



Vinayak Nayak

Dec 12, 2020 · 6 min read · Listen

[Twitter](#) [Facebook](#) [LinkedIn](#) [GitHub](#) [Medium](#) [...](#)

Semantic Image Segmentation with DeepLabv3-pytorch

Build your own blurring/background modification functionality in videos

Photo by [Rodion Kutsaev](#) on [Unsplash](#)

Introduction

With the surge in use of video calling services during the COVID lockdown, many players are offering a service where the user of the service could blur the background or add a custom background etc. (Zoom, MS-Teams etc.) This is a classic use case of image segmentation where the object of interest is located and the pixels barring this region are modified/substituted.

There are many deep learning architectures which could be used to solve the instance segmentation problem and today we're going to use DeepLab-v3 which is a State of the Art semantic image segmentation model which comes in many flavors. Its goal is to assign semantic labels (e.g., person, sheep, airplane and so on) to every pixel in the input image. We are going to particularly be focusing on using the DeepLabv3 model with a ResNet-101 backbone that is offered out of the box with the torch library.

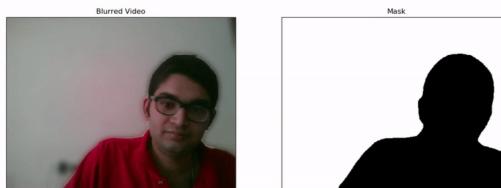


Image by Vinayak

At the end of this post, you'll be able to build something as shown above. We'll go about building this in a step by step fashion as follows:

- [Dependencies needed](#)
- [Using cv2 to set up a video session and prepare output screens](#)
- [Using deeplab for predicting labels](#)
- [Applying segmentation \(labels\) map and blurring](#)
- [Bonus: Using custom background](#)

So without any further ado, let's get started!

Dependencies

For reproducing the code below, we will need to have Python installed on our system and in addition to that we will also need the following:

- torch
- Pillow
- numpy

Search



Vinayak Nayak

68 Followers

AI Developer, Data Science Enthusiast

[Follow](#)

You can now subscribe to
get stories delivered directly
to your inbox.
[Got it](#)

[Data Science tutorial with K-means and Python](#)

Ronny Polle in Level Up Coding
[Blood Spectroscopy to Image Classification](#)



Bibhabasu Mohapatra
[Start with Transformers for Beginners](#)



Adrienne Kline in CodeX
[Image Processing: Color Conversions](#)



[Help](#) [Status](#) [Writers](#) [Blog](#) [Careers](#) [Privacy](#) [Terms](#) [About](#)
Knowable

- matplotlib
- cv2

These could be simply installed by using the command

```
pip install packagename
```

If you're working with conda, I would recommend you create a separate environment for this project using the following command. You can choose a higher version for python as well by changing the last portion to python=3.7 or python=3.8 etc.

```
conda create -n deeplabenv python=3.6
```

Once you install the above packages, we're ready to go to the next step.

...

Getting Input & Setting the stage for Output

We will be using opencv to interface a webcam for reading in input from our screens and we'll use matplotlib's pyplot module to render the processed video feed to output.

```
1 import cv2
2
3 # Start a video cam session
4 video_session = cv2.VideoCapture(0)
5
6 # Given a video capture object, read frames from the same and convert it to RGB
7 def grab_frame(cap):
8     _, frame = cap.read()
9     return cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

[input_prep.py hosted with ❤ by GitHub](#)

[view raw](#)

opencv's VideoCapture object is used to get the image input for video. If you have multiple webcams you could create multiple such objects by passing the appropriate index; by default nowadays, most monitors have one inbuilt camera which could be indexed at 0th position.

Subsequently, opencv reads images in a BGR format but while rendering we need to show it in RGB format; so we've written a tiny function that captures a frame in realtime and converts it from BGR format to RGB format above. With this, we're set with the input preprocessing steps. Let's look at how we'll set the stage for output now.

```
1 import matplotlib.pyplot as plt
2 from utils import grab_frame
3
4 # Define two axes for showing the mask and the true video in realtime
5 # And set the ticks to none for both the axes
6 fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (15, 8))
7 ax2.set_title("Mask")
8
9 ax1.set_xticks([])
10 ax1.set_yticks([])
11 ax2.set_xticks([])
12 ax2.set_yticks([])
13
14 # Create two image objects to picture on top of the axes defined above
15 im1 = ax1.imshow(grab_frame(video_session))
16 im2 = ax2.imshow(grab_frame(video_session))
17
18 # Switch on the interactive mode in matplotlib
19 plt.ion()
20 plt.show()
```

[output_prep.py hosted with ❤ by GitHub](#)

[view raw](#)

Why use matplotlib.pyplot and not cv2.imshow? There are some inconsistencies with cv2.imshow when it comes to ubuntu distributions; I had [one such issue](#) which caused me to look at alternate methods and since pyplot is easy to implement and mostly included with all major python distributions like anaconda by default, I thought of using pyplot for rendering the output.

So basically we set up two subplots as seen in the gif on the top; one to see the blurred version and another one to actually look at the labels mask which the deeplab model has predicted. We switch on the interactive mode for pyplot and display the image captured directly at the very beginning of the stream. Now, that we have the stage set, let's discuss the part to obtain predictions from the deeplab-v3 model.

Deeplab-v3 Segmentation

The model offered at torch-hub for segmentation is trained on [PASCAL VOC](#)

[dataset](#) which contains 20 different classes of which the most important one for us is the person class with label 15.

