LEC 7

Convolution layer

- linear classifier로는 얻을 수 없던 spatial information을 보존시켜 줌

- input image batch shape이 N\*Cin\*H\*W, filter shape이 Cout\*Cin\*Kw\*Kh일 때, Cout개의 filter가 이미지를 슬라이딩하며 dot product 연산을 거쳐 N\*Cout\*H’\*W’ 크기의 activation map을 생성함

- linear operation만을 수행하기 때문에 여러 layer를 쌓아도 linear classifier처럼 하나의 layer와 같아지므로, layer 사이에 nonlinear activation function을 추가함

-첫번째 layer filter는 edge 방향, 대비 색상과 같은 공간 정보를 가진 K\*K 크기의 local image template Cout개를 학습함

Padding / Strided Convolution / Pooling

- layer를 지날 때마다 spatial dimension이 줄어들어 생기는 정보 손실을 방지하기 위해 input 주변에 zero padding을 추가해 줌

- 하지만 L번째 layer가 1+L\*(K-1) 크기의 receptive field를 가지므로, input의 global context 정보를 얻으려면 너무 많은 layer가 필요하기 때문에, stride를 적용해 downsampling 할 수 있음

- 또한 pooling layer를 통해서도 downsampling이 가능하며, parameter 학습이 필요하지 않으며, transitional invariance를 얻을 수 있다는 장점이 있음

Calculation

- output size를 H’=(H-K+2P)/S+1 와 같이 구할 수 있음

- 학습가능한 parameter 수는 (Cin\*K\*K+1)\*Cout 개 이며, multiply-add 연산 횟수는 output shape\*filter 한 개의 shape 만큼 이루어짐

Batch Normalization

- 각 layer마다 input 분포가 변하는 internal covariate shift 문제를 해결하기 위해, output을 batch의 각 dimension 별로 정규화해주는 batch normalization을 사용해 deep network의 학습에 도움을 줄 수 있음

- test 시에는 input이 서로 독립적이어야 하므로, 평균과 분산을 train data의 running average 값으로 사용함

- 큰 learning rate 값을 사용해 빠르게 수렴할 수 있고, test 때는 convolution layer와 합쳐 사용할 수 있다는 장점도 있음

LEC 8

AlexNet

- AlexNet은 trial-and-error 방식으로 설계되었고, 5층의 conv layer를 사용하고, 중간에 max pooling으로 output크기를 줄이며, 마지막에 fc layer로 flatten해주는 구조로 이루어져 있음

- 초반 conv layer의 output이 고해상도이므로, memory 사용량이 높음

- output을 flatten시키기 때문에, 대부분의 파라미터가 fc layer에 위치함

- 많은 개수의 filter가 고해상도 output과 conv 연산 수행하므로, 연산량은 conv layer에 집중됨

VGG

- VGG는 이전과 다르게 2014년에 매우 깊은 layer을 처음 사용했음

- 모든 conv layer를 3x3으로 통일하고, 2\*2 max pool만을 사용하고, pooling 이후에 채널을 2배 늘리도록 설정한 design rule에 따라 설계됨

- 3x3 conv layer 2개를 겹쳐 사용해, 5x5 conv layer와 같은 receptive field를 가지면서 파라미터와 연산량을 줄임

- max pooling 이후 채널을 2배 늘리더라도, 해상도가 반씩 줄어들기 때문에 연산량은 같음

GoogLeNet

- GoogLeNet은 VGG와 달리 메모리, 파라미터, 연산량을 줄이려는 노력을 했음

- 초반에 Stem Network를 통해 size를 224에서 28로 downsampling해 효율성을 높임

- 각 layer에 parallel branch를 만들어 kernel size를 튜닝할 필요 없이 동시에 다양한 크기의 filter를 거치는 Inception module을 만듦

- 이 때 1x1 bottleneck layer를 다른 conv 연산 전에 수행해 채널 수를 줄여, 연산량을 줄임

- 대부분의 파라미터를 갖는 fc layer의 weight 크기를 줄이기 위해, 앞에 채널당 하나의 평균값만을 남기는 global average pooling을 추가함

