Task1. YOLO + OpenCV를 이용한 embedded edge computing (NVIDIA Jetson Nano) 환경에서의 real time object detection을 구현하라.

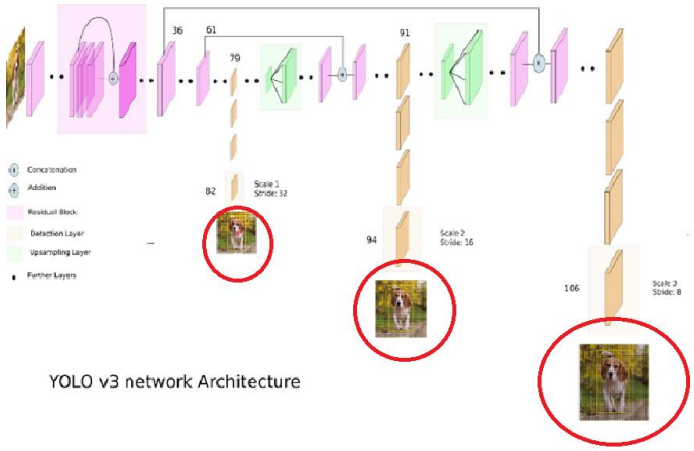
먼저 object detection이란 특정 이미지에서 bounding box를 통한 영역을 설정하고 그 영역 안에 있는 물체의 종류 및 존재 유무를 판별하는 것이다. 우리는 물체인식을 위해 yolo와 openCV를 사용한다. Yolo를 사용한 real time object detection을 구현하기 위해 먼저 yolo의 네트워크, 구조, 원리 등과 함께 OpenCV를 이용하는 방법 등을 이해하고 있어야 한다.

1-A-1 Yolo(you only look once)에 대한 소개 및 네트워크, 구조

yolo는 물체 인식을 위해 만들어진 신경망으로 bounding box를 조정하고 그 영역 내의 물체가 무엇인지 분류를 동시에 실행한다. Yolov3가 사용하는 네트워크는 Darknet53으로 그림1으로 이루어져 있으며 구조는 그림 2와 같다. Darknet53은 YOLOv2에서 사용했던 Darknet19보다 더 작은 이미지에 대한 detection을 수행할 수 있으며 더 높은 정확도를 보인다. 또한 표1에서 보이듯 ResNet-101와 ResNet-152보다 효율적이라는 것을 볼 수 있다. YOLOv3의 가장 큰 특징은 세 가지 다른 scale로 detection을 진행한다. 네트워크 속 각각 다른 3개 위치에서 3개의 다른 크기의 feature map에 1x1 커널을 사용하며 detection한다.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

(그림 1-A-1)



(그림 1-A-2)

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

(표 1-A-1)

1-A-2 간단한 yolo의 원리

Yolo가 input data로 이미지를 받으면 SxS개의 격자로 나눈다. 그후 이미지 전체를 신경망에 넣어 특징을 추출하고 prediction tensor를 생성한다. 격자 별 예측 정보를 바탕으로 bounding box를 조정하고 분류한다.

1-A-3 OpenCv

OpenCV는 컴퓨터 비전과 영상처리를 위해 만들어진 라이브러리로 인텔사에서 개발하여 무료로 배포하였다. 사용하기 쉽고 단순한 함수들을 조금만 숙지해도 쉽게 응용할 수 있다.

1-A-4 코드 구현하기

코드를 구현하기 앞서 우리가 사용하는 Jetson nano의 제한된 성능으로 YOLOv3_tf를 사용한다. YOlOv3_tf를 제외하고 가장 중요한 2개의 라이브러리는 Tensorflow, cv2이다. 따라서 설치해 주고 import하여 주피터 환경에서 실행할 수 있도록 한다. 또한 YOLOv3_tf를 통한 이미지 물체 인식을 위해 주피터 환경으로 불러와 weight와 이런 설치가 완료된 후 어떤 절차로 Realtime object detection이 진행될지 파악한다. 우리 팀이 고안한 절차는 다음과 같다.

I. webcam을 통해 실시간으로 동영상을 찍는다.

II. 동영상을 캡처해 이미지를 불러온다

III. 캡처한 이미지에 대해서 yolo가 처리할 수 있는 데이터로 전처리 해준다.

IV. 전처리한 이미지를 YOLOv3_tf에 input-data로 주어 object detection한 결과(boxes, scores, classes, num)를 받아온다.

V. 이 데이터들을 처음에 캡처했던 이미지와 합친다.

VI. 이 합친 이미지를 다시 반환하여 보여준다.

VII. 1-6과정을 무한 반복한다.

우리는 2,6과정을 OpenCv의 set(), VideoCapture(), read(), imshow(),... 등의 함수를 통해 동영상의 크기를 setting, 캡처, 불러오기, 화면으로 보여주기 등의 기능을 사용하여 진행하였다. 3 과정은 OpenCv의 cvtColor() 함수를 이용해 BGR순서로 되어있는 데이터를 RGB순서로 바꿔주고 Tensorflow를 사용해 차원을 변경하고 크기를 리사이즈 하여 Yolov3_tf가 처리할 수 있는 형태로 바꿔주었다. 4,5번과정은 앞선 과정에서 가공된 data를 Yolov3_tf의 함수들을 이용해 이미지를 예측하고 또 합쳐 주었다. 위와 같은 과정으로 이루어진 Real time object detection은 성공적이었다.

(그림 1-A-3)

1-B Hyperparameter에 따른 object detection

우리는 webcam에서 불러오는 동영상의 해상도와 YOLO를 통해 예측하기전 리사이즈 하는 크기 그리고 마지막으로 fps를 바꿔 보기로 했다.

우리는 매개변수를 바꾸고 실행시켜보는 과정에서 잦은 오류로 인한 재부팅과 class 중첩문제가 빈번하게 일어났다. 따라서 조금 더 효율적으로 매개변수들과 물체인식의 상관관계를 파악하기 위해 컴퓨터에 anaconda python을 설치하고 pycharm이라는 edit tool을 이용해 3개의 매개변수를 바꿔가며 비교했다.

3개의 파라미터를 바꿔가며 비교해본 detection accuracy 결과는 아래의 표 2와 같다.

