**Pixel Link Pytorch Implementation**

|  |
| --- |
| **목차**   1. **Introduction to functions** 2. **Dataset 구성하기** 3. **Network 이해하기** 4. **Traning Dataset을 GPU에 연결하기** 5. **Weight Initialization 과 Optimization 객체 생성** 6. **Data Augmentation과 Link Prediction(Instance Segmentation)** 7. **Forward Propagation(PixelLink Network Architecture)** |

**201421553**

**소프트웨어학과**

**김 한**

**1.Introduction to functions**

|  |
| --- |
| **Main.py : 프로그램의 시작과 train / retrain 함수 등으로 구성되어 있다.**  **Net.py : Pixellink의 Network (VGG16 + Linkage)가 구성되어 있다.**  **Config.py: 본 프로그램의 필요한 Parameter 목록들로 구성되어 있다.**  **Datasets.py: Data 학습을 위한 변형과 전처리, 논문의 핵심 함수 등으로 구성되어 있다.** |

**2. Dataset 구성하기**

|  |  |
| --- | --- |
|  | **<explanation>**  Import가 되는 순간 사용자정의 클래스인 net.py, datasets.py, criterion.py 등이 호출 된다. 단, 내부의 정의된 함수와 init이나 getitem 등은 실행이 되지 않고 class의 껍데기만 호출이 된다.  <사용하는 라이브러리>  Numpy : 각종 array 연산  Torch: CNN 신경망 구성  Torch.utils.data : DataLoader를 통해 raw한 이미지들을 dataset으로 구성.  Cv2 : 각종 이미지에 대한 전처리 |

|  |
| --- |
|  |
| **<explanation>**  Argparse는 **“Command Line”에** 사용자로부터 argument를 입력받아서 프로그램을 시킬 때 각 argument를 분리(pasring)해주는 것을 도와주는 module이다. 따라서 argparse.ArgumentParser함수를 통해 parser가 먼저 생성된다. 다음으로 add\_argument를 이용하여 입력받고자 하는 인자의 조건을 설정한다. **위의 코드에선 –-train 혹은 –retrain**이다. 다음으로 parser.parse\_args함수는 인자들을 parsing하여 args에 저장한다. |

|  |
| --- |
|  |
| **<explanation>**  Args 의 namespace에 train : true / retrain : false와 같이 미리 정의된 argument와 type의 값이 저장되어 있다. 따라서 위의 코드를 만나게 되면 **입력된 type 값에 맞게 해당함수가 호출**된다. **Python main.py –train True라고 입력할 경우** elif 문 안에 main() 함수를 호출하게 된다. |

|  |
| --- |
|  |
| **<explanation>**  다음으로 main함수가 호출이되면 datasets이라는 py의 함수인 PixelLinkIC15Dataset를 호출하게 되는데 argument로 config.py의 train\_image\_dir 와 train\_labels\_dir에서 값을 사용하게 된다.  두번째 코드는 config.py의 train\_image\_dir와 train\_labels\_dir 인데, 변수의 이름처럼 **학습을 위한 이미지와 정답 label에 해당하는 Ground Truth의 경로**가 적혀있다. |

|  |
| --- |
|  |
| **<explanation>**  pixelLinkIC15Dataset 클래스의 객체가 생성되는 동시에 init이라는 생성자를 호출하게 된다. 다시 **super를 통해 상속받는 parent 클래스인 ICDAR15Dataset 클래스의 생성자를 부르게 된다**. |

|  |
| --- |
|  |
| **<explanation>**  Parent 클래스인 ICDAR15Dataset의 생성자가 불릴 때, image\_dir 과 label\_dir을 저장하게 된다. 위에서 설명했듯, 여기에는 학습을 위한 이미지와 label이 담겨있는 경로이다. 이것을 argument로 하여 read\_labels을 부르게 된다. Read\_label 함수의 argument로 label\_dir와 config.all\_trains을 사용한다. 그 결과를 all\_label에 저장하게 된다.  두번째 코드는 config.py에 들어있는 all\_trains라는 변수인데 학습을 위한 이미지의 총 개수인 1000이 저장되어있다. |

