HOW TO TRAIN YOUR DRAGON

TRAINING VISION MODEL FOR MONOCULAR DEPTH ESTIMATION AND MASK GENERATION

DATASET IS THE **BEAST**

1.2 Million images,

Uncompressed size = ~20GB

Time to Extract Entire Dataset = "Forever" (~15 m to 2h on Colab)
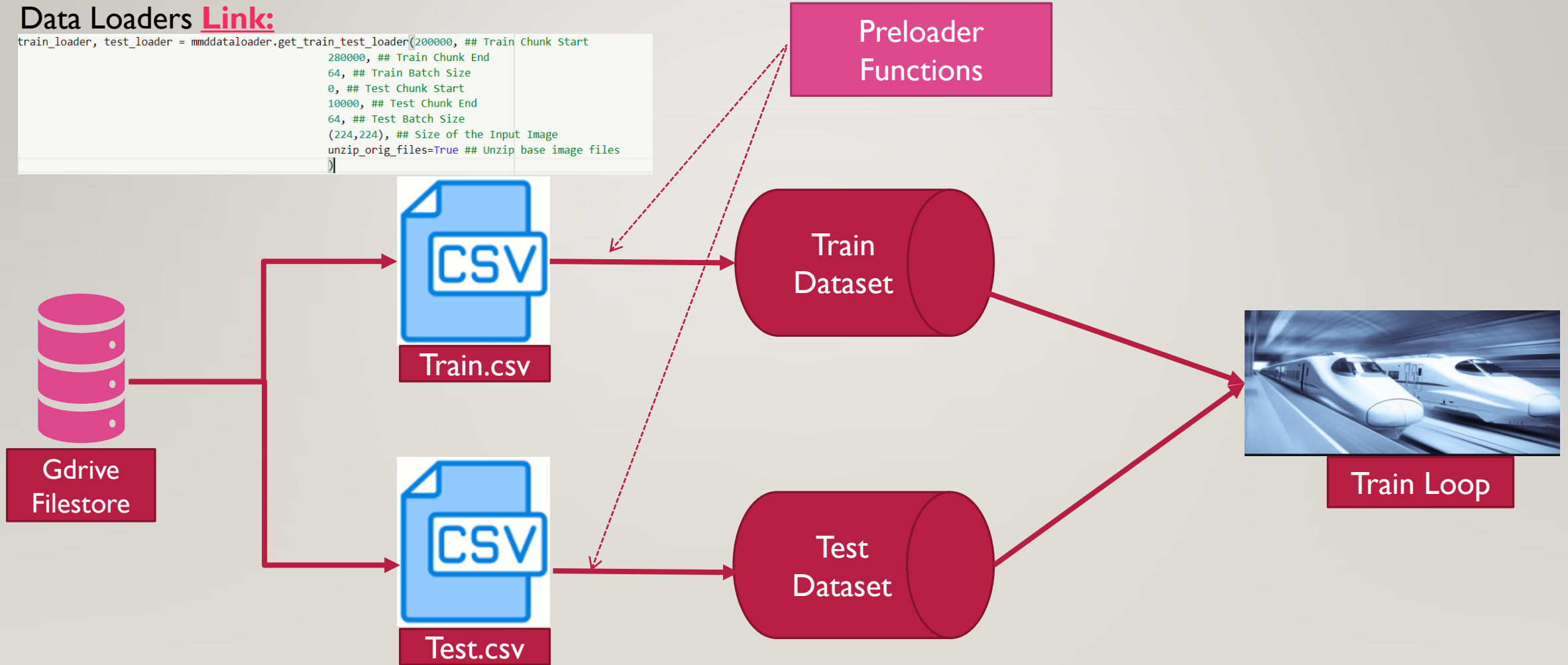
Data Sources Profiling link

Data Management is a major challenge. Following options were considered:

Option	Observations
Bulk load all data	Easy access. Highly unreliable during extraction and long preload time
Create HDF5 dataset	Awesome r/w speeds. Diskspace even with compression ~10x
GCS Buckets	Easy access r/w speeds. High latency ~2.4ms
Torch dataloaders with Zip	Decent r/w speeds and per-epoch speeds. Unreliable as training proceeds due to parallel access.
Torch dataloaders with selective preload	Much better r/w speeds and per-epoch speeds. Preload time limited to the chunk getting loaded.