해상도	320*240		640*480	
Size / Fps	160	320	160	320
24	71.07%	54.40%	76.58%	37.18%
40	69.86%	60.67%	77.93%	47.14%

(표 1-B-2)

분석 결과는 다음과 같다. 같은 해상도와 프레임 조건에서 이미지를 리사이즈를 할 때 더 작은 160으로 리사이즈 한다면 훨씬 더 높은 detection accuracy를 갖는 것을 확인할 수 있었다. 또한 처음 불러오는 해상도의 크기가 클수록 좋은 detection accuracy를 갖는 것을 알 수 있었다.

하지만 Fps에 따른 detection accuracy는 특정 경향을 찾아보기 어려웠다.

1-1-1 Fps과 detection accuracy에 대한 고찰

Fps와 detection accuracy과의 관계가 궁금하여 우리가 구현했던 코드들을 하나하나 따라가며 추적해봤다. 먼저 fps는 정의된 이후 while문(동영상을 캡처해 detection하고 합성하여 다시 보여주는 과정)의 마지막 부분의 딜레이 값을 결정하기 위한 변수였다. 즉, fps는 오직 delay의 값 만을 결정하기 위한 변수이므로 실질적으로 object detection을 수행하는 YOLOv3_tf 모델에는 전혀 영향을 주지 않았다. 따라서 Fps와 detection accuracy의 상관관계가 없기 때문에 FPS에 따른 detection accuracy의 경향성을 볼 수 없던 것이다.

Fps에 대한 추가적인 고찰 - fps를 24라고 설정한다면 정말로 object detection 하여 보여주는 실시간 동영상이 정말로 24fps일까? 정답은 24fps가 아니다. 왜냐하면 실제로 object detection한 동영상의 fps는 1초 동안 while문이 반복되는 횟수와 같다. 위의 문단에서 fps는 유일하게 while문 마지막에 delay를 결정하기 위한 변수였다고 설명했다. 즉 while문이 끝나갈 때 추가로 어느 정도의 시간을 delay시킬지를 결정한다. 하지만 우리는 Yolo가 object detection한 시간도 계산해야 한다. 식을 통해 살펴본다면 설정한 Fps는 아래의 식1과 같고 실제 Fps는 식2와 같다

$$Fps = \frac{1s}{딜레이}$$ ... (식 1-B-1, 설정한 Fps)

$$Fps = \frac{1}{object\,dection에\,소요된\,시간 + 합성\,및\,남은\,작업\,소요\,시간 + 딜레이}$$ ...(식 1-B-2, 실제 Fps)

따라서 실제 FPS가 설정한 Fps보다 작을 수밖에 없고 그것이 30Fps만을 지원하는 webcam에서 40Fps를 설정하더라도 문제가 없는 이유이다.

느낀 점

이 프로젝트는 어려운 분야라고 생각해왔던 딥 러닝을 조금 더 쉽고 재미있게 공부할 수 있는 좋은 기회였다. 또한 새로운 우리만의 학습 네트워크를 만들어 보기도 하고 기존 잘 만들어진 네트워크를 다루기도 하며 학습하는 과정의 구조와 원리를 제대로 이해할 수 있게 되었다. 다루는 과정속에서 여러 문제가 발생했고 이를 해결하기 위한 새로운 방법들을 찾고 배워가며 조금 더 깊은 이해를 할 수 있게 되었다. 하지만 이와 함께 좋은 성능을 갖는 네트워크를 만드는 것은 매우 복잡하고 어려우며 특정 한계점들이 있다는 것을 알 수 있었다.

진행과정

Jetson nano와 webcam 및 실습에 필요한 물건들을 수령한후 바로 Term-project는 시작되었다. 법학도서관 스터디룸을 빌려 프로젝트의 시작을 열려고 했으나 인터넷선을 연결할 수 없어 다른 장소를 물색하게 되었다. 하지만 프로젝트를 시행하기에 마땅한 공간을 찾는 것은 매우 어려운 일이었다. 우리는 보드게임 카페의 방을 잠시 빌려 그곳에서 프로젝트를 시작했고 그날 Jetson 과 Tensorflow, opencv등의 설치를 했고 Task1-A과 Task2, 3한

Task 2-A Colab 환경에서 CNN from scratch (각 팀에서 설계한) 모델을 CIFAR-100 data에 대하여 training 시켜라. 단, 총 파라미터의 수가 3천만개를 넘지 않게 설계할 것.

시작하기 앞서 합성곱 신경망(CNN: Convolutional Neural Network)이란 심층 신경망(DNN: Deep Neural Network)의 한 종류로 하나 이상의 컨볼루션 계층과 통합 계층, 완전하게 연결된 계층으로 구성된 신경망을 뜻한다. 그 구조는 다음과 같다.

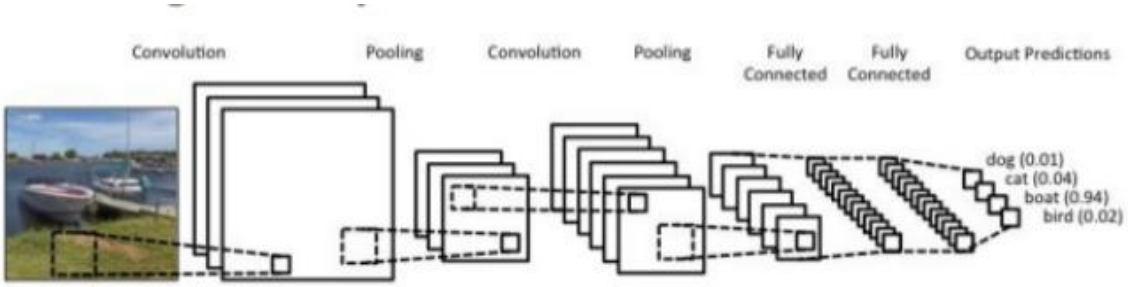


그림 2-1

CNN은 2차원 데이터의 학습에 적합한 구조를 가지고 있으며, 역전달을 통해 훈련될 수 있다.

Cifar-100 dataset은 100개의 분류를 가진 60000개의 이미지 데이터 셋이다. 이 중 50000개는 학습을 위한 데이터 셋이며 나머지 10000개는 평가를 위한 데이터 셋이다.

2-1: 모델 만들기

2-1-1 기본지식 학습

우리는 모델을 만들기 위해 실제로 사용되는 모델에서 자주 등장하는 계층을 분류하고 그 계층이 어떤 작용을 하는지 살펴보기로 했다. 그 결과 다음과 같은 기구나 계층이 많이 등장한다는 것을 알 수 있었다.

