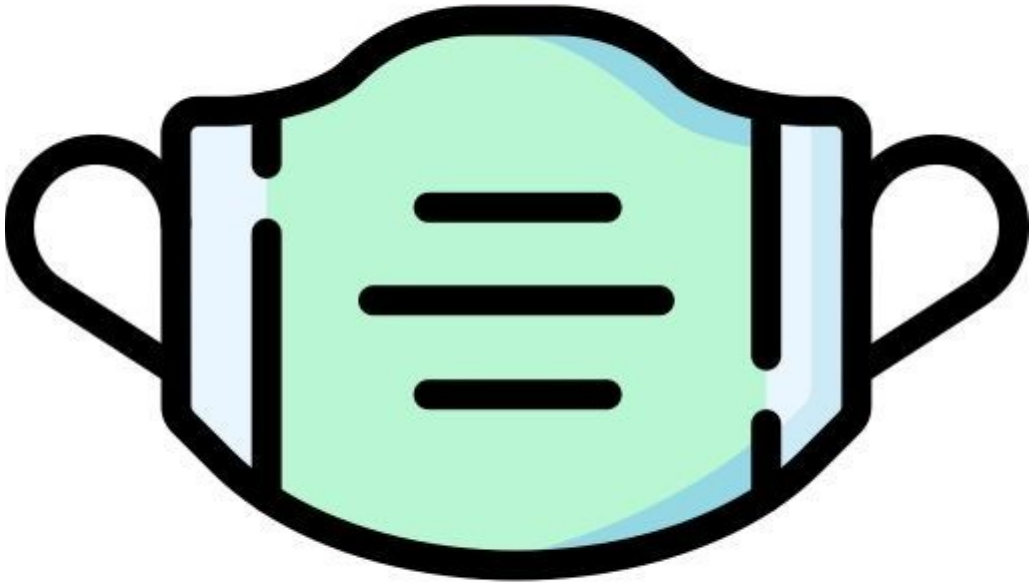
Project 제안서

(Mask 착용 유무 Classification)



- 작성자 : 31기 20정유철
- 작성일 : 2022. 09. 16

목 차

1. Project 소개

(1) 선정 동기

현재 우리는 4차산업혁명의 시대에 살고 있다. 기술의 발전속도는 무서울 정도로 빠르게 발전하고있는 상황이다.
현재 싱가포르 텔레콤 CDO에서는 2018년에 1인 1 비서로봇을 도입하였고, 업무 효율을 끌어올렸다고 발표한 바가 있다. 그래서 향후 미래에는 인공지능비서로봇을 개인별로 소유하는 시대가 올 것이라고 생각했다.
이러한 인공지능 비서로봇을 만드는 것이 내가 하고 싶은 궁극적인 목표이다. 하지만 미래에 있을법한 그러한 로봇은 정말 다양한 분야들과 기술들의 총 집합체일 것이라 생각이들어서, 향후 제 진로를 어느방향으로 정해야할지 고민을 하고 있었다.

그러던와중 방학중에 김효민 박사님께서 진행하신 진로세미나를 듣고, 내가 정진하고 싶은 분야를 생각해보았다. 인공지능비서로봇의 핵심 기술은, 사람처럼 생각하고 사람처럼 말하는 것이라는 생각이 들었고, 그러기 위해서는 우선 사람의 말을 잘 알아들어야한다는 결론에 도달하였다.

그러한 핵심 기술을 공부하기에 앞서, 방학기간 동안 배운 딥러닝을 조금 더 자세히 알아보고, 응용해보고싶었기에 딥러닝을 활용할만한게 없을까 생각을 하게 되었다. 어찌되었건 이러한 고민을 하는 궁극적인 이유는 미래에 사람들에게 도움이 되게끔 하는 것일 것이다.

최근 사스, 메르스, 신종플루, 에볼라, 코로나 등 여러 전염병들이 발발하였고, 이제는 마스크가 없는 생활은 상상할 수 없게 되었다. 이러한 이슈가 생기기 전 전염병에 관련하여 빌 게이츠는 전염병에대해 준비가 전무한 현상황을 비판하고 경고하였다.
세계보건기구 WHO에서도 매번 전염병에대한 대처가 느린 이유도 전염병 발생국가의 보고를 받고나서 움직이기 때문에 전염병을 예방할 수 있는 방법으로는 마스크 쓰기와 손씻기가 유일하다.

이러한 이유로 마스크 착용유무 분류기를 생각하였고, 이번 여름방학에 파이토치를 이용한 딥러닝 스터디를 진행하면서 따로 'Teachable Machine' 을 통한 마스크 착용 유무 딥러닝 모델을 만들어 보았는데, 이 과정에서 내 얼굴만을 한정해서 잘 작동하는 것을 보고 아쉬움을 많이 느꼈다.

'Pytorch' 를 포함함 딥러닝 프레임워크도 그렇고 'Teachable Machine' 에서도 사용자가 쓰기 편하게 만들어 놓아 세부적인 개념을 알아도 이것이 실제 코드로 어떻게 독작하는지 알기 힘든면이 있었고, 그래서 이를 직접 프레임워크 없이 Numpy, Scikit-learn, SciPy, Matplotlib 등을 이용해 자세한 구현과정을 거쳐보고 싶은 생각이 들어 작품을 준비하게 되었다.

(2) Project 시나리오

1. 학습 데이터 셋을 위한 사진을 수집하거나 촬영한다. (마스크 쓴 사진, 벗은 사진)
2. 웹캠이나 노트북내장 카메라를 이용하여 실시간으로 학습된 모델에 적용해 마스크 착용 유무를 잘 분류하는지 확인한다.
3. 잘 인식되고 있는지에 대한 척도를 판단하기 위해 인식률이나 정확도를 수치적으로 계산해 보여준다.

(3) 목표

1. 모델 제작

이미지 전처리를 위한 최소한의 torch를 제외한 torch는 사용하지 않고, numpy로 마스크 착용유무 판단 분류기를 구현한다.

2. 훈련 데이터 제작 및 테스트

마스크 착용과 미착용 두가지로 분류하여 이미지들을 'kaggle' 의 'Covid Face Mask Detection Dataset' 받아와 모델을 학습시킵니다. 모델 제작 후 마스크를 쓴 사진과 안쓴 사진으로 모델이 잘 분류하는지 테스트한다.

