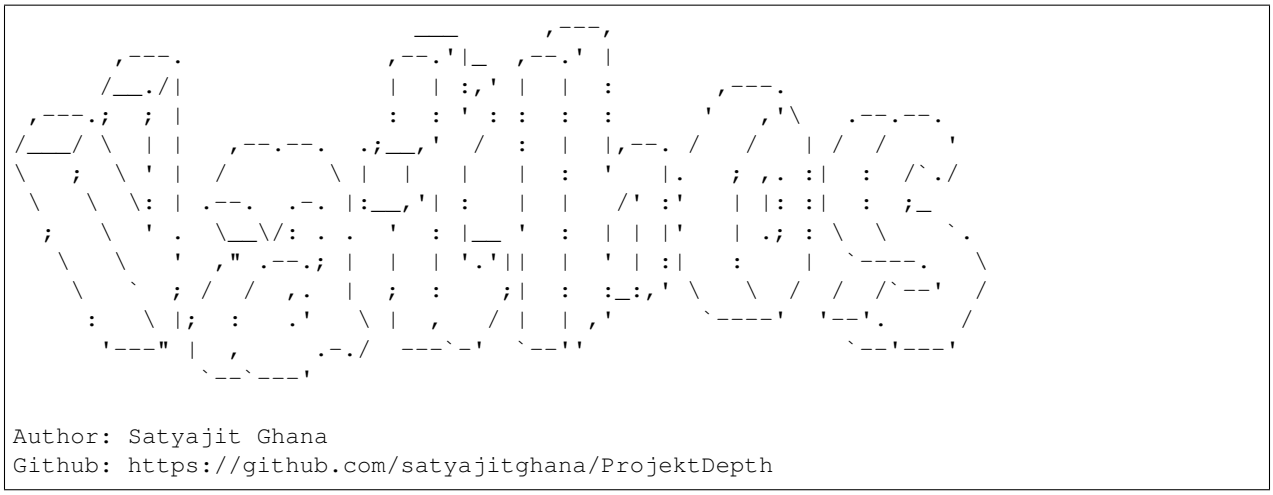
# ProjektDepth - vathos

***Release master***

**Nov 20, 2020**

# Get Started



```
Author: Satyajit Ghana
Github: https://github.com/satyajitghana/ProjektDepth
```

You must me curious what this is eh ? See The Journey to know how this project was made

CHAPTER 1

# vathos.model

## 1.1 vathos.model.resunet

Residual - Unet - Enhanced

**class ResUNet**
A ResNet - Unet inspired custom model for monocular depth estimation

**class ResDoubleConv**(*in_channels*, *out_channels*)
Basic DoubleConv of a ResNetV2

Performs basic Pre Activated ResNet Double Convolution

**Parameters**

- **in_channels** – input channels
- **out_channels** – output channels

**class ResDownBlock**(*in_channels*, *out_channels*)
Basic DownBlock of a ResNetV2

Performs a Residual Down operation

**Parameters**

- **in_channels** – input channels
- **out_channels** – output channels

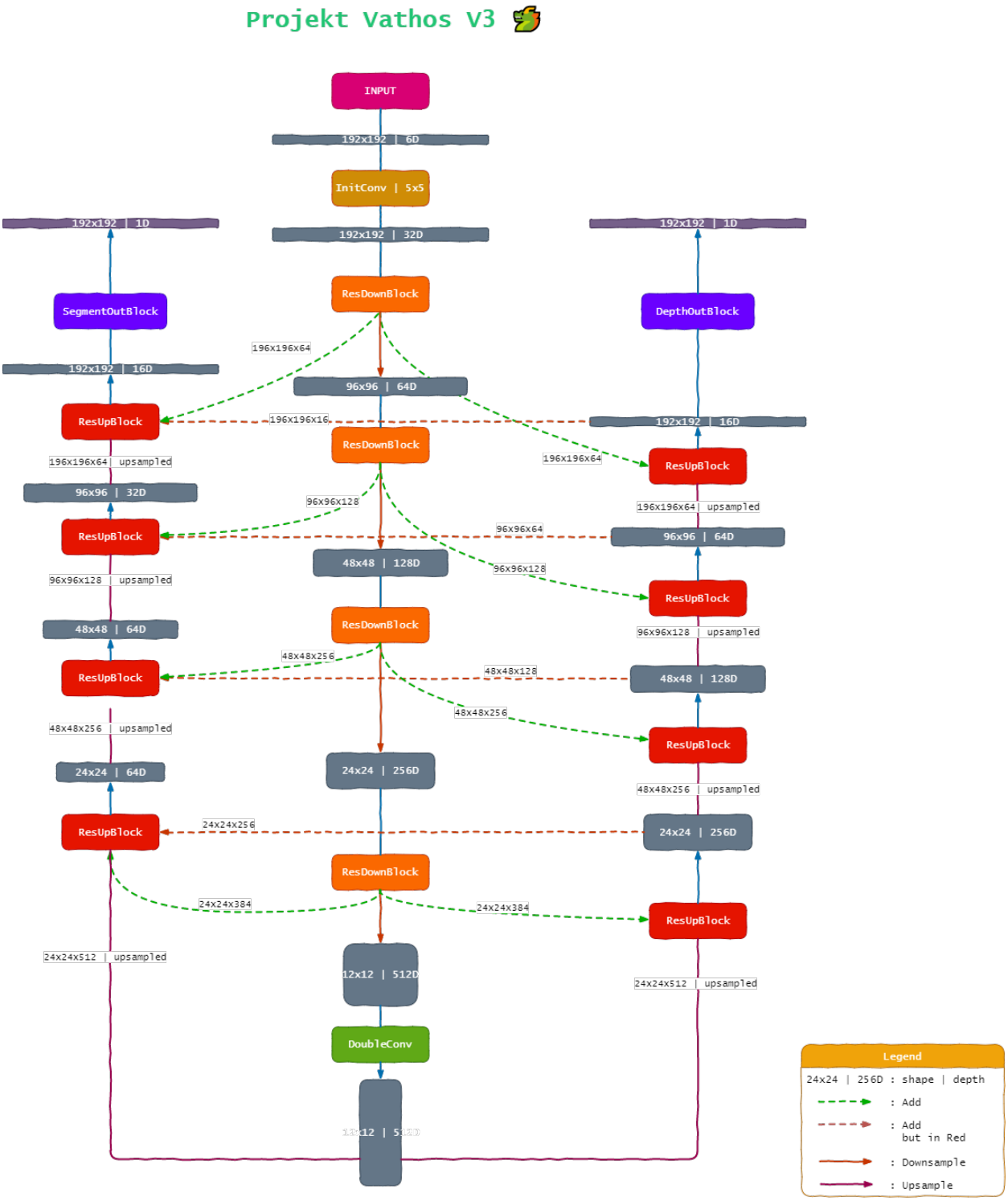
output: $(N, C, H/2, W/2)$

**class ResUpBlock**(*in_channels*, *out_channels*, *skip_channels*, *dense_channels=None*)
Basic UpBlock of a ResNetV2

Performs Residual Up Convolution on the input, uses PixelShuffle to produce checkerboard-free outputs

**Parameters**

- **in_channels** – input channels


Projekt Vathos V3
INPUT
192x192 | 6D
InitConv | 5x5
192x192 | 32D
ResDownBlock
96x96 | 64D
ResDownBlock
48x48 | 128D
ResDownBlock
24x24 | 256D
ResDownBlock
12x12 | 512D
DoubleConv
12x12 | 512D
196x196x64
196x196x16
96x96x128
48x48x256
24x24x384
SegmentOutBlock
192x192 | 1D
192x192 | 16D
ResUpBlock
196x196x64 | upsampled
96x96 | 32D
ResUpBlock
96x96x128 | upsampled
48x48 | 64D
ResUpBlock
48x48x256 | upsampled
24x24 | 64D
ResUpBlock
24x24x512 | upsampled
DepthOutBlock
192x192 | 1D
192x192 | 16D
ResUpBlock
196x196x64 | upsampled
96x96 | 64D
ResUpBlock
96x96x128 | upsampled
48x48 | 128D
ResUpBlock
48x48x256 | upsampled
24x24 | 256D
ResUpBlock
24x24x512 | upsampled
Legend
24x24 | 256D : shape | depth
: Add
: Add but in Red
: Downsample
: Upsample

- **out_channels** – output channels
- **skip_channels** – skip input channels
- **dense_channels** – dense input channels (from another decoder)

**Note:** The input is applied with a 1x1 convolution and then pixel shuffle to keep the channels constant and also produce checkerboard-free outputs, the rest is then followed by double convolution

## 1.2 vathos.model.resunext

ResidualNeXt - UNet - Enhanced

**class ResUNeXt**
A ResNeXt - Unet inspired custom model for monocular depth estimation and segmentation

**class ResDoubleConv**(*in_channels*, *out_channels*)
Basic DoubleConv of a ResNeXt

Performs basic Pre Activated ResNet Double Convolution

**Parameters**

- **in_channels** – input channels
- **out_channels** – output channels

**class ResDownBlock**(*in_channels*, *out_channels*)
Basic DownBlock of a ResNetV2

**class ResUpBlock**(*in_channels*, *out_channels*, *skip_channels*, *dense_channels=None*)
Basic UpBlock of a ResNetV2

## 1.3 vathos.model.loss

The Various Loss Functions that can be used with the model

**Note:** All the loss functions take the Logits as input, internally they will sigmoid the input perform their operations and then return the mean error

### 1.3.1 Segmentation Loss

**class DiceLoss**
Criterion that computes Sørensen-Dice Coefficient loss.

According to [1], we compute the Sørensen-Dice Coefficient as follows:

$$\text{Dice}(x, class) = \frac{2|X| \cap |Y|}{|X| + |Y|}$$

where: - $X$ expects to be the scores of each class. - $Y$ expects to be the one-hot tensor with the class labels.

the loss, is finally computed as:

$$\text{loss}(x, class) = 1 - \text{Dice}(x, class)$$

[1] https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient

**class BCEDiceLoss**
Performs BCE and Dice Loss and adds them both

loss = bce_loss + 2 * dice_loss

**class TverskyLoss**(*alpha: float*, *beta: float*)
Performs Tversky Loss on Logits

According to [1], we compute the Tversky Coefficient as follows:

$$\text{S}(P, G, \alpha; \beta) = \frac{|PG|}{|PG| + \alpha|P\ G| + \beta|G\ P|}$$

**where:**

- $P$ and $G$ are the predicted and ground truth binary labels.
- $\alpha$ and $\beta$ control the magnitude of the penalties for FPs and FNs, respectively.

#### Notes

- $\alpha = \beta = 0.5$ => dice coeff
- $\alpha = \beta = 1$ => tanimoto coeff
- $\alpha + \beta = 1$ => F beta coeff

**Reference:** [1] https://kornia.readthedocs.io/en/latest/losses.html

**class BCETverskyLoss**
Performs BCE and Tversky Loss and adds them both

loss = bce_loss + 2 * tversky_loss

### 1.3.2 Depth Loss

**class RMSELoss**(*eps=1e-06*)
Performs RMSE Loss

we simply sigmoid the input, pass it through *nn.MSELoss* and then do a *torch.sqrt* on it

**class BerHuLoss**(*threshold: float = 0.2*)
Implementation of the BerHu Loss from [1]

$$B(y, y') = (1/n) * |y' - y| if |y' - y| <= c$$
$$B(y, y') = (1/n) * ((y' - y)^2 + c^2)/2 * cothwerwise$$
$$c = 1/5 * max(|y' - y|)$$

[1] http://cs231n.stanford.edu/reports/2017/pdfs/203.pdf

[2] https://arxiv.org/abs/1207.6868

**class GradLoss**
Performs Gradient Loss

The Image XY Gradients are computed for input and target and the mean L1Loss between these gradients is returned