모델명	사용된 기구
EffNet-L2	Avg pooling, batch normalization, max pooling, stride
BiT-L	Avg pooling, batch normalization, learning schedule, max pooling, relu, sgd+momentum
VGG-19	Adam, batch normalization, drop out, max pooling
EffNet-B7	Avg pooling, batch normalization, change lr, max pooling, stride
Tresnet-Xl	Avg pooling, batch normalization, max pooling
Gpipe	Adam, avg pooling, drop out, max pooling, relu

표 2-1

다음 기능 또는 계층이 하는 역할은 다음과 같다.

Adam: 1차 및 2차 모멘트의 적응적 추정을 기반으로 하는 확률적 경사하강법, 직관적으로는 모멘텀과 AdaGrad를 융합한 방식이다. SGD에 비해 느리지만 해를 찾는 과정에서 발산할 가능성이 작다. 아래와 같은 수식으로 표현 가능하다. 아래 식에서 베타1은 모멘텀이 감소하는 비율, 베타2는 모멘텀이 감소하는 기간을 나타내는 상수이다.

$$a_{n+1} := \beta_1 \cdot a_n + (1-\beta_1) \cdot \nabla f(x_n), \quad a_0 := 0$$
$$b_{n+1} := \beta_2 \cdot b_n + (1-\beta_2) \cdot \nabla f(x_n) \odot \nabla f(x_n), \quad b_0 := 0$$
$$x_{n+1} := x_n - \frac{\widehat{\alpha}_{n+1}}{\sqrt{b_{n+1}} + \varepsilon} \odot a_{n+1}$$

Average, Max pooling: 풀링은 가로 세로 방향의 공간을 줄이는 연산으로 n*n영역의 원소를 하나로 집약하여 공간의 크기를 줄인다. 이때 원소의 수가 줄어들기 때문에 오버피팅을 방지하는데 도움을 준다. Average pooling은 영역 안의 원소의 평균을 꺼낸다. 이로 인해 전 계층의 자극이 줄어들어 깊은 모델에서는 성능 저하의 문제로 이어질 수 있다. Max pooling은 영역 안의 원소 중 최대값을 꺼낸다. 이로 인해 전 계층의 자극이 강조되기 때문에 성능저하의 문제는 없지만 오버피팅을 방지하는 정도가 Avg pooling보다 못하다.

Change lr (learning rate schedule): 학습률(learning rate)는 경사하강법 알고리즘에서 보폭에 해당한다. 학습률이 너무 작은 경우 최저점까지 도달하는데 걸리는 시간이 너무 길고 학습률이 너무 큰 경우 곡선의 최저점을 무질서하게 이탈할 우려가 있다. 이 상충 상황을 해결하기 위해 학습 초기에는 큰 학습률, 후기에는 작은 학습률을 적용하는 방법을 사용할 수 있다. 이것이 learning rate schedule이다.

Drop out: drop out은 뉴런을 임의로 삭제하면서 학습시키는 방법이다. Drop out 비율을 정해주면 은닉층에서 정해진 비율만큼 뉴런을 삭제하고 학습시킨다. 시험 때는 모든 뉴런에 신호를 전달한다. 이를 통해 오버피팅 문제를 완화할 수 있고 계산량이 줄어들기 때문에 학습속도가 빨라지는 효과를 볼 수 있다.

Sgd with momentum: sgd는 가장 기본적인 학률적 경사 하강법으로 랜덤하게 추출한 일부 데이터에 대해 가중치를 조절한다. 그로 인해 속도는 빠르지만 정확도가 낮다는 단점을 가지고 있다. 다음과 같이 표현된다.

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

이를 개선하기 위해 모멘텀을 적용할 수 있다. 모멘텀은 문자 그대로 관성이라는 의미를 갖고 있다. 모멘텀을 더하면 가중치를 수정하기 전 이전 수정방향을 참고하여 같은 방향으로 일정한 비율만 수정하게 된다. 이로 인해 지그재그 현상이 좀 줄어들고 정확도가 상승한다. 그 외에도 기울기가 0인 안장점에서 탈출하도록 하는 효과가 있다. 수식으로 표현하면 다음과 같다.

$$\mathrm{V(t)} = m * V(t-1) - \alpha \frac{\partial}{\partial w} Cost(w)$$
$$\mathrm{W(t+1)} = W(t) + V(t)$$

Batch normalization: 배치 정규화는 가중치의 초기값을 설정하는데 있어 활성화 값의 분포를 표준 정교 분포로 강제하는 것이다. 이로 인해 학습 속도 개선, 오버 피팅 문제를 해결할 수 있다.

2-1-2 모델구현

위와 같은 학습 과정을 거친 후 우리는 3가지 모델을 만들기로 하였다. 각각 학습 속도, 깊이를 중시한 두 모델과 이 둘을 적절히 조화해 최적의 성능을 낼 수 있는 모델을 고안해 보았다. 다음은 우리가 최종적으로 고안한 세번째 모델의 구조이다.

모든 conv2d층 다음에는 배치정규화, 활성함수 relu, dropout 층이 추가되어 있다.

모든 dropout은 0.2의 비율을 갖고 진행되었다.

처음과 두번째 conv2d층에는 초기 난수를 정규분포의 초기값으로 설정하는 커널 초기화로 glorot_normal을 나머지 conv2d층에서는 정규분포에 따라 텐서를 생성하는 커널초기화로 Randomnormal(stddev=0.01)을 사용하였다.

Dense 층에서는 softmax를 사용하였다.