3. 피드백 및 수정 반복

4. 실제 평가

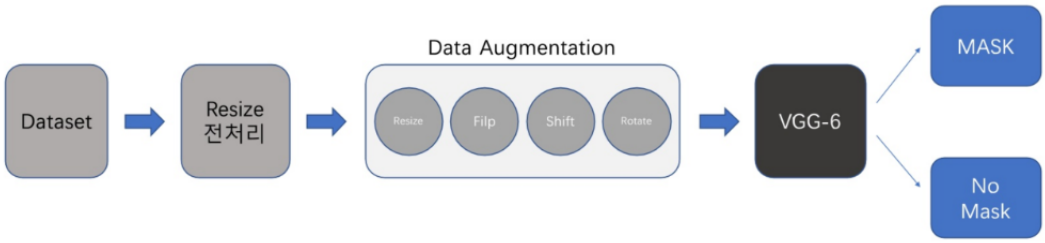
Accuracy, Precision, Recall

5. 데이터 셋

data 폴더 안에 Test, Train, Validation 으로 구성된 세가지 폴더를 만들고 그 각각의 폴더 안에 Mask, Non Mask 로 두가지 폴더를 만들어주고 그 안에는 각각 마스크를 착용한 사진, 착용하지 않은 사진을 구별하여 넣어준다.

2. Project 내용

(2) System Architecture



(3) 목표 구현 내용

1. 모델 제작

1-1. 환경구축

제작 환경은 Anaconda, Jupyter Notebook 에서 진행할 계획이며. 사용할 패키지는 다음과 같다.

ImageFolder : 폴더명을 레이블로하여 이미지가 분류되어있을 경우 data split을 하기 위한 패키지이다.
Data Augementation: Resize, Flip, Rotate, Shift : 이미지를 전처리
numpy로는 conv2d, batch normalization, ReLU, Maxpool, FC layer, optimizer, loss fuction, F1 Score, Recall, Precision 을 구현할 예정이다.

1-2 데이터 수집

데이터 수집은 'kaggle' 에서 'Covid Face Mask Detection Dataset' 에서 받아올 예정이다. 이 Dataset은 Test, Train, Validation 각 폴더안에 Mask, Non Mask 로 이미지 파일들이 분류되어 있다.

Train - Mask(300개), Non Mask(300개)
Validation - Mask(152개), Non Mask(153개)
Test - Mask(50개), Non Mask(50개)

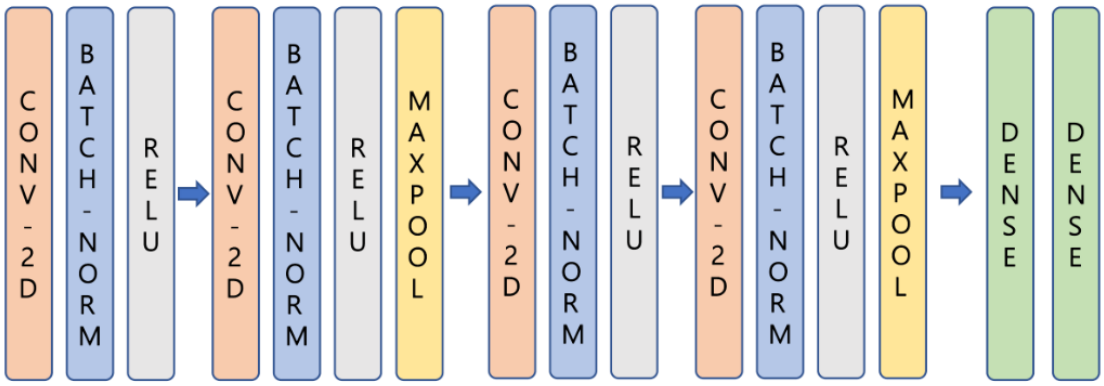
Data Augmetation(Resize, Rotate, Flip, Shift)를 이용하여 학습시킬 데이터를 늘려준다.

데이터 전처리 과정의 경우 앞서 소개한 파일의 각각의 폴더안 이미지 사이즈들이 아래 사진들처럼 제각각이기 때문에, torchvision.transforms.Resize 로 224 * 224 로 크기를 통일시켜준다.

<train set에서 무작위로 뽑은 예시>

1-3. Model Architecture

모델은 VGG net(VGG 개요참고)에서 영감을 받아와 VGG - 6 라는 이름으로 직접 만든모델로 구현해보려한다. Epoch은 30회, Batch size는 100으로 시작해볼 예 정이다.



내가 구현하고자하는 Model Architecture 는 위 그림과 같다. Conv layer는 첫번째 층부터 아래와 같이 구성할 예정이다.

Conv2d(3, 32, kernel_size=3, stride=2, padding=1)
BatchNorm2d(32)
ReLU

Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
BatchNorm2d(64)
ReLU
Maxpool2d(2)

Conv2d(64, 128, kernel_size=3, stride=2, padding=1)
BatchNorm2d(128)
ReLU

Conv2d(128, 128, kernel_size=3, stride=2, padding=1)
BatchNorm2d(128)
ReLU
Maxpool2d(2)

이렇게 모든 층을 거치고 마지막에 추출되는 특징의 크기는 128*7*7=6,272 개이며 이후 Dense층, 즉 FC layer를 거친다. 입력차원과 출력차원을 간략히 적어보면

