


UNet

Introducing Symmetry in Segmentation



Heet Sankesara

Follow

Jan 23, 2019 · 6 min read

Introduction

Vision is one of the most important senses humans possess. But have you ever wondered about the complexity of the task? The ability to capture the reflected light rays and get meaning out of it is a very convoluted task and yet we do it so easily. We developed it due to millions of years of evolution. So how can we give machines the same ability in a very small period of time? For computers, these images are nothing but matrices and understanding the nuances behind these matrices has been an obsession for many mathematicians for years. But after the emergence of artificial intelligence and particularly CNN architectures, the research has made progress like never before. Many problems which are previously considered untouchable are now showing astounding results.

One such problem is the image segmentation. In Image Segmentation, the machine has to partition the image into different segments, each of them representing a different entity.



Image Segmentation Example

As you can see above, how the image turned into two segments, one represents the cat and the other background. Image segmentation is useful in many fields from self-driving cars to satellites. Perhaps the most important of them all is medical imaging. The subtleties in medical images are quite complex and sometimes even challenging for trained physicians. A machine that can understand these nuances and can identify necessary areas can make a profound impact in medical care.

Convolutional Neural Networks gave decent results in easier image segmentation problems but it hasn't made any good progress on complex ones. That's where UNet comes in the picture. UNet was first designed especially for medical image segmentation. It showed such good results that it used in many other fields after. In this article, we'll talk about why and how UNet works. If you don't know intuition behind CNN, please read this first. You can check out UNet in action here.

The Intuition Behind UNet

The main idea behind CNN is to learn the feature mapping of an image and exploit it to make more nuanced feature mapping. This works well in classification problems as the image is converted into a vector which used further for classification. But in image segmentation, we not only need to convert feature map into a vector but also reconstruct an image from this vector. This is a mammoth task because it's a lot tougher to convert a vector into an image than vice versa. The whole idea of UNet is revolved around this problem.

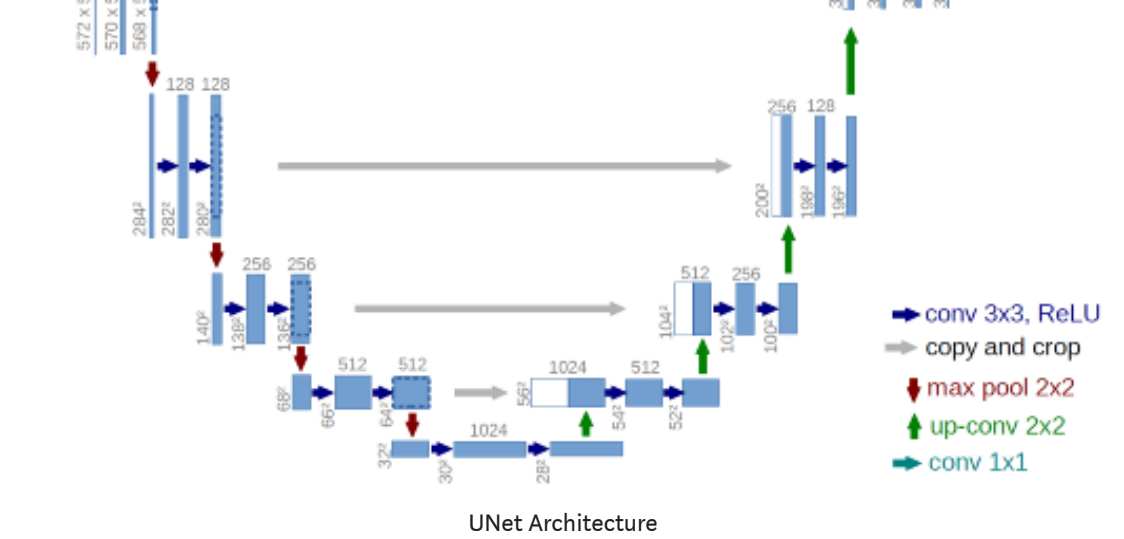
While converting an image into a vector, we already learned the feature mapping of the image so why not use the same mapping to convert it again to image. This is the recipe behind UNet. Use the same feature maps that are used for contraction to expand a vector to a segmented image. This would preserve the structural integrity of the image which would reduce distortion enormously. Let's understand the architecture more briefly.