Layer (type)	Output Shape	Param #
conv2d (64,3,3)	(None, 32, 32, 64)	1792
max_pooling2d (3,3)	(None, 30, 30, 64)	0
dropout_1 (Dropout)	(None, 30, 30, 64)	0
conv2d_1 (128,3,3)	(None, 30, 30, 128)	73856
average_pooling2d (3,3)	(None, 28, 28, 128)	0
dropout_3 (Dropout)	(None, 28, 28, 128)	0
conv2d_2 (256,3,3)	(None, 14, 14, 256)	295168
conv2d_3 (256,3,3)	(None, 7, 7, 256)	590080
max_pooling2d_1 (3,3)	(None, 5, 5, 256)	0
dropout_6 (Dropout)	(None, 5, 5, 256)	0
conv2d_4 (512,3,3)	(None, 5, 5, 512)	1180160
conv2d_5 (512,3,3)	(None, 5, 5, 512)	2359808
average_pooling2d_1 (3,3)	(None, 3, 3, 512)	0
dropout_9 (Dropout)	(None, 3, 3, 512)	0
conv2d_6 (512,3,3)	(None, 3, 3, 512)	2359808
conv2d_7 (512,3,3)	(None, 3, 3, 512)	2359808
max_pooling2d_2 (3,3)	(None, 1, 1, 512)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 100)	51300

Total params: 9,282,788
Trainable params: 9,277,284
Non-trainable params: 5,504

그림 2-2

2-2: 하이퍼 파라미터 최적

우리는 튜닝할 하이퍼 파라미터로 시작 학습률, 모멘텀, 배치 사이즈, validation split을 선정하였다. 이 중 학습률은 Grid search 방법을 모멘텀, 배치 사이즈, validation split은 Random search방법을 적용하기로 하였다. Random search의 경우 다음 식을 근거하여 상위 5%의 성능을 발휘하는 하이퍼 파라미터 집합을 찾기 위해 60번 반복하였다.

$$1-(1-0.05)^{60} \approx 0.95$$

보통 최적의 하이퍼 파라미터를 찾을 땐 작동시간을 줄이기 위해 random search를 사용하지만 학습률의 경우 옵티마이저의 수식을 고려해보면 0.0005의 배수처럼 딱 떨어지는 숫자를 사용하는 것이 바람직하다 판단해 grid search를 사용하였다.

다음 표는 모델 3을 대상으로 Grid search로 찾아낸 최적의 학습률 0.005를 적용하여 epoch 수를 10으로 설정하고 Random search를 한 결과의 일부이다.

batch size	validation split	momentum	top 1 error
962	0.1352	0.97256	19.17%
44	0.2142	0.98262	17.72%
530	0.1186	0.96966	17.41%
734	0.2057	0.91524	15.78%
728	0.2929	0.95069	12.87%
903	0.1633	0.91372	12.69%
873	0.108	0.94731	11.72%
858	0.2384	0.99002	7.05%
358	0.147	0.99556	6.49%
972	0.2827	0.9187	6.01%

표 2-2

60번을 반복수행한 결과 가장 높은 top 1 정확도는 19.17로 이 때의 하이퍼 파라미터를 이용해 학습을 진행하였다.

2-3: 학습 진행 및 평가

Sgd, momentum, learning schedule을 적용하여 학습을 진행하였다. 이때 사용한 learning schedule의 설정은 boundaries=[10000,25000,40000], values=[lr,lr*0.1,lr*0.01,lr*0.001] 이다. 총 300 epoch의 학습을 진행하였고 세 모델의 결과는 다음과 같다.

	Top 1 error	Top 5 error	Time/epoch
1st model	42.42%	72.67%	5.103s
2nd model	43.04%	69.06%	57.425s
Final model	57.14%	82.13%	32.753s

표 2-3

Task 2-B. Task 2-A에서 training한 모델을 사용하여 CIFAR-100 data를 Jetson Nano에서 evaluate 하라.

위의 학습시킨 모델을 Jetson Nano 환경에서 evaluate 해보았다. 아래 사진은 그 결과이다.

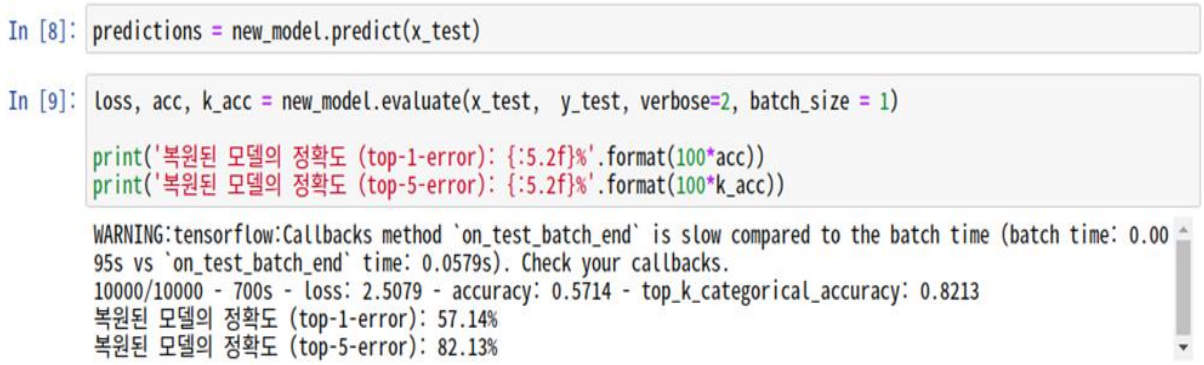


그림 2-3

이 때 정확도는 pc환경과 동일 했지만 evaluate time은 700s로 pc환경보다 훨씬 오래걸렸다.

Task3-A. Colab 환경에서 image classification을 위한 네트워크모델 중에서 기존의 잘 알려진 3개를 이용하여 transfer learning을 진행하라.

3-A-1. Transfer learning

일반적으로 딥러닝 모델들을 제대로 훈련시키려면 많은 양의 데이터와 시간, 돈이 필요하다. 그렇기에 사용목적에 맞는 데이터를 일일이 수집하여 목적에 적합한 모델을 학습시키기란 매우 어려운 일이다. 그래서 나온 개념이 전이학습(transfer learning)이다. 전이학습이란, 해결하고자 하는 문제에는 정답이 소량만 존재하는 반면, 해결하고자 하는 문제와 비슷한 문제에는 정답이 대량으로 존재할 경우 사용하는 방법으로, ImageNet이 제공하는 거대한 데이터셋을 학습한 가중치 값들을 실제 사용 목적에 효과적으로 활용하는 방식이다. 여기서 ImageNet이란 전 세계 연구자들이 쉽게 접근할 수 있는 WordNet(동의어) 계층 구조에 따라 구성된 이미지 dataset이다. WordNet에는 100,000개 이상의 class들이 존재하며, 각각 class에는 평균 1000개의 이미지를 가지고 있다. 다시 전이학습으로 돌아와서, 전이학습의 방법에는 3가지가 존재하는데, 모델을 그대로 사용하는 방법, 모델의 파라미터를 새롭게 바꾼 후 학습시키는 방법, 그리고 모델을 수정하고 다시 학습시키는 방법이 있다. 우리 E팀은 transfer learning을 진행하기 위해서, image classification을 위한 네트워크 모델 중, InceptionResnetV2, VGG16, Resnet50V2를 선정하였다.