- 중간중간에 Auxiliary Classifier에서 loss를 계산해, gradient 전파에 도움을 줌

Residual Network

- train 시에 deep 모델이 shallow 모델보다 성능이 안 좋은 원인이 overfitting이 아닌 underfitting 때문임이 알려지자, deep 모델이 identity function을 학습해 shallow 모델을 모방하고자 하는 아이디어에서 residual network가 제시됨

- 기존 conv block에 input X를 그대로 전파해주는 shortcut을 추가한 residual block을 통해 gradient가 바로 전파 가능해짐

- ResNet-50부터 사용한 bottleneck residual block에서는 3x3 conv 앞뒤로 1x1 conv를 추가해 채널 수를 줄인 후, 다시 늘림. 따라서 연산량은 줄고, 비선형성은 높아지는 장점을 얻음

LEC 9

Activation Functions

- Sigmoid는 절댓값이 커지면 gradient가 0에 가까워지는 saturation 문제가 생김

- gradient가 항상 positive하므로, 모든 weight의 부호가 같아져 optimization 과정이 지그재그로이뤄지는 문제도 있지만, 이 문제는 mini-batch training으로 해결 가능함

- Tanh는 zero-centered하지만, saturation 문제를 여전히 갖고 있음

- ReLU는 saturation 문제가 없고, 계산도 효율적이며 실제로 수렴도 빠르게 되는 장점이 있음

- 그러나 sigmoid처럼 항상 positive한 값이 되는 문제가 있고, x가 음수인 경우 gradient가 0이 돼 일부 training data는 아예 학습이 안 되는 문제가 있음

- 학습이 죽는 문제를 해결하기 위해 음수일 때의 그래프를 보완한 leaky ReLU, ELU, SELU 등의 함수도 제안됨

- 실제 학습 때는 ReLU를 사용하는 게 좋고, sigmoid나 tanh는 gradient 수렴에 문제가 되므로 쓰지 않는 게 좋음

Weight Initialization

- weight를 0으로 초기화하면 모든 data에 대해 output이 0이고, gradient도 0이 되어 학습이 되지 않음

- 작은 랜덤 값으로 초기화하면 activation 값이 0이 돼 gradient 또한 0이 되고, 큰 랜덤 값으로 초기화해도 값이 saturate되어 gradient가 0이 됨

- Xavier 초기화는 분산을 1/Din으로 설정해 input 과 output 분산을 같도록 만들어 주며, zero-centered 함수의 경우 잘 작동함

- ReLU 사용할 때는 표준편차에 2를 곱해주는 Kaiming 초기화를 사용할 수 있음

Regularization

- forward pass마다 일정 확률로 랜덤하게 뉴런을 0으로 설정하는 Dropout은 각 뉴런이 feature들을 중복 학습하는 것을 방지하는 효과가 있음

- 또한 각 binary mask마다 다른 모델이라 생각하면, 앙상블 효과도 있음

- test 때는 모든 뉴런을 사용하며, train output들의 평균과 같도록 test output에 p를 곱해 줌

- 이외에도 다양한 정규화 방법이 있지만, train 시에는 특정 randomness를 추가하고, test 때는randomness를 적분해 없애는 패턴은 일반적으로 동일함

LEC 10

Learning rate schedules

- learning rate schedule를 사용하면 높은 학습률의 빠른 수렴과 낮은 학습률의 global minimum 도달이라는 각 장점을 모두 활용할 수 있음

- 정해진 epoch마다 학습률을 낮추는 step 방식은 하이퍼파라미터 튜닝에 오랜 시간이 걸린다는 단점이 있음

- 하이퍼파라미터 수가 적은 방식들 중 cosine 방식은 computer vision 분야에서 자주 쓰이고, linear 방식은 NLP에서 자주 쓰임

- validation accuracy가 줄어들기 전에 모델 파라미터를 저장하는 early stopping은 항상 권장됨

Choosing Hyperparameters

- 각 하이퍼파라미터마다 특정 값을 정한 뒤 모든 조합을 확인하는 grid search, 범위만 설정하고 매번 랜덤한 값들을 사용하는 random search 방법이 있음

- 중요한 파라미터 기준으로 생각할 때, grid search는 적은 값만을 테스트하기 때문에 최적값을 찾기 어렵지만, random search는 다양한 값들을 확인하기에 중요한 파라미터의 분포에 대해 많은 정보를 얻을 수 있음