Top highlight

UNet Architecture

How UNet Works





The architecture looks like a ‘U’ which justifies its name. This architecture consists of three sections: The contraction, The bottleneck, and the expansion section. The contraction section is made of many contraction blocks. Each block takes an input applies two 3X3 convolution layers followed by a 2X2 max pooling. The number of kernels or feature maps after each block doubles so that architecture can learn the complex structures effectively. The bottommost layer mediates between the contraction layer and the expansion layer. It uses two 3X3 CNN layers followed by 2X2 up convolution layer.

But the heart of this architecture lies in the expansion section. Similar to contraction layer, it also consists of several expansion blocks. Each block passes the input to two 3X3 CNN layers followed by a 2X2 upsampling layer. Also after each block number of feature maps used by convolutional layer get half to maintain symmetry. However, every time the input is also get appended by feature maps of the corresponding contraction layer. This action would ensure that the features that are learned while contracting the image will be used to reconstruct it. The number of expansion blocks is as same as the number of contraction block. After that, the resultant mapping passes through another 3X3 CNN layer with the number of feature maps equal to the number of segments desired.

Loss calculation in UNet

What kind of loss one would use in such an intrinsic image segmentation? Well, it is defined simply in the paper itself.

The energy function is computed by a pixel-wise soft-max over the final feature map combined with the cross-entropy loss function

UNet uses a rather novel loss weighting scheme for each **pixel** such that there is a higher weight at the border of segmented objects. This loss weighting scheme helped the U-Net model segment cells in biomedical images in a *discontinuous fashion* such that individual cells may be easily identified within the binary segmentation map.

First of all pixel-wise softmax applied on the resultant image which is followed by cross-entropy loss function. So we are classifying each pixel into one of the classes. The idea is that even in segmentation every pixel have to lie in some category and we just need to make sure that they do. So we just converted a segmentation problem into a multiclass classification one and it performed very well as compared to the traditional loss functions.