```
1 import torch
2 import torchvision
3
4 def load_model():
5     # Load the DeepLab v3 model to system
6     device = "cuda" if torch.cuda.is_available() else "cpu"
7     model = torch.hub.load('pytorch/vision:v0.6.0', 'deeplabv3_resnet101', pretrained=True)
8     model.to(device).eval()
9
10    return model
```

deeplab_v3_model_load.py hosted with ❤ by GitHub [view raw](#)

Using the above code we can download the model from torch-hub and use it for our segmentation task. Note that since this is based on top of a Resnet-101 model which is quite heavy, the inference will take time and rendering will lag heavily if you do not have a medium sized GPU if not a high end one. Now that we have the model loaded, let's discuss how we could get predictions from it.

```
1 import torch
2 import torchvision
3
4 def get_pred(img, model):
5     # See if GPU is available and if yes, use it
6     device = "cuda" if torch.cuda.is_available() else "cpu"
7
8     # Define the standard transforms that need to be done at inference time
9     imagenet_stats = [[0.485, 0.456, 0.406], [0.485, 0.456, 0.406]]
10    preprocess = torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
11                                                 torchvision.transforms.Normalize(mean = imagenet_stats[0],
12                                                 std = imagenet_stats[1])])
13
14    input_tensor = preprocess(img).unsqueeze(0)
15    input_tensor = input_tensor.to(device)
16
17    # Make the predictions for labels across the image
18    with torch.no_grad():
19        output = model(input_tensor)[“out”][0]
20        output = output.argmax(0)
21
22    # Return the predictions
23    return output.cpu().numpy()
```

deeplab_v3_model_predict.py hosted with ❤ by GitHub [view raw](#)

We will first load the image using Pillow or directly get it from the VideoCapture object of cv2 defined above. Once we have this, we will normalize it using imagenet stats and convert it to a tensor. Then if we have a GPU available at our disposal, we can move the tensor to GPU. All pre-trained models expect input images in mini-batches of 3-channel RGB images of shape $(N, 3, H, W)$, where N is the number of images, H and W are height and width of the two images respectively. Since our video capture object captures single frames, it's output is $(3, H, W)$. We therefore unsqueeze the tensor along the first dimension to make it $(1, 3, H, W)$.

Once we do that, we then put the model in eval mode and without autograd perform a forward pass through the same. The forward pass gives `aux` and `out` objects from which, the `out` object is of interest to us during inference. The `aux` values have loss value per pixel and are useful during train time. So, we get the `out` object and do a argmax along the 1st dimension to obtain the labels map which is of the same height and width as the original image with a single channel. This mask could be used for segmentation which we'll look at in the next section.

• • •

Human Segmentation

Now that we have the predictions, we can use them for segmenting the human and blurring the background.

```
1 from utils import *
2 import numpy as np
3 import cv2
4
5 # Define the kernel size for applying Gaussian Blur
6 blur_value = (51, 51)
7
8 frame = utils.grab_frame(video_session)
9 # Read the frame's width, height, channels and get the labels' predictions from utilities
10 width, height, channels = frame.shape
11 labels = utils.get_pred(frame, model)
12
13 # Wherever there's empty space/no person, the label is zero
14 # Hence identify such areas and create a mask (replicate it across RGB channels)
15 mask = labels == 0
16 mask = np.repeat(mask[:, :, np.newaxis], channels, axis = 2)
17
18 # Apply the Gaussian blur for background with the kernel size specified in constants above
19 blur = cv2.GaussianBlur(frame, blur_value, 0)
20 frame[mask] = blur[mask]
21 ax1.set_title("Blurred Video")
```

```

23 # Set the data of the two images to frame and mask values respectively
24 im1.set_data(frame)
25 im2.set_data(mask * 255)
26 plt.pause(0.01)

```

segment.py hosted with ❤ by GitHub [view raw](#)

The labels extracted out of the image are two dimensional. We need to replicate this mask across all three channels i.e. RGB. So, we use numpy to repeat this segmentation mask across all three channels.

Next, we use opencv's Gaussian Blur to blur the entire frame output and get a new image called blur. The extent of blur is determined by the `blur_value` variable which is a tuple representing the kernel size for blurring. Higher the value, more the blur and vice versa.

Next, with the help of mask, we replace the background pixels of the frame with the blurred frame's pixels. Now that we have both the mask and the resulting manipulated frame, we set the data of the two image variables defined in the *Setting the stage for output* section to these two frames respectively and define an interval for interactive pyplot object for refreshing the frame (think of it as an equivalent of frame rate).

This is how you could use DeepLabv3 for making your very own background blurring feature on custom videos or live vidcams with Image Segmentation.

Bonus: Background Substitution with custom image

Just like we blurred the background above, we could also substitute the background with a custom image and it only requires some minor modifications to the code.