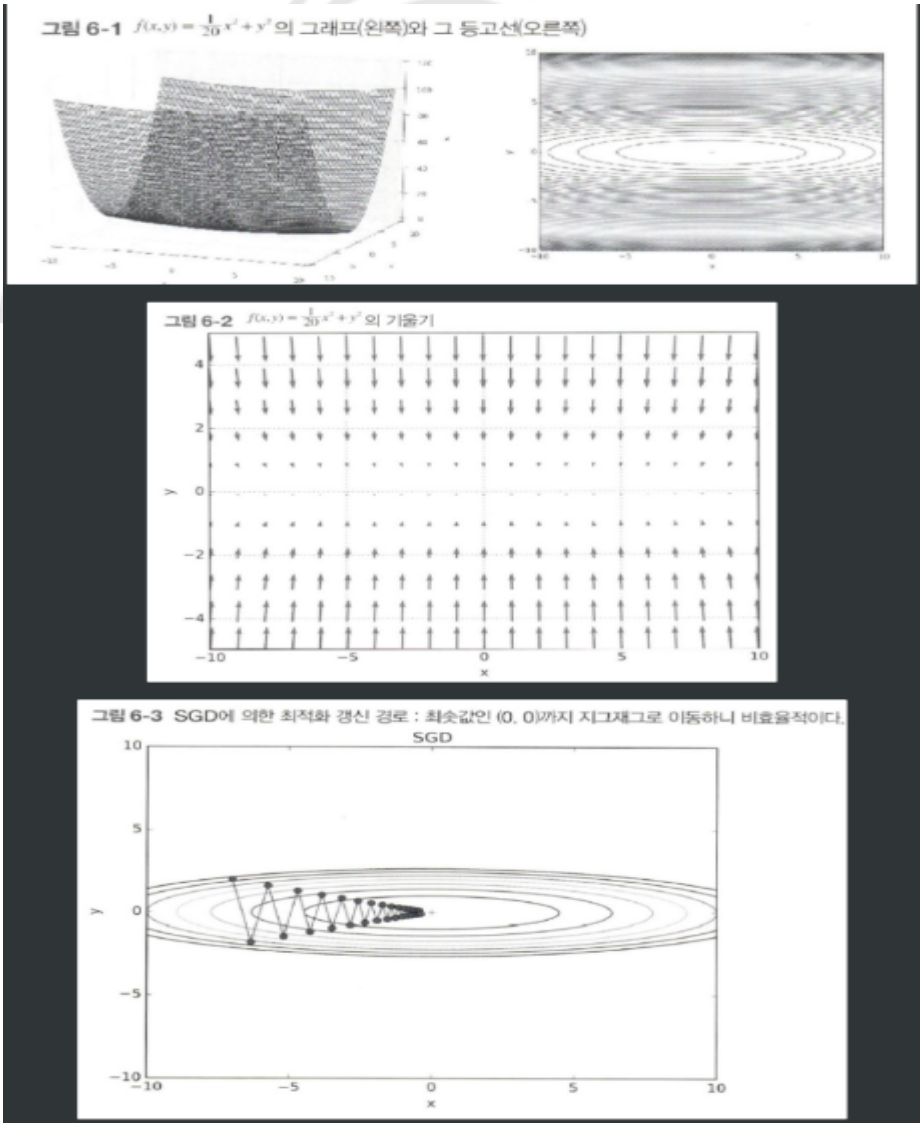
Linear(6272, 512) -> Linear(512, 2) 로 표현할 수 있겠다.

1-4. Optimizer

Optimizer 는 Adam optimizer를 사용할 예정이다.

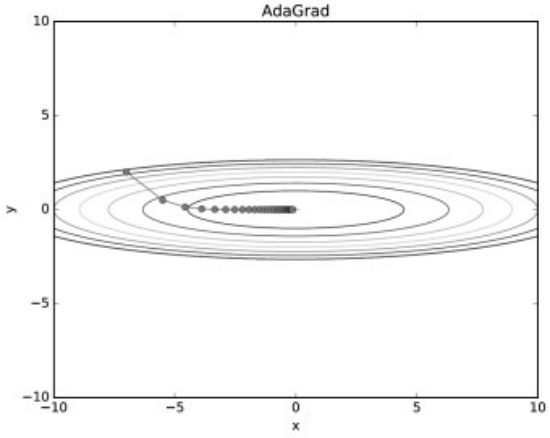
이유는 Stepsize가 Gradient의 Rescaling에 영향을 받지 않고, Gradient가 커져도 Stepsize는 bound되어 있어서 어떠한 Loss Function을 사용한다 하더라도 안정적으로 최적화를 위한 하강이 가능하다는 점과 Stepsize를 과거의 Gradient 크기를 참고하여 Adapted 시킬 수 있기 때문이다. 그렇다면 이전에 나온 Optimizer들의 단점들이 어떠하길래 Adam이 좋은지 알아보겠다.

먼저 SGD의 단점은 f(x,y) = 1/20* x^2 + y^2 같은 비등방성 함수에서 발생한다.



위에서 예시로 든 함수는 '밥그릇' 모양이 x축 방향으로 늘인 듯한 모습이고, 실제로 등고선은 x축 방향으로 늘인 타원으로 되어 있다. 함수의 기울기를 그려보면 두번째 사진과 같이 y축방향으로의 기울기는 크고 x축 방향은 작다는 것이 특징이다. 그래서 마지막 사진을 보면 SGD는 심하게 굽이진 움직임을 보이고 있다. 그렇기 때문에, 반복이 충분하면 최저점을 찾을 확률은 높지만, 항상 그렇지도 않을뿐더러 노이즈도 심한 단점을 보완하기 위해 AdaGrad, Adam 이라는 방법을 사용할 수 있다.

AdaGrad는 개별 매개변수에 적응적으로 학습률을 조정하면서 학습을 진행한다. 연산을 진행하며 매개변수의 원소 중에서 많이 움직인(크게 개선된) 원소는 학습률을 낮춰주는 방식으로 작동한다.
아래 그림과 같이 SGD보다 효율적으로 움직이는 것을 볼 수 있다. 이 AdaGrad 방법에 Momentum 이라는 방식이 합쳐지면 Adam 이 된다.



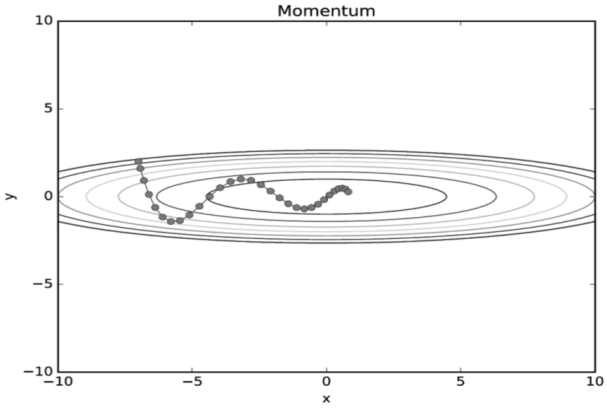
$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$

모멘텀의 속도 갱신 수식

$$W \leftarrow W + v$$

모멘텀 가중치 갱신 수식

모멘텀은 가중치 갱신 수식이 W+v 이다. 여기서 v는 속도를 의미한다. 속도 또한 갱신될 때, 매개변수 α를 v와 곱해서 손실함수를 따라 Global Minimum을 향해 내려갈 때, 아무힘을 받지 않아도 서서히 하강한다. 움직이는 모습은 아래 그림과 같다. SGD와 비교할 때, x축 방향으로 일정하게 가속되고, 방향의 변화가 아주 작게 일어나므로 SGD보다 효율적이라 할 수 있다.