**class SSIMLoss**
Performs SSIM Loss

window sizes uses are 5x5 and 11x11

we tried adding other window sizes too, but there wasn't a significant benefit

**Note:** we do ssim loss for various window sizes, add them and return the mean

**class RMSEwSSIMLoss**
Performs RMSE and SSIM Loss

loss = $\sqrt{}$

### 1.3.3 Accuracy Functions

**iou**(*outputs: torch.Tensor*, *labels: torch.Tensor*)

**rmse**(*outputs: torch.Tensor*, *labels: torch.Tensor*)

CHAPTER 2

# vathos.data_loader

The Dataloader classes that can be used with vathos

Some DataVisualization

BG Images

FG_BG Images

FG_BG_MASK Images

DEPTH_FG_BG Images

Mean and Standard Deviation

```
bg_stat = (['0.573435604572296', '0.520844697952271', '0.457784473896027'], ['0.
↪207058250904083', '0.208138316869736', '0.215291306376457'])
fg_bg_stat = (['0.568499565124512', '0.512103974819183', '0.452332496643066'], ['0.
↪211068645119667', '0.211040720343590', '0.216081097722054'])
fg_bg_mask_stat = (['0.062296919524670', '0.062296919524670', '0.062296919524670'], [
↪'0.227044790983200', '0.227044790983200', '0.227044790983200'])
depth_fg_bg_stat = (['0.302973538637161', '0.302973538637161', '0.302973538637161'], [
↪'0.101284727454185', '0.101284727454185', '0.101284727454185'])
```

**class DenseDepth**(*root*, *source_zipfolder*, *train=True*, *transform=None*, *target_transform=None*)
DenseDepth Dataset

**Parameters**

- **root** – the directory where to place the dataset unzipped files
- **source_zipfiles** – the directory where the dataset zip files are stored (mount your drive and give a absolute path)
- **transform** – torchvision transform for input images
- **target_transform** – torchvision transform for ouput images

Input is fg_bg image AND bg image

Target is fg_bg_mask AND depth_fg_bg image

**static apply_on_batch**(*batch*, *apply_func*)
applies a given function the batch

**Parameters**

- **batch** – a batch of data
- **apply_func** – a function that is to be applied to each data in batch

**extractall**()
extracts the zip files into the root dir

**static plot4_batch**(*batch*)
Plots 4 images for batch

**Parameters** **batch** – the batch of data of this DenseDepth dataset

**Returns** the created figure

**Return type** matplotlib.pyplot.figure

**static plot_results**(*batch*)
Plots 8 images for batch's model results

**Parameters** **batch** – the batch of data of this DenseDepth dataset

**Returns** the created figure

**Return type** matplotlib.pyplot.figure

**static plot_sample**(*sample*)
Plots a given sample of the dataset

**Parameters** **batch** – the batch of data of this DenseDepth dataset

**Returns** the created figure

**Return type** matplotlib.pyplot.figure

# CHAPTER 3

# vathos.trainer

## 3.1 Trainers

**class BaseTrainer**(*model*, *loss_fns*, *optimizer*, *config*, *train_subset*, *test_subset*, *state_dict=None*)
BaseTrainer: An Abstract Meta Class for all trainers (GPU, CPU, TPU)

**Parameters**

- **model** – the model to be trained, (can be on cpu/gpu)
- **loss_fns** (*Tuple*) – (seg_loss, depth_loss)
- **optimizer** – the optimizer (can be on cpu/gpu)
- **config** – config in dict format
- **train_subset** (*torch.utils.data.Subset*) – train dataset wrapped in a subset containing the indices
- **test_subset** (*torch.utils.data.Subset*) – test dataset wrapped in a subset containing the indices
- **state_dict** (*Optional*) – the saved state in a dictionary format

**optimizer_to**(*optim*, *device*)
moves the optimizer to device

**Parameters**

- **optim** – the optimizer
- **device** – device to which to move to

**scheduler_to**(*sched*, *device*)
moves the scheduler to device

**Parameters**

- **sched** – the scheduler

- **device** – device to which to move to

**class GPUTrainer**(**args*, ***kwargs*)

GPUTrainer: Trains the vathos model on GPU

see `BaseTrainer` for args

### Examples

```
>>> gpu_trainer = GPUTrainer(model, loss_fns, optimizer, cfg, train_subset, test_
↪subset, state_dict=state_dict)
>>> gpu_trainer.start_train()
```

**start_train**()

trains the model for self.epochs times

the model and training state is saved at every epoch

summary is flushed to disk every epoch

**test_epoch**(*epoch*)

tests the model for one epoch

**Parameters** **epoch** – the epoch number

**Returns** miou, mrmse, seg_loss, depth_loss

**Return type** Dict

**train_epoch**(*epoch*)

trains the model for one epoch

**Parameters** **epoch** – the epoch number

**Returns** miou, mrmse, seg_loss, depth_loss

**Return type** Dict

**class TPUTrainer**(**args*, ***kwargs*)

TPUTrainer: Trains the vathos model on TPU

# CHAPTER 4

# vathos.runner

**class Runner**(*config*)

Runner that encapsulated a Trainer

**Parameters** **config** – a dict that contains the current experiment info

**setup_train**()

sets up the training for the config provided

**start_train**()

a wrapper that calls self.trainer.start_train()

# CHAPTER 5

## vathos.utils

The various utility functions that are essential for vathos

**load_config** (*filename: str*) → dict
Load a configuration file as YAML and returns a dict

**Parameters** **filename** (*str*) – location of the file

**Returns** (Dict) of the config

**setup_device** (*model: torch.nn.modules.module.Module*, *target_device: int*) → Tuple[torch.device, List[int]]
sets up the device for the model

**Parameters**

- **model** (*nn.Module*) – the model
- **target_device** (*int*) – index of the target device

**Returns** Tuple[torch.device, List[int]]

**setup_param_groups** (*model: torch.nn.modules.module.Module*, *config: Dict*) → List

**get_instance** (*module: module*, *name: str*, *config: Dict*, **args*) → Any
creates an instance from a constructor name and module name

**Parameters**

- **module** (*ModuleType*) – the module which contains the class
- **name** (*str*) – name of the class
- **config** (*Dict*) – configuration of experiment
- **args** (*Any*) – any arguments that needs to be passed to the class

**get_instance_v2** (*module*, *ctor_name*, **args*, ***kwargs*)
creates an instance from a constructor name and module name

**Parameters**

- **module** – the module which contains the ctor_name
- **ctor_name** – name of the constructor
- **args** (*Optional*) – positional arguments that needs to be passed to ctor
- **kwargs** (*Optional*) – keywords arguments that needs to be passed to ctor

**setup_logger**(*name*)

# CHAPTER 6

# Vathos - The Journey

Q : So what does the model do ? Ans: It takes in 2D images and outputs segmented image + a depth map, basically recognizes the object for which it was trained for, segments it out, and also creates a 3D depth map of the image.

Read through The Journey of how the dataset was made, how the model was researched and finally how it was trained, if you want to see the results of the model, head over to `Training on Small Dataset`, a few more model outputs can be found in `Loss Functions`.

## 6.1 1. The Model

GitHub Link : https://github.com/satyajitghana/ProjektDepth/blob/master/notebooks/10_DepthModel_ModelImprovisation.ipynb Colab Link : https://colab.research.google.com/github/satyajitghana/ProjektDepth/blob/master/notebooks/10_DepthModel_ModelImprovisation.ipynb

Given the task to do monocular depth estimation and also segmentation, we researched upon a few possibilities

The obvious answer was to use a Encoder-Decoder architecture, since the reference model we found were these

1. https://github.com/OniroAI/MonoDepth-PyTorch/
2. https://github.com/wolverinn/Depth-Estimation-PyTorch
3. https://heartbeat.fritz.ai/research-guide-for-depth-estimation-with-deep-learning-1a02a439b834

All of them using a variation of Encoder-Decoder architecture

Various Papers were surveyed to find that UNet with Residual connections work best, also multiple encoder-decoder networks can be combined together to form a new network.

1. Stacked U-Nets: A No-Frills Approach to Natural Image Segmentation: https://arxiv.org/pdf/1804.10343.pdf : This inspired me to stack the decoder part of my network to give two outputs

2. Depth Estimation and Semantic Segmentation from a Single RGB Image Using a Hybrid Convolutional Neural Network: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6514714/ : Various loss functions were inspired from this

3. Road Extraction by Deep Residual U-Net https://arxiv.org/abs/1711.10684 : I referred this to create the ResUNet

4. Transfer Learning for Brain Tumor Segmentation https://arxiv.org/abs/1912.12452 : Brain Tumor Segmentation done using UNet

5. Image Super-Resolution Using Deep Convolutional Networks https://arxiv.org/pdf/1501.00092 : For the Decoder Network to prevent checkerboard issues, PixelShuffle was used

So we went with that combined with my knowledge that residual networks have proved to be really good, and for segmentation a UNet kind of architecture is good, finally we created a custom Residual-UNet architecture, with Single Encoder - Double Decoder and Residual Connections between them, the model diagram will make it more clear to understand how these residual connections between the decoders and the encoder play out.

The model was made over 3 iterations, in which various configurations were tried, but the model was too densly connected since we had used *concatenation* to add the residual connections, which created a huge backprop size, that couldn't fit into memory, hence we converted the *concat* ops to *add* ops, various model param counts were also tried.

The input to our model will be *fg_bg* and *bg* image and output will be *depth_fg_bg* and *fg_bg_mask*

input: $(3, 196, 196), (3, 196, 196)$

output: $(1, 196, 196), (1, 196, 196)$

### 6.1.1 Model Architecture

Total Params: 15,855,712

### 6.1.2 Summary of the Model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 192, 192]           4,800
            Conv2d-2         [-1, 64, 192, 192]           2,048
    BatchNorm2d-3         [-1, 64, 192, 192]             128
    BatchNorm2d-4         [-1, 32, 192, 192]              64
            ReLU-5         [-1, 32, 192, 192]               0
            Conv2d-6         [-1, 64, 192, 192]          18,432
    BatchNorm2d-7         [-1, 64, 192, 192]             128
            ReLU-8         [-1, 64, 192, 192]               0
            Conv2d-9         [-1, 64, 192, 192]          36,864
    ResDoubleConv-10         [-1, 64, 192, 192]               0
        MaxPool2d-11           [-1, 64, 96, 96]               0
    ResDownBlock-12  [[-1, 64, 96, 96], [-1, 64, 192, 192]]               0
        Conv2d-13          [-1, 128, 96, 96]           8,192
    BatchNorm2d-14          [-1, 128, 96, 96]             256
    BatchNorm2d-15           [-1, 64, 96, 96]             128
            ReLU-16          [-1, 64, 96, 96]               0
        Conv2d-17          [-1, 128, 96, 96]          73,728
    BatchNorm2d-18          [-1, 128, 96, 96]             256
            ReLU-19          [-1, 128, 96, 96]               0
        Conv2d-20          [-1, 128, 96, 96]         147,456
    ResDoubleConv-21          [-1, 128, 96, 96]               0
        MaxPool2d-22          [-1, 128, 48, 48]               0
    ResDownBlock-23  [[-1, 128, 48, 48], [-1, 128, 96, 96]]               0
        Conv2d-24          [-1, 256, 48, 48]          32,768
    BatchNorm2d-25          [-1, 256, 48, 48]             512
    BatchNorm2d-26          [-1, 128, 48, 48]             256
            ReLU-27          [-1, 128, 48, 48]               0
```

(continues on next page)

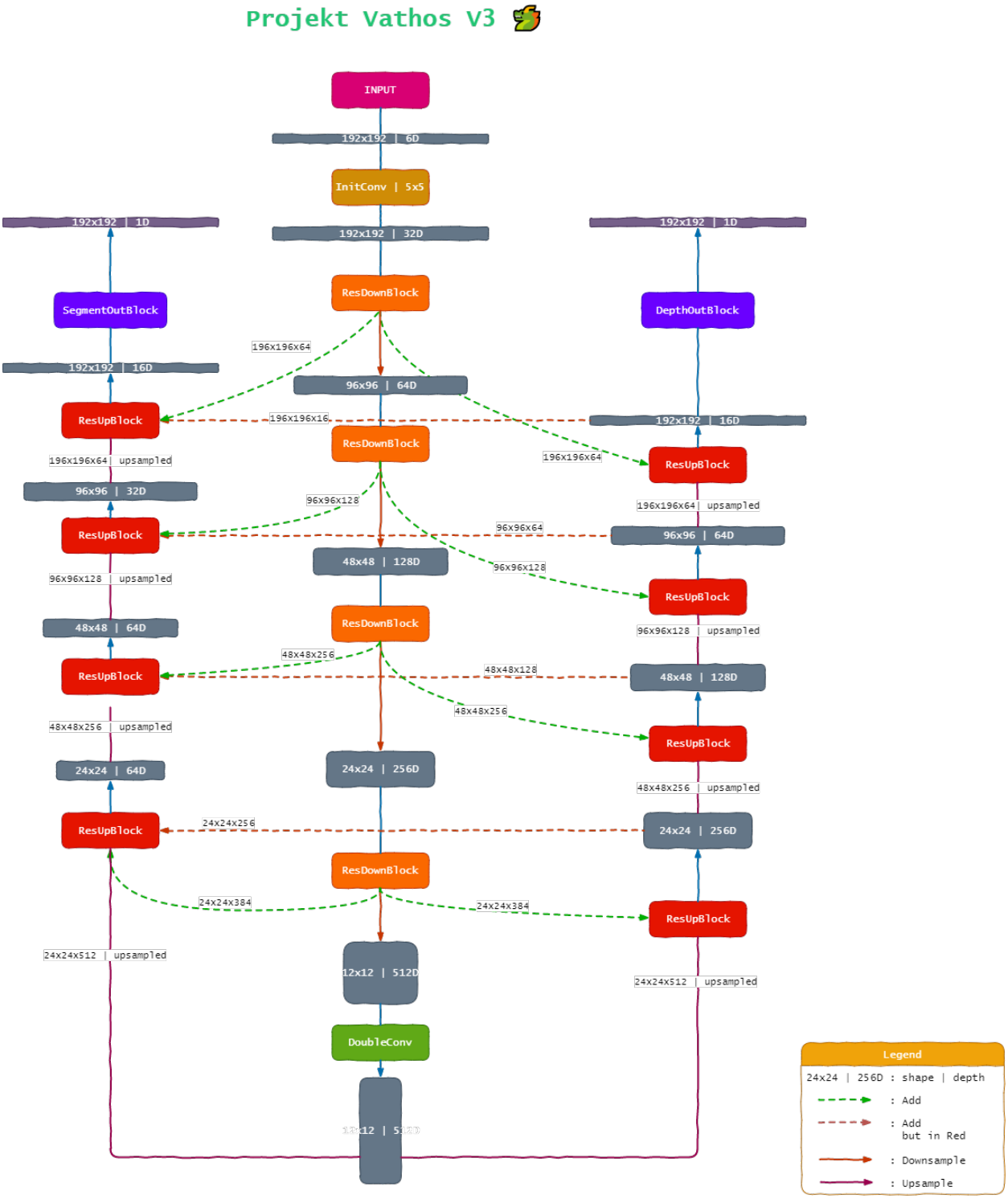



Fig. 1: ResUNet-V3

(continued from previous page)

