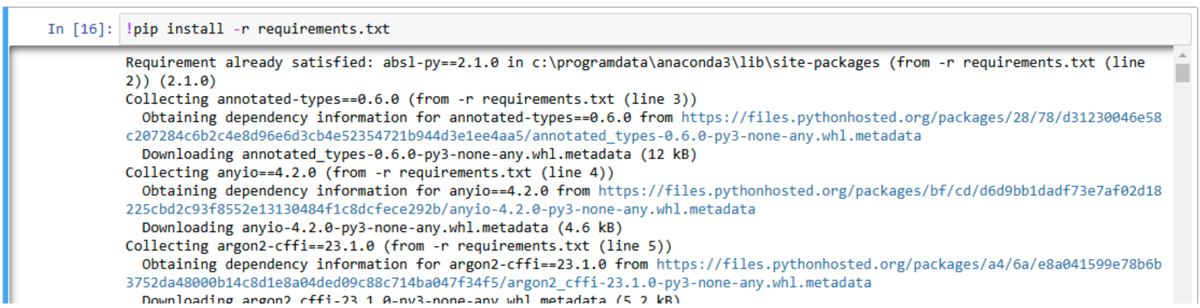
Listed below are the software used in the project execution:

- **Programming Language:**
  Python - The primary language used for programming the model and it's execution

- **Python Libraries:**
  - NumPy: For numeric operations when images are converted into numeric arrays
  - PIL(Pillow): For Image manipulation and enhancements
  - TorchVision: Ensures pre-processing consistency and normalization to a scale of 0 to 1
  - MatPlotLib: For Visualization and plotting graphs of loss function and side by side comparison of expected and actual results.
  - h5py: To read the .mat (MATLAB) file containing NYU-DepthV2 indoor dataset.
  - TensorFlow.keras: To aid in constructing neural network, data feed creation for model training, creating different layers, feature extraction on each layer and merging different feature maps for complex learning mappings.
  - cv2: for image reading, processing and saving operations
  - mono_640x192: ResNet pre-trained weights on outdoor images and related libraries as teacher model and related files.
  - os: for path related operation to remove dependencies on Linux path slash and Window path slash.
  - logging: To log the process information into various log files.

- **Development Environment:**
  Jupyter Notebook 6 or above (Notebook Environment package details added in the Code Artifacts)

- **Library installation:**
  A requirement.txt file has been shared with the Code Artifacts. We can use **!pip install -r requirements.txt** command within jupyter notebook. The output of installation looks similar to the following image. Ref 1



```
In [16]: !pip install -r requirements.txt
Requirement already satisfied: absl-py==2.1.0 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line
2)) (2.1.0)
Collecting annotated-types==0.6.0 (from -r requirements.txt (line 3))
  Obtaining dependency information for annotated-types==0.6.0 from https://files.pythonhosted.org/packages/28/78/d31230046e58
c207284c6b2c4e8d96e6d3cb4e52354721b944d3e1ee4aa5/annotated_types-0.6.0-py3-none-any.whl.metadata
  Downloading annotated_types-0.6.0-py3-none-any.whl.metadata (12 kB)
Collecting anyio==4.2.0 (from -r requirements.txt (line 4))
  Obtaining dependency information for anyio==4.2.0 from https://files.pythonhosted.org/packages/bf/cd/d6d9bb1dadf73e7af02d18
225cbd2c93f8552e13130484f1c8dcfece292b/anyio-4.2.0-py3-none-any.whl.metadata
  Downloading anyio-4.2.0-py3-none-any.whl.metadata (4.6 kB)
Collecting argon2-cffi==23.1.0 (from -r requirements.txt (line 5))
  Obtaining dependency information for argon2-cffi==23.1.0 from https://files.pythonhosted.org/packages/a4/6a/e8a041599e78b6b
3752da48000b14c8d1e8a04ded09c88c714ba047f34f5/argon2_cffi-23.1.0-py3-none-any.whl.metadata
  Downloading argon2_cffi-23.1.0-py3-none-any.whl.metadata (5.2 kB)
```

Figure 1: Installing requirements.txt into conda/jupyter/pycharm environment

3.3 Dataset

The dataset we used for our study is **nyu_depth_v2_labeled.mat** which is a publicly available dataset and easily accessible through NYU-DepthV2 or can download from my Google Drive location GoogleDrive-Utsav. The data comes as a .mat file which has the first image as the RGB image and second image as the depth map. We use the h5py to read these images and then split them into training and test data. The couplets are indexed and thus processing these images, train-test split and, model training data needs to be indexed as well. We cannot have RGB image[0] encoded to be later decoded by depth map[1].