UNet Implementation

I implemented the UNet model using Pytorch framework. You can check out the UNet module [here](#). Images for segmentation of optical coherence tomography images with diabetic macular edema are used. You can checkout UNet in action [here](#).

```
1 import torch
2 from torch import nn
3 import torch.nn.functional as F
4 import torch.optim as optim
5
6 class UNet(nn.Module):
7     def contracting_block(self, in_channels, out_channels, kernel_size=3):
8         block = torch.nn.Sequential(
9             torch.nn.Conv2d(kernel_size=kernel_size, in_channels=in_channels, out_channels=out_channels),
10            torch.nn.ReLU(),
11            torch.nn.BatchNorm2d(out_channels),
12            torch.nn.Conv2d(kernel_size=kernel_size, in_channels=out_channels, out_channels=out_channels),
13            torch.nn.ReLU(),
14            torch.nn.BatchNorm2d(out_channels),
15        )
16        return block
17
18     def expansive_block(self, in_channels, mid_channel, out_channels, kernel_size=3):
19         block = torch.nn.Sequential(
20            torch.nn.Conv2d(kernel_size=kernel_size, in_channels=in_channels, out_channels=mid_channel),
21            torch.nn.ReLU(),
22            torch.nn.BatchNorm2d(mid_channel),
23            torch.nn.Conv2d(kernel_size=kernel_size, in_channels=mid_channel, out_channels=mid_channel),
24            torch.nn.ReLU(),
25            torch.nn.BatchNorm2d(mid_channel),
26            torch.nn.ConvTranspose2d(in_channels=mid_channel, out_channels=out_channels, kernel_size=kernel_size),
27        )
28        return block
29
30     def final_block(self, in_channels, mid_channel, out_channels, kernel_size=3):
31         block = torch.nn.Sequential(
32            torch.nn.Conv2d(kernel_size=kernel_size, in_channels=in_channels, out_channels=mid_channel),
33            torch.nn.ReLU(),
34            torch.nn.BatchNorm2d(mid_channel),
```

```
35         torch.nn.Conv2d(kernel_size=kernel_size, in_channels=mid_channel, out_channe
36         torch.nn.ReLU()),
37         torch.nn.BatchNorm2d(mid_channel),
38         torch.nn.Conv2d(kernel_size=kernel_size, in_channels=mid_channel, out_channe
39         torch.nn.ReLU()),
40         torch.nn.BatchNorm2d(out_channels),
41     )
42     return block
43
44     def __init__(self, in_channel, out_channel):
45         super(UNet, self).__init__()
46         #Encode
47         self.conv_encode1 = self.contracting_block(in_channels=in_channel, out_channels=64)
48         self.conv_maxpool1 = torch.nn.MaxPool2d(kernel_size=2)
49         self.conv_encode2 = self.contracting_block(64, 128)
50         self.conv_maxpool2 = torch.nn.MaxPool2d(kernel_size=2)
51         self.conv_encode3 = self.contracting_block(128, 256)
52         self.conv_maxpool3 = torch.nn.MaxPool2d(kernel_size=2)
53         # Bottleneck
54         self.bottleneck = torch.nn.Sequential(
55             torch.nn.Conv2d(kernel_size=3, in_channels=256, out_channels=512),
56             torch.nn.ReLU(),
57             torch.nn.BatchNorm2d(512),
58             torch.nn.Conv2d(kernel_size=3, in_channels=512, out_channels=512),
59             torch.nn.ReLU(),
60             torch.nn.BatchNorm2d(512),
61             torch.nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_s
62         )
63         # Decode
64         self.conv_decode3 = self.expansive_block(512, 256, 128)
65         self.conv_decode2 = self.expansive_block(256, 128, 64)
66         self.final_layer = self.final_block(128, 64, out_channel)
67
68     def crop_and_concat(self, upsampled, bypass, crop=False):
69         if crop:
70             c = (bypass.size()[2] - upsampled.size()[2]) // 2
71             bypass = F.pad(bypass, (-c, -c, -c, -c))
72         return torch.cat((upsampled, bypass), 1)
73
74     def forward(self, x):
75         # Encode
76         encode_block1 = self.conv_encode1(x)
77         encode_pool1 = self.conv_maxpool1(encode_block1)
78         encode_block2 = self.conv_encode2(encode_pool1)
79         encode_pool2 = self.conv_maxpool2(encode_block2)
80         encode_block3 = self.conv_encode3(encode_pool2)
81         encode_pool3 = self.conv_maxpool3(encode_block3)
82         # Bottleneck
83         # ...
84         # ...
85         # ...
86         # ...
87         # ...
88         # ...
89         # ...
90         # ...
91         # ...
92         # ...
93         # ...
94         # ...
95         # ...
96         # ...
97         # ...
98         # ...
99         # ...
100        # ...
101        # ...
102        # ...
103        # ...
104        # ...
105        # ...
106        # ...
107        # ...
108        # ...
109        # ...
110        # ...
111        # ...
112        # ...
113        # ...
114        # ...
115        # ...
116        # ...
117        # ...
118        # ...
119        # ...
120        # ...
121        # ...
122        # ...
123        # ...
124        # ...
125        # ...
126        # ...
127        # ...
128        # ...
129        # ...
130        # ...
131        # ...
132        # ...
133        # ...
134        # ...
135        # ...
136        # ...
137        # ...
138        # ...
139        # ...
140        # ...
141        # ...
142        # ...
143        # ...
144        # ...
145        # ...
146        # ...
147        # ...
148        # ...
149        # ...
150        # ...
151        # ...