```
        Conv2d-28          [-1, 256, 48, 48]         294,912
   BatchNorm2d-29          [-1, 256, 48, 48]             512
          ReLU-30          [-1, 256, 48, 48]               0
        Conv2d-31          [-1, 256, 48, 48]         589,824
   ResDoubleConv-32          [-1, 256, 48, 48]               0
        MaxPool2d-33          [-1, 256, 24, 24]               0
   ResDownBlock-34  [[-1, 256, 24, 24], [-1, 256, 48, 48]]               0
        Conv2d-35          [-1, 512, 24, 24]         131,072
   BatchNorm2d-36          [-1, 512, 24, 24]           1,024
   BatchNorm2d-37          [-1, 256, 24, 24]             512
          ReLU-38          [-1, 256, 24, 24]               0
        Conv2d-39          [-1, 512, 24, 24]       1,179,648
   BatchNorm2d-40          [-1, 512, 24, 24]           1,024
          ReLU-41          [-1, 512, 24, 24]               0
        Conv2d-42          [-1, 512, 24, 24]       2,359,296
   ResDoubleConv-43          [-1, 512, 24, 24]               0
        MaxPool2d-44          [-1, 512, 12, 12]               0
   ResDownBlock-45  [[-1, 512, 12, 12], [-1, 512, 24, 24]]               0
   BatchNorm2d-46          [-1, 512, 12, 12]           1,024
          ReLU-47          [-1, 512, 12, 12]               0
        Conv2d-48          [-1, 512, 12, 12]       2,359,296
   BatchNorm2d-49          [-1, 512, 12, 12]           1,024
          ReLU-50          [-1, 512, 12, 12]               0
        Conv2d-51          [-1, 512, 12, 12]       2,359,296
   ResDoubleConv-52          [-1, 512, 12, 12]               0
        Conv2d-53         [-1, 2048, 12, 12]       1,048,576
   PixelShuffle-54          [-1, 512, 24, 24]               0
        Conv2d-55          [-1, 512, 24, 24]         262,144
        Conv2d-56          [-1, 256, 24, 24]         131,072
   BatchNorm2d-57          [-1, 256, 24, 24]             512
   BatchNorm2d-58          [-1, 512, 24, 24]           1,024
          ReLU-59          [-1, 512, 24, 24]               0
        Conv2d-60          [-1, 256, 24, 24]       1,179,648
   BatchNorm2d-61          [-1, 256, 24, 24]             512
          ReLU-62          [-1, 256, 24, 24]               0
        Conv2d-63          [-1, 256, 24, 24]         589,824
   ResDoubleConv-64          [-1, 256, 24, 24]               0
   ResUpBlock-65          [-1, 256, 24, 24]               0
        Conv2d-66         [-1, 1024, 24, 24]         262,144
   PixelShuffle-67          [-1, 256, 48, 48]               0
        Conv2d-68          [-1, 256, 48, 48]          65,536
        Conv2d-69          [-1, 128, 48, 48]          32,768
   BatchNorm2d-70          [-1, 128, 48, 48]             256
   BatchNorm2d-71          [-1, 256, 48, 48]             512
          ReLU-72          [-1, 256, 48, 48]               0
        Conv2d-73          [-1, 128, 48, 48]         294,912
   BatchNorm2d-74          [-1, 128, 48, 48]             256
          ReLU-75          [-1, 128, 48, 48]               0
        Conv2d-76          [-1, 128, 48, 48]         147,456
   ResDoubleConv-77          [-1, 128, 48, 48]               0
   ResUpBlock-78          [-1, 128, 48, 48]               0
        Conv2d-79          [-1, 512, 48, 48]          65,536
   PixelShuffle-80          [-1, 128, 96, 96]               0
        Conv2d-81          [-1, 128, 96, 96]          16,384
        Conv2d-82           [-1, 64, 96, 96]           8,192
   BatchNorm2d-83           [-1, 64, 96, 96]             128
   BatchNorm2d-84          [-1, 128, 96, 96]             256
```

(continues on next page)

(continued from previous page)

```
            ReLU-85          [-1, 128, 96, 96]               0
          Conv2d-86           [-1, 64, 96, 96]          73,728
     BatchNorm2d-87           [-1, 64, 96, 96]             128
            ReLU-88           [-1, 64, 96, 96]               0
          Conv2d-89           [-1, 64, 96, 96]          36,864
     ResDoubleConv-90           [-1, 64, 96, 96]               0
     ResUpBlock-91           [-1, 64, 96, 96]               0
          Conv2d-92          [-1, 256, 96, 96]          16,384
     PixelShuffle-93          [-1, 64, 192, 192]               0
          Conv2d-94         [-1, 64, 192, 192]           4,096
          Conv2d-95         [-1, 16, 192, 192]           1,024
     BatchNorm2d-96          [-1, 16, 192, 192]              32
     BatchNorm2d-97          [-1, 64, 192, 192]             128
            ReLU-98          [-1, 64, 192, 192]               0
          Conv2d-99         [-1, 16, 192, 192]           9,216
     BatchNorm2d-100          [-1, 16, 192, 192]              32
            ReLU-101          [-1, 16, 192, 192]               0
          Conv2d-102          [-1, 16, 192, 192]           2,304
ResDoubleConv-103         [-1, 16, 192, 192]               0
     ResUpBlock-104          [-1, 16, 192, 192]               0
          Conv2d-105           [-1, 1, 192, 192]              16
          Conv2d-106         [-1, 2048, 12, 12]       1,048,576
     PixelShuffle-107           [-1, 512, 24, 24]               0
          Conv2d-108          [-1, 512, 24, 24]         262,144
          Conv2d-109          [-1, 512, 24, 24]         131,072
          Conv2d-110           [-1, 64, 24, 24]          32,768
     BatchNorm2d-111           [-1, 64, 24, 24]             128
     BatchNorm2d-112          [-1, 512, 24, 24]           1,024
            ReLU-113          [-1, 512, 24, 24]               0
          Conv2d-114          [-1, 64, 24, 24]         294,912
     BatchNorm2d-115           [-1, 64, 24, 24]             128
            ReLU-116           [-1, 64, 24, 24]               0
          Conv2d-117          [-1, 64, 24, 24]          36,864
ResDoubleConv-118          [-1, 64, 24, 24]               0
     ResUpBlock-119           [-1, 64, 24, 24]               0
          Conv2d-120          [-1, 256, 24, 24]          16,384
     PixelShuffle-121           [-1, 64, 48, 48]               0
          Conv2d-122          [-1, 64, 48, 48]          16,384
          Conv2d-123          [-1, 64, 48, 48]           8,192
          Conv2d-124          [-1, 64, 48, 48]           4,096
     BatchNorm2d-125           [-1, 64, 48, 48]             128
     BatchNorm2d-126           [-1, 64, 48, 48]             128
            ReLU-127           [-1, 64, 48, 48]               0
          Conv2d-128          [-1, 64, 48, 48]          36,864
     BatchNorm2d-129           [-1, 64, 48, 48]             128
            ReLU-130           [-1, 64, 48, 48]               0
          Conv2d-131          [-1, 64, 48, 48]          36,864
ResDoubleConv-132          [-1, 64, 48, 48]               0
     ResUpBlock-133          [-1, 64, 48, 48]               0
          Conv2d-134          [-1, 256, 48, 48]          16,384
     PixelShuffle-135           [-1, 64, 96, 96]               0
          Conv2d-136          [-1, 64, 96, 96]           8,192
          Conv2d-137          [-1, 64, 96, 96]           4,096
          Conv2d-138          [-1, 32, 96, 96]           2,048
     BatchNorm2d-139          [-1, 32, 96, 96]              64
     BatchNorm2d-140          [-1, 64, 96, 96]             128
            ReLU-141          [-1, 64, 96, 96]               0
```

(continues on next page)

(continued from previous page)

```
        Conv2d-142           [-1, 32, 96, 96]          18,432
    BatchNorm2d-143           [-1, 32, 96, 96]              64
            ReLU-144           [-1, 32, 96, 96]               0
        Conv2d-145           [-1, 32, 96, 96]           9,216
ResDoubleConv-146          [-1, 32, 96, 96]               0
    ResUpBlock-147           [-1, 32, 96, 96]               0
        Conv2d-148          [-1, 128, 96, 96]           4,096
    PixelShuffle-149         [-1, 32, 192, 192]               0
        Conv2d-150         [-1, 32, 192, 192]           2,048
        Conv2d-151         [-1, 32, 192, 192]             512
        Conv2d-152         [-1, 16, 192, 192]             512
    BatchNorm2d-153          [-1, 16, 192, 192]              32
    BatchNorm2d-154          [-1, 32, 192, 192]              64
            ReLU-155          [-1, 32, 192, 192]               0
        Conv2d-156         [-1, 16, 192, 192]           4,608
    BatchNorm2d-157          [-1, 16, 192, 192]              32
            ReLU-158          [-1, 16, 192, 192]               0
        Conv2d-159         [-1, 16, 192, 192]           2,304
ResDoubleConv-160         [-1, 16, 192, 192]               0
    ResUpBlock-161          [-1, 16, 192, 192]               0
        Conv2d-162           [-1, 1, 192, 192]              16
================================================================
Total params: 15,855,712
Trainable params: 15,855,712
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.84
Forward/backward pass size (MB): 14099753.81
Params size (MB): 60.48
Estimated Total Size (MB): 14099815.14
----------------------------------------------------------------
```

Another model that we made but wasn't used, was a ResUNeXt model, which was inspired from ResNeXt and UNet this model will be very useful for multiclass segmentation output, but for our needs it wasn't required since we had atmost 100 different classes to segment, which should be pretty easy.

anyways this was the summary of ResUNeXt

Refer to *vathos.model* for more details

Total Params: 16,409,856

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 192, 192]           4,800
            Conv2d-2         [-1, 64, 192, 192]           2,048
    BatchNorm2d-3         [-1, 64, 192, 192]             128
    BatchNorm2d-4         [-1, 32, 192, 192]              64
            ReLU-5         [-1, 32, 192, 192]               0
            Conv2d-6        [-1, 320, 192, 192]          10,240
    BatchNorm2d-7        [-1, 320, 192, 192]             640
            ReLU-8        [-1, 320, 192, 192]               0
            Conv2d-9        [-1, 320, 192, 192]          28,800
    BatchNorm2d-10        [-1, 320, 192, 192]             640
            ReLU-11        [-1, 320, 192, 192]               0
        Conv2d-12         [-1, 64, 192, 192]          20,480
```