Adam 은 2015년에 제안된 새로운 방법이다. Adam Optimizer의 수식은 아래 사진과 같다.

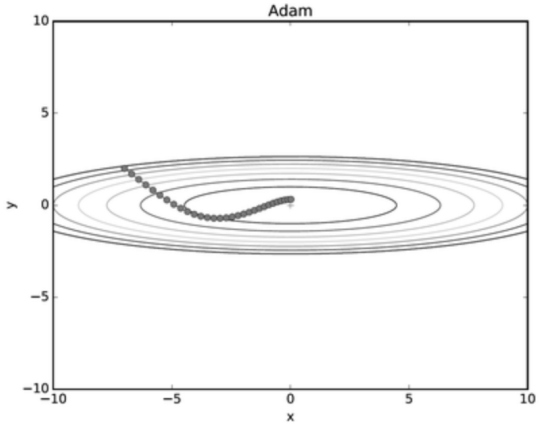
$$\mathrm{M(t)} = \beta_1 M(t-1) + (1-\beta_1)\frac{\partial}{\partial w(t)} Cost(w(t))$$

$$\mathrm{V(t)} = \beta_2 V(t-1) + (1-\beta_2)\left(\frac{\partial}{\partial w(i)} Cost(w(i))\right)^2$$

$$\widehat{M}(t) = \frac{M(\mathrm{t})}{1-{\beta_1}^t} \qquad \widehat{V}(t) = \frac{V(\mathrm{t})}{1-{\beta_2}^t}$$

$$\mathrm{W(t+1)} = W(t) - \alpha * \frac{\widehat{M}(t)}{\sqrt{\widehat{V}(t) + \epsilon}}$$

추가적으로 α = 0.001, β1 = 0.9, β2 = 0.999, ϵ = 10^-8 으로 설정하는 것이 가장 좋은 default값이라고 논문에 명시되어 있다. Adam optimizer의 학습그래프를 나타내면 아래그림과 같은 형태이다.



1-5 손실함수

Loss Function으로는 MSE Loss fuction을 사용할 예정입니다. MSE 손실함수는 선형회귀에서 많이 사용됩니다. 예측값과 실체값의 오차의 제곱을 활용한다.

$$cost(W,b) = \frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} - H(x^{(i)})\right]^2$$

MSE 함수를 코드로 구현해보면 다음과 같다. y를 신경망의 출력 t를 정답레이블이라한다면, 그 사이 오차의 제곱의 합으로 손실함수를 나타낸다. 제곱을 하는 이유는 -오차와 +오차간의 합이 0이 나오는 경우에는 기계가 오차가 0이다라고 해석할 수 있는 문제가 있기 때문에 제곱을 함으로써 절대값 처리를 해준다고 보면 된다.

우리는 마스크를 착용했는지 안했는지 두가지를 다뤄야한다. 그러므로 Dense Layer의 말단에는 2개의 노드가 있어야한다.

2. 실제 평가

실제 평가에서는 Precision 과 Recall 그리고 F1-score 를 통해 모델의 성능을 수치적으로 평가한다.

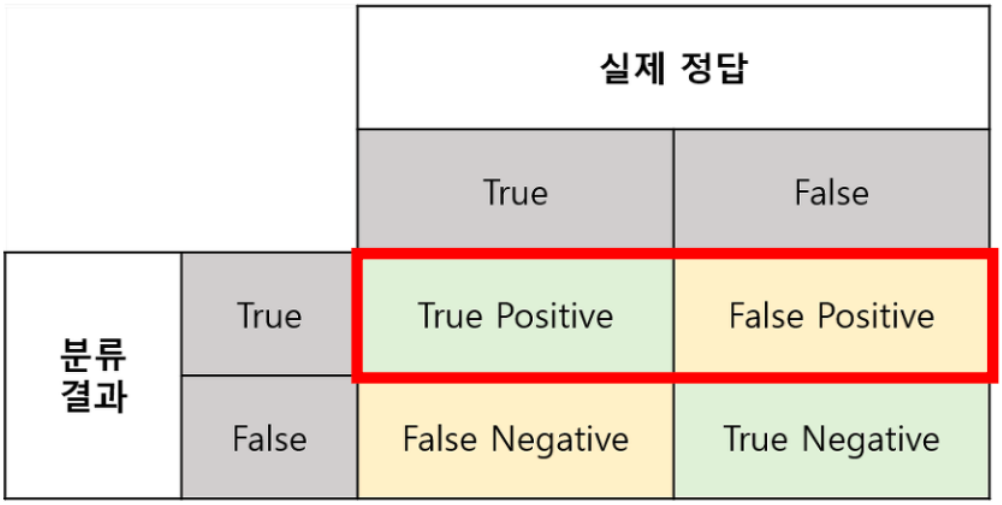
이때 Overfitting을 방지하기 위해
- Validation set을 이용
- 부족하다면 추가적인 학습 데이터를 추가
- 뽑아내는 Feature(특징)들 줄이기
- Validation Loss 가 더 낮아지기 전에 Early Stop
- Network size를 줄이기
- Weight decay
- Drop out
- Batch Normalization

등의 방법들을 사용할 예정이다.

2-1. Precision(정밀도)

모델이 도출한 정답과 실제 정답의 관계를 이용해 평가를 진행할 수 있다.
True라고 분류한 것 중 실제 True의 비율을 말한다.