3-A-2. CIFAR-100 dataset 소개

CIFAR-100 dataset은 CIFAR-10 dataset과 함께 Alex Krizhevsky, Vinod Nair, Geoffery Hinton에 의해서 수집된 8천만개의 작은 image data set의 label이 지정된 부분집합이다. CIFAR-100은 32x32 크기의 컬러 image로 구성되어 있다. CIFAR-100 dataset은 dataset의 크기와 class의 개수를 제외하면 CIFAR-10 dataset 동일하다. CIFAR-100 dataset은 하나의 class당 500개의 학습 image와 100개의 test image를 가지고 있다. 또한 CIFAR-100 dataset은 앞서 말한 학습, test image를 분류하는 포함한 class를 100개 가지고 있는데, 이 100개의 class를 포괄하는 super class가 있다. (그림3-A-1) CIFAR-10 dataset의 부분집합을 (그림3-A-2)는 CIFAR-100 dataset의 class 분류를 나타낸 것이다.

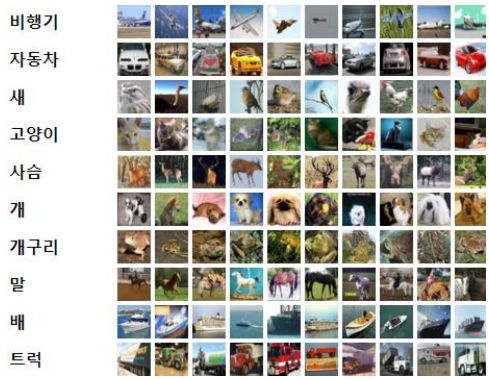


그림 3-A-1

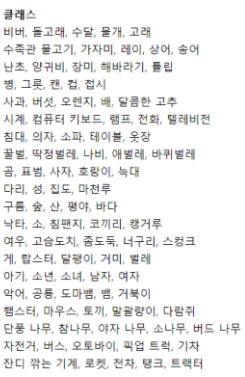


그림 3-A-2

3-A-3. InceptionResnetV2 소개

InceptionResnet이란 inception 네트워크에 Residual Connection을 적용한 네트워크를 적용한 네트워크를 말한다. 먼저 Inception에 대한 설명이다. 이전 모델들은 네트워크가 더 많은 레이어를 가질수록 성능이 향상되는 것을 중점으로 하여, 네트워크가 깊어질수록 학습해야 하는 파라미터의 수가 늘어나 convolution을 실행할 때마다 많은 양의 층이 쌓여 연산에 강한 부담을 주게 된다. 또한 많은 양의 hyperparameter를 설정하는데 부담이 걸릴 수밖에 없었다. 하지만 (그림3-A-3)과 같이 기존의 한 층에 convolution을 한 번 실행한 방식을 한 층에 여러 번 convolution하여 비선형적인 관계를 더욱 잘 표현하게 되었다. 이 방법을 통해서 Inception이라는 이름을 얻게 되었다. 다음으로는 Residual Connection에 대한 설명이다. Residual connection이란, 2015년에 등장한 Resnet 네트워크에서 등장한 개념으로 (그림3-A-4) output image에 input image를 합하여 값을 반환하는 과정을 말한다.

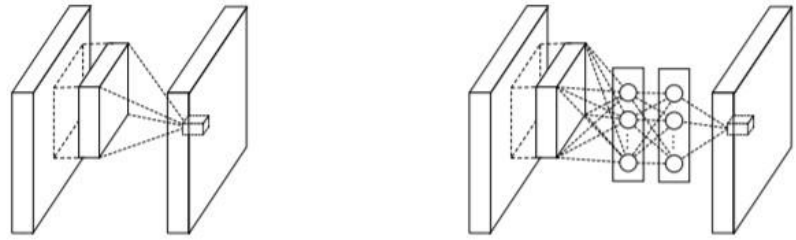


그림 3-A-3

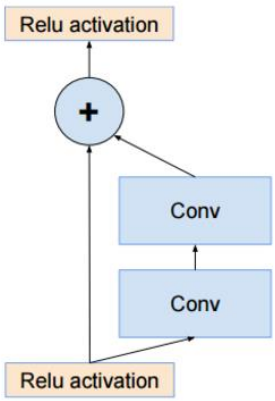


그림 3-A-4

3-A-4. InceptionResnetV2의 transfer learning

우리는 InceptionResnetV2 모델이 image를 분류할 수 있도록 input shape (32,32,3)을 (150,150,3)으로 resize 해주는 layer를 추가하고, VGG16 모델과 flatten layer와 activation이 softmax인 prediction layer을 하나의 모델로 묶어 transfer learning 실행하였다. 또한 (표3-A-1)와 같은 설정

을 통해서 InceptionResnetV2 모델의 transfer learning을 진행하였다

Opimizer	Adam	**손실 함수**	Cross entropy
Batchsize	32	**Callback 함수**	modelcheckpoint
Epcoch의 수	20	**고정 layer**	100
Validation split	0.2	**Trainable parameter**	1,382,500

표 3-A-1

3-A-5. VGG16 소개

VGG 네트워크 모델은 네트워크의 깊이를 깊게 만드는 것이 성능에 어떠한 영향을 미치는지를 확인하고자 만들어진 모델이다. 그래서 VGG 네트워크 모델 개발진들은 깊이의 영향만을 중점적으로 볼 수 있도록 convolution filter의 사이즈 중 가장 작은 3x3로 고정하였다. VGG 네트워크 개발진들은 11층, 13층, 16층, 19층으로 깊어지는 과정에서 분류 오류가 감소하여, 성능이 좋아진다는 것을 발견하였다. 그 중 VGG16은 (그림3-A-5)과 같이 층이 16개인 VGG 네트워크 모델을 의미한다.