(continues on next page)

(continued from previous page)

```
    ResDoubleConv-13         [-1, 64, 192, 192]               0
        MaxPool2d-14           [-1, 64, 96, 96]               0
    ResDownBlock-15  [[-1, 64, 96, 96], [-1, 64, 192, 192]]               0
        Conv2d-16          [-1, 128, 96, 96]           8,192
    BatchNorm2d-17          [-1, 128, 96, 96]             256
    BatchNorm2d-18           [-1, 64, 96, 96]             128
            ReLU-19           [-1, 64, 96, 96]               0
        Conv2d-20          [-1, 672, 96, 96]          43,008
    BatchNorm2d-21          [-1, 672, 96, 96]           1,344
            ReLU-22          [-1, 672, 96, 96]               0
        Conv2d-23          [-1, 672, 96, 96]         127,008
    BatchNorm2d-24          [-1, 672, 96, 96]           1,344
            ReLU-25          [-1, 672, 96, 96]               0
        Conv2d-26          [-1, 128, 96, 96]          86,016
    ResDoubleConv-27          [-1, 128, 96, 96]               0
        MaxPool2d-28          [-1, 128, 48, 48]               0
    ResDownBlock-29  [[-1, 128, 48, 48], [-1, 128, 96, 96]]               0
        Conv2d-30          [-1, 256, 48, 48]          32,768
    BatchNorm2d-31          [-1, 256, 48, 48]             512
    BatchNorm2d-32          [-1, 128, 48, 48]             256
            ReLU-33          [-1, 128, 48, 48]               0
        Conv2d-34         [-1, 1344, 48, 48]         172,032
    BatchNorm2d-35         [-1, 1344, 48, 48]           2,688
            ReLU-36         [-1, 1344, 48, 48]               0
        Conv2d-37         [-1, 1344, 48, 48]         508,032
    BatchNorm2d-38         [-1, 1344, 48, 48]           2,688
            ReLU-39         [-1, 1344, 48, 48]               0
        Conv2d-40          [-1, 256, 48, 48]         344,064
    ResDoubleConv-41          [-1, 256, 48, 48]               0
        MaxPool2d-42          [-1, 256, 24, 24]               0
    ResDownBlock-43  [[-1, 256, 24, 24], [-1, 256, 48, 48]]               0
        Conv2d-44          [-1, 512, 24, 24]         131,072
    BatchNorm2d-45          [-1, 512, 24, 24]           1,024
    BatchNorm2d-46          [-1, 256, 24, 24]             512
            ReLU-47          [-1, 256, 24, 24]               0
        Conv2d-48         [-1, 2720, 24, 24]         696,320
    BatchNorm2d-49         [-1, 2720, 24, 24]           5,440
            ReLU-50         [-1, 2720, 24, 24]               0
        Conv2d-51         [-1, 2720, 24, 24]       2,080,800
    BatchNorm2d-52         [-1, 2720, 24, 24]           5,440
            ReLU-53         [-1, 2720, 24, 24]               0
        Conv2d-54          [-1, 512, 24, 24]       1,392,640
    ResDoubleConv-55          [-1, 512, 24, 24]               0
        MaxPool2d-56          [-1, 512, 12, 12]               0
    ResDownBlock-57  [[-1, 512, 12, 12], [-1, 512, 24, 24]]               0
    BatchNorm2d-58          [-1, 512, 12, 12]           1,024
            ReLU-59          [-1, 512, 12, 12]               0
        Conv2d-60         [-1, 2720, 12, 12]       1,392,640
    BatchNorm2d-61         [-1, 2720, 12, 12]           5,440
            ReLU-62         [-1, 2720, 12, 12]               0
        Conv2d-63         [-1, 2720, 12, 12]       2,080,800
    BatchNorm2d-64         [-1, 2720, 12, 12]           5,440
            ReLU-65         [-1, 2720, 12, 12]               0
        Conv2d-66          [-1, 512, 12, 12]       1,392,640
    ResDoubleConv-67          [-1, 512, 12, 12]               0
        Conv2d-68         [-1, 2048, 12, 12]       1,048,576
    PixelShuffle-69          [-1, 512, 24, 24]               0
```

(continues on next page)

(continued from previous page)

```
        Conv2d-70          [-1, 512, 24, 24]         262,144
        Conv2d-71          [-1, 256, 24, 24]         131,072
    BatchNorm2d-72          [-1, 256, 24, 24]             512
    BatchNorm2d-73          [-1, 512, 24, 24]           1,024
            ReLU-74          [-1, 512, 24, 24]               0
        Conv2d-75         [-1, 1344, 24, 24]         688,128
    BatchNorm2d-76         [-1, 1344, 24, 24]           2,688
            ReLU-77         [-1, 1344, 24, 24]               0
        Conv2d-78         [-1, 1344, 24, 24]         508,032
    BatchNorm2d-79         [-1, 1344, 24, 24]           2,688
            ReLU-80         [-1, 1344, 24, 24]               0
        Conv2d-81          [-1, 256, 24, 24]         344,064
    ResDoubleConv-82          [-1, 256, 24, 24]               0
    ResUpBlock-83          [-1, 256, 24, 24]               0
        Conv2d-84         [-1, 1024, 24, 24]         262,144
    PixelShuffle-85          [-1, 256, 48, 48]               0
        Conv2d-86          [-1, 256, 48, 48]          65,536
        Conv2d-87          [-1, 128, 48, 48]          32,768
    BatchNorm2d-88          [-1, 128, 48, 48]             256
    BatchNorm2d-89          [-1, 256, 48, 48]             512
            ReLU-90          [-1, 256, 48, 48]               0
        Conv2d-91          [-1, 672, 48, 48]         172,032
    BatchNorm2d-92          [-1, 672, 48, 48]           1,344
            ReLU-93          [-1, 672, 48, 48]               0
        Conv2d-94          [-1, 672, 48, 48]         127,008
    BatchNorm2d-95          [-1, 672, 48, 48]           1,344
            ReLU-96          [-1, 672, 48, 48]               0
        Conv2d-97          [-1, 128, 48, 48]          86,016
    ResDoubleConv-98          [-1, 128, 48, 48]               0
    ResUpBlock-99          [-1, 128, 48, 48]               0
        Conv2d-100          [-1, 512, 48, 48]          65,536
    PixelShuffle-101          [-1, 128, 96, 96]               0
        Conv2d-102          [-1, 128, 96, 96]          16,384
        Conv2d-103           [-1, 64, 96, 96]           8,192
    BatchNorm2d-104           [-1, 64, 96, 96]             128
    BatchNorm2d-105          [-1, 128, 96, 96]             256
            ReLU-106          [-1, 128, 96, 96]               0
        Conv2d-107          [-1, 320, 96, 96]          40,960
    BatchNorm2d-108          [-1, 320, 96, 96]             640
            ReLU-109          [-1, 320, 96, 96]               0
        Conv2d-110          [-1, 320, 96, 96]          28,800
    BatchNorm2d-111          [-1, 320, 96, 96]             640
            ReLU-112          [-1, 320, 96, 96]               0
        Conv2d-113           [-1, 64, 96, 96]          20,480
ResDoubleConv-114           [-1, 64, 96, 96]               0
    ResUpBlock-115           [-1, 64, 96, 96]               0
        Conv2d-116          [-1, 256, 96, 96]          16,384
    PixelShuffle-117          [-1, 64, 192, 192]               0
        Conv2d-118          [-1, 64, 192, 192]           4,096
        Conv2d-119          [-1, 16, 192, 192]           1,024
    BatchNorm2d-120          [-1, 16, 192, 192]              32
    BatchNorm2d-121          [-1, 64, 192, 192]             128
            ReLU-122          [-1, 64, 192, 192]               0
        Conv2d-123          [-1, 64, 192, 192]           4,096
    BatchNorm2d-124          [-1, 64, 192, 192]             128
            ReLU-125          [-1, 64, 192, 192]               0
        Conv2d-126          [-1, 64, 192, 192]           1,152
```

(continues on next page)

(continued from previous page)

```
    BatchNorm2d-127         [-1, 64, 192, 192]             128
             ReLU-128         [-1, 64, 192, 192]               0
         Conv2d-129         [-1, 16, 192, 192]           1,024
ResDoubleConv-130         [-1, 16, 192, 192]               0
    ResUpBlock-131         [-1, 16, 192, 192]               0
         Conv2d-132          [-1, 1, 192, 192]              16
         Conv2d-133        [-1, 2048, 12, 12]       1,048,576
    PixelShuffle-134           [-1, 512, 24, 24]               0
         Conv2d-135         [-1, 512, 24, 24]         262,144
         Conv2d-136         [-1, 512, 24, 24]         131,072
         Conv2d-137          [-1, 64, 24, 24]          32,768
    BatchNorm2d-138          [-1, 64, 24, 24]             128
    BatchNorm2d-139         [-1, 512, 24, 24]           1,024
             ReLU-140         [-1, 512, 24, 24]               0
         Conv2d-141         [-1, 320, 24, 24]         163,840
    BatchNorm2d-142         [-1, 320, 24, 24]             640
             ReLU-143         [-1, 320, 24, 24]               0
         Conv2d-144         [-1, 320, 24, 24]          28,800
    BatchNorm2d-145         [-1, 320, 24, 24]             640
             ReLU-146         [-1, 320, 24, 24]               0
         Conv2d-147          [-1, 64, 24, 24]          20,480
ResDoubleConv-148          [-1, 64, 24, 24]               0
    ResUpBlock-149          [-1, 64, 24, 24]               0
         Conv2d-150         [-1, 256, 24, 24]          16,384
    PixelShuffle-151           [-1, 64, 48, 48]               0
         Conv2d-152          [-1, 64, 48, 48]          16,384
         Conv2d-153          [-1, 64, 48, 48]           8,192
         Conv2d-154          [-1, 64, 48, 48]           4,096
    BatchNorm2d-155          [-1, 64, 48, 48]             128
    BatchNorm2d-156          [-1, 64, 48, 48]             128
             ReLU-157          [-1, 64, 48, 48]               0
         Conv2d-158         [-1, 320, 48, 48]          20,480
    BatchNorm2d-159         [-1, 320, 48, 48]             640
             ReLU-160         [-1, 320, 48, 48]               0
         Conv2d-161         [-1, 320, 48, 48]          28,800
    BatchNorm2d-162         [-1, 320, 48, 48]             640
             ReLU-163         [-1, 320, 48, 48]               0
         Conv2d-164          [-1, 64, 48, 48]          20,480
ResDoubleConv-165          [-1, 64, 48, 48]               0
    ResUpBlock-166          [-1, 64, 48, 48]               0
         Conv2d-167         [-1, 256, 48, 48]          16,384
    PixelShuffle-168           [-1, 64, 96, 96]               0
         Conv2d-169          [-1, 64, 96, 96]           8,192
         Conv2d-170          [-1, 64, 96, 96]           4,096
         Conv2d-171          [-1, 32, 96, 96]           2,048
    BatchNorm2d-172          [-1, 32, 96, 96]              64
    BatchNorm2d-173          [-1, 64, 96, 96]             128
             ReLU-174          [-1, 64, 96, 96]               0
         Conv2d-175         [-1, 160, 96, 96]          10,240
    BatchNorm2d-176         [-1, 160, 96, 96]             320
             ReLU-177         [-1, 160, 96, 96]               0
         Conv2d-178         [-1, 160, 96, 96]           7,200
    BatchNorm2d-179         [-1, 160, 96, 96]             320
             ReLU-180         [-1, 160, 96, 96]               0
         Conv2d-181          [-1, 32, 96, 96]           5,120
ResDoubleConv-182          [-1, 32, 96, 96]               0
    ResUpBlock-183          [-1, 32, 96, 96]               0
```

(continues on next page)

(continued from previous page)

```
        Conv2d-184          [-1, 128, 96, 96]           4,096
    PixelShuffle-185          [-1, 32, 192, 192]               0
        Conv2d-186         [-1, 32, 192, 192]           2,048
        Conv2d-187         [-1, 32, 192, 192]             512
        Conv2d-188         [-1, 16, 192, 192]             512
    BatchNorm2d-189          [-1, 16, 192, 192]              32
    BatchNorm2d-190          [-1, 32, 192, 192]              64
             ReLU-191          [-1, 32, 192, 192]               0
        Conv2d-192         [-1, 64, 192, 192]           2,048
    BatchNorm2d-193          [-1, 64, 192, 192]             128
             ReLU-194          [-1, 64, 192, 192]               0
        Conv2d-195         [-1, 64, 192, 192]           1,152
    BatchNorm2d-196          [-1, 64, 192, 192]             128
             ReLU-197          [-1, 64, 192, 192]               0
        Conv2d-198         [-1, 16, 192, 192]           1,024
ResDoubleConv-199        [-1, 16, 192, 192]               0
    ResUpBlock-200          [-1, 16, 192, 192]               0
        Conv2d-201          [-1, 1, 192, 192]              16
================================================================
Total params: 16,409,856
Trainable params: 16,409,856
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.84
Forward/backward pass size (MB): 14098296.45
Params size (MB): 62.60
Estimated Total Size (MB): 14098359.89
----------------------------------------------------------------
```