|  |
| --- |
|  |
| **<explanation>**  우선, res에 위와 같은 코드를 실행하게 되면, 2차원 배열안에 num개수만큼의 1차원 배열이 생성되게 된다. 따라서 num이라는 파라미터로 all\_trains라는 이미지 1000개의 개수가 passing 되므로 1000개의 1차원 배열을 생성하게 된다. |

|  |
| --- |
|  |
| **<explanation>**  기본 파일 open으로 ground\_truth가 열리지 않고 오류가 생기기 때문에 codecs.open 함수와 utf-8 방식읭 인코딩을 통해 정상적으로 파일을 open한다. argument로는 label이 저장된 경로인 train\_images/ground\_truth/gt\_img\_1.txt 라는 최종경로에서 for문을 통해 하나씩 불러오게된다.  다음으로 tmp라는 dictionary를 선언하고 coordinate(text영역의 각 꼭짓점 4쌍의 좌표), content(들어있는 text의 내용), ignore(무시할만한 text), area(text 영역의 넓이)라는 key를 선언한다.    Ground\_Truch의 raw한 data값은 위와 같다. 총 8개의 coordinate 값과 text 값으로 이루어져 있다. 따라서 먼저 이를 “,” 기준으로 split하여 list로 담은 후에 reshape(4,2)를 이용하여 [[377,117],[463,117],[465,130],[378,130]](첫번째 줄 예시)와 같이 coordinate 값을 알맞게 재구성한다. 다음으로 tmp[‘content’]에는 인식이 된 text 문자열을 담는다. 또한 무시되어도 좋을만한 ###(ignore) 이라면 True로 그 값을 저장하고 마땅히 인식되어야 할 값이라면 False로 값을 저장한다.  마지막으로 tmp[‘area’]는 cv2의 contourArea에 coor(좌표값)을 argument로 넘겨서 넓이를 구한 값을 저장한다.  따라서, read\_label이라는 함수를 거치고 나면 ground\_truth의 txt 파일은 아래과 같은 구조로 res에 담기게 된다.    즉, **1000개의 이미지 파일에 대한 ground\_truth로써 좌표값, text 내용, 무시여부, text 넓이**를 담은 dictionary 1000개가 담겨있다. |

|  |
| --- |
|  |
| **<explanation>**  생성된 현재 dataset 객체에는 ground\_truth label에 대한 전처리가 완료된 상태로 남아있다.  다음으로 WeightedRandomSampler를 이용하여 총 1000개의 이미지 중에서 **config.batch\_size(24), replacement(중복)=True로 하여 24개의 이미지를 임의로 선택**하게 된다.  다음으로 dataloader를 통해 앞으로 학습을 진행할 data를 load할 수 있도록 완성한다.  여기까지가 **기존의 raw한 이미지 data를 활용하기위한 dataset 구성방식**이다. |

**3.Network 이해하기**

|  |
| --- |
|  |
| **<explanation>**  Dataloader가 완성되면 다음으로 net.py 의 class net의 객체를 생성하고 초기화를 진행한다. |

|  |
| --- |
|  |
| **<explanation>**  Class net은 torch 의 nn에 들어있는 **module을 상속받아서 신경망 관련 함수들을 overriding하여 이용**한다. 위에 보이는 network 의 구조는 **pixellink에서 언급된 VGG Network의 기본프레임을 조금 변형하여 이용**하였다. 먼저 convolution stage1(conv1\_1 + relu + conv1\_2 + relu + pool)로 정의되었고 각 파라미터에서 stride와 padding은 1로 한다. 최초 input channel은 RGB이므로 3개이며, kernel(filter)의 사이즈는 3x3으로 VGG와 같다. Output channel 64는 hyperparameter로써 convolution stage를 거칠때마다 64->128->256->512로 늘어나며 따라서 최초의 input channel 의 개수를 제외하고는 모든 imput channel의 개수는 당연히 이전 output channel의 개수와 동일하다. Pooling 같은 경우는 maxpooling 방식과 stride 2로 정의되어있다. 기본 kernel size는 2x2이며, stride도 2이다. 따라서 이미지의 크기는 pooling을 통과하면서 줄어든다. 마지막 convolution stage5의 pool에서는 kernel 의 크기를 3x3으로 하며 stride와 padding을 1로 정의한다. |