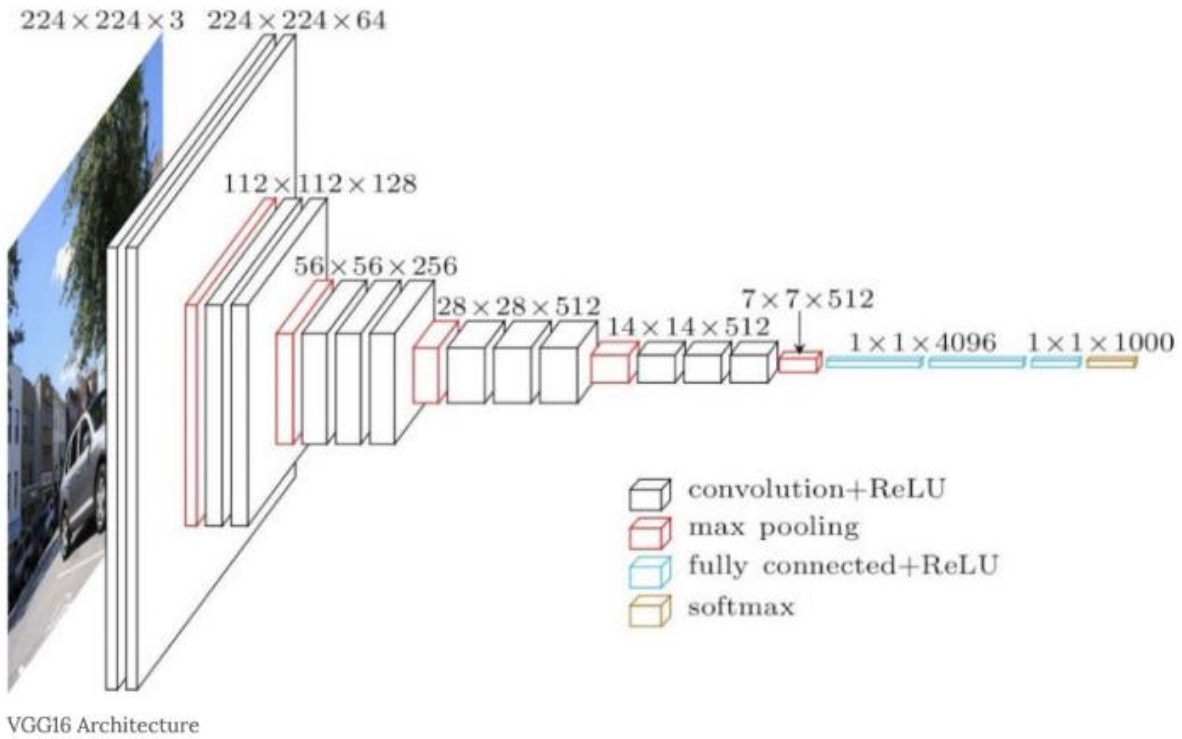


그림 3-A-5

3-A-6. VGG16의 transfer learning

우리는 VGG16 모델이 image를 분류할 수 있도록 input shape (32,32,3)을 (224,224,3)로 resize 해주는 layer를 추가하고, VGG16 모델과 flatten layer와 activation이 softmax인 prediction layer을 하나의 모델로 묶어 transfer learning 실행하였다. 또한 표와 같은 설정을 통해서 VGG16 모델의 transfer learning을 진행하였다.

Opimizer	Adam	**손실 함수**	Cross entropy
Batchsize	32	**Callback 함수**	modelcheckpoint
Epcoch의 수	20	**고정 layer**	10
Validation split	0.2	**Trainable parameter**	16,078,180

표 3-A-2

3-A-7. Resnet50V2의 소개

모델의 신경망이 깊어질수록 역전파 과정에서 입력층으로 갈수록 기울기가 점차적으로 작아지는 현상이 발생한다. 이로 인해 가중치들이 제대로 계산되지 않아 결국 최적의 모델을 찾을 수 없게 되는 것을 gradient vanishing이라고 한다. 반대로 기울기가 점차 커져 가중치들이 비정상으로 큰 값을 갖게 되는 경우를 gradient exploding(기울기 폭주)이라고 한다. Resnet50V2는 인공신경망이 깊어질수록 생기는 gradient vanishing, exploding의 문제의 발생 증가를 해결하기 위해서, 앞서 3-A-1에서 언급한 Residual connection을 이용한 방법인 Resnet 네트워크 모델을 도입하였다. 그 중에서 layer가 50개인 Resnet을 Resnet50이라고 하며, Resnet50V2는 (그림3-A-6)과 같이 Residual connection 과정에서 배열의 순서가 다르다.

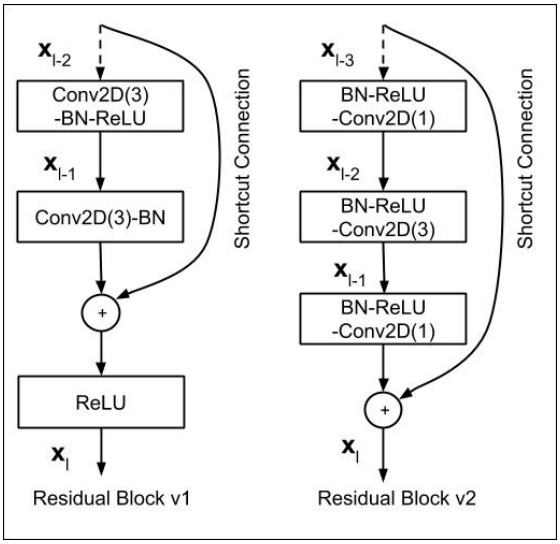


그림 3-A-6

3-A-8. Resnet50V2의 transfer learning

우리는 Resnet50V2 모델이 image를 분류할 수 있도록 input shape (32,32,3)을 (224,224,3)로 resize 해주는 layer를 추가하고, VGG16 모델과 flatten layer와 activation이 softmax인 prediction layer을 하나의 모델로 묶어 transfer learning 실행하였다. 또한 표와 같은 설정을 통해서 Resnet50V2 모델의 transfer learning을 진행하였다.