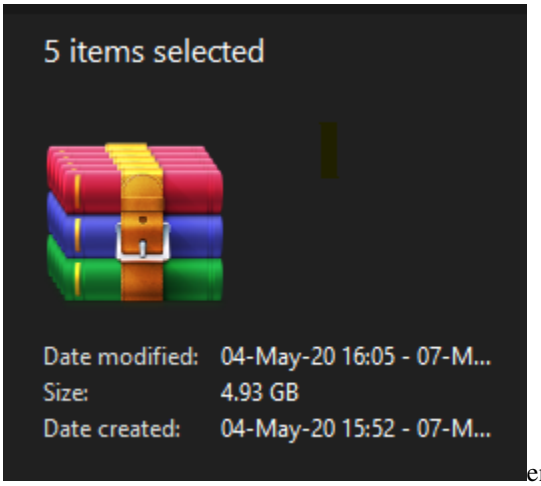
## 6.2 2. The Dataset

### 6.2.1 Dataset Google Drive link

https://drive.google.com/open?id=10MBvlf6pMB78o-bWNe7tVlqNaIP3DtKQ

`.zip`, no compression algorithm was used, `ZIP_STORE` option was used

#### Total Size



enter image description here

### 6.2.2 Dataset Creation

Github Link: https://github.com/satyajitghana/TSAI-DeepVision-EVA4.0/blob/master/14_RCNN/01_DenseDepth_DatasetCreation.ipy

https://github.com/satyajitghana/TSAI-DeepVision-EVA4.0/blob/master/14_RCNN/01_02_DenseDepth_DatasetCreation.ipynb

Colab Link: https://colab.research.google.com/github/satyajitghana/TSAI-DeepVision-EVA4.0/blob/master/14_RCNN/01_DenseDepth_DatasetCreation.ipynb

https://colab.research.google.com/github/satyajitghana/TSAI-DeepVision-EVA4.0/blob/master/14_RCNN/01_02_DenseDepth_Dataset

### 6.2.3 Depth Map creation

Colab link: https://colab.research.google.com/github/satyajitghana/TSAI-DeepVision-EVA4.0/blob/master/14_RCNN/02_DepthModel_DepthMap.ipynb

### 6.2.4 Mean and Standard Deviation

Github Link: https://github.com/satyajitghana/TSAI-DeepVision-EVA4.0/blob/master/14_RCNN/03_DepthModel_MeanStd.ipynb

Colab Link: https://colab.research.google.com/github/satyajitghana/TSAI-DeepVision-EVA4.0/blob/master/14_RCNN/03_DepthModel_MeanStd.ipynb

#### Dataset Stats:

1. BG Images
   - Mean: `['0.573435604572296', '0.520844697952271', '0.457784473896027']`
   - Std: `['0.207058250904083', '0.208138316869736', '0.215291306376457']`
1. FG_BG Images
   - Mean: `['0.568499565124512', '0.512103974819183', '0.452332496643066']`
   - Std: `['0.211068645119667', '0.211040720343590', '0.216081097722054']`

1. FG_BG_MASK Images
   - Mean: `['0.062296919524670', '0.062296919524670', '0.062296919524670']`
   - Std: `['0.227044790983200', '0.227044790983200', '0.227044790983200']`
1. DEPTH_FG_BG
   - Mean: `['0.302973538637161', '0.302973538637161', '0.302973538637161']`
   - Std: `['0.101284727454185', '0.101284727454185', '0.101284727454185']`

### 6.2.5 Dataset Visualization

Github Link: https://github.com/satyajitghana/TSAI-DeepVision-EVA4.0/blob/master/14_RCNN/04_DepthModel_DataViz.ipynb

Colab Link: https://colab.research.google.com/github/satyajitghana/TSAI-DeepVision-EVA4.0/blob/master/14_RCNN/04_DepthModel_DataViz.ipynb

Note: To view them larger, `right click -> Open image in new tab`

#### BG Images


enter
image description here

#### FG Images


enter
image description here

#### FG_BG Images


enter
image description here

#### FG_BG_MASK Images


enter
image description here

#### Depth_FG_BG Images


enter
image description here

### 6.2.6 How the dataset was created

Since we need to apply the foreground images on background images and also creating a mask of the fg images, i used transparent background png images, a image crawler was run on Bing to gather people, animals, dogs, cats, bears, goats, deer, cow and human images for the fg, mall interior, interior and indoor images were searched and crawled.

Now we converted the fg png images to mask by filling the transparent part with white and rest image with black using this code,

```
img = cv2.imread(fg_images[0], cv2.IMREAD_UNCHANGED)
ret, mask = cv2.threshold(im[:, :, 3], 0, 255, cv2.THRESH_BINARY)
```

For the BG Images, they were resized and cropped to `200x200` using this,

```
def crop_center(pil_img):
        img_width, img_height = pil_img.size
        crop_dim = img_width if img_width < img_height else img_height
        crop_width = crop_height = crop_dim

        return pil_img.crop(((img_width - crop_width) // 2, (img_height - ␣
↪crop_height) // 2, (img_width + crop_width) // 2, (img_height + crop_
↪height) // 2))
```

Once we've process this, we'll have `fg (100)`, `bg (100)`, `fg_mask (100)`, now we need to create the fg_bg images

Now to create the fg_bg images and also the fg_bg_mask images, we will place the fg images on top of bg images at random positions, 10 times, and do this with flipped fg images, in total we will have `fg (100) x bg (100) x flip (2) x place_random (10) = fg_bg (400,000) + fg_bg_mask (400, 000)`

Code to do this,

```
idx = 0
for bidx, bg_image in enumerate(tqdm(bgc_images)):

    if (bidx < last_idx):
        continue

        Path(f 'depth_dataset_cleaned/labels/').mkdir(parents = True, exist_ok = True)
        label_info = open(f "depth_dataset_cleaned/labels/bg_{bidx:03d}_label_info.txt
↪", "w+")

        idx = 4000 * bidx

        print(f 'Processing BG {bidx}')
        Path(f 'depth_dataset_cleaned/fg_bg/bg_{bidx:03d}').mkdir(parents = True,␣
↪exist_ok = True)
        Path(f 'depth_dataset_cleaned/fg_bg_mask/bg_{bidx:03d}').mkdir(parents = True,
↪ exist_ok = True)

        for fidx, fg_image in enumerate(tqdm(fgc_images)): #do the add fg to bg 20␣
↪times
            for i in range(20): #do this twice, one with flip once without
            for should_flip in [True, False]:
            background = Image.open(bg_image)
        foreground = Image.open(fg_image)
        fg_mask = Image.open(fgc_mask_images[fidx])
```

(continues on next page)

(continued from previous page)

```
        if should_flip:
            foreground = foreground.transpose(PIL.Image.FLIP_LEFT_RIGHT)
        fg_mask = fg_mask.transpose(PIL.Image.FLIP_LEFT_RIGHT)

        b_width, b_height = background.size
        f_width, f_height = foreground.size
        max_y = b_height - f_height
        max_x = b_width - f_width
        pos_x = np.random.randint(low = 0, high = max_x, size = 1)[0]
        pos_y = np.random.randint(low = 0, high = max_y, size = 1)[0]
        background.paste(foreground, (pos_x, pos_y), foreground)

        mask_bg = Image.new('L', background.size)

        fg_mask = fg_mask.convert('L')
        mask_bg.paste(fg_mask, (pos_x, pos_y), fg_mask)

        background.save(f 'depth_dataset_cleaned/fg_bg/bg_{bidx:03d}/fg_{fidx:03d}_bg_
↪{bidx:03d}_{idx:06d}.jpg', optimize = True, quality = 30)
        mask_bg.save(f 'depth_dataset_cleaned/fg_bg_mask/bg_{bidx:03d}/fg_{fidx:03d}_
↪bg_{bidx:03d}_mask_{idx:06d}.jpg', optimize = True, quality = 30)
        label_info.write(f 'fg_bg/bg_{bidx:03d}/fg_{fidx:03d}_bg_{bidx:03d}_{idx:06d}.
↪jpg\tfg_bg_mask/bg_{bidx:03d}/fg_{fidx:03d}_bg_{bidx:03d}_mask_{idx:06d}.jpg\t{pos_
↪x}\t{pos_y}\n')

        idx = idx + 1

        label_info.close()
        last_idx = bidx
```

For efficiency i wrote the generated file to .zip file, why was this done though ?

https://medium.com/@satyajitghana7/working-with-huge-datasets-800k-files-in-google-colab-and-google-drive-bcb175c79477

Once this was done, we need to create the depth map, by running the DenseDepth Model on our fg_bg images, this was done by taking batches of 1000, since otherwise we had memory bottleneck issues, moreover i had to manually use the python's garbage collector to make sure we free the memory after every batch

```
def run_processing(fr = 0, to = 10):
    print(f 'running process from {fr}(inclusive) to {to}(exclusive) BGs')
        for bdx, b_files in enumerate(tqdm(grouped_files[fr: to])):

            print(f 'Processing for BG {fr + bdx}')

        out_zip = ZipFile('depth_fg_bg.zip', mode = 'a', compression = zipfile.ZIP_
↪STORED)

        batch_size = 1000
        batch_idx = 0
        for batch in make_batch(b_files, batch_size):
            images = []
        print(f 'Processing Batch {batch_idx}')
        for idx, b_file in enumerate(tqdm(batch)):
            imgdata = fg_bg_zip.read(b_file)
        img = Image.open(io.BytesIO(imgdata))
        img = img.resize((640, 480))
        x = np.clip(np.asarray(img, dtype = float) / 255, 0, 1)
```

(continues on next page)

(continued from previous page)

```
        images.append(x)

        images = np.stack(images, axis = 0)
        print(f 'Running prediction for BG {fr + bdx} Batch {batch_idx}')
        t1 = time()
        output = predict(model, images)
        outputs = output.copy()
        t2 = time()
        print(f 'Prediction done took {(t2-t1):.5f} s')

        # resize the outputs to `200x200` and extract channel 0
        outputs = [resize(output, (200, 200))[: ,: , 0]
            for output in outputs
        ]

        # create a temporary directory to save the png outputs of current bg directory
        Path(f 'temp_b').mkdir(parents = True, exist_ok = True)

        print('Saving to Zip File')# for every output, save the output by appending␣
↪mask to it
        for odx, output in enumerate(tqdm(outputs)):
            _, parent_f, f_name = b_files[batch_idx * batch_size + odx].split(os.sep)
        f_name = f_name.split('.')[0]
        img = Image.fromarray(output * 255)
        img = img.convert('L')
        img.save(f 'temp_b/temp.png')

        out_zip.write('temp_b/temp.png', f 'mask_fg_bg/{parent_f}/mask_{f_name}.png')

        # cleanup files
        del output, outputs, images

        # garbage collect
        gc.collect()

        batch_idx = batch_idx + 1

        out_zip.close()
```

## 6.3 3. Loss Functions

GitHub Link : https://github.com/satyajitghana/ProjektDepth/blob/master/notebooks/11_DepthModel_ModelTrain_PlayWithLossFunctions.ipynb Colab Link : https://colab.research.google.com/github/satyajitghana/ProjektDepth/blob/master/notebooks/11_DepthModel_ModelTrain_PlayWithLossFunctions.ipynb

Segmentation Loss Functions:

- BCEWithLogits
- DiceLoss
- BCEDice
- TverskyLoss
- BCETversky

Depth Loss Functions:

- BerHu
- GradLoss
- BCEWithLogits
- RMSE
- SSIM

Various combinations of the above losses were tried with 1/16th of the dataset

A Small experiment was done to check if 30M params is really required or no, we found that 15M params model performed similar to 30M, so the higher param model was scraped