4 Environment Setup

We follow the following steps to setup our environment. Start the Jupyter Notebook, Import the project as-is. The following is the expected folder structure for the project to run smoothly. Ref 2

```
C:\Users\Administrator\Thesis\FinalDraftChangeInThisFolderHenceforth\indoorDepthEstimation_TeacherStudentModel>tree /f
Folder PATH listing for volume Windows
Volume serial number is 901F-7E5C
C:.
│   IndoorDepthEstimate_TeacherStudentModel.ipynb
│   layers.py
│   mono_640x192.zip
│   nyu_depth_v2_labeled.mat
│   README.md
│   requirements.txt
│   utils.py
│   __init__.py
│
├───logs
├───models
│   └───mono_640x192
│           depth.pth
│           encoder.pth
│           pose.pth
│           pose_encoder.pth
│
├───networks
│       depth_decoder.py
│       pose_cnn.py
│       pose_decoder.py
│       resnet_encoder.py
│       __init__.py
│
├───output
├───preprocessing
│       BlockDropoutRegularization.py
│       CutFlipDataAugmentation.py
│       GeometricTransformationImageFlip.py
│       GeometricTransformationImageWarp.py
│       QuadrantShuffleDataAugmentation.py
│       ShiftAndRotateAllImages.py
│       StripeDropoutRegularization.py
│       __init__.py
│
└───savedStudentModel
        student_model_v1_02-10-2024.h5
        student_model_v2_03-12-2024.h5
        student_model_v3_05-12-2024.h5
        student_model_v4_06-12-2024.h5
        student_model_v5_07-12-2024.h5
```

Figure 2: Expected Folder Structure before Code Execution

4.1 Order Of Execution

4.1.1 Synthesizing Data

Open the command prompt in the **preprocessing** folder to synthesize the data from NYU mat file in the following order

- **python GeometricTransformationImageFlip.py**
  - Task: Generate Vertical and Horizontal Flips of RGB and Depth Maps
  - Log: logs\SynthesizedData_ImageFlippings_log.txt file.
  - RGB Data Stored Location: preprocessing\SynthesizedData\FlippedRGBImage
  - Depth Data Stored Location: preprocessing\SynthesizedData\FlippedDepthImage
- **python GeometricTransformationImageWarp.py**
  - Task: Generate X-Sheared and Y-Sheared Images of RGB and Depth Maps
  - Log: logs\SynthesizedData_ImageWarping_log.txt file.
  - RGB Data Stored Location: preprocessing\SynthesizedData\XshearedRGB & preprocessing\SynthesizedData\XshearedDepth
  - Depth Data Stored Location: preprocessing\SynthesizedData\YshearedDepth
- **python CutFlipDataAugmentation.py**
  - Task: Generate Cutflips of Mirrored Images of RGB and Depth Maps
  - Log: logs\SynthesizedData_ImageWarping_log.txt file.
  - RGB Data Stored Location: preprocessing\SynthesizedData\CutFlipRGB
  - Depth Data Stored Location: preprocessing\SynthesizedData\CutFlipDepth
- **python QuadrantShuffleDataAugmentation.py**
  - Task: Generate Quadrants and Shuffles of Images of RGB and Depth Maps
  - Log: logs\SynthesizedData_ImageWarping_log.txt file.
  - RGB Data Stored Location: preprocessing\SynthesizedData\QuadShuffleRGB
  - Depth Data Stored Location: preprocessing\SynthesizedData\QuadShuffleDepth
- **python BlockDropoutRegularization.py**
  - Task: Drops 4 blocks from 4x4 distributed image matrix from RGB and Depth Map
  - Log: logs\SynthesizedData_BlockDropReg_log.txt file.
  - RGB Data Stored Location: preprocessing\SynthesizedData\BlockDropRegRGB
  - Depth Data Stored Location: preprocessing\SynthesizedData\BlockDropRegDepth
- **python StripeDropoutRegularization.py**
  - Task: Drops 2 Stripes from 1x5 distributed image matrix from RGB and Depth Map
  - Log: logs\SynthesizedData_StripeDropReg_log.txt file.
  - RGB Data Stored Location: preprocessing\SynthesizedData\StripeDropRegRGB
  - Depth Data Stored Location: preprocessing\SynthesizedData\StripeDropRegDepth