Opimizer	Adam	**손실 함수**	Cross entropy
Batchsize	32	**Callback 함수**	modelcheckpoint
Epcoch의 수	20	**고정 layer**	100
Validation split	0.2	**Trainable parameter**	30,595,172

표 3-A-3

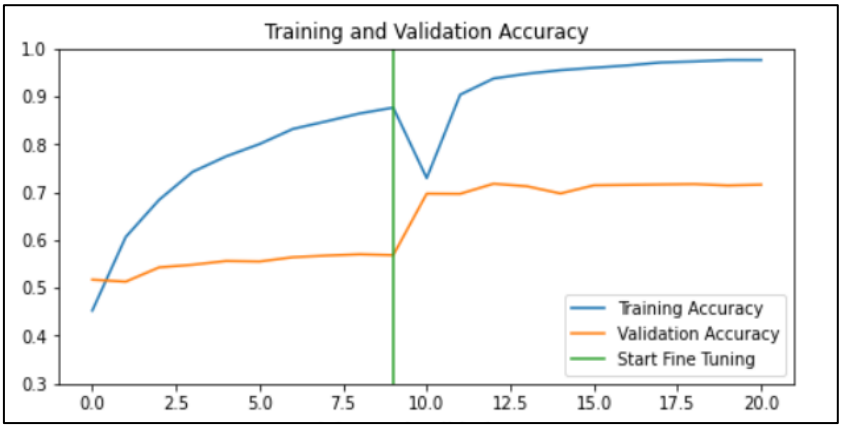
Task3-B. Colab 환경에서 Task 3-A에서 만들어진 3개의 모델에 대해 evaluate

3-B-1. InceptionResnetV2의 transfer learning 모델 evaluate

InceptionResnetV2의 transfer learning 모델을 evaluate한 결과는 다음 (표3-B-1)와 (그래프3-B-1)와 같다.

Total training Time	**Evaluate Time**	**Top1 정확도**	**Top1 정확도**
95minutes	22s	71.54%	91.50%

표 3-B-1



그래프 3-B-1

3-B-2. Vgg16의 transfer learning 모델 evaluate

Vgg16의 transfer learning 모델을 evaluate한 결과는 다음 (표3-B-2)와 (그래프3-B-2)와 같다.

Total training Time	Evaluate Time	Top1 정확도	Top5 정확도
49minutes	40s	48.43%	75.24%

표 3-B-2

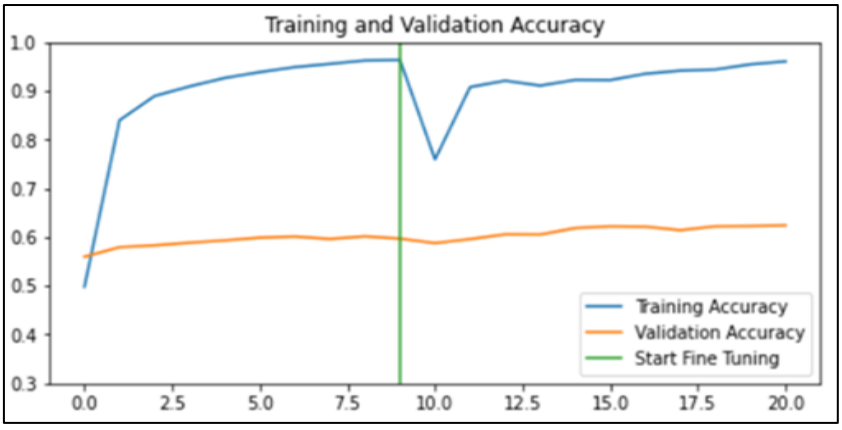
그래프 3-B-2

3-B-3. Resnet50V2의 transfer learning 모델 evaluate

Resnet50V2의 transfer learning 모델을 evaluate한 결과는 다음 (표3-B-3)와 (그래프3-B-3)와 같다.

Total training Time	Evaluate Time	Top1 정확도	Top5 정확도
44minutes	15s	62.39%	85.84%

표 3-B-3



그래프 3-B-3

Task 3-C. Task 3-B의 evaluate 결과와 Task2-B의 evaluate 결과를 비교 분석하라.

	Final Model	InceptionResNetV2	ResNet50V2	Vgg16
Top1 Error	57.14%	71.54%	62.39%	48.43%
Top5 Error	82.13%	91.50%	85.84%	75.24%
1 Epoch time	32.753s	287s	132s	148s

transfer learning한 모델들의 결과값과 task2에서 결과를 비교한 값은 다음과 같습니다. 우리의 모델이 정확도 측면에서 Vgg16보다 높고 InceptionResNetV2와 ResNet50V2보다는 낮은 것을 확인하실 수 있을 겁니다. 직접 제작한 모델임에도 불구하고, 타 모델들과 어느정도 비슷한 수준의 정확도를 보여주고 있습니다. InceptionResnet과 ResNet이

Task 3-D. Task 3-A에서 만든 모델을 최상의 accuracy를 갖도록 hyper parameter를 optimal tuning 하라.

3-D-1 모델 선택 이유

task A에서 우리는 ResNet50V2, Vgg16, InceptionResNetV2 3가지를 사용했다. Vgg16, inceptionresnetv2는 아래에 그래프에서 보시다시피 fine tuning을 10 epoch fine, tuning 10 epoch을 하니 아래처럼 validation 그래프가 진동을 하는 모습을 보였다. 그러나 ResNet50V2 는 비교적 진동없이 그래프가 형성되었기 때문에 이 그래프를 고르기로 했다.

Vgg16 - training, validation 정확도 그래프

inceptionresnetv2 - training, validation 정확도

그래프 3-D-1

3-D-2 파라미터 변경

3-D-2.1 Optimizer 조정

1. Optimizer

- 최적화

  - 신경망 학습의 목적은 손실함수의 값을 가능한 낮추는 매개변수 즉 가중치와 편향을 찾는 것

- 경사 하강법 (Gradient Descent)

  -공식 : $x_{n+1} = x_n \times r$ (r : learning rate)

  -위 과정을 반복하여 매개변수의 최적의 값을 찾아냄

- Gradient Descent 계보

3-D-2 gradient decent 계보

1. Stochastic Gradient Descent(SGD) : 모든 데이터(batch)의 loss function을 계산하는 것이 아니라 일부 데이터(mini-batch)의 loss function을 계산한다.

   장점

   ① 계산 속도가 매우 빠르다.

   ② GD와 다르게 local minima에 빠지지 않고 더 좋은 방향으로 수렴할 수 있다.

2. SGD+Momentum : SGD로 계산을 하되, 계산한 것에 관성을 주는 것이다. 스텝을 계산해서 움직인 후 그 방향대로 가중치를 추가한 다음 error를 수정한다. 자세한 건 2-1-1에 기본지식에서 참고하자.