Without tons of GPUs

- GPU 자원이 부족한 경우, 우선 initial loss와 small sample overfitting을 통해 모델과 최적화 과정에 문제가 없는지 확인함

- 그 다음, 100번 정도의 iteration동안 loss가 감소하는 학습률을 찾고, 그 값들과 다른 하이퍼파라미터 값들로 작게 grid search를 진행함. 그렇게 찾은 조합들로 길게 학습한 후, learning curve에서 찾은 문제점들을 개선함

Transfer Learning

- 데이터 수가 많지 않더라도, 기존의 pretrained 모델의 layer를 그대로 이용하고, 위에 몇 개의 layer를 추가해 fine-tuning하는 방식의 transfer learning으로 성능을 높일 수 있음

- 이 때 갖고 있는 dataset의 크기와 그 dataset이 pretrain된 dataset과 얼마나 유사한 지가 fine-tuning layer 개수를 선택하는 데에 도움을 줌

LEC 11

Grouped Convolution

- grouped conv은 input을 각각 Cin/G 채널씩 나눠 G개의 그룹으로 만든 후, 하나의 filter도 그에 맞춰 Cin/G 채널을 가짐. 이후 Cout개의 filter마다 각자 해당하는 그룹과 conv을 수행함

- output 채널은 input의 Cin개 채널에만 영향을 받고, 이 과정은 G개의 parallel layerㄹ ㅗ나눠 conv을 수행하는 것과 같아짐

- grouped conv은 기존 conv에 비해 파라미터 수와 연산량을 1/G만큼 줄여줌. 또한 병렬 처리에 유리하며, 그룹별로 correlation이 높은 채널끼리 학습될 수 있음

Improving ResNets

- ResNeXt는 bottleneck residual block에 grouped conv를 적용해 기존 ResNet과 같은 연산량을 유지하면서도 성능을 높임

- SENet은 ResNet block에 global average pooling 이후 채널을 줄였다가 늘리는 SE Block을 추가해, global context 정보를 얻어 각 채널당 가중치를 부여해줌. 연산량 또한 유사함

- 모바일 기기에 사용되는 MobileNet은 grouped conv에서 그룹을 Cin개 만큼 나누는 depthwise conv과 1x1 conv를 사용해 기존 conv보다 연산량을 9배 줄임

- MobileNetV2는 저차원 layer에 필요한 정보가 압축되어 있다는 가정 하에 ResNet과 반대로 채널을 늘렸다 줄이는 방식을 택함

- ShuffleNet은 group conv layer들 사이에 채널 순서를 바꿔주는 channel shuffle을 수행해 추가적인 conv layer 없이도 채널 사이의 정보를 얻도록 했음

Neural Architecture Search

- NAS는 RNN controller를 통해 네트워크 구조를 출력하고, 샘플링한 모델 학습해 나온 결과를 다시 controller에 업데이트함. 강화학습을 통해 점점 좋은 네트워크를 출력하게 됨

- 최근에는 NAS보단 직접 ResNet에 변형을 준 모델들이 제시되면서 성능을 높임

- NFNets은 bach norm 대신 scale 파라미터를 사용했고, weight를 직접 정규화했음

- RegNets은 design space에서 모델 구조의 파라미터들을 random search하며 최적의 모델을 찾음

LEC 12

GPU / Deep learning frameworks

- CPU는 코어 개수가 적지만, 각 코어가 빠른 clock speed를 가짐

- 반면 GPU는 각 코어는 느리지만, 수천개의 코어를 가져 행렬 연산과 같은 병렬 처리에 장점을 가짐

- Deep learning framework는 새로운 아이디어를 빠르게 구현하도록 다양한 common layer나 utility를 제공하며, gradient를 자동으로 계산해주고, GPU에서 효율적으로 작동되어야 함

PyTorch

- Tensor는 GPU에서 작동하는 다차원 array를 의미함

- Autograd는 requires\_grad=True인 tensor를 대상으로 forward pass를 따라가며 계산 그래프를 생성하고, backward()가 호출되면 각 tensor의 gradient를 계산해 줌