4.1.2 Shifting All Images in an unified location

From the same folder cmd line run the command,

- **python ShiftAndRotateAllImages.py**
  - Task: Shifts all RGB Images and Depth Images consolidated into one folder each while rotating the image 90 degrees clockwise
  - Log: logs\SynthesizedData_ImageTransfer_log.txt file.
  - RGB Data Stored Location: preprocessing\SynthesizedData\RGB
  - Depth Data Stored Location: preprocessing\SynthesizedData\Depth

Once all the scripts have ran, we should get the following image count in each folder within SynthesizedData folder. Ref 3

```
C:\Users\Administrator\Thesis\FinalDraftChangeInThisFolderHenceforth\indoorDepthEstimation_TeacherStudentModel\preprocessing\SynthesizedData>for /d %i in (*) do @echo %i && dir "%i" /a:-d /s /b | find /c /v ""
BlockDropRegDepth
1449
BlockDropRegRGB
1449
CutFlipDepth
2898
CutFlipRGB
2898
Depth
13041
FlippedDepthImage
2898
FlippedRGBImage
2898
QuadShuffleDepth
1449
QuadShuffleRGB
1449
RGB
13041
StripeDropRegDepth
1449
StripeDropRegRGB
1449
XshearedDepth
1449
XshearedRGB
1449
YshearedDepth
1449
YshearedRGB
1449
```

Figure 3: Expected Data Count after Scripts Run

It is advised to delete all other folders except **Depth** and **RGB** as we will be using this folder for further operations.

5 Depth Estimation Algorithm

We now start executing the main code for depth estimation.
Open the **IndoorDepthEstimate_TeacherStudentModel.ipynb** file in configured jupyter notebook.
Run the import block (Ref 4)

```
In [2]: from __future__ import absolute_import, division, print_function
        %matplotlib inline

        import os
        import numpy as np
        from PIL import Image,ImageFilter
        import matplotlib.pyplot as plt
        from torchvision.transforms import ToTensor
        import torch
        from torchvision import transforms
        import cv2
        import tensorflow as tf
        from tensorflow.keras.utils import Sequence

        from tensorflow.keras.layers import Input, Conv2D, UpSampling2D, Concatenate, Multiply, GlobalAveragePooling2D, Dense, Reshape
        from tensorflow.keras.models import Model
        from tensorflow.keras.applications import DenseNet169
        import tensorflow as tf
        from tensorflow.keras.callbacks import EarlyStopping

        import networks
        from utils import download_model_if_doesnt_exist
```

Figure 4: Import Block

We get all the necessary imports to run our code
Second we load the pre-trained model as the teacher, the encoder.pth and depth.pth files should be in "models/mono_640x192" path to load properly. Ref 5

```
In [2]: model_name = "mono_640x192"

        download_model_if_doesnt_exist(model_name)
        encoder_path = os.path.join("models", model_name, "encoder.pth")
        depth_decoder_path = os.path.join("models", model_name, "depth.pth")

        encoder = networks.ResnetEncoder(18, False)
        depth_decoder = networks.DepthDecoder(num_ch_enc=encoder.num_ch_enc, scales=range(4))

        loaded_dict_enc = torch.load(encoder_path, map_location='cpu')
        filtered_dict_enc = {k: v for k, v in loaded_dict_enc.items() if k in encoder.state_dict()}
        encoder.load_state_dict(filtered_dict_enc)

        loaded_dict = torch.load(depth_decoder_path, map_location='cpu')
        depth_decoder.load_state_dict(loaded_dict)

        encoder.eval()
        depth_decoder.eval();

C:\ProgramData\anaconda3\Lib\site-packages\torchvision\models\_utils.py:135: UserWarning: Using 'weights' as positional paramet
er(s) is deprecated since 0.13 and may be removed in the future. Please use keyword parameter(s) instead.
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enum or
`None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing
`weights=None`.
  warnings.warn(msg)
```