152        # ...
153        # ...
154        # ...
155        # ...
156        # ...
157        # ...
158        # ...
159        # ...
160        # ...
161        # ...
162        # ...
163        # ...
164        # ...
165        # ...
166        # ...
167        # ...
168        # ...
169        # ...
170        # ...
171        # ...
172        # ...
173        # ...
174        # ...
175        # ...
176        # ...
177        # ...
178        # ...
179        # ...
180        # ...
181        # ...
182        # ...
183        # ...
184        # ...
185        # ...
186        # ...
187        # ...
188        # ...
189        # ...
190        # ...
191        # ...
192        # ...
193        # ...
194        # ...
195        # ...
196        # ...
197        # ...
198        # ...
199        # ...
200        # ...
201        # ...
202        # ...
203        # ...
204        # ...
205        # ...
206        # ...
207        # ...
208        # ...
209        # ...
210        # ...
211        # ...
212        # ...
213        # ...
214        # ...
215        # ...
216        # ...
217        # ...
218        # ...
219        # ...
220        # ...
221        # ...
222        # ...
223        # ...
224        # ...
225        # ...
226        # ...
227        # ...
228        # ...
229        # ...
230        # ...
231        # ...
232        # ...
233        # ...
234        # ...
235        # ...
236        # ...
237        # ...
238        # ...
239        # ...
240        # ...
241        # ...
242        # ...
243        # ...
244        # ...
245        # ...
246        # ...
247        # ...
248        # ...
249        # ...
250        # ...
251        # ...
252        # ...
253        # ...
254        # ...
255        # ...
256        # ...
257        # ...
258        # ...
259        # ...
260        # ...
261        # ...
262        # ...
263        # ...
264        # ...
265        # ...
266        # ...
267        # ...
268        # ...
269        # ...
270        # ...
271        # ...
272        # ...
273        # ...
274        # ...
275        # ...
276        # ...
277        # ...
278        # ...
279        # ...
280        # ...
281        # ...
282        # ...
283        # ...
284        # ...
285        # ...
286        # ...
287        # ...
288        # ...
289        # ...
290        # ...
291        # ...
292        # ...
293        # ...
294        # ...
295        # ...
296        # ...
297        # ...
298        # ...
299        # ...
300        # ...
301        # ...
302        # ...
303        # ...
304        # ...
305        # ...
306        # ...
307        # ...
308        # ...
309        # ...
310        # ...
311        # ...
312        # ...
313        # ...
314        # ...
315        # ...
316        # ...
317        # ...
318        # ...
319        # ...
320        # ...
321        # ...
322        # ...
323        # ...
324        # ...
325        # ...
326        # ...
327        # ...
328        # ...
329        # ...
330        # ...
331        # ...
332        # ...
333        # ...
334        # ...
335        # ...
336        # ...
337        # ...
338        # ...
339        # ...
340        # ...
341        # ...
342        # ...
343        # ...
344        # ...
345        # ...
346        # ...
347        # ...
348        # ...
349        # ...
350        # ...
351        # ...
352        # ...
353        # ...
354        # ...
355        # ...
356        # ...
357        # ...
358        # ...
359        # ...
360        # ...
361        # ...
362        # ...
363        # ...
364        # ...
365        # ...
366        # ...
367        # ...
368        # ...
369        # ...
370        # ...
371        # ...
372        # ...
373        # ...
374        # ...
375        # ...
376        # ...
377        # ...
378        # ...
379        # ...
380        # ...
381        # ...
382        # ...
383        # ...
384        # ...
385        # ...
386        # ...
387        # ...
388        # ...
389        # ...
390        # ...
391        # ...
392        # ...
393        # ...
394        # ...
395        # ...
396        # ...
397        # ...
398        # ...
399        # ...
400        # ...
401        # ...
402        # ...
403        # ...
404        # ...
405        # ...
406        # ...
407        # ...
408        # ...
409        # ...
410        # ...
411        # ...
412        # ...
413        # ...
414        # ...
415        # ...
416        # ...
417        # ...
418        # ...
419        # ...
420        # ...
421        # ...
422        # ...
423        # ...
424        # ...
425        # ...
426        # ...
427        # ...
428        # ...
429        # ...
430        # ...
431        # ...
432        # ...
433        # ...
434        # ...
435        # ...
436        # ...
437        # ...
438        # ...
439        # ...
440        # ...
441        # ...
442        # ...
443        # ...
444        # ...
445        # ...
446        # ...
447        # ...
448        # ...
449        # ...
450        # ...
451        # ...
452        # ...
453        # ...
454        # ...
455        # ...
456        # ...
457        # ...
458        # ...
459        # ...
460        # ...
461        # ...
462        # ...
463        # ...
464        # ...
465        # ...
466        # ...
467        # ...