|  |
| --- |
|  |
| **<explanation>**  이부분은 **기존의 FCnet에 해당했던 VGG의 구조를 Pixellink에서 소개했던 논문의 방식(conversion FCnet to Conv)처럼 수정된 것**을 보여준다. 6번째 convolution에서 이용한 방식은 dilated convolution이다. 일단, **receptive field는 output layer의 뉴런 하나에 영향을 주는 input layer 쪽 뉴런들의 공간 사이즈**를 말한다. **즉, filter 의 사이즈와 같다**고 할 수 있는데, **dilated convolution은 이때의 커널의 사이 공간을 벌림으로써 receptive filed를 확장하는 방법**을 말한다.    위의 그림처럼 3x3의 filter가 있다고 할 때 dialated convolution을 할 경우 필터를 사이 공간을 다음과 같이 벌림으로써 convolution을 진행한다. 보통 pooling을 하게되면 이미지가 줄어들어 연산량이 많아진다. 그렇다고 이 연산량을 극복하기 위해 filter의 크기를 키워서 receptive field를 늘리면 파리미터가 많아져서 오버피팅의 위험이 있다. Dialated convolution은 이럴 경우 구해야하는 파라미터의 개수는 적되 계산량을 줄일 수 있는 있는 방식이며 보다 이미지 주위의 **global한 문맥정보를 파악할 수 있다는 장점**을 가진다. 또한 **이미지의 width와 height 사이즈를 유지할 수 있는 동시에 해상도 역시 유지**된다. 마지막 conv7에서는 stride 1 padding 0으로 하여 1024개의 channel를 얻으며 진행된다. |

|  |
| --- |
|  |
| **<Explanation>**   |  |  | | --- | --- | |  | Pixellink의 Network Architecture이다. Conv stage 1,2 를 진행한 후에 out1\_1로 conv1 1x1로 channel size 2와 16을 각각 구한다. 이때 stride는 1, padding=0으로 한다. 아래의 코드들 모두 마찬가지로 **적절한 stage를 forward를 한 결과를 해당 out에 집어넣어 conv를 하는 것**이라고 할 수 있다. 마지막 최종 prediction의 경우는 final1\_1,2에 집어넣어 결과를 출력한다. 자세한건 forward의 과정에서 다루도록 한다. | |

**4. Training Dataset 을 GPU에 연결하기**

|  |
| --- |
|  |
| **<explanation>**  Config.gpu가 True라면, 해당 PC에 GPU를 Data Training에 이용하게 된다. Torch의 device함수와 cuda 함수를 이용하여 이를 연동한다. 자세한 설명한 주석과 같다.  만약 GPU가 여러 개라면 DataParallel 함수를 이용하며, 아예 없을 경우는 CPU로 연산을 진핸한다. |

**5. Weight Initialization 과 Optimization 객체 생성**

|  |
| --- |
|  |
| **<explanation>**  위와같이 nn module에서 apply 함수를 이용하여 weight 파라미터에 대해서 초기화를 진행한다. 본 논문의 내용처럼 **Pixellink는 Stochastic Gradient Descent를 이용하여 Optimization을 진행**한다. Image-net에서 pretrained model을 활용하는 방식이 아니라 **Xavier method를 이용**하여 weight 파라미터에 대해서 초기화를 진행한다. Optim.SGD 라는 패키지내의 함수를 torch에서 이용을하며 parameter()의 목록을 함수를 통해 불러와 argument로 전달한다. Learning rate와 momentum, weight\_decay는 아래와 같다. 즉, 이러한 방식으로 optimization 객체를 생성해둔다. |

**6. Data Augmentation과 Link Prediction(Instance Segmentation)**

|  |
| --- |
|  |
| **<explanation>**  **Training Dataset 1000개를 한번 도는 것을 1 epoch**이라 할 때 학습을 위해 **총 60000번**의 training을 진행하겠다. I\_batch는 이전에 구성 되어진 dataloader에서 불러온 해당 data에 대한 index에 해당하며, sample은 해당 data에 들어있는 정보를 말하며 dictionary 형태로 되어있음을 알 수 있다. **여기서 enumerate를 진행할 때, dataloader에 있는 PixelLinkIC15Dataset 객체의 \_\_getitem\_\_ 함수가 불리게 된다.** |