DATA PIPELINE

Data Loaders **Link:**

```
train_loader, test_loader = mmddataloader.get_train_test_loader(200000, ## Train Chunk Start
                                    280000, ## Train Chunk End
                                    64, ## Train Batch Size
                                    0, ## Test Chunk Start
                                    10000, ## Test Chunk End
                                    64, ## Test Batch Size
                                    (224,224), ## Size of the Input Image
                                    unzip_orig_files=True ## Unzip base image files
                                    )
```
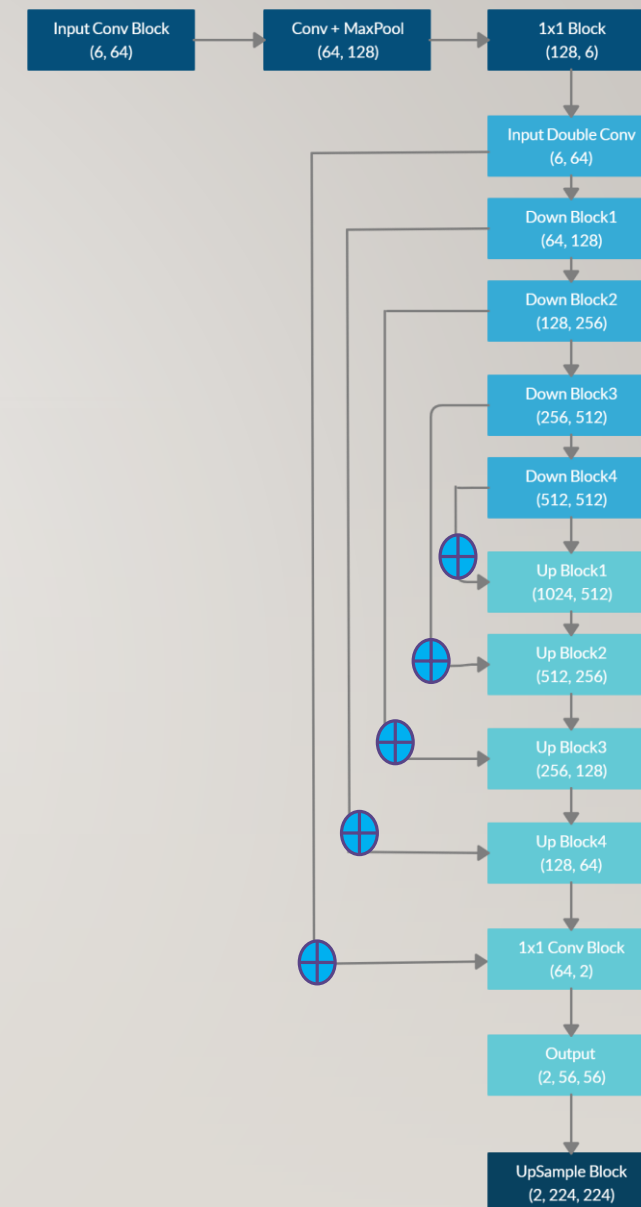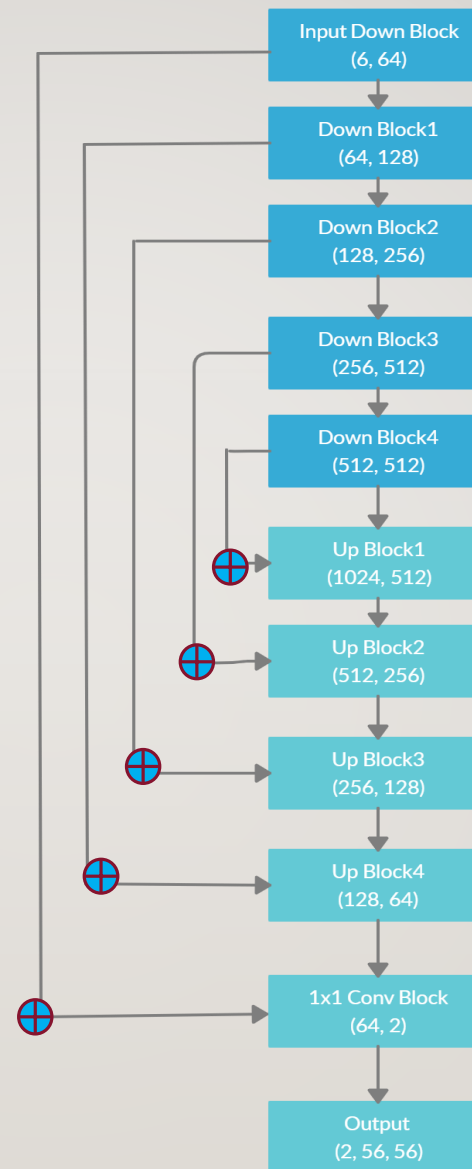