- Module은 state와 weight를 저장가능한 neural network layer를 정의해주며, 기존 module을 subclass들로 이용해 custom 모델을 만들 수 있음

Dynamic / Static Computation Graphs

- 매 iteration마다 그래프를 생성하고 삭제하는 dynamic graph는 input에 따라 모델 구조가 바뀌는 recurrent, recursive, modular network에 적용됨

- 반면 매번 graph를 재사용할 수 있는 static graph는 최적화를 통해 효율적 계산이 가능하고, serialization으로 code 없이 다른 언어에서 그래프를 사용가능하다는 장점이 있음

LEC 13

Object Detection

- object detection은 multiple objects들 각각에 대해 category label과 bounding box 좌표를 output으로 출력하는 classification과 regression 두 task를 수행함

- 기존 image classification과 달리 한 image 당 여러 object를 찾아야 하고, label과 bounding box 두 개의 output을 가져야 하며, 고해상도의 이미지가 필요해 계산량이 증가하는 문제가 있음

- single object detection의 경우 기존 이미지 분류 모델에 bounding box에 대한 regression loss를 적용하는 multitask loss를 사용할 수 있음

- multi object의 경우로 확장할 때, sliding window 방식을 생각해볼 수 있지만, H\*H\*W\*W 정도 크기의 너무 많은 box가 생김

- 대신 적은 수의 object 후보 영역을 찾는 region proposal을 이용할 수 있음

R-CNN

- R-CNN은 region proposal로 찾은 2000개의 RoI 영역을 224x224 크기로 warping한 image를 각각 convnet의 input으로 넣음

- 이후 class 확률과 bounding box transform 파라미터를 output으로 출력하는 과정으로 이루어짐

IoU / Non-Max Suppression

- 예측 box의 정확도를 나타내는 metric으로 IoU를 사용함. IoU는 ground-truth box와 예측 box 간의 (Intersection 영역 / Union 영역)으로 구할 수 있음

- R-CNN은 학습 시에 IoU에 따라 region proposal을 positive, negative, neutral로 분류함. positive box만 class 예측과 box offset 예측을 둘 다 수행하며, negative box는 background class 예측만 수행함

- 동일 object에 대해 중복되는 box를 제거하기 위해 NMS를 사용가능함. 가장 높은 clsss score를 가진 box와 IoU가 threshold 이상인 box들을 제거하는 과정을 반복해서 진행함

Mean Average Precision

- MAP는 object detector 모델의 성능을 평가하는 metric임

- 우선 NMS 이후 남은 box를 클래스마다 score 순서대로 정렬함. 그 후, 순차적으로 예측 box와 ground-truth box를 비교하며 precision과 recall을 구해 PR curve를 그려, 영역 크기의 클래스 별 평균으로 mAP를 구함. 마지막으로, threshold별 평균을 구해 모델의 mAP를 구함

LEC 14

Fast R-CNN

- 기존 R-CNN은 image당 2000개의 forward pass를 거치고, 이 때 중복되는 부분에 대해 여러 번계산되므로 매우 느림

- 이 문제를 해결하기 위해 Fast R-CNN은 먼저 image 전체를 한 번만 CNN에 넣어 출력된 image features를 얻어냄. 그 후, 각 RoI 영역들로 feature를 resize하고 crop한 후 가벼운 CNN에 입력으로 넣어 최종 bounding box와 class 출력을 구하기 때문에 계산량이 줄어듦

RoI Pool

- 이 때 feature를 crop할 때 RoI pooling을 이용함

- feature에서 바로 RoI를 구할 수 없기 때문에, input 이미지에서 구한 RoI 영역을 feature grid에 맞춰 snap함.