Figure 5: Import Block

Third, we execute the below block so that the teacher pre-trained on outdoor images can generate it's predictions (or pseudo depth maps). Ref 6,7

```
In [7]: log_file="/logs/process_log_v5.txt"

        def generate_teacher_predictions_from_png(rgb_folder, depth_folder, output_folder, feed_height, feed_width, encoder, depth_decode
            os.makedirs(output_folder, exist_ok=True)
            rgb_files = sorted([f for f in os.listdir(rgb_folder) if f.endswith(".png")])
            depth_files = sorted([f for f in os.listdir(depth_folder) if f.endswith(".png")])

            with open(log_file, "w") as log:
                log.write("Processing log for RGB and depth files\n")
                log.write("=" * 40 + "\n")
                for idx, (rgb_file, depth_file) in enumerate(zip(rgb_files, depth_files)):

                    rgb_image_path = os.path.join(rgb_folder, rgb_file)
                    rgb_image = Image.open(rgb_image_path).convert("RGB")

                    rgb_image_resized = rgb_image.resize((feed_width, feed_height))
                    rgb_tensor = ToTensor()(rgb_image_resized).unsqueeze(0)

                    with torch.no_grad():
                        features = encoder(rgb_tensor)
                        outputs = depth_decoder(features)
                        disp = outputs[("disp", 0)]
                        disp_resized = torch.nn.functional.interpolate(
                            disp, (rgb_image.height, rgb_image.width), mode="bilinear", align_corners=False
                        )

                    disp_np = disp_resized.squeeze().cpu().numpy()
                    output_path = os.path.join(output_folder, f"prediction_{idx}.npy")
                    np.save(output_path, disp_np)
                    log.write(f"Processing RGB: {rgb_file} with Depth: {depth_file}\nCreated prediction file: {output_path}\n")


            print(f"Processed {len(rgb_files)} images and saved predictions to {output_folder}")
            print(f"Processing log saved to {log_file}")
            log.close()
```

```
In [8]:
        generate_teacher_predictions_from_png(
            rgb_folder="preprocessing/SynthesizedData/RGB",
            depth_folder="preprocessing/SynthesizedData/Depth",
            output_folder="v5_path_to_teacher_predictions",
            feed_height=192,
            feed_width=640,
            encoder=encoder,
            depth_decoder=depth_decoder
        )
```

Figure 6: Generating .npy files by teacher model

The folder v5_path_to_teacher_predictions is generated with pseudo depth map prediction by the teacher .npy files

We also get the .npy files dumped as follows. Note that this step is a checklist and we need to be verified that files are created. This step will take a long time.

Name	Date modified	Type	Size
prediction_0.npy	07-12-2024 15:53	NPY File	1,201 KB
prediction_1.npy	07-12-2024 15:53	NPY File	1,201 KB
prediction_2.npy	07-12-2024 15:53	NPY File	1,201 KB
prediction_3.npy	07-12-2024 15:53	NPY File	1,201 KB
prediction_4.npy	07-12-2024 15:53	NPY File	1,201 KB

Figure 7: Teacher Prediction Folder Content

We can co-relate which RGB-Depth Map pairs up with which prediction npy file from the logs\process_log_v5.txt. Ref 8

```
Processing log for RGB and depth files
========================================
Processing RGB: cutflip_0_1.png with Depth: cutflip_0_1.png
Created prediction file: v5_path_to_teacher_predictions\prediction_0.npy
Processing RGB: cutflip_0_10.png with Depth: cutflip_0_10.png
Created prediction file: v5_path_to_teacher_predictions\prediction_1.npy
Processing RGB: cutflip_0_100.png with Depth: cutflip_0_100.png
Created prediction file: v5_path_to_teacher_predictions\prediction_2.npy
Processing RGB: cutflip_0_1000.png with Depth: cutflip_0_1000.png
```