Preloader Functions

Gdrive Filestore

CSV — Train.csv

CSV — Test.csv

Train Dataset

Test Dataset

Train Loop

DNN MODELS

Model	Parameter Count	Link
Modified Unet-224	Total params: **17,532,622** Trainable params: **17,532,622** Non-trainable params: 0 Input size (MB): 1.15 Forward pass size (MB): 324.75 Params size (MB): 66.88 Estimated Total Size (MB): 392.78	Base Model **Link** Final Model **Link**

1. The final model consists of two parts:
   - Base Unet-56: Base U-Net styled model with
     - Input_size of (6, 56, 56) and
     - outputs (2, 56, 56)
     - Param Count: ~17.3 M
   - Double Conv Layer + Base Unet-56 + UpSampling layer:
     - Input_size of (6, 224, 224) and
     - Output Size (2, 224, 224)
     - Param Count: ~17.6M

DNN MODELS – OTHER MODELS EVALUATED

Model	Parameter Count	Challenges/Observations	Link

Plain Resnet50 with last layers modified to (2, 224, 224)	`Total params: ` **`20,132,864`**
	`Trainable params: `**`20,132,864`**
	`Non-trainable params: 0`
	`Input size (MB): 1.15`
	`Forward pass size (MB): 2751.66`
	`Params size (MB): 76.80`
	`Estimated Total Size (MB): 2829.61`
		Checkerboard/Vertical Strides observed.
		
			MonoMaskDepthResnet

Resnet-18 + Unet	`Total params: ` **`28,476,354`**
	`Trainable params: `**`28,476,354`**
	`Non-trainable params: 0`
	`Input size (MB): 1.15`
	`Forward pass size (MB): 2751.66`
	`Params size (MB): 76.80`
	`Estimated Total Size (MB): 2829.61`
		Long convergence time and difficult to train larger batch size.	ResnetUnet

TRAINING STRATEGY

Training Stage-1
(Strengthen Spine)

1. Use BaseModel
2. 56x56 Image Size
3. Large Batch Size
4. Cover Full Dataset in chunks of 50K
5. Target decent performance.

Main Notebook Link(For Stg2 and Stg3):
https://github.com/rajy4683/S15FinalSubmission/blob/master/S15EVA4_FinalOutput.ipynb

Training Stage-1: Sample
https://github.com/rajy4683/S15FinalSubmission/blob/master/S15Unet6_Large_56.ipynb

Training Stage-2
(Add head and tail, train only head and tail)
1. Head and tail added to Baselined Model from Stage-1
2. Freeze all Spine Layers
3. Image Size (224x224)
4. Medium Batch Size(64)
5. Cover Full Dataset in chunks of 50K

Training Stage-3
(Full-Scale Training)

1. Unfreeze all layers and perform necessary tuning
2. Train full Image Size
3. Cover Full Dataset in chunks of 100K

TRAINING STRATEGY

1. Adopt multi-task learning strategy(i.e train for both Masks and Depth Prediction) from the beginning (mainly inspired from EIP4 and HyperFace).
2. Randomly sample full training dataset and create chunks of 50K. Each chunk had a decent combination of various fg/bg combinations. Classes of BG were: (classrooms, college_outdoors, corridors, dining_room, lobby, malls, meeting_rooms)
3. Training Stage-1 (Strengthen Spine)
   1. For training each chunk:
      1. Base Unet model with input_size (6,56,56) was used (fg_bg images and plain bg images were concatenated)
      2. Initial iterations used simple loss functions(RMSE) and basic augmentations like Horizontal Flip.
      3. Large batch size of up to 256 (on P100 whenever available)
      4. Checkpoint major improvements and retrain from the previous checkpoint.
      5. Evaluated Multiple Loss Functions and finalized on
4. Training Stage-2 (Add Head and Tail)
   1. Add Double-Conv Layer to accept (6, 224, 224) as the input and UpSample output from previous stage to 4x
   2. Repeat Training Loop over in chunks

5. Training Stage-3 (Unfreeze all layers)
   1. Tune hyperparameters and repeat Training Loop over in chunks

LOSS FUNCTIONS

Loss Functions Evaluated	Task	Observation
RMSE	Depth Prediction and Mask Formation	Very helpful in mask prediction tasks. Acts as a good regularizer when used with other loss functions
L1 Loss	Depth Prediction and Mask Formation	Provides better convergence than RMSE/L2. Acts as a good regularizer when used with other loss functions
BCELossWithLogits	Mask Formation	Resulted in noisy output on multiple datasets
Dice Loss	Mask Formation	For masks, feature formation was much faster with Dice Loss
SSIM Loss	Depth Prediction	Quite reliable and provides faster feature formation
Pixel Weighted L1/L2 Loss	Mask Formation	Intention was to use it with other loss functions to regularize but seems to cause large noise over images.