- 그 후, 비슷한 크기의 2x2 subregion으로 나눠 max pooling을 해 최종적으로 모든 region이 같은 크기의 feature를 갖도록 함

Faster R-CNN

- Fast R-CNN은 heuristic한 selective search를 이용해 region proposal을 하기 때문에, 이 부분이 bottleneck이 됨

- 대신 Faster R-CNN은 CNN으로 학습 가능한 region proposal network를 모델 중간에 추가함

- RPN은 image feature의 각 point마다 K개의 anchor box를 만들어 각 anchor box마다 객체가 있는지 여부에 대한 binary classification loss와 box regression loss 예측을 수행함

- 이 과정이 first stage이고, 이후에 각 region마다 RoI pooling을 수행해 최종적으로 class classification loss와 box regression loss를 출력하는 과정이 second stage임

Single-Stage Object Detection

- single-stage detector는 faster r-cnn의 first stage만을 사용하는 대신, RPN의 anchor box가 직접 category 분류까지 수행하도록 함

LEC 15

RoI Align

- RoI pooling은 snapping으로 인해 proposal 영역이 image에서 feature로 전해질 때 misalignment 문제와 box 좌표로 역전파가 불가능하다는 문제가 생김

- RoI align은 proposal 영역을 snapping 없이 그대로 feature 영역에 옮긴 뒤, 같은 크기의 subregion마다 일정 간격의 point들을 bilinear interpolation으로 샘플링함. 따라서 위의 두 가지 문제 모두 해결가능함

Detection without Anchors

- CornetNet은 anchor box 없이 upper left corner와 lower right corner 확률에 관한 pixel 단위의 heatmap 두 개를 카테고리마다 학습함

- test time에는 upper left embedding과 lower right embedding 벡터가 유사한 픽셀끼리 묶어 object detection을 함

Semantic Segmentation

- semantic segmentation은 이미지의 모든 픽셀에 카테고리 label을 부여하는 task이고, instance 구분은 하지 않음

- 네트워크가 전부 conv layer인 모델을 이용해 마지막에 input과 같은 H\*W size의 score를 출력하는 fully convolutional network가 segmentation에 사용됨

- 하지만 receptive field를 고려했을 때 layer가 너무 깊어지고, 고해상도 image로 conv 연산은 비싸다는 문제가 있기 때문에, downsampling 이후 upsampling의 과정이 필요함

Upsampling

- 3개의 neighbor 값들을 바탕으로 픽셀 값을 에측하는 Bicubic interpolation이 자주 쓰임

- 학습가능한 방식으로는 transposed convolution이 사용됨. 기존 conv과 달리 input의 한 픽셀과filter 전체에 곱함. 또한 input이 한 픽셀씩 움직일 때 filter는 output에서 stride만큼의 pixel을 움직임.

- conv을 행렬곱으로 표현했을 때, transposed conv은 기존 conv 행렬을 transpose시킨 연산과 같아짐

Instance Segmentation

- Instance Segmentation은 각 object instance에 해당하는 픽셀을 찾는 task이며, 기존 object detection 모델의 끝에 mask prediction head를 추가해 region proposal마다 mask 예측을 함

- 이외에도 같은 방식으로 keypoint estimation, caption prediction, 3d shape prediction task도 수행할 수 있음

LEC 16

Recurrent Neural Networks

- RNN은 임의의 길이를 가진 sequence input이나 output을 다룰 수 있기 때문에, image captioning, machine translation 등 다양한 task를 수행할 수 있음

- non-sequential data도 여러 부분으로 나눠 sequential data처럼 사용할 수 있음

- 기본적인 vanilla rnn 구조는 매 time step마다 이전 hidden state와 현재 input을 입력으로 받아 새로운 hidden state를 출력하며, 매번 같은 weight를 공유함

- 계산 그래프를 time step으로 펼쳤을 때, 매 step마다 내려온 weight의 gradient를 합해 역전파 계산 가능함

- seq2seq 모델은 encoder에서 input sequence를 받는 부분과, 최종 hidden state를 decoder가 받아 output sequence를 출력하는 부분으로 나눠져 있음

- one-hot vector로 인코딩된 input을 weight matrix와 행렬곱하면 column 하나만 추출하므로, 그대신 각 input에 해당하는 column을 학습해 추출하는 embedding layer를 추가하기도 함

Truncated Backpropagation Through Time

- 긴 sequence의 경우 한 번에 forward pass와 backward pass를 계산하면 메모리가 많이 필요하므로, sequence를 chunk로 나눔

- 앞 chunk에서 forward 와 backward 연산 후 마지막 hidden state만 남기고 모두 메모리에서 삭제함. 그 후, hidden state를 다음 chunk에 전해주는 과정으로 진행됨