**With 30M Params:**


**With 15M Params:**


### 6.3.1 Playing with Loss Functions

Table 1: Loss Functions Comparison

| Seg Loss Funcion | Depth Loss Function | mIOU | mRMSE |
|---|---|---|---|
| BCE | BCE | 0.1278 | 0.1149 |
| Dice | BCE | 0.4515 | 0.0858 |
| Tversky | BCE | 0.3221 | 0.1018 |
| BCEDice | BCE | 0.4578 | 0.0993 |
| BCETversky | BCE | 0.3831 | 0.0936 |
| BCEDice | BerHu | 0.4214 | 0.2882 |
| BCEDice | RMSE | 0.4366 | 0.0774 |
| BCEDice | Grad | 0.4542 | 0.1795 |
| BCEDice | SSIM | 0.4413 | 0.1304 |

**Note:**

- mIOU: higher is better
- mRMSE: lower is better

**1. seg_loss = BCEWithLogits, depth_loss = BCEWithLogits**

**2. seg_loss = DiceLoss, depth_loss = BCEWithLogits**

**3. seg_loss = TverskyLoss, depth_loss = BCEWithLogits**

**4. seg_loss = BCEDiceLoss, depth_loss = BCEWithLogits**


Fig. 2: bce_bce

```
mIOU : 0.12789911031723022
mRMSE : 0.11494194716215134
total time : 196.1347 s
```


Fig. 3: dice_bce

```
mIOU : 0.4515075087547302
mRMSE : 0.08582810312509537
total time : 193.0364 s
```


Fig. 4: tversky_bce

```
mIOU : 0.32213953137397766
mRMSE : 0.10182604193687439
total time : 193.5620 s
```

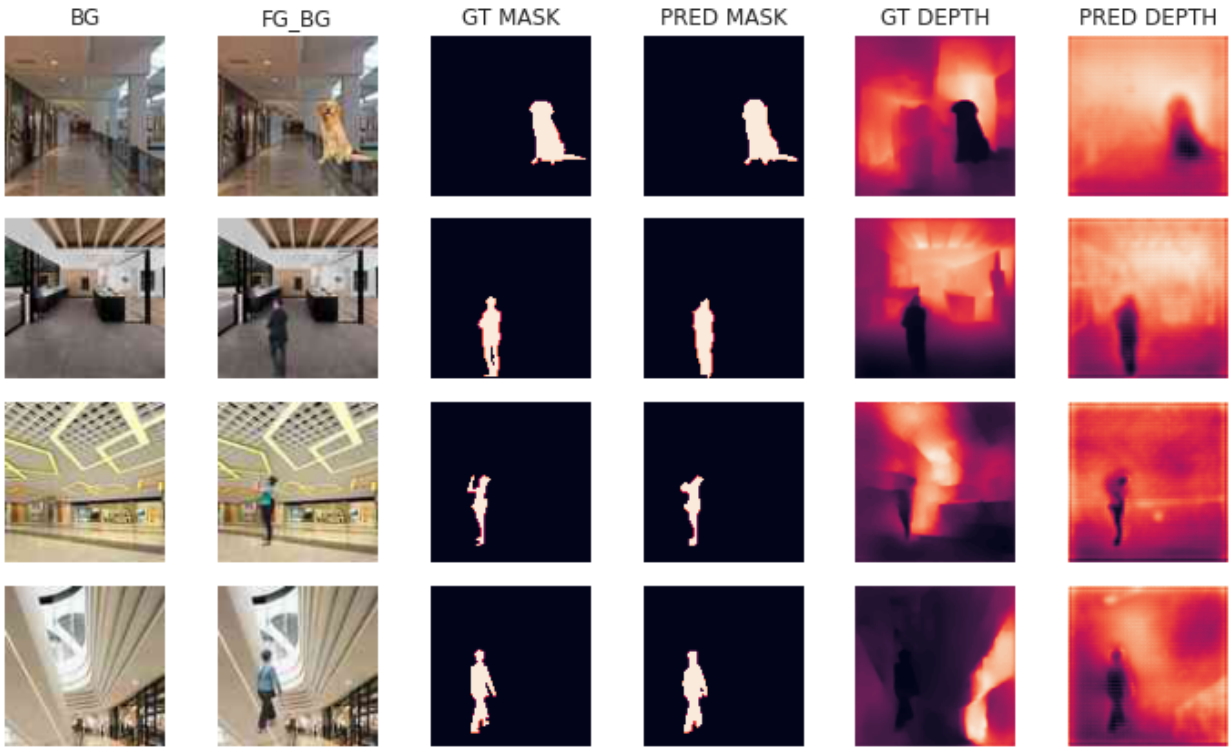



Fig. 5: bcedice_bce

```
mIOU : 0.4578476846218109
mRMSE : 0.09939917922019958
total time : 191.8000 s
```

**5. seg_loss = BCETverskyLoss, depth_loss = BCEWithLogits**


Fig. 6: bcetversky_bce

```
mIOU : 0.3831656873226166
mRMSE : 0.0936645045876503
total time : 192.6121 s
```

**6. seg_loss = BCEDiceLoss, depth_loss = BCEWithLogits**

**7. seg_loss = BCEDiceLoss, depth_loss = BerHuLoss**

**8. seg_loss = BCEDiceLoss, depth_loss = RMSELoss**

**9. seg_loss = BCEDiceLoss, depth_loss = GradLoss**

**10. seg_loss = BCEDiceLoss, depth_loss = SSIMLoss**

## 6.4 4. Testing Timings and Hyper Params

GitHub Link : https://github.com/satyajitghana/ProjektDepth/blob/master/notebooks/08_DepthModel_Experiments_Timings.ipynb Colab Link : https://colab.research.google.com/github/satyajitghana/ProjektDepth/blob/master/notebooks/08_DepthModel_Experiments_Timings.ipynb

GPU: Tesla P100

### 6.4.1 Playing with batch_size

```
BATCH_SIZE = 16
```

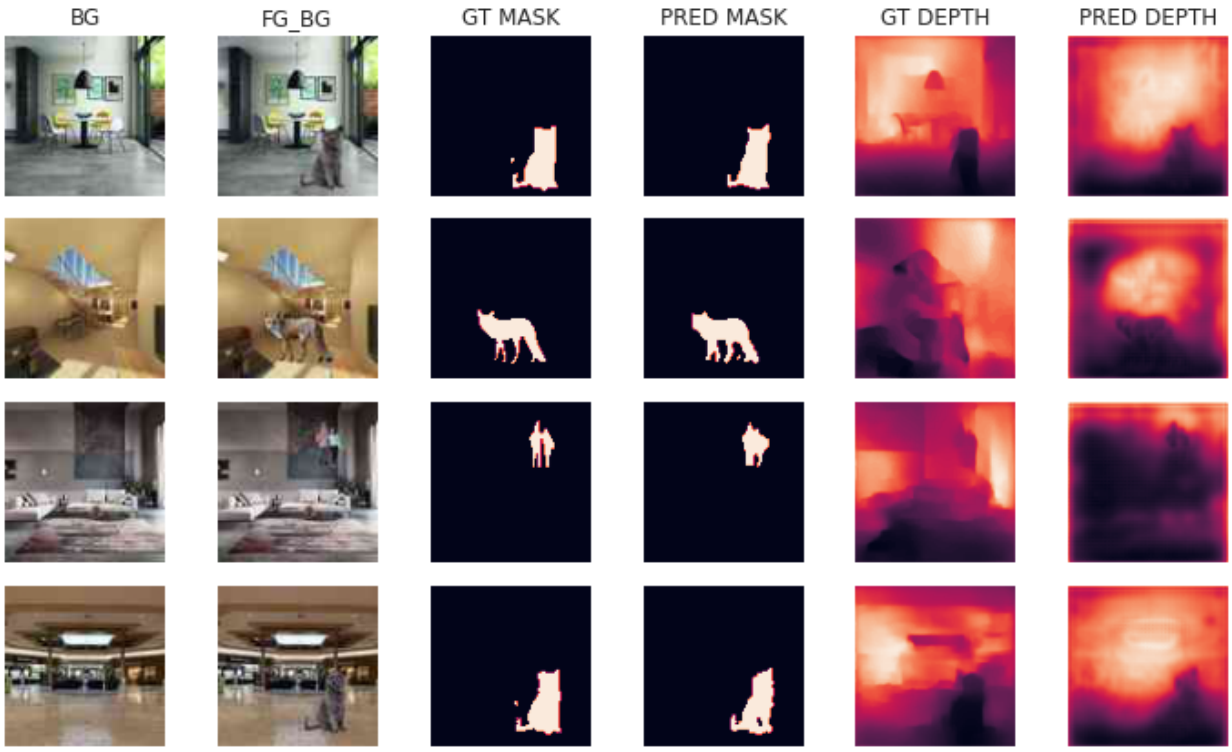



Fig. 7: bcedice_bce

```
mIOU : 0.4485453963279724
mRMSE : 0.12491746991872787
total time : 193.3488 s
```


Fig. 8: bcedice_berhu

```
mIOU : 0.42147812247276306
mRMSE : 0.2882708013057709
total time : 193.7522 s
```


Fig. 9: bcedice_rmse

```
mIOU : 0.4366089999675751
mRMSE : 0.07745874673128128
total time : 180.7616 s
```

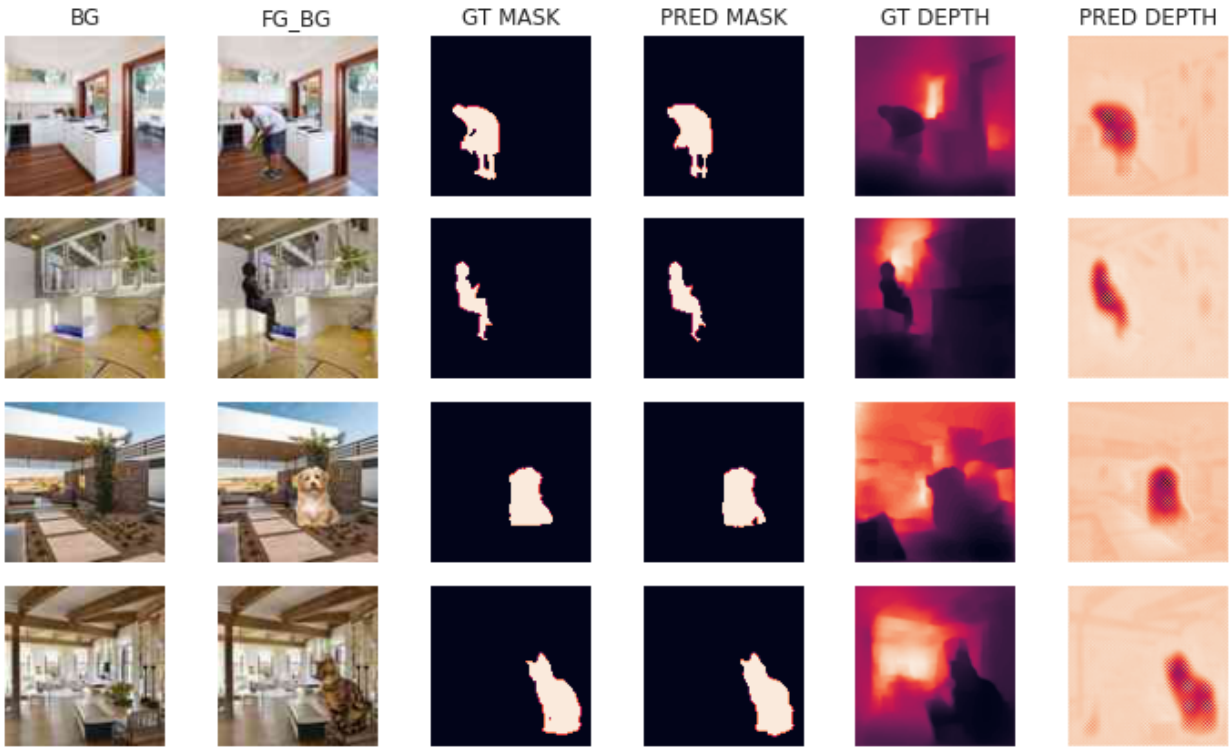



Fig. 10: bcedice_grad

```
mIOU : 0.4542521834373474
mRMSE : 0.1795133352279663
total time : 185.2947 s
```


Fig. 11: bcedice_ssim

```
mIOU : 0.4413087069988251
mRMSE : 0.1304335743188858
total time : 189.7473 s
```

```
total time : 170.1182 s
the model took : 154.6660 s i.e. 0.9092 % of total execution
data loading took : 0.9110 s i.e. 0.0054 % of total execution
others took : 13.0727 s i.e. 0.0768 % of total execution
```

