# Image Background Removal

## Requirements-

1. Linux 20.0.4 LTS (tested)
2. GPU >= 2.0 GB
3. Ram >= 4.0 GB
4. CPU - At least 4 core processor
5. Python v3.7
6. Pytorch GPU v1.7.1

Note - Read the "requirements.txt" for all python packages with their significant versions and installations.

Folder Structure -

*Final_Bundle-/*

*|_model*
*|_alphas*
*|_tests*
*|_inputs*
*|_results*
*|_utils*

## *Instructions without Conda*

1. Create all the folders structured above.
2. You can use ‘pip install -r requirements.txt’ to install all necessary packages.
3. In the models folder put pertained weights of the trained model and "modnet.py" in it and the backbone network inside of it.
4. In utils folder put "image_manipulation.py"
5. To run the model use the "image_matting.py" and put all images in "inputs" folder
6. Generated alphas and results will be stored in their respective folders.
7. You also need to install a supported CUDA version for PyTorch-GPUv 1.7.1 in order to run the inference.

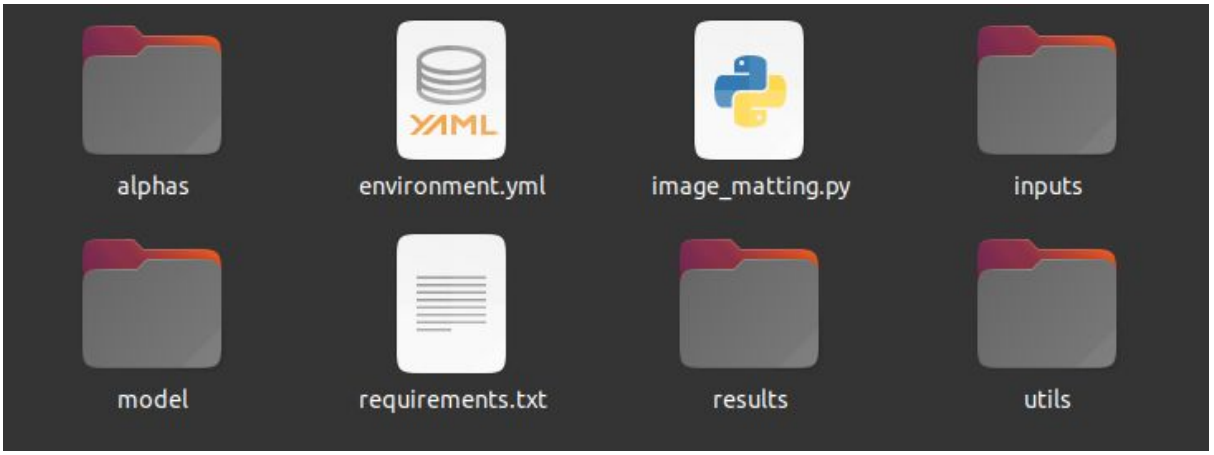
Noted - Tested on Core i5 10th gen, 16 GB system memory, Nvidia 1650ti 4GB memory GPU, Linux 20.04 LTS os. ( It takes an average of 3sec to perform background removal in my system ).

## *Instruction with Conda*

1. *If you have installed Conda already then create a new environment using ‘conda env create --file environment.yml’ assuming you also have a file named ‘environment.yml’.*
2. *Activate the env using ‘conda activate Pytorch-gpu’.*
3. *Now you can easily run the inference in the terminal using command "/PATH_TO_DIRECTORY/python3 image_matting.py" in the terminal to run the test.*
4. *See the results in the results folder.*

**CODE description:**

1. This is how the directory should look like.




2. Explaining Image_matting.py

```
import os
import re
import sys
import cv2
import random
import warnings
import numpy as np
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as transformsge
from utils.image_manipulations import stack_images, save_image
from model.modnet import MODNet
```

These are all necessary package imports.

```
path = f'{os.getcwd()}'.replace('\\', '/')
model_dir = path + '/model/modnet_image_matting.ckpt'
input_dir = path + '/input/'
alpha_dir = path + '/alphas/'
output_dir = path + '/results/'
```

Defining the working directory.

```
# create MODNet and load the pre-trained ckpt
modnet = MODNet(backbone_pretrained=False)
modnet = nn.DataParallel(modnet).cuda()
modnet.load_state_dict(torch.load(model_dir))
modnet.eval()
print('[INFO]...model loaded successfully.')
```