468        # ...
469        # ...
470        # ...
471        # ...
472        # ...
473        # ...
474        # ...
475        # ...
476        # ...
477        # ...
478        # ...
479        # ...
480        # ...
481        # ...
482        # ...
483        # ...
484        # ...
485        # ...
486        # ...
487        # ...
488        # ...
489        # ...
490        # ...
491        # ...
492        # ...
493        # ...
494        # ...
495        # ...
496        # ...
497        # ...
498        # ...
499        # ...
500        # ...
501        # ...
502        # ...
503        # ...
504        # ...
505        # ...
506        # ...
507        # ...
508        # ...
509        # ...
510        # ...
511        # ...
512        # ...
513        # ...
514        # ...
515        # ...
516        # ...
517        # ...
518        # ...
519        # ...
520        # ...
521        # ...
522        # ...
523        # ...
524        # ...
525        # ...
526        # ...
527        # ...
528        # ...
529        # ...
530        # ...
531        # ...
532        # ...
533        # ...
534        # ...
535        # ...
536        # ...
537        # ...
538        # ...
539        # ...
540        # ...
541        # ...
542        # ...
543        # ...
544        # ...
545        # ...
546        # ...
547        # ...
548        # ...
549        # ...
550        # ...
551        # ...
552        # ...
553        # ...
554        # ...
555        # ...
556        # ...
557        # ...
558        # ...
559        # ...
560        # ...
561        # ...
562        # ...
563        # ...
564        # ...
565        # ...
566        # ...
567        # ...
568        # ...
569        # ...
570        # ...
571        # ...
572        # ...
573        # ...
574        # ...
575        # ...
576        # ...
577        # ...
578        # ...
579        # ...
580        # ...
581        # ...
582        # ...
583        # ...
584        # ...
585        # ...
586        # ...
587        # ...
588        # ...
589        # ...
590        # ...
591        # ...
592        # ...
593        # ...
594        # ...
595        # ...
596        # ...
597        # ...
598        # ...
599        # ...
600        # ...
601        # ...
602        # ...
603        # ...
604        # ...
605        # ...
606        # ...
607        # ...
608        # ...
609        # ...
610        # ...
611        # ...
612        # ...
613        # ...
614        # ...
615        # ...
616        # ...
617        # ...
618        # ...
619        # ...
620        # ...
621        # ...
622        # ...
623        # ...
624        # ...
625        # ...
626        # ...
627        # ...
628        # ...
629        # ...
630        # ...
631        # ...
632        # ...
633        # ...
634        # ...
635        # ...
636        # ...
637        # ...
638        # ...
639        # ...
640        # ...
641        # ...
642        # ...
643        # ...
644        # ...
645        # ...
646        # ...
647        # ...
648        # ...
649        # ...
650        # ...
651        # ...
652        # ...
653        # ...
654        # ...
655        # ...
656        # ...
657        # ...
658        # ...
659        # ...
660        # ...
661        # ...
662        # ...
663        # ...
664        # ...
665        # ...
666        # ...
667        # ...
668        # ...
669        # ...
670        # ...
671        # ...
672        # ...
673        # ...
674        # ...
675        # ...
676        # ...
677        # ...
678        # ...
679        # ...
680        # ...
681        # ...
682        # ...
683        # ...
684        # ...
685        # ...
686        # ...
687        # ...
688        # ...
689        # ...
690        # ...
691        # ...
692        # ...
693        # ...
694        # ...
695        # ...
696        # ...
697        # ...
698        # ...
699        # ...
700        # ...
701        # ...
702        # ...
703        # ...
704        # ...
705        # ...
706        # ...
707        # ...
708        # ...
709        # ...
710        # ...
711        # ...
712        # ...
713        # ...
714        # ...
715        # ...
716        # ...
717        # ...
718        # ...
719        # ...
720        # ...
721        # ...
722        # ...
723        # ...
724        # ...
725        # ...
726        # ...
727        # ...
728        # ...
729        # ...
730        # ...
731        # ...
732        # ...
733        # ...
734        # ...
735        # ...
736        # ...
737        # ...
738        # ...
739        # ...
740        # ...
741        # ...
742        # ...
743        # ...
744        # ...
745        # ...
746        # ...
747        # ...
748        # ...
749        # ...
750        # ...
751        # ...
752        # ...
753        # ...
754        # ...
755        # ...
756        # ...
757        # ...
758        # ...
759        # ...
760        # ...
761        # ...
762        # ...
763        # ...
764        # ...
765        # ...
766        # ...
767        # ...
768        # ...
769        # ...
770        # ...
771        # ...
772        # ...
773        # ...
774        # ...
775        # ...
776        # ...
777        # ...
778        # ...
779        # ...
780        # ...
781        # ...
782        # ...