Long Short Term Memory (LSTM)

- vanilla rnn은 hidden state gradient를 역전파할 때 똑같은 weight matrix를 반복해서 곱해줘야 하고, tanh non-linearity를 거치게 되는 문제가 있음

- 만약 largest singular value가 1보다 크면 gradient가 폭발하고, 1보다 작으면 gradient가 소실되는 문제가 생김

- 1보다 큰 경우는 gradient clipping으로 일부 해결 가능하지만, 1보다 작은 경우 새로운 RNN 구조가 필요해 LSTM을 사용함

- LSTM은 hidden state 말고도 cell state를 가짐. 4개의 gate가 이번 time step의 값을 cell state에 얼마나 쓰고, 지울 지 여부를 알려주는 역할을 하고, 다음 hidden state를 업데이트하는 데 cell state를 이용함

- cell state를 역전파할 때 forget gate와 elementwise 곱만을 수행하기 때문에, 반복적인 W 행렬곱 연산도 없으며 non-linearity 문제도 줄어듦

- LSTM의 이러한 uninterrupted gradient flow는 resnet의 skip connection과도 유사함

LEC 17

Seq2Seq with RNNs and Attention

- rnn을 이용한 기존 seq2seq는 긴 input sequence를 하나의 final hidden state로만 decoder에 전달하는 bottleneck 문제가 있음

- 대신 attention은 decoder의 매 time step마다 어떤 input 시점에 집중해야 할지 알려주는 context vector를 계산함

- 이전 시점의 decoder state와 모든 encoder state를 mlp에 넣어 alignment score를 구해 softmax로 attention weight를 구한 뒤, encoder state들의 가중합으로 context vector를 계산함

- 이 때 hidden state들의 순서는 고려하지 않기 때문에, image feature grid를 hidden state로 이용해 image captioning에 attention을 사용할 수 있음

Attention Layer

- 몇 가지 과정을 거쳐 attention layer를 일반화할 수 있음

- similarity function으로 mlp 대신 dot product로 효율적으로 계산함. softmax의 vanishing gradient 문제와 차원이 클 때의 문제 해결을 위해 scaling도 추가함

- query가 여러 개일 때의 multiple query 경우로 확장함

- input을 key, value로 나눠 key는 query와 유사도를 구할 때 사용하고, value는 output을 구할 때 사용해 유연성을 높임

Self-Attention Layer

- self-attention query 또한 input vector로 생성해, input끼리 서로 비교하는 특수한 attention을 의미함

- self-attention은 input 순서에 영향을 받지 않는 permutation equivariant한 특성을 가지므로, 순서가 중요한 task의 경우 positional encoding을 input과 합침

- 뒤 sequence를 미리 알면 안 되는 language modeling 같은 task의 경우, masked self-attention을 이용해 similarity matrix에 (–)무한대 mask를 처리해 줌

- multihead self-attention은 input 차원을 h개로 나눠 각 attenhion head로 병렬 처리가 가능하도록 해줌

The Transformer

- self attention은 긴 sequence를 처리가능한 rnn과 병렬 연산이 가능한 1d conv의 장점을 모두 갖고 있음

- transformer는 self-attention만을 이용한 block을 쌓아 만들며, attention 이후의 layer normalization과 mlp 과정은 input vector마다 독립적으로 계산 가능해 gpu에 적합함

LEC 18

How to use Attention / Transformers for Vision?

- vision 분야에서 attention만을 이용해 학습하는 여러 아이디어가 있을 수 있음

- query는 input의 receptive field의 가운데 pixel, key는 receptive field 전체, value는 output의 같은 영역으로 정의한 후, attention을 이용해 계산하는 local attention을 conv 대신 사용할 수 있음

- 또는 image pixel을 직접 transformer에 input으로 넣는 방법이 있지만, 메모리가 많이 필요함

- vision transformer는 image 패치를 linear projection으로 d차원 vector로 만든 후 input으로 넣어줌

Improving ViT

- 큰 dataset으로 pretrain했을 때는 resnet보다 뛰어난 성능을 보여주지만, 작은 dataset의 경우 성능이 떨어짐

- augmentation과 regularization 외에도, teacher cnn의 output을 모방하는 student vit를 학습하는 distillation을 통해 성능을 개선할 수 있음