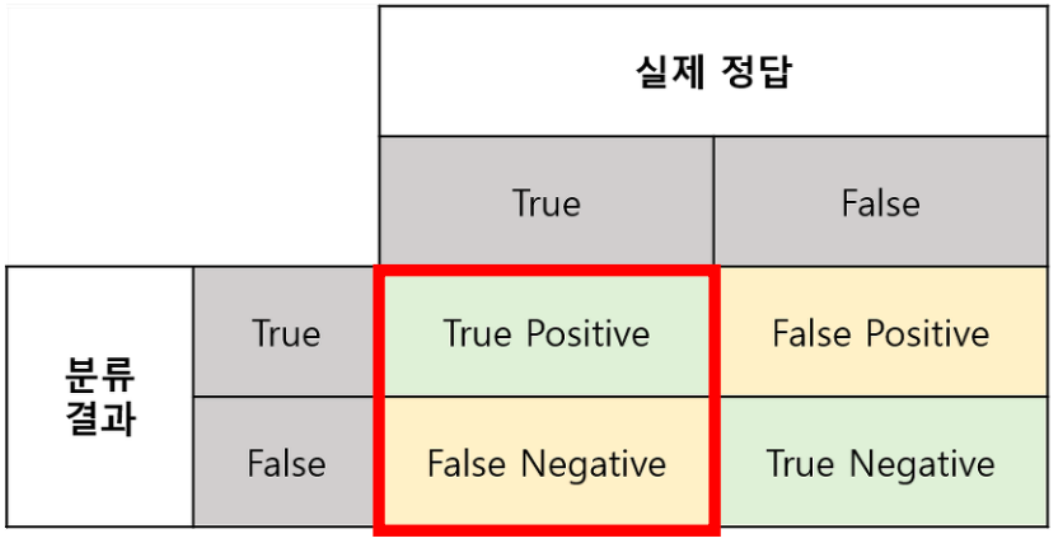
TP: 실제 True인 것을 True라 예측(정답)
FP: 실제 False인 것을 True라 예측(오답)
FN: 실제 True인 것을 False라 예측(오답)
TN: 실제 False인 것을 False라 예측(정답)



Precision = TP/(TP + FP)

2-2. Recall(검출율 or 재현율)

인식/탐지 기술의 성능을 평가하기 위해서는 Precision과 Recall을 동시에 고려해야 한다. Precision과 Recall의 성능 변화를 전체적으로 살펴야한다.
실제 True중 올바르게 True를 맞춘 비율을 말한다.



Recall = TP/(TP + FN)

2-3. Accuracy

전체 샘플들의 개수 중에서 얼마나 나의 모델이 정답이라고 예측한 샘플이 포함되어있는지를 알아보는 방법이다.

$$Accuracy = \frac{\text{올바르게 예측한 샘플 개수}}{\text{전체 샘플 개수}} = \frac{TP + TN}{TP + TN + FP + FN}$$

3. Project 진행 일정

진행 목표	9월			10월				11월		
	3	4	5	1	2	3	4	1	2	3
Data Preprocessing										
Convolution						중간고사				
Optimizer										
Foward, Backward, Test										
Overfitting 방지										
디버깅 및 Cam Video										

4. 용어 정리

Torch

torch란, Facebook에서 제공하는 딥러닝 도구로서, Numpy와 효율적인 연동을 지원하는 편리한 도구이다.

Numpy

numpy란, 다차원 배열을 쉽게 처리하고 효율적으로 사용할 수 있도록 지원하는 파이썬 패키지입니다. numpy는 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공한다. 데이터 분석을 할 때, Pandas와 함께 자주 사용하는

도구로 등장한다.

CNN

Convolution Neural Network의 약자로 일반 Deep Neural Network에서 이미지나 영상과 같은 데이터를 처리할 때 발생하는 문제점들을 보완한 방법이다.

https://yuchulnote.github.io/deep_learning_study/Lab10-1/

Channel

이미지 처리의 기본적인 용어인 채널에 대해서 간단히 정리하면

기계는 글자나, 이미지보다 텐서를 더 잘 처리할 수 있다. 이미지는 (높이, 너비, 채널) 이라는 3차원 텐서이다.
여기서 높이는 이미지의 세로방향 픽셀수, 너비는 이미지의 가로방향 픽셀수 그리고 채널은 이미지의 색깔 성분을 의미한다.

흑백이미지는 채널 수가 1이고, 일반적인 이미지는 RGB로 색을 분류할 수 있기 때문에 채널 수가 3이 된다.

그래서 첫번째 convolution layer의 입력 채널값으로 3이 들어가게 되는 이유이다.

Stride

Kernel(커널)이 이동하는 스텝의 크기이다. 보통 커널사이즈로는 3x3, 5x5를 사용한다.

Conv2d(convolution)

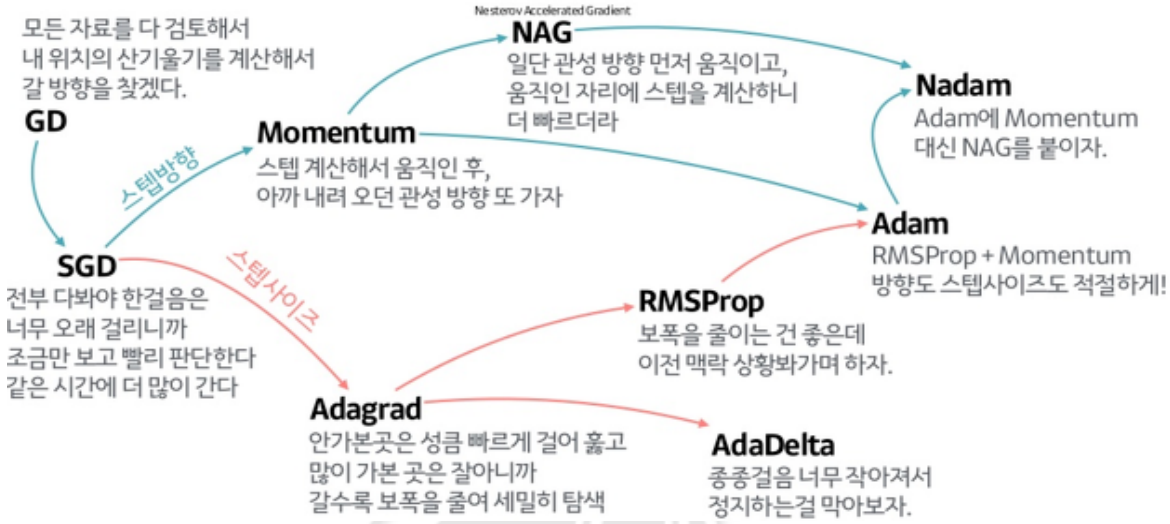
convolution이란 이미지 위에서 stride 값 만큼 filter(kernel)를 이동시키면서 겹쳐지는 부분의 각 원소의 값을 곱해서 모두 더한 값을 출력으로 하는 연산을 말한다.