BATCH_SIZE = 32

```
total time : 157.4680 s
the model took : 149.0861 s i.e. 0.9468 % of total execution
data loading took : 0.7408 s i.e. 0.0047 % of total execution
others took : 6.7184 s i.e. 0.0427 % of total execution
```

BATCH_SIZE = 64

```
total time : 149.5570 s
the model took : 144.6822 s i.e. 0.9674 % of total execution
data loading took : 0.6090 s i.e. 0.0041 % of total execution
others took : 3.5036 s i.e. 0.0234 % of total execution
```

BATCH_SIZE = 128

```
total time : 145.5745 s
the model took : 142.2546 s i.e. 0.9772 % of total execution
data loading took : 0.5900 s i.e. 0.0041 % of total execution
others took : 1.8308 s i.e. 0.0126 % of total execution
```

BATCH_SIZE = 256

```
RuntimeError: CUDA out of memory. Tried to allocate 1.12 GiB (GPU 0; 15.90 GiB total␣
↪capacity; 12.72 GiB already allocated; 1023.81 MiB free; 14.20 GiB reserved in␣
↪total by PyTorch)
```

### 6.4.2 Testing Model Timings

```
model = ResUNet()
model = model.to(device)
lossfn = nn.BCEWithLogitsLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

meow = torch.utils.data.Subset(train_subset, range(0, len(train_subset)//1))
meow_loader = torch.utils.data.DataLoader(meow, batch_size=128, shuffle=True, num_
↪workers=2, pin_memory=True)
pbar = tqdm(meow_loader, dynamic_ncols=True)

other_time = 0

start = time()

other_s = time()
model.train()
other_e = time()

other_time += other_e - other_s
```

(continues on next page)

(continued from previous page)

```
data_load_time = 0
model_time = 0

meow_time = 0

for batch_idx, data in enumerate(pbar):

    other_s = time()
    optimizer.zero_grad()
    other_e = time()

    other_time += other_e - other_s

    load_s = time()

    data['bg'] = data['bg'].to(device)
    data['fg_bg'] = data['fg_bg'].to(device)
    data['depth_fg_bg'] = data['depth_fg_bg'].to(device)
    data['fg_bg_mask'] = data['fg_bg_mask'].to(device)

    load_e = time()

    data_load_time += load_e - load_s

    model_s = time() # model start

    x = torch.cat([data['bg'], data['fg_bg']], dim=1)
    d_out, s_out = model(x)

    l1 = lossfn(d_out, data['depth_fg_bg'])
    l2 = lossfn(s_out, data['fg_bg_mask'])

    loss = 2*l1 + l2

    loss.backward()
    optimizer.step()
    model_e = time() # model end

    model_time += model_e - model_s

    other_s = time()

    pbar.set_description(desc=f'loss={loss.item():.10f} batch_id={batch_idx}')


    # del data # and this shit was taking 0.07% of mah time
    if batch_idx % 200 == 0:
        torch.cuda.empty_cache() # this shit takes 8% of my frikking time

    other_e = time()

    other_time += other_e - other_s
end = time()

print(f'total time : {end-start:.4f} s')
print(f'the model took : {model_time:.4f} s i.e. {(model_time / (end - start)):.4f} %␣
↪of total execution')
```

(continues on next page)

(continued from previous page)

```
print(f'data loading took : {data_load_time:.4f} s i.e. {(data_load_time / (end - start)):.4f} % of total execution')
print(f'others took : {other_time:.4f} s i.e. {(other_time / (end - start)):.4f} % of total execution')
```

OUTPUT

```
total time : 2331.8488 s
the model took : 2289.7168 s i.e. 0.9819 % of total execution
data loading took : 10.2884 s i.e. 0.0044 % of total execution
others took : 28.5874 s i.e. 0.0123 % of total execution
```

## 6.5 5. Run on TPU !

Github Link : https://github.com/satyajitghana/ProjektDepth/blob/master/notebooks/09_DepthModel_Experiments_TPU.ipynb Colab Link : https://colab.research.google.com/github/satyajitghana/ProjektDepth/blob/master/notebooks/09_DepthModel_Experiments_TPU.ipynb

Refer to this article: https://medium.com/@satyajitghana7/speed-up-your-model-training-w-tpu-on-google-colab-c55ac0f634d9

Running on TPU almost halfed the training time, but at the time of writing this documentation, the code doesn't work, or is buggy since the PyTorch-XLA package is being updated frequently and there has been a lot of changes that's being introduced, which is the reason i decided to stay away from it when actually training my model. But its good for small trains.

## 6.6 6. Training on Small Dataset

Github Link : https://github.com/satyajitghana/ProjektDepth/blob/master/notebooks/13_DepthModel_TrainOnSmallDataset.ipynb Colab Link : https://colab.research.google.com/github/satyajitghana/ProjektDepth/blob/master/notebooks/13_DepthModel_TrainOnSmallDataset.ipynb

Now the model was trained on `96x96` images, so then later we can perform transfer learning to train on `192x192` images

The entire training took about 5.5 hrs, but its actually took more, since there were bugs that i had to patch up and then restart the run.

Albeit the program does checkpoint every epoch and also stores the best accuracy model, every model metric is logged using Tensorboard.

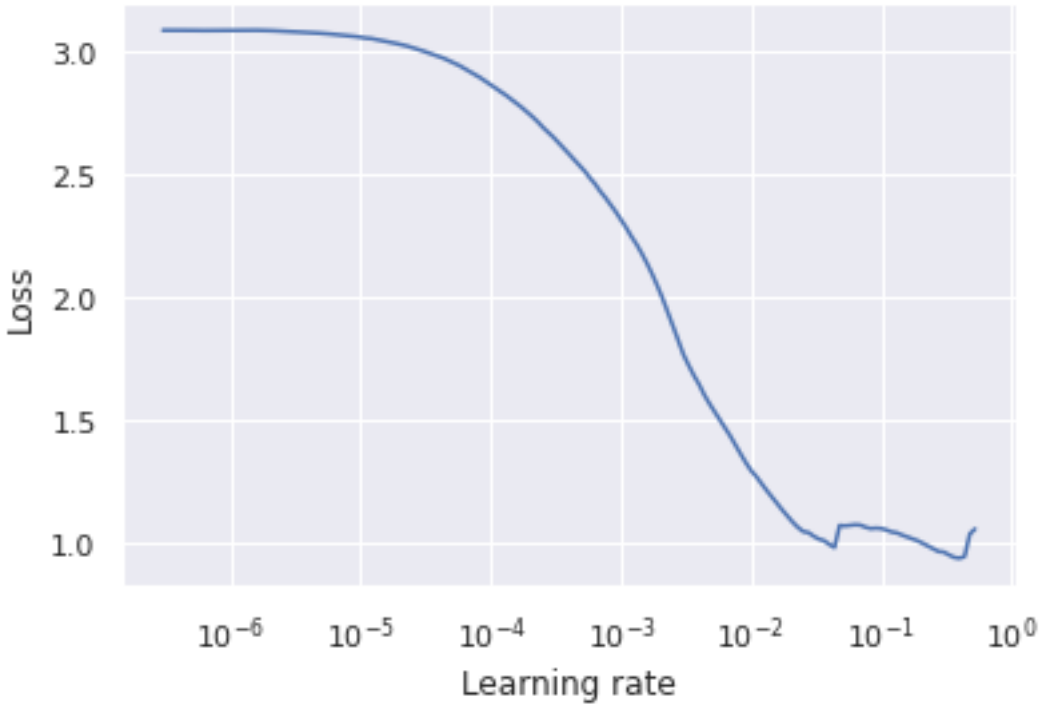
Quite a tiring experience overall

### 6.6.1 LR Range Test

GitHub Link : https://github.com/satyajitghana/ProjektDepth/blob/master/notebooks/12_DepthModel_LRRangeTest.ipynb Colab Link : https://colab.research.google.com/github/satyajitghana/ProjektDepth/blob/master/notebooks/12_DepthModel_LRRangeTest.ipynb

So the max_lr should be about `0.2-0.4`

Colab was kept alive by using my chrome extension https://github.com/satyajitghana/colab-keepalive




### 6.6.2 Train for 15 Epochs

#### LR Value

#### Train Loss

Fig. 12: Train - Segmentation Loss

#### Train Accuracy

#### Test Loss

#### Test Accuracy

### 6.6.3 Results

#### After 1st epoch

#### After 2nd epoch

#### After 4th epoch

#### After 9th epoch

Fig. 13: Train Depth Loss

Fig. 14: Train mIOU

#### After 11th epoch

#### After 15th epoch

## 6.7 7. Dataset and Model Exploration Log

### 6.7.1 Ideas

- Train the model on `96x96` images then move on to larger `192x192` images, since i made the images dataset of `200x200` i'll have to resize them, dumb me, should have thought of taking the size in multiple of `32`, that's what the GPU s.
- Encoder-Decoder Model ? might be overkill ? should try it though, but seems like i could get away without using ED model
- Go through the various loss functions: https://pytorch.org/docs/stable/nn.html#loss-functions
- First train the network images (Encoder) so it learns to identify the objects and segment them, then teach the network to do depth estimation on large images as well as segmentation. - [X] Nope we'll train the entire model with smaller images first and then train with large images
- https://github.com/fastai/fastai/blob/master/fastai/layers.py#L202 read this to understand how to initialize the pixel shuffle
- Use https://github.com/OniroAI/MonoDepth-PyTorch/blob/master/models_resnet.py for reference
- Use PreActivated ResNetV2, since its proven to be working better

There are so many mistakes in sir's code, why did he use interpolation ? why not use deconvolution ? it will not introduce the checkerboard issue. I figures i should use PixelShuffle, let's see how good it works

- Use Pixel Shuffel algorithm to increase resolution
- Maybe Use DepthWise Separation Convolution (MobileNet uses this)
- Use ResNeXt like architecture, let each kernel choose which object to segment
- Use IoU as a metric to tell how good my model is
- Try DiceLoss ?
- Try BCE Loss ?
- Try DICE + BCE Loss
- Try Jaccard Loss ? (Hybrid Loss as mentioned in Stacked Unet)
- Try image gradient loss ? from here https://github.com/wolverinn/Depth-Estimation-PyTorch
- Read this https://heartbeat.fritz.ai/research-guide-for-depth-estimation-with-deep-learning-1a02a439b834

Fig. 15: Train mRMSE

Fig. 16: Test - Segmentation Loss

Fig. 17: Test Depth Loss

- Study https://github.com/wolverinn/Depth-Estimation-PyTorch/blob/master/fyn_main.py
- Sudy UNet
- Study ResNeXt
- Add a pre convolution

Higher Batch Size, Higher Learning Rate

Make Unet-Resnet Architecture, Then add Pixel Shuffling, Then use ResNeXt

Combine all these ideas to make the final model, All the best

BOOOO

### 6.7.2 TODO

- [x] Make Unet-Resnet
- [ ] Try the different loss functions
- [x] Add Pixel Shuffling
- [x] Check if everything works with ResNet and smaller dataset
- [x] Add ResNeXt
- [ ] Try DataAugmentation
- [ ] ~~Check if everything works with ResNeXt with smaller dataset~~
- [x] Create a Deeper Network
- [x] Reduce the Filters on the Segmentation Decoder
- [ ] Make the library
- [ ] Add save model checkpoint
- [ ] Add Tensorboard
- [ ] Train ! Train ! Train !