```

1 from utils import *
2 import cv2
3 import numpy as np
4
5 # Read the background image to memory
6 bg_image = cv2.imread(BG_PTH)
7 bg_image = cv2.cvtColor(bg_image, cv2.COLOR_BGR2RGB)
8
9 # Start a video cam session
10 video_session = cv2.VideoCapture(0)
11
12 # Read frame from the video capture object and get the labels from get_pred function
13 frame = utils.grab_frame(video_session)
14 width, height, channels = frame.shape
15 labels = utils.get_pred(frame, model)
16
17 # Define a blurring value kernel size for cv2's Gaussian Blur
18 blur_value = (51, 51)
19
20 # The PASCAL VOC dataset has 20 categories of which Person is the 16th category
21 # Hence wherever person is predicted, the label returned will be 15
22 # Subsequently repeat the mask across RGB channels
23 mask = labels == 15
24 mask = np.repeat(mask[:, :, np.newaxis], 3, axis=2)
25
26 # Resize the image as per the frame capture size
27 bg = cv2.resize(bg_image, (height, width))
28 bg[mask] = frame[mask]
29 frame = bg
30 ax1.set_title("Background Changed Video")

```

custom_background.py hosted with ❤ by GitHub [view raw](#)

We read a background image of our choice for substitution using cv2 and bring it to RGB format. Next, in the part where we replace the frame's non-human pixels with blurred pixels, we now use this background image pixels instead.

Basically, we resize the background image to fit the frame capture source or the video source, then mask all the human pixels in this image with the frame's human pixels. This gives the source human overlaid on top of the background image like shown in the gif below.

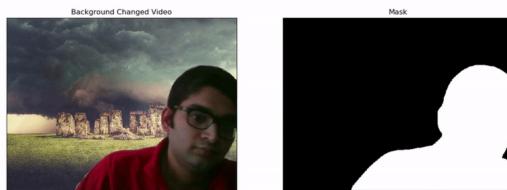


Image by Vinayak

Hope you enjoyed reading this post as much as I did while writing it. If you want to read further, I've mentioned some links in the resource section which may prove helpful. Cheers!

References

1. [Deeplabv3-Resnet101 Pytorch](#)
2. [All the code for the above example](#)
3. [More on Image Segmentation](#)
4. [PyImageSearch Article on Instance Segmentation with opencv](#)

92 2 ⌂ ⌂ ...

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

Get this newsletter

Emails will be sent to shubham.iateetud@gmail.com.
[Not you?](#)

Follow

More from Towards Data Science

Your home for data science. A Medium publication sharing concepts, ideas and codes.

Diane Tunnicliffe · Dec 12, 2020

Schools, Starbucks, and... Scientology?

Unexpected findings using linear regression models, and my personal musings on group work. — It was nearing the end of November. Turkeys, politics, and Covid-19 were the only prevalent topics of conversation. I...



Linear Regression 8 min read

⌂ ...

Share your ideas with millions of readers. [Write on Medium](#)

Manpreet Singh Minhas · Dec 12, 2020 ★

A Compact CNN for Weakly Supervised Textured Surface Anomaly Detection

In this article, I'll be discussing a paper [1] that proposes a compact convolutional neural network (CNN) for detecting anomalies/defects fro...



Data Science 10 min read

⌂ ...

Monica P. · Dec 12, 2020 ★

David and Goliath — Markov Model vs GPT-2 Model

For my final project in my Artificial Intelligence class for my Data Science Masters, I chose to compare two models; one using Markov principles and the other a Deep learning model created by OpenAI for Natural Language...



Deep Learning 8 min read

⌂ ...

Terence Shin · Dec 12, 2020 ★

How to Simulate a Pandemic in Python

A Step-by-Step Data Science Project — Introduction What's a better time to simulate the spread of a disease than during a global pandemic? I don't have much more to say — let's jump right into programming a simple...



Data Science 13 min read

⌂ ...

Okoh Anita · Dec 12, 2020 ★

From Medium Post to Audio

How I converted my medium post to realistic Audio using AWS Polly — Introduction I recently discovered that I retain more information when I read a book in audio format than in written format. This discovery...



AWS Lambda 6 min read

⌂ ...

[Read more from Towards Data Science](#)