VGG

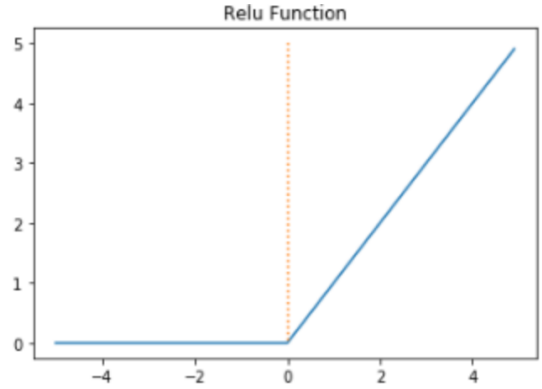
Oxford VGG(Visual Geometry Group)에서 만든 Network입니다. VGG는 원래 Layer 개수에 따라 VGG11~VGG19 까지 만들 수 있도록 되어 있다. 사용방법은 torchvision.models.vgg로 사용할 수 있다.

Optimizer

Optimization은 손실함수의 결과 값을 최소화하는 모델의 파라미터를 찾는 것을 의미한다. 이러한 Optimization의 알고리즘을 Optimizer라고 한다.



ReLU



인공 신경망에서 최고의 인기를 얻고있는 활성화 함수인 ReLU함수는 f(x)=max(0,x) 로 아주 간단하다.

렐루함수는 0이하의 음수부터는 0을 출력하고 0을 넘어가는 양수값에서는 자기자신, 즉 항등함수 x가 출력되는 함수이다.

렐루함수는 하이퍼블릭탄젠트함수나 시그모이드함수처럼 연산이 필요한 것이 아니라 단순 임계값이므로 연산 속도도 빠르고, 특정 양수값에 수렴하지 않으므로 깊은 신경망에서 더 잘 작동한다.

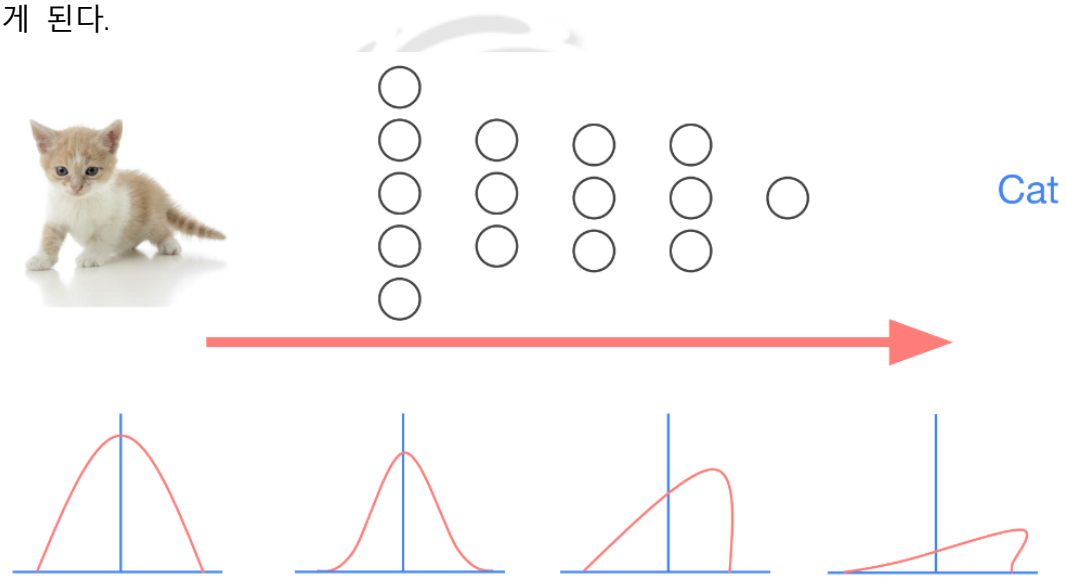
Batch Normalization

ReLU 계열의 함수와 He 초기화를 사용하는 것만으로도 어느정도 기울기 소실과 폭주를 예방할 수 있지만, 100%는 아니기 때문에, 또 다른 방법으로 제안된 것이 배치 정규화이다.

배치 정규화(Batch Normalization)는 인공신경망의 각 층에 들어가는 입력을 평균과 분산으로 정규화하여 학습을 효율적으로 만드는 방법이다.

Internal Covariate Shift(내부 공변량 변화)

내부 공변량 변화란, 학습 과정에서 층별로 입력 데이터 분포가 달라지는 현상을 말한다. 즉, 이전 층들의 학습에 의해 이전층의 가중치가 바뀌게 되면, 현재 층에 전달되는 입력 데이터의 분포가 현재 층이 학습했던 시점의 분포와 차이가 발생하게 된다.



예를 들어서 사진을 보고 고양이인지 아닌지 분류하는 분류기를 학습시킨다고 가정하였을 때, 층 하나를 지날 때마다, 사진의 그래프처럼 문제가 발생한다.
배치 정규화는 말 그대로 한 번에 들어오는 배치 단위로 정규화하는 것을 말한다.
배치 정규화는 각 층에서 활성화 함수를 통과하기 전에 수행된다.

순서를 요약하면 다음과 같다.

1. 입력에 대해 평균을 0으로 만들고, 정규화를 한다.
2. 정규화된 데이터에 대해서 스케일과 시프트를 수행한다.

배치 정규화의 식을 살펴보면 다음과 같습니다. BN은 배치 정규화를 의미합니다.

input: 미니 배치 $B = x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

output: $y^{(i)} = BN_{\gamma,\beta}(x^{(i)})$

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$ 👉 미니 배치에 대한 평균

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_B)^2$$ 👉 미니 배치에 대한 분산

$$\hat{x}^{(i)} \leftarrow \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$ 👉 정규화

$$y^{(i)} \leftarrow \gamma\hat{x}^{(i)} + \beta = BN_{\gamma,\beta}(x^{(i)})$$ 👉 스케일 조정과 시프트

- m은 미니 배치에 있는 샘플의 수
- μ_B는 미니 배치 B에 대한 평균.
- σ_B는 미니 배치 B에 대한 표준편차.
- $\hat{x}^{(i)}$은 평균이 0이고 정규화 된 입력 데이터.
- ε은 분모가 0이 되는 것을 막는 작은 수. 보편적으로 10^{-5}
- γ는 정규화 된 데이터에 대한 스케일 매개변수로 학습 대상
- β는 정규화 된 데이터에 대한 시프트 매개변수로 학습 대상
- $y^{(i)}$는 스케일과 시프트를 통해 조정한 BN의 최종 결과

배치 정규화의 장점은 다음과 같다.