Figure 8: RGB-Depth Mapping with prediction.npy

We then build the student model.

```
In [10]:
def SE_Block(input_tensor, reduction=16):
    channels = input_tensor.shape[-1]
    se = GlobalAveragePooling2D()(input_tensor)
    se = Dense(channels // reduction, activation='relu')(se)
    se = Dense(channels, activation='sigmoid')(se)
    se = Reshape((1, 1, channels))(se)
    return Multiply()([input_tensor, se])


def DenseDepth(input_shape=(320, 320, 3)):
    backbone = DenseNet169(weights='imagenet', include_top=False, input_shape=input_shape)

    conv4 = backbone.get_layer("conv4_block6_concat").output
    conv3 = backbone.get_layer("conv3_block12_concat").output
    conv2 = backbone.get_layer("conv2_block6_concat").output
    conv1 = backbone.get_layer("conv1_relu").output

    up1 = UpSampling2D()(conv4)
    up1 = Concatenate()([up1, conv3])
    up1 = Conv2D(128, (5, 5), activation='relu', padding='same')(up1)
    up1 = SE_Block(up1)
    up1 = Conv2D(64, (3, 3), activation='relu', padding='same', dilation_rate=2)(up1)

    up2 = UpSampling2D()(up1)
    up2 = Concatenate()([up2, conv2])
    up2 = Conv2D(64, (5, 5), activation='relu', padding='same')(up2)
    up2 = SE_Block(up2)
    up2 = Conv2D(32, (3, 3), activation='relu', padding='same', dilation_rate=2)(up2)

    up3 = UpSampling2D()(up2)
    up3 = Concatenate()([up3, conv1])
    up3 = Conv2D(32, (5, 5), activation='relu', padding='same')(up3)
    up3 = SE_Block(up3)
    up3 = Conv2D(16, (5, 5), activation='relu', padding='same', dilation_rate=2)(up3)

    up4 = UpSampling2D()(up3)
    up4 = Conv2D(16, (5, 5), activation='relu', padding='same')(up4)
    up4 = SE_Block(up4)

    output = Conv2D(1, (5, 5), activation='sigmoid', padding='same')(up4)

    model = Model(inputs=backbone.input, outputs=output)
    model.compile(optimizer='adam', loss="mean_squared_error")

    return model

model = DenseDepth()
model.summary()
```

Figure 9: Student Model Building

The model is so designed that each encoder layer shrinks the image and each decoder upscales the image to it's original resolution. Filter count changes with the need of the

smoothness requirements. Each decoder takes input from it's preceding decoder layer and encoder layer having the same image resolution. Ref 9
Now, we generate indoor predictions data from these .npy files generated by teacher model.

```
In [35]:
log_file_student="student_process_log_v5.txt"

class NYUDataGenerator(Sequence):

    def __init__(self, rgb_folder, depth_folder, prediction_folder, batch_size, input_shape):
        self.rgb_folder = rgb_folder
        self.depth_folder = depth_folder
        self.prediction_folder = prediction_folder
        self.batch_size = batch_size
        self.input_shape = input_shape
        self.rgb_files = sorted([f for f in os.listdir(rgb_folder) if f.endswith('.png')])
        self.depth_files = sorted([f for f in os.listdir(depth_folder) if f.endswith('.png')])
        assert len(self.rgb_files) == len(self.depth_files), "Mismatch between RGB and depth map counts."

    def __len__(self):
        return len(self.rgb_files) // self.batch_size

    def __getitem__(self, idx):
        batch_images = []
        batch_predictions = []
        with open(log_file_student, "a") as log:
            log.write("Student Learning \nProcessing log for RGB and depth files\n")
            log.write("=" * 40 + "\n")
            for i in range(self.batch_size):
                img_idx = idx * self.batch_size + i
                rgb_path = os.path.join(self.rgb_folder, self.rgb_files[img_idx])
                rgb_image = Image.open(rgb_path).convert("RGB")
                rgb_image = rgb_image.resize(self.input_shape[:2])
                rgb_image = np.array(rgb_image) / 255.0
                batch_images.append(rgb_image)
                pred_path = os.path.join(self.prediction_folder, f"prediction_{img_idx}.npy")
                teacher_pred = np.load(pred_path)
                teacher_pred = np.expand_dims(teacher_pred, axis=-1)
                teacher_pred = tf.image.resize(teacher_pred, self.input_shape[:2]).numpy()
                batch_predictions.append(teacher_pred)
                log.write(f"Student Processing RGB: {rgb_path} with Teacher Prediction: {pred_path}\n")
            return np.array(batch_images), np.array(batch_predictions)
```