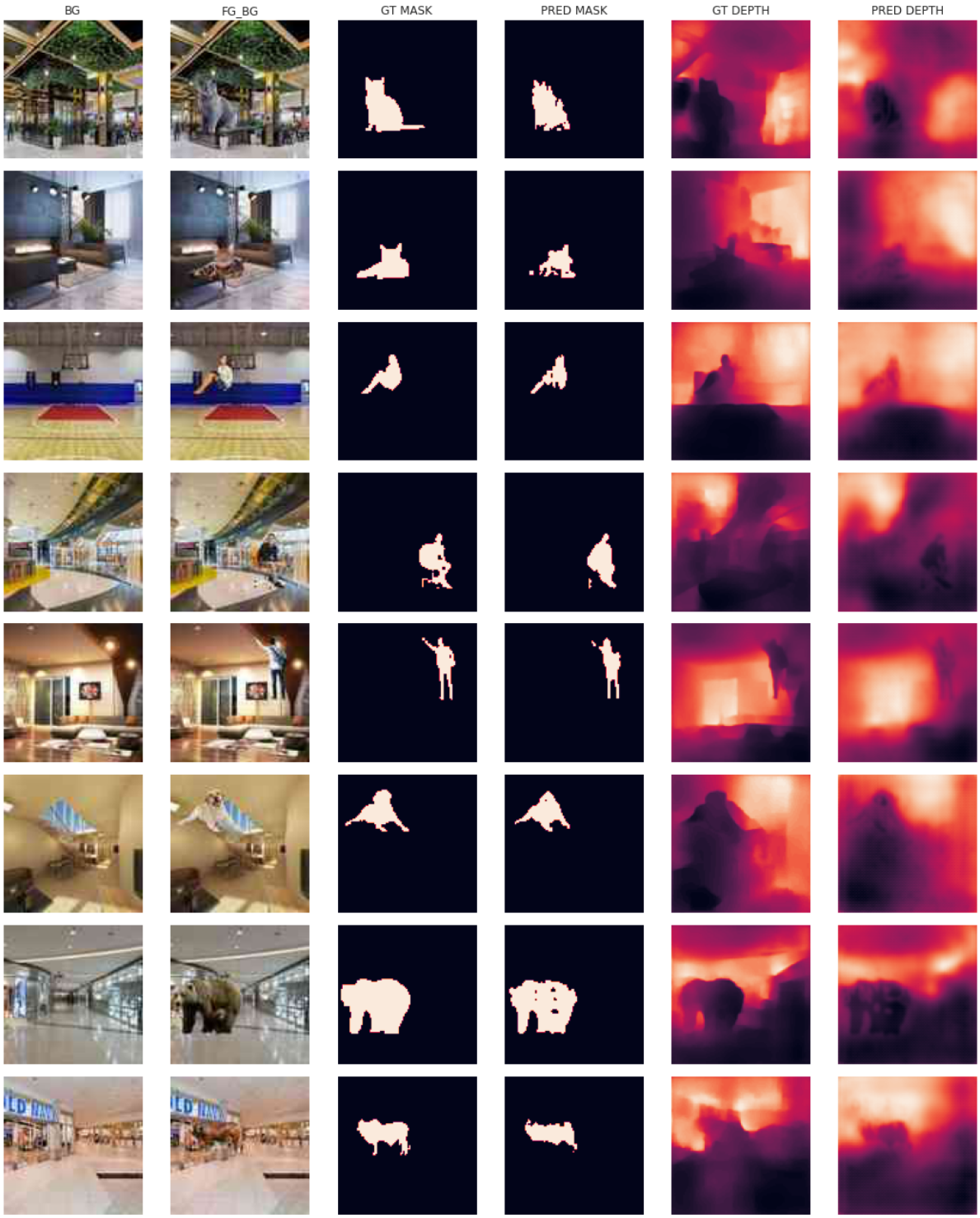
### 6.7.3 Model

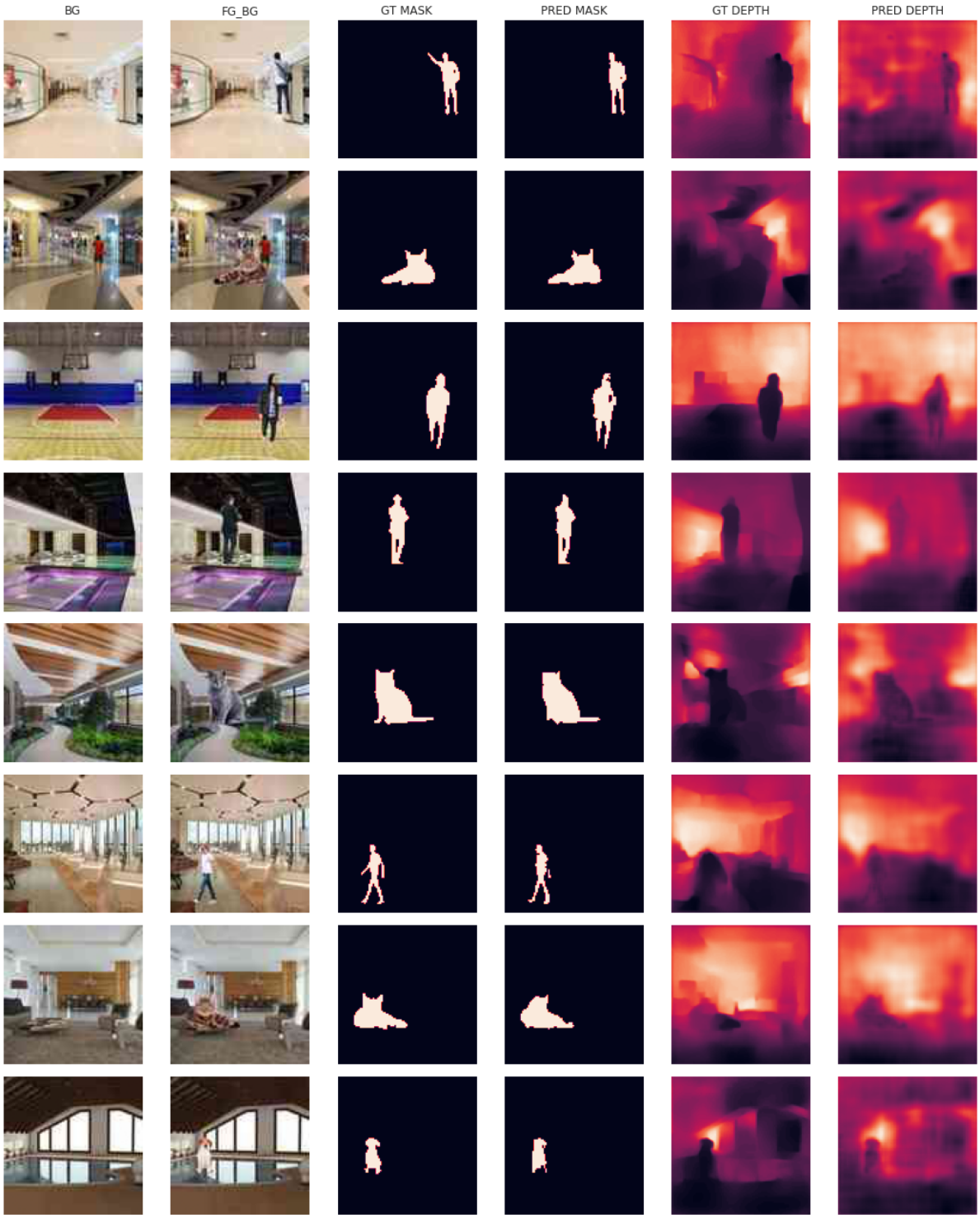
```
Encoding -> Bridge -> Decoder1
                   -> Decoder2
```

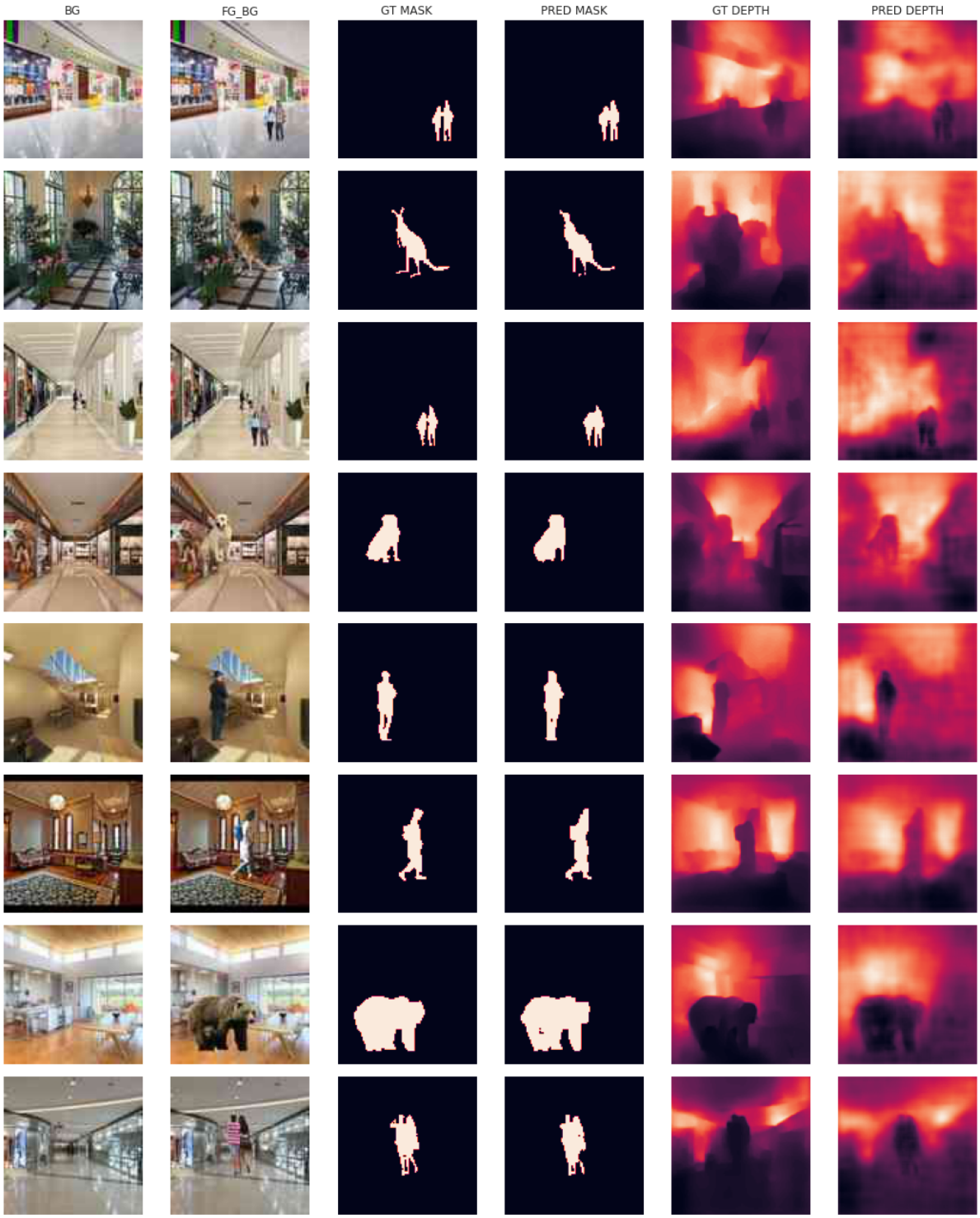
Fig. 18: Test mIOU

Fig. 19: Test mRMSE


BG
FG_BG
GT MASK
PRED MASK
GT DEPTH
PRED DEPTH


BG
FG_BG
GT MASK
PRED MASK
GT DEPTH
PRED DEPTH


BG
FG_BG
GT MASK
PRED MASK
GT DEPTH
PRED DEPTH


BG
FG_BG
GT MASK
PRED MASK
GT DEPTH
PRED DEPTH


BG
FG_BG
GT MASK
PRED MASK
GT DEPTH
PRED DEPTH

### 6.7.4 Requirements

- 70-30 train-test split

### 6.7.5 Notes

1. All convolution operations are with 3*3 filters and with the SAME padding thus the size of the feature map remains the same on each level of contracting path and corresponding expanding path. With the same padding, the boundary information is preserved and it also allows for more convolutions to be added.
2. Because the feature size remains the same on a single level, cropping of the feature map from the contracting path is not required in order to concatenate with the corresponding feature map of the expanding path. No cropping means no loss of information.
3. Along with the long skip connection between every level of contracting and expanding paths, we have local skip connection between convolutions on each level. Skip connection helps in getting a smooth loss curve and also helps to avoid gradient disappearance and explosion.

### 6.7.6 Colab Accounts Management

- Dataset Referenced Links : shadowleaf.deeplearning@gmail.com
- Actual Dataset: shadowleaf.contact@gmail.com
- Notebooks: shadowleaf.contact@gmail.com

### 6.7.7 Further Improvements

- Try better quality images dataset, probably i compressed the jpeg too much

### 6.7.8 Targets

- 11th May - Research about the possible models
- 12th May - Research about the shortlisted models
- 13th May - Create Models Conceptually, Research on Possible Loss Functions
- 14th May - Implement the Models and create the smaller dataset of 96x96
- 15th May - Try different loss functions and train with smaller dataset, ~~Create the library therefore with Tensorboard~~
- 16th May - ~~Train Train Train~~ Failed at running the model
- 17th May - ~~Train Train Train~~ Failed at running the model
- 18th May - The model works ! no memory leaks !
- 19th May - Tested the model works on TPU, reduced time by half, lower the model size ? create the library, fix the model
- 20th May - Create the library and train on`96x96`

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Index