Final Loss Function used:
1. For Mask Prediction: **L1Loss + Dice Loss**

1. For Depth Estimation: **RMSE Loss + SSIM Loss**

Losses Module **Link**:

PROFILING RESULTS

For profiling runs, cProfile python module was used for training loops
Two striking observations:
1. SSIM Loss takes a whopping 56ms per call
2. Any .item() call causes CUDA synchronization which consumes ~171ms

ncalls	tottime	percall	cumtime	percall	TimePercent	filename:lineno(function)
1	0.036	0.036	181.691	181.691	100%	mmdtraintest.py:216(execute_model)
2	0.128	0.064	177.648	88.824	98%	mmdtraintest.py:114(train_batch)
644	0.194	0	45.075	0.07	25%	mmdlosses.py:20(construct_criterion)
322	0.002	0	44.001	0.137	24%	mmdlosses.py:59(LocalSSIMLoss)
322	0.275	0.001	43.755	0.136	24%	ssim.py:82(forward)
322	0.013	0	6.239	0.019	3%	mmdmodels.py:169(forward)
5152/3542	0.08	0	4.966	0.001	3%	container.py:98(forward)
3542	0.014	0	4.697	0.001	3%	mmdmodels.py:30(forward)
322	0.015	0	4.562	0.014	3%	mmdmodels.py:105(forward)
314	0.003	0	3.915	0.012	2%	tensor.py:170(backward)
314	0.003	0	3.912	0.012	2%	__init__.py:45(backward)
314	3.892	0.012	3.892	0.012	2%	{method 'run_backward' of 'torch._C._EngineBase' objects}
2	0.001	0.001	3.561	1.78	2%	mmdtraintest.py:172(test_batch)
1288	0.041	0	2.347	0.002	1%	mmdmodels.py:63(forward)
1610	0.005	0	2.334	0.001	1%	mmdmodels.py:44(forward)
7728	0.025	0	2.087	0	1%	conv.py:348(forward)
7728	0.039	0	2.048	0	1%	conv.py:340(_conv_forward)
7406	0.642	0	2.011	0	1%	batchnorm.py:83(forward)
7084	0.014	0	0.433	0	0%	activation.py:93(forward)
322	0.003	0	0.353	0.001	0%	mmdlosses.py:35(LocalFocalLoss)
1610	0.007	0	0.342	0	0%	upsampling.py:130(forward)
322	0.191	0.001	0.328	0.001	0%	mmdlosses.py:80(forward)
1932	0.008	0	0.278	0	0%	pooling.py:138(forward)
322	0.101	0	0.218	0.001	0%	mmdlosses.py:63(dice_loss)
322	0.022	0	0.109	0	0%	mmdlosses.py:38(LocalPixelLoss)
322	0.003	0	0.104	0	0%	mmdlosses.py:50(LocalRMSELoss)
322	0.003	0	0.094	0	0%	mmdlosses.py:47(LocalL1Loss)
644	0.002	0	0.091	0	0%	loss.py:431(forward)
322	0.001	0	0.087	0	0%	mmdmodels.py:83(forward)
322	0.001	0	0.064	0	0%	loss.py:87(forward)
322	0.001	0	0.017	0	0%	mmdlosses.py:76(__init__)
16	0	0	0.001	0	0%	mmddataloader.py:272(__len__)