Hierarchical ViT: Swin Transformer

- cnn은 image가 모델을 지나면서 해상도는 줄고 채널수는 증가하는데, object가 다양한 크기로 나타날 수 있으므로 유용함. 반면 vit는 모든 block의 해상도와 채널수가 같음

- swin transformer는 block들 사이에 patch merging을 해 patch 크기를 2배씩 키우는 계층적 vit모델 구조를 가짐

- 하지만 모델 초반에 큰 패치 크기로 인해 attention matrix도 커지는 문제가 생기는데, 각 patch를 M\*M으로 나눠 같은 window끼리만 attention을 계산하고, 중간에 shifted window로 변형을 줘 다른 window와도 attention할 수 있도록 함

Object Detection with Transformers: DETR

- detr은 transformer를 이용해 object detection을 수행함

- encoder에 image feature를 넣어주고, decoder에 object query를 입력해 예측 box들을 출력함. 이후 ground truth box와의 bipartite matching loss를 구해 학습함

LEC 19

Unsupervised Learning

- 비지도학습은 지도학습과 달리 label이 주어지지 않았을 때, data가 내포하고 있는 구조를 학습하며, 많은 양의 unlabel data를 사용 가능하다는 점에서 유용함

Generative Models

- discriminative model은 p(ylx) 분포를 학습하며, 각 image에 대한 label끼리만 경쟁하기 때문에 unreasonable input을 감지하지 못해 adversarial attack에 취약함

- data에 label을 부여하거나, 지도학습에서 사용됨

- generative model은 p(x) 분포를 학습하고, image 자체의 likelihood를 학습함

- outlier를 감지하거나, 새로운 데이터를 샘플링하거나, 비지도학습에서 사용됨

- conditional generative model은 p(ylx)를 학습하며, 각 label에서의 image의 likelihood를 학습함

- 라벨에 따른 새로운 data를 생성하는 데 사용됨

Autoregressive models

- autoregressive model은 MLE를 이용해 data들의 가능도를 최대화하는 분포 p(x)를 직접 구함

- chain rule을 이용하면 이전 subpart들이 주어졌을 때의 다음 subpart 확률의 곱과 같아짐

- PixelRNN은 왼쪽과 위쪽의 픽셀 값과 hidden state를 이용해 각 픽셀값을 rnn을 통해 예측하는 방법이며, 순차적으로 계산되므로 학습과 생성 모두 느림

- PixelCNN도 비슷하지만 CNN으로 계산하고, 각 픽셀 별 병렬 계산이 가능해 학습은 빠르지만 생성은 여전히 느림

- autoregressive model은 p(x)를 명시적으로 계산하고, 이 분포가 좋은 metric이 된다는 장점이 있지만, 순차적 생성 방식이므로 느리다는 단점이 있음

Variational Autoencoders

- variational autoencoder는 p(x)를 직접 구하기 어렵기 때문에 lower bound를 최대화함

- autoencoder는 encoder에서 input의 저차원의 feature를 추출해 decoder에서 다시 input을 복원하는 비지도학습 방식이며, latent feature z를 학습하게 됨

- encoder를 다양한 downstream task에 사용 가능하지만, 새로운 data를 샘플링하지는 못함

- vae는 p(z)를 gaussian과 같은 간단한 분포로 가정하고, autoencoder로 학습한 z를 p(z)에서 샘플링한 후, decoder에서 얻은 p(xlz)로부터 x를 샘플링함

- 각 x에 대해 z를 구할 수 있다면 conditional generative model로 p(xlz)를 학습 가능하지만, z를 직접 구할 수 없음

- z를 marginalize out하는 방법도 z에 대해 적분할 수 없으므로 불가능함

- 베이즈 정리를 이용하면 P(zlx)를 구할 수 없지만, encoder가 q(zlx)를 학습해 p(zlx)를 예측할 수 있음

- logp(x)의 z에 대한 평균에 관한 최종 식에서 마지막 term은 p(zlx)를 모르기 때문에 제외하면, likelihood의 lower bound를 얻을 수 있고, 이것을 최대화하도록 vae를 학습하면 됨