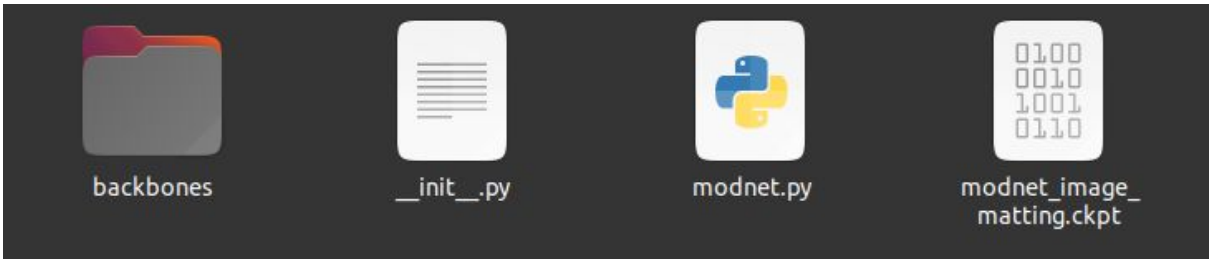
Loading the model

```python
# read image
im = Image.open(os.path.join(input_dir, im_name))

# unify image channels to 3
im = np.asarray(im)
if len(im.shape) == 2:
    im = im[:, :, None]
if im.shape[2] == 1:
    im = np.repeat(im, 3, axis=2)
elif im.shape[2] == 4:
    im = im[:, :, 0:3]

# convert image to PyTorch tensor
im = Image.fromarray(im)
im = im_transform(im)

# add mini-batch dim
im = im[None, :, :, :]

# resize image for input
im_b, im_c, im_h, im_w = im.shape
if max(im_h, im_w) < ref_size or min(im_h, im_w) > ref_size:
    if im_w >= im_h:
        im_rh = ref_size
        im_rw = int(im_w / im_h * ref_size)
    elif im_w < im_h:
        im_rw = ref_size
        im_rh = int(im_h / im_w * ref_size)
else:
    im_rh = im_h
    im_rw = im_w

im_rw = im_rw - im_rw % 32
im_rh = im_rh - im_rh % 32
im = F.interpolate(im, size=(im_rh, im_rw), mode='area')

# inference
_, _, matte = modnet(im.cuda(), True)

# resize and save matte
matte = F.interpolate(matte, size=(im_h, im_w), mode='area')
matte = matte[0][0].data.cpu().numpy()
# code for stacking images
matte_name = im_name.split('.')[0] + '_alpha.png'
Image.fromarray(((matte * 255).astype('uint8')), mode='L').save(os.path.join(alpha_dir, matte_name))
```

These functions are used to run the inference and resizing the input and output of the files according to the actual size.

```python
for name in names:

    print('Process image: {0}'.format(name[0]))

    alpha = cv2.imread(os.path.join(alpha_dir, name[1]))
    og = cv2.imread(os.path.join(input_dir, name[0]))

    og = cv2.cvtColor(og, cv2.COLOR_BGR2RGB)
    alpha = cv2.cvtColor(alpha, cv2.COLOR_BGR2GRAY)

    final = stack_images(og, alpha)
    image_name = output_dir + name[0].split('.')[0] + '_bg_removed.png'
    save_image(image_name, final)
```

This is used to load the input images and save the corresponding output images.

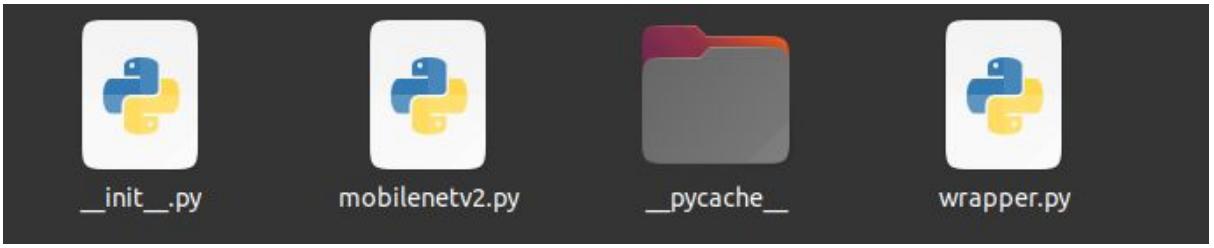



This is how the model folder looks like.

3. These functions are used to Stack Images and Save Images. src = image_manipulation.py

```python
from PIL import Image
import numpy as np

def save_image(path, image):

    assert image.dtype in [np.uint8, np.float32, np.float64]

    if image.dtype in [np.float32, np.float64]:
        image = np.clip(image * 255, 0, 255).astype(np.uint8)


    Image.fromarray(image).save(path)

def stack_images(*images):

    images = [
        (image if len(image.shape) == 3 else image[:, :, np.newaxis])
        for image in images
    ]
    return np.concatenate(images, axis=2)
```

4. This is how the backbone folder looks like




Note - Its '__init__.py' contains wrapper methods.