FINAL RESULTS
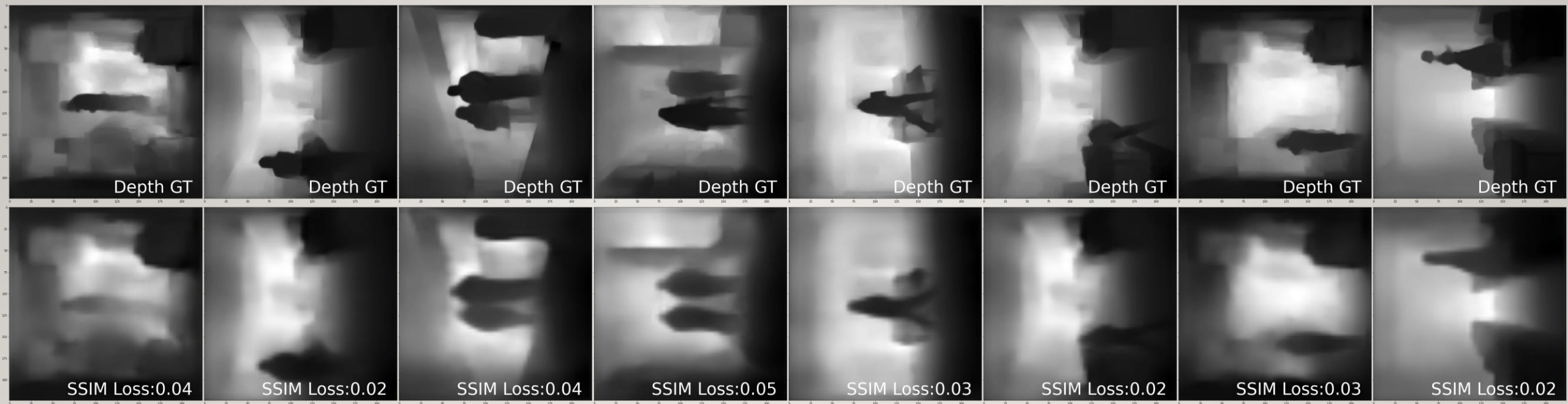
Losses	Values
Overall Loss	0.94152086
Overall Mask	0.87551310
Overall Depth Loss	0.06600758

FINAL RESULTS
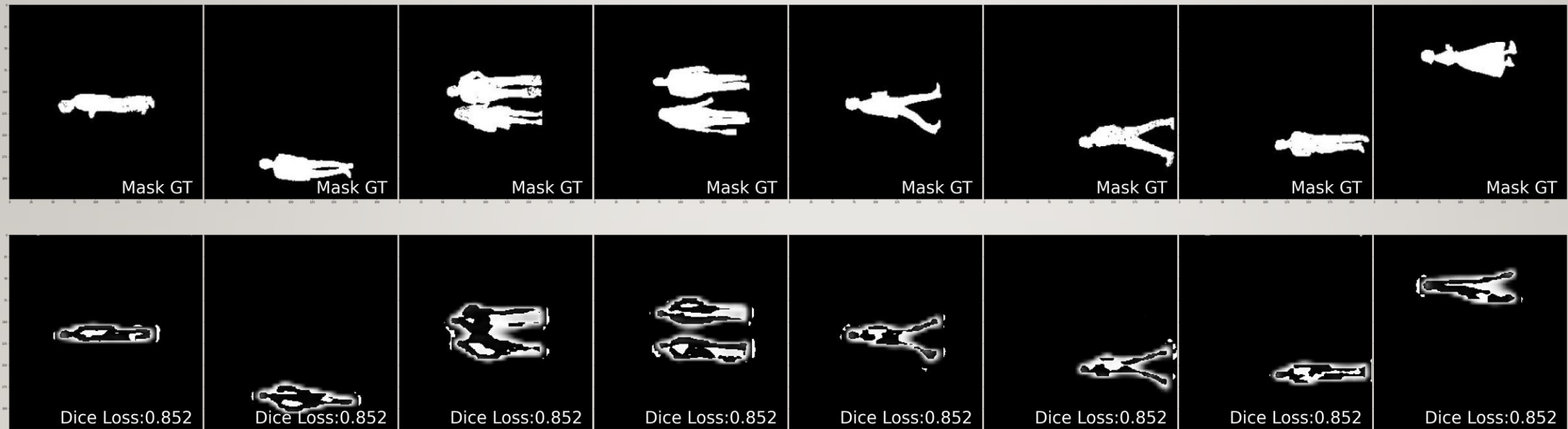
Losses	Values
Overall Loss	0.94152086
Overall Mask	0.87551310
Overall Depth Loss	0.06600758

HANDLING COLAB

- One central Google Drive was created which contains:

  - Base Modules

  - Raw Image Zip files

  - Model Checkpoint directories

  - Logs and other useful files

- Multiple Google accounts were linked back to the shared Google Drive.

- All notebooks, upon Colab expiry were pushed into private GitHub repository and reloaded directly from GitHub to allow for easy resuming of training

REFERENCE CITATIONS CREDITS