783        # ...
784        # ...
785        # ...
786        # ...
787        # ...
788        # ...
789        # ...
790        # ...
791        # ...
792        # ...
793        # ...
794        # ...
795        # ...
796        # ...
797        # ...
798        # ...
799        # ...
800        # ...
801        # ...
802        # ...
803        # ...
804        # ...
805        # ...
806        # ...
807        # ...
808        # ...
809        # ...
810        # ...
811        # ...
812        # ...
813        # ...
814        # ...
815        # ...
816        # ...
817        # ...
818        # ...
819        # ...
820        # ...
821        # ...
822        # ...
823        # ...
824        # ...
825        # ...
826        # ...
827        # ...
828        # ...
829        # ...
830        # ...
831        # ...
832        # ...
833        # ...
834        # ...
835        # ...
836        # ...
837        # ...
838        # ...
839        # ...
840        # ...
841        # ...
842        # ...
843        # ...
844        # ...
845        # ...
846        # ...
847        # ...
848        # ...
849        # ...
850        # ...
851        # ...
852        # ...
853        # ...
854        # ...
855        # ...
856        # ...
857        # ...
858        # ...
859        # ...
860        # ...
861        # ...
862        # ...
863        # ...
864        # ...
865        # ...
866        # ...
867        # ...
868        # ...
869        # ...
870        # ...
871        # ...
872        # ...
873        # ...
874        # ...
875        # ...
876        # ...
877        # ...
878        # ...
879        # ...
880        # ...
881        # ...
882        # ...
883        # ...
884        # ...
885        # ...
886        # ...
887        # ...
888        # ...
889        # ...
890        # ...
891        # ...
892        # ...
893        # ...
894        # ...
895        # ...
896        # ...
897        # ...
898        # ...
899        # ...
900        # ...
901        # ...
902        # ...
903        # ...
904        # ...
905        # ...
906        # ...
907        # ...
908        # ...
909        # ...
910        # ...
911        # ...
912        # ...
913        # ...
914        # ...
915        # ...
916        # ...
917        # ...
918        # ...
919        # ...
920        # ...
921        # ...
922        # ...
923        # ...
924        # ...
925        # ...
926        # ...
927        # ...
928        # ...
929        # ...
930        # ...
931        # ...
932        # ...
933        # ...
934        # ...
935        # ...
936        # ...
937        # ...
938        # ...
939        # ...
940        # ...
941        # ...
942        # ...
943        # ...
944        # ...
945        # ...
946        # ...
947        # ...
948        # ...
949        # ...
950        # ...
951        # ...
952        # ...
953        # ...
954        # ...
955        # ...
956        # ...
957        # ...
958        # ...
959        # ...
960        # ...
961        # ...
962        # ...
963        # ...
964        # ...
965        # ...
966        # ...
967        # ...
968        # ...
969        # ...
970        # ...
971        # ...
972        # ...
973        # ...
974        # ...
975        # ...
976        # ...
977        # ...
978        # ...
979        # ...
980        # ...
981        # ...
982        # ...
983        # ...
984        # ...
985        # ...
986        # ...
987        # ...
988        # ...
989        # ...
990        # ...
991        # ...
992        # ...
993        # ...
994        # ...
995        # ...
996        # ...
997        # ...
998        # ...
999        # ...
1000       # ...
```

```
unet = Unet(in_channel=1,out_channel=2)
#out_channel represents number of segments desired
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(unet.parameters(), lr = 0.01,
momentum=0.99)
optimizer.zero_grad()
outputs = unet(inputs)
# permute such that number of desired segments would be on 4th
dimension
outputs = outputs.permute(0, 2, 3, 1)
m = outputs.shape[0]
# Resizing the outputs and label to caculate pixel wise softmax loss
outputs = outputs.resize(m*width_out*height_out, 2)
labels = labels.resize(m*width_out*height_out)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
```

Conclusion

Image segmentation is an important problem and every day some new research papers are published. UNet contributed significantly in such research. Many new architectures are inspired by UNet. But still, there is so much to explore. There are so many variants of this architecture in the industry and hence it is necessary to understand the first one to understand them better. So if you have any doubts please comment below or refer to the resources page.

Resources

- UNet original paper
- UNet Pytorch implementation
- UNet Tensorflow implementation
- More about Semantic Segmentation
- Practical Image Segmentation

Author’s Note

This tutorial is the second article in my series of DeepResearch articles. If you like this tutorial please let me know in comments and if you don’t please let me know in comments more briefly. If you have any doubts or any criticism just flood the comments with it. I’ll reply as soon as I can. If you like this tutorial please share it with your peers.

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. Upgrade