1. 시그모이드나 하이퍼블릭탄젠트 함수를 사용해도 기울기 소실 문제가 크게 개선된다.
2. 가중치 초기화에 덜 민감해진다.
3. 훨씬 큰 학습률을 사용할 수 있어 학습 속도가 개선된다.
4. 과적합(Overfitting)을 방지하는 효과가 미미하지만 존재한다.

배치 정규화의 단점은 다음과 같다.

1. 미니 배치 크기에 의존적이다.
2. RNN에 적용하기 어렵다.

Maxpool

일반적으로 합성곱 연산 + 활성화 함수 단계를 거치고 풀링 단계를 거친다.
풀링 연산은 크게 1. 최대 풀링(max pooling), 2. 평균 풀링(average pooling)이 사용된다.

이 중 max pooling은 합성곱 연산을 마치고 나온 특성맵을 새로운 kernel로 pooling 할 때, 필터에 들어가는 값중 최대값만을 출력시켜 특성맵의 크기가 다운샘플링되게 만드는 것이다.

FC Layer

FC Layer(Fully-Connected Layer)는 CNN 마지막에서 분류(Classification)을 결정하는 단계이다.

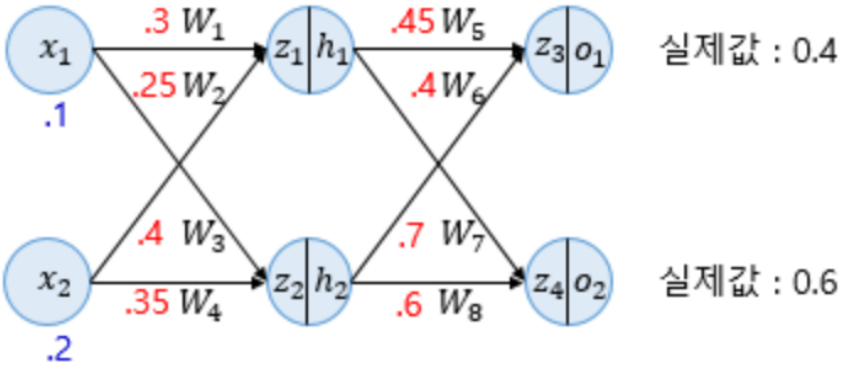
쉽게 생각해서, 모든 노드들끼리 연결이 되어있는 신경망형태라고 보면 된다.

BackPropagation

오늘 날 Artificial Neural Network를 학습시키기 위한 일반적인 알고리즘 중 하나이다. 역전파 법이 있기 전에는 그냥 순전파를 진행하는 방식 밖에 없었기에 MLP(Multi Layer Perceptron)을 학습시킬 방법이 없었지만.
역전파법이 생기면서 각 Layer의 편미분값을 구하고 연쇄법칙을 응용하여 학습을 더 빠르게 할 수 있게 되었다. 또한 MLP도 학습시킬 수 있게 되었다.

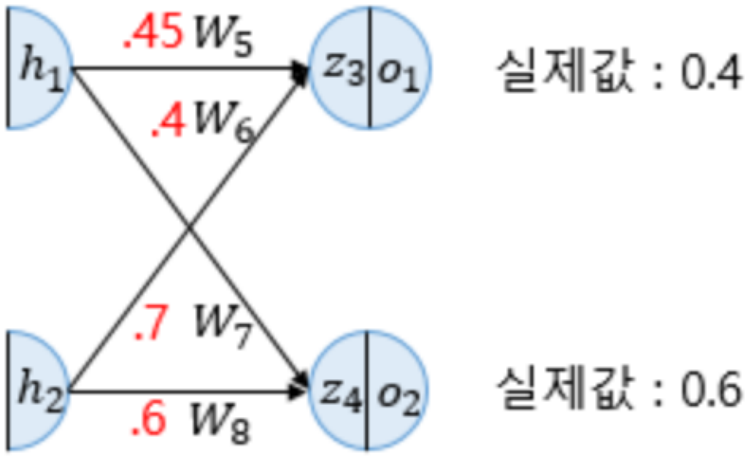
역전파법은 내가 뽑고자 하는 target 값과 실제 모델이 계산한 output이 얼마나 차이가 나는지 구한 후 그 오차값을 다시 뒤로 전파해가면서 각 노드가 가지고 있는 변수들을 갱신하는 하나의 알고리즘이다.

역전파를 간단히 소개하기 위해 아래와 같은 FC Layer가 있다고 가정해보겠다.



순전파부터 설명하면 왼쪽의 입력값 x를 시작으로 W(가중치, weight)를 곱해서 더해진 z 그리고 h는 이러한 연산값인 z를 활성화함수의 정의역으로 가지는 치역이다. 이렇게 은닉층에서 활성화 함수를 거치고 나온 값에 또 가중치들이 곱해지고 더해진 z를 마지막으로 활성화함수를 통과하고 나온 값인 도식이다.

역전파는 말 그대로 이제 이러한 연산의 값을 거꾸로 연산해나간다고 생각하면 된다. 여기서 필요한 개념은 편미분에 대한 개념이다.