|  |
| --- |
|  |
| **<explanation>**  위처럼 getitem에 train이 true로 되어 있기 때문에, train\_data\_transform 함수를 해당하는 이미지의 index와 함께 호출한다. |

|  |
| --- |
|  |
| **<explanation>**  Training Data의 수를 늘리기 위한 Data Augmentation에 해당하는 부분이다. 먼저 training하고자 하는 이미지와 ground\_truth를 불러온 후, 이미지에 대해서 회전(rotate)과 특정영역을 자른다(crop). 모든 이미지를 rotate하거나 crop하는 것이 아니라 50퍼센트의 확률을 넘을 경우에 대해서만 이를 작동시킨다. 변형된 이미지는 다시 512x512의 크기로 재조정 된다. 즉, 정리를 하자면 모든 이미지를 회전시키거나 특정영역을 잘라서 새로운 데이터로 이용하는 것이 아니라 샘플링된 이미지 중에서 50퍼센트의 확률로 이미지를 변형하여 데이터로 이용하는 방식이다. 마지막으로 이미지를 torch 라이브러리가 인식할 수 있도록 Channel, Height, Width 순서로 바꿔준다. 이후에 변형된 이미지와 label을 반환한다. |

|  |
| --- |
|  |
| **<explanation>**  반환된 image는 batch size(샘플링된 이미지의 수) x Channel x Height x Width로 저장되어있다. 이를 tensor형태의 4차원으로 전환해준다.  다음으로 PixelLinkIC15Dataset에 있는 static method인 label\_to\_mask\_pixel\_pos\_weight 함수를 호출하게 되는데, label과 512라는 이미지의 크기 그리고 현재 이용하려하는 version 2가 argument로 넘어가게 된다. |

|  |
| --- |
|  |
| **<explanation>**  Pixellink 논문에 핵심적으로 소개된 linkage를 이용한 instance 별 segmentation 방식이 해당 method에 소개되어있다. 우선 앞서 전처리를 통해 dictionary에 ignore와 coordinate가 저장되어 있다는 것을 알 수 있다. Ignore 변수와 label 변수에 이를 각각 집어넣고 numpy를 통해 변환되며, 특히 coordinate는 4쌍의 형식으로 reshape된다. Pixel\_mask 와 link\_mask는 version 2이므로 논문의 내용처럼 1/2배의 사이즈인 256으로 한다. 이를 모두 0으로 초기화한다. |

|  |
| --- |
|  |
| **<explanation>**  Label.shape는 특정 이미지에서 발견된 text의 개수를 담고 있다. 즉, for문을 통해 해당이미지 text로 detected 된 것들 중에서 if not ignore statement을 통해 마땅히 detected 되어야하는 text area에 대해서 무언가를 진행하겠다는 뜻이다. if 문 아래를 좀 더 살펴보면, mask\_tmp를 하나 만들고 여기에 **drawContour라는 함수를 이용해 text 영역을 직접 그리게 된다**. 이때 **그리는 과정에서 mask\_tmp에 영역으로 그려지는 픽셀에 해당하는 부분이 1로 바뀌게 되고** 이를 pixel\_mask에 값을 추가하게 된다. 즉, for문을 모두 수행하게 되면 **pixel\_mask 변수에는 text 영역에 해당하는 pixel들만이 모두 1이되고 일종의 이미지에 대한 비트맵 형식으로 저장**되게 된다. 아래의 neg\_pixel\_mask는 이 값을 뒤집어 0인 pixel을 1로 바꾸어 반대의 비트맵이 저장된다.  또한 pixel\_mask[pixel\_mask !=1] = 0 로 하여 1이 아닌 값. 즉, 0이라면 그대로 0 2이상의 값들 역시 0으로 표시하겠다는 것은 **중복된 영역은 text area로 판단하지 않는다**는 것을 알 수 있다.  다음으로 pixel\_mask\_area를 1로 점힌찍힌 모든부분에 대해서 area로 계산하여 변수로 저장한다. 아래의 for문은 구해진 pixel\_mask에 아까의 과정을 한번 더 반복하여 진짜로 1에 해당하는 부분만 다시 체크하는 것이고 확인이 끝난 후 **area의 값이 존재한다면 이를 text\_box로 간주하고 real\_box\_num이라는 bounding box의 개수를 카운트**한다. 박스가 하나도 없다는 것은 이미지 내에서 text가 존재하지 않는다는 것이므로 이를 바로 정리하여 반환하고 만약 text가 있으면 box당 평균 weight를 area/box\_num으로 하여 새로운 변수를 생성한다.    667번 이미지에서 drawContour 함수를 통해 찾은 텍스트영역은 위와 같다. Bounding box의 수는 7개이다. |