Figure 10: Data Generation from.npy file

This data generated will be crucial for the student model to recreate (Teacher-student framework principle). This data will not be stored in any physical location on the device and thus is memory consuming. Ref 10
With this, we are ready for training our student model.

```
In [38]: def root_mean_squared_error(y_true, y_pred):
             return tf.sqrt(tf.reduce_mean(tf.square(y_pred - y_true)))

In [39]: student_model = DenseDepth()
         student_model.compile(optimizer='adam', loss=root_mean_squared_error)

In [40]: log_dir = "logs/student_model_v5_07-12-2024"
         tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

In [41]: history = student_model.fit(data_gen, epochs=10)
         model_save_path = "student_model_v5_07-12-2024.h5"
         student_model.save(model_save_path)
         print(f"Model saved to {model_save_path}")

Epoch 1/10
C:\Users\Administrator\AppData\Roaming\Python\Python311\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:12
2: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `wor
kers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3192s 3s/step - loss: 0.1760
Epoch 2/10
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3142s 3s/step - loss: 0.1406
Epoch 3/10
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3139s 3s/step - loss: 0.1298
Epoch 4/10
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3134s 3s/step - loss: 0.1249
Epoch 5/10
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3123s 3s/step - loss: 0.1207
Epoch 6/10
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3128s 3s/step - loss: 0.1148
Epoch 7/10
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3145s 3s/step - loss: 0.1129
Epoch 8/10
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3127s 3s/step - loss: 0.1090
Epoch 9/10
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3129s 3s/step - loss: 0.1057
Epoch 10/10
1086/1086 ━━━━━━━━━━━━━━━━━━━━ 3140s 3s/step - loss: 0.1019
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.
save_model(model, 'my_model.keras')`.
Model saved to student_model_v5_07-12-2024.h5
```

Figure 11: Fit the student model from the teacher learning/predictions

This is the highest time consuming step, as depending on the filter count, image resolution and image quantity, the model training can take anywhere from 12 hrs to 18 hrs. We save the model as .h5 file so that we can load the model for prediction whenever instead of training the model again. Ref 11
And we can load the model in the following manner. Ref 12

```
In [5]: from tensorflow.keras.models import load_model
        model_path = "student_model_v5_07-12-2024.h5"
        custom_objects = {
                'optimizer':'adam', 'loss':'root_mean_squared_error'
        }
        model = load_model("student_model_v5_07-12-2024.h5", compile=False)
        print(f"Model loaded from {model_path}")

        Model loaded from student_model_v5_07-12-2024.h5
```

Figure 12: Loading .h5 file as model to predict

And finally we predict the depth map from the image feed. It is imperative to know that we need to normalize the test image before it can be used for prediction. If the image is taken from the .mat file, the steps have been shown in 13. The steps will be different if we are using a png, jpeg or jpg images and have not been demonstrated in this code snippet.

```
In [255]: test_rgb_image = np.transpose(test_rgb_image, (1, 2, 0))
          test_depth_maps1_image = ((test_depth_image - test_depth_image.min()) /
                                  (test_depth_image.max() - test_depth_image.min()) * 255).astype(np.uint8)
          test_rgb_image = ((test_rgb_image - test_rgb_image.min()) /
                                  (test_rgb_image.max() - test_rgb_image.min()) * 255).astype(np.uint8)

In [256]: test_rgb_image_pil = Image.fromarray(test_rgb_image)
          resized_rgb_image_pil = test_rgb_image_pil.resize((320, 320), Image.LANCZOS)
          resized_rgb_image_pil = np.rot90(resized_rgb_image_pil, k=-1)
          resized_rgb_image = np.array(resized_rgb_image_pil)
          test_depth_maps1_image_pil = Image.fromarray(test_depth_maps1_image)
          test_depth_maps1_image_pil = test_depth_maps1_image_pil.resize((320, 320), Image.LANCZOS)
          test_depth_maps1_image_pil = np.rot90(test_depth_maps1_image_pil, k=-1)
          test_depth_maps1_image_resized = np.array(test_depth_maps1_image_pil)

In [257]: test_rgb_image = np.rot90(test_rgb_image, k=-1)
          output_path = "test_rgb_image.png"
          image = Image.fromarray(test_rgb_image)
          image.save(output_path)

In [258]: def preprocess_image(image, input_shape):
              image = image.resize(input_shape[:2])
              image = np.array(image) / 255.0
              image = np.expand_dims(image, axis=0)
              return image

          input_shape = (320, 320, 3)
          preprocessed_image = preprocess_image(image, input_shape)
          predicted_depth_map = model.predict(preprocessed_image)
          print(predicted_depth_map.shape)

          1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
          (1, 320, 320, 1)
```

Figure 13: Test Image Pre-processing and Student Model Prediction

6 Conclusion

With this Configuration Manual, users should be able to execute the indoor depth estimations smoothly. The order of execution is important as without pre-processing, there will be no data synthesized and with no image feed, there nothing to train the teacher and student model.

References

Kumar, T., Brennan, R., Mileo, A. and Bendechache, M. (2024). Image data augmentation approaches: A comprehensive survey and future directions, *IEEE Access* .