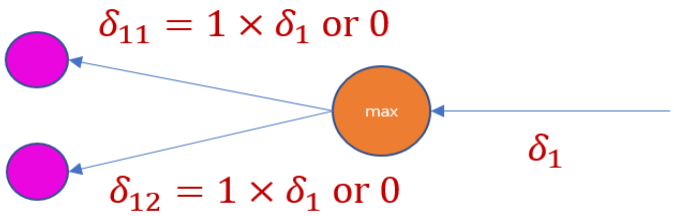


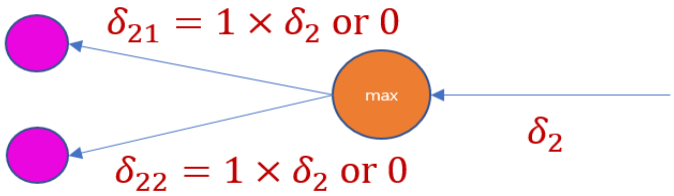
layer간의 BackPropagation을 설명하기 위해 역전파 1단계에 해당하는 위 그림만 살펴보겠다. 역전파 1단계에서 업데이트해야할 가중치는 W_5,6,7,8 이다.

이 중 W_5만 살펴보면 전체 total 값의 o1방향으로의 편미분을 구한후 합성함수의 Chain Rule에 의해 o1에 대한 Z_3방향을의 편미분을 곱하고 그리고 Z_3에 대한 W_5 방향으로의 편미분 값을 곱하면 total 결과값에 대한 W_5방향으로의

편미분 값을 구할 수 있게된다.
이러한 방식으로 역전파를 통하여 가중치들을 업데이트 할 수 있다.

그 다음 CNN에서의 BackPropagation을 살펴보겠습니다. Maxpooling에 대한 역전파법은

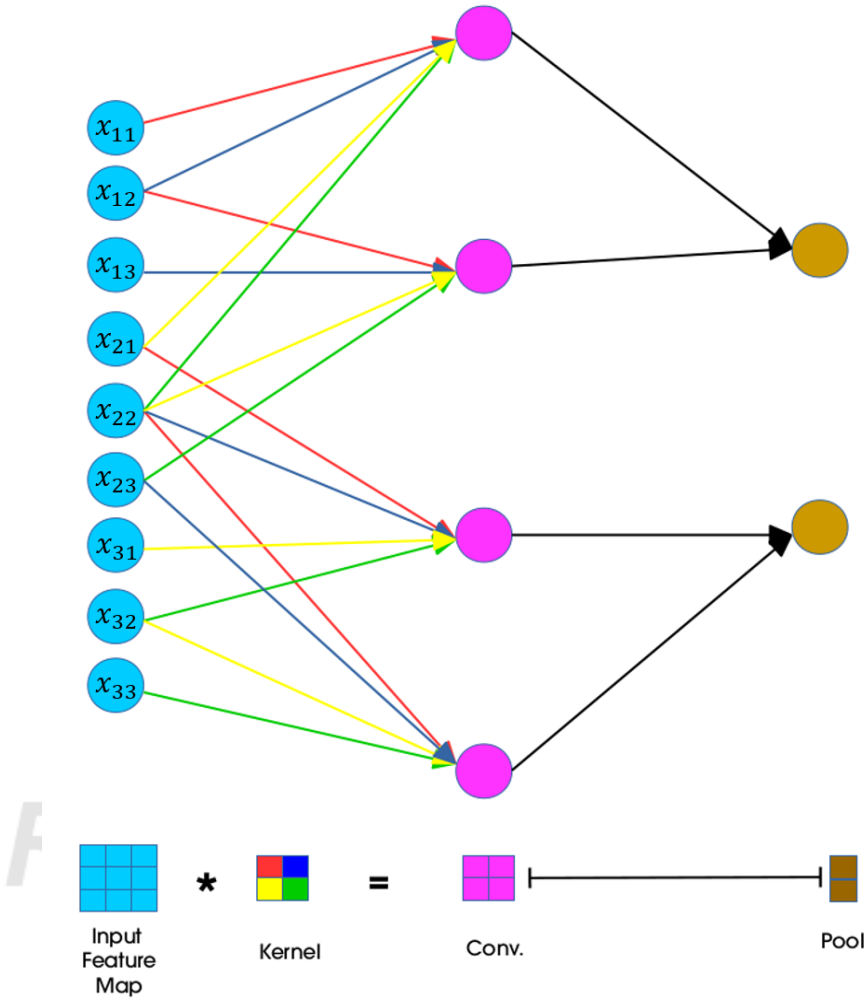




최대값이 속해 있는 요소의 로컬 Gradient는 1, 나머지는 0이기 때문에 여기에 흘러 들어온 Gradient를 곱해 구하게 된다.

그 다음 ReLU 함수의 BackPropagation에 대해서 알아보겠다. 기본적으로 ReLU함수는 입력값 x가 0보다 클때 x 자기 자신을 출력하고, 0보다 작거나 같을 때는 0을 출력하는 비선형함수이다. 이를 y에대한 x방향으로의 편미분을 계산하게되면 0보다 클 때는 1을 출력하고 0보다 작을 때는 0을 출력하게 된다.

그 다음 Convolution 층에서의 BackPropagation을 살펴 보겠다.



위 그림에서를 예시로 들면 입력값이 3 * 3 인 행렬이고, kernel size는 2*2 이다. stride를 1이라고 가정하였을 때, x_11를 살펴보겠습니다. 순전파에서는 x_11은 kernel의 W_1과 곱해지기 때문에, 역전파에서도 동일하게 한번의 연산을 거치게 된다.

Loss Function

손실 함수는 지도학습 시 알고리즘이 예측한 값과 실제 정답의 차이를 비교하기 위한 함수이다. 즉, '학습 중에 알고리즘이 얼마나 잘못 예측하는 정도'를 확인하기 위한 함수로써 최적화(Optimization)를 위해 최소화하는 것이 목적인 함수이다.

5. 참고 자료

[1] BoostCourse - 파이토치로 시작하는 딥러닝

[2] 밑바닥부터 시작하는 딥러닝 - 사이토 고키

[3] CNN의 역전파

[4] https://www.slideshare.net/yongho/ss-79607172

[5] 마스크착용유무 딥러닝 모델 : https://yuchulnote.github.io/models/mask/

[6] 데이터셋 : Covid Face Mask Detection Dataset : https://www.kaggle.com/datasets/prithwirajmitra/covid-face-mask-detection-dataset

[7] VGG 개요 : https://yuchulnote.github.io/deep_learning_study/Lab10-3/

[8] 역전파법 개요 : https://yuchulnote.github.io/deep_learning_study/Lab08-2/

[9] ADAM : A METHOD FOR STOCHASTIC OPTIMIZATION

[10] Face Mask Detection In Real-time Using Python - Suzon_Abdul Karim