|  |
| --- |
|  |
| **<explanation>**  위의코드는 **구해진 pixel\_mask에서 detected된 box를 크기에 따라 weight를 주는 방식으로 drawContour를 한 것을 의미**한다. 로직자체는 위의 pixel\_mask를 구한 방식과 유사하게 tmp를 이용하여 summation된 weight를 구하는 방식이다. Weight를 추가한 drawcontour의 결과는 아래와 같다.    다음으로 우선 Link\_mask는 각 pixel의 8방향의 이웃에 해당하는 값에 1을 더하는 코드를 np.clip을 통해 완성한다. 즉, **기존의 pixel\_mask의 비트맵에서 8방향 중 특정 한방향에 1이 더해진 새로운 비트맵이 8개가 생성**된다. 이후 **logical\_and 를 이용하여 기존의 pixel\_mask 비트맵과 이웃방향에 1이 더해진 새로운 비트맵 에서 1이 중복되는 값으로 link\_mask가 생성**된다.  아래의 그림은 해당 설명에 대한 예시이며, 8방향의 이웃이 들어와 Linkage가 되었을떄link\_mask에 포함되는 것들을 표시한 것과 같다.    즉, 그림처럼 **근처에 붙은 pixel만이 1이 최종 link\_mask가 된다는 것을 알 수 있다. 이를 통해 instance segmentation이 이루어진다**는 것을 알 수 있다.   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |
| 위와같이 lInkage를 통해 각 방향별로 이웃에 대한 link\_mask의 결과를 확인할 수 있다.    위의 그림은 논문에서 소개한 pixellink의 방식이다. 즉, 위와 같은 방식으로 진행이 된다는 것을 알 수 있다.  Torch를 이용하여 최종적으로 pixel\_mask, weight, link\_mask, neg\_pixel\_mask 등이 4차원으로 전환되어 반환된다.    최종적으로 dictionary의 형태로 하나에 sample에 대하여 원본 이미지, pixel\_mask, neg\_pixel\_mask, label, pixel\_weight, link\_mask가 getitem 호출의 결과로 반환이 되게 된다. |

|  |
| --- |
|  |
| **<explanation>**  따라서 **enumerate가 dataloader를 호출하여 getitem을 부르게 됨으로써 i\_batch에는 random으로 sampling된 이미지의 번호가 들어가게 되고, sample에는 방금전에 반환된 dictionary가 들어가게된다.**  다음으로 dictionary의 각 key들에 대해 GPU연산을 위해 GPU에 연결이 된다. 다음으로 forward를 통해 image가 argument로 넘어가게 되면서 실질적인 CNN의 연산이 시작되게 된다. |

**7. Forward Propagation(PixelLink Network Architecture)**

|  |
| --- |
|  |
| |  |  | | --- | --- | |  | **<explanation>**  논문에선 **fc6, fc7이 convolutional layer로 전환**됐다고 언급되어 있다. 위의 코드와 **VGG의 기본 프레임을 따르는 동시에 fcnet을 conv로 전환한 FC net(Fully Convolutional Net) 방식으로 구성**되어 있다.  VGG16의 기본프레임과 왼쪽과 같다. | |

|  |
| --- |
|  |
| **<explanation>**  Upsampling 역시 위에서 소개한 pixellink의 network 구조처럼 진행된다**. Bilinear interporation을 통해 이미지를 확장하게 되면서 사이의 값들을 예측함으로써 이미지에 최대한 영향을 주지 않도록 한다**. 순서는 최종 out부터 다시 거슬러 올라가는 동시에 더해가며 upsampling이 진행된다.  **최종적으로 out\_1과 out\_2를 구함으로써 text/non-text prediction과 link prediction를 forward 함수의 결과로 반환**한다. |