3. Adam(adaptive momentum estimation) : 최소값에 가까울 때 동일한 learning rate를 사용하면 최소값 주변에서 진동 할 수 있다. 그래서 한 스탭 갈때마다 learning rate를 감소하는 방법인 RMSProp 방식과 Momentum 방식을 둘 다 사용하는 방법이다. SGD에 비해 느리지만 해를 찾는 과정에서 발산할 가능성이 작다. 일반적으로 가장 자주 사용되는 optimizer이다.

   주요 파라미터

   - x_0 : 초기 learning rate
   - B_1 : 모멘텀 감소하는 비율 (default : 0.9)
   - B_2 : 모멘텀 감소하는 기간 (default : 0.99)

4. Nadam : Adam방식에서 Momentum 대신 NAG를 사용하는 방식이다. NAG는 Momentum에서 가중치를 추가하는 것 대신의 일부만 update하고 그 지점에서 gradient를 계산한다. Momentum보다 속도가 더 빠르다.

2. 기본 파라미터

ResNet50V2에서 기본 parameter를 이미지 크기는 (224,224,3), optimizer은 adam, batch size는 32, epoch은 10번, validation split은 0.2, callback 함수는 modelcheckpoint로 해 놓았다. 그리고 imagenet 데이터를 가지고 학습된 weight 값이 CIFAR 100에 100% 맞다고 생각들지 않아서 layer

수를 절반으로 줄이고 학습을 진행했다.

3. Optimizer 변경

우리는 어떤 optimizer를 사용할지 고민했다. 일단 가장 일반적으로 사용되는 adam을 사용했다. 그리고 가장 기본인 sgd를 사용했는데 이때 그냥 sgd만 사용한 것이 아니라 momentum을 추가하여 사용했다. 왜냐하면 그냥 sgd는 해를 찾을 때 모든 방향을 다 고민하기 때문에 해를 찾는 과정에서 진동을 많이 하게 된다. 그래서 momentum을 넣어서 진동을 감소하고 매끄럽게 최적의 값을 찾게 해준다. 일단은 momentum은 0.9로 정해놓고 학습을 시켜보았다.

	Sgd+momentum	adam
Top1 정확도	63.39%	62.89%
Top5 정확도	91.72%	85.84%

표 3-D-2-

위 표는 sgd+momentum과 adam에 test data에 top1 정확도와 top5 error의 정확도를 보여주는 그래프이다. 보시다시피 sgd+momentum 기법이 top1 정확도는 1%가량 top5 정확도에서는 6% 가량 높은 것을 알 수 있었다. 그래서 우리는 sgd 기법이 adam 보다 ResNet50V2에서 더 높은 정확도를 가지고 있다고 생각하여 sgd+momentum 기법에서 momentum의 최적화를 찾으려고 해 보았다.

3-D-2.2 Momentum 조정

1. 기본 파라미터

ResNet50V2에서 기본 parameter를 이미지 크기는 (224,224,3), optimizer은 sgd, batch size는 64, epoch은 10번, validation split은 0.2, callback 함수는 modelcheckpoint, imagenet layer를 0.5배로 해 놓고 최적의 momentum을 찾으려고 한다.

2. Momentum 변경

Momentum 09 의 정확도가 굉장히 높아서 모멘텀을 낮춰가면서 학습을 해 보았다. 그런데 0.8 에서 매우 높은 정확도를 보여서 점차 낮췄다. 그래서 Momentum 을 0.7, 0.5, 0.3, 0.1 로 점차 내리면서 정확도를 파악했다.

	0.1	0.3	0.5	0.7	0.9
Top1 정확도	75.98%	76.32%	76.77%	76.58%	72.64%
Top5 정확도	94.75%	95.11%	95.43%	94.85%	92.9%

표 3-D-2.2

위 표에서 보시다시피 momentum 이 0.5 일 때 가장 높은 정확도를 보여주었다. Momentum 이 너무 작거나 너무 크면 정확도가 감소하는 현상을 볼 수 있었다. 그래서 우리는 최적의 Momentum 을 0.5 라고 두고 batch size 를 조절하려고 했다.

3-D-2.3 Batch size 조정

1. **Epoch, Batch 정의**

데이터 수가 너무 많으면 모든 데이터를 다 다루기에는 계산 시간이 부족하고 메모리도 부족하다. 그래서 최적화를 할 때 데이터를 나눠서 여러 번 학습을 진행한다.

- Epoch
  - 전체 데이터셋에 대해 한번 학습을 완료한 상태로 forward pass, backward pass 를 한번씩 한 상태이다.
  - Forward pass : input layer 부터 output layer 까지 각 계층의 weight 를 계산하는 과정
  - Backward pass : Forward pass 와 반대 방향인 input layer 부터 output layer 까지 weight 값을 수정해 나가는 과정

- Batch

  - 전체 데이터를 몇 개의 묶음으로 나눌건지를 의미한다. 예를 들어 1000 개의 데이터를 20 개로 나누면 1 묶음이 50 개인 20 개의 묶음이 생성된다. 여기서 batch size 는 50 이고

iteration 은 20 이라고 한다. 그리고 batch 1 개를 돌때마다 한번씩 weight 가 갱신된다. 1 epoch 에 20 iteration 이 있으면 20 번 weight 가 갱신된다.

- batch size 가 너무 크면 계산 속도가 느리고 메모리의 문제가 생긴다. 또 너무 작으면 가중치를 너무 자주 바꿔서 불안정해진다. 그래서 적절한 batch size 를 찾는 것이 중요하다.

- batch size 는 16,32,64,128,256 등 메모리가 허락하는 한 크게 잡을수록 정확도가 높아진다고 한다.

- 다만 컴퓨터에서 사용하는 것과 달리 vram 등 실제로 사용되는 곳에서는 batch size 가 64 인게 최대치라고 한다.

- batch size 는 일반적으로 2^n 으로 한다. 반드시 그럴 필요는 없지만 cpu, gpu 내장 메모리가 2 의 제곱수여서 batch size 가 2^n이어야 데이터 송수신의 효율을 높일 수 있기 때문이다.

- Batch size 에 따른 2 가지 변화

1. 최적화 난이도

- 수렴이 빠를수록, 학습용 데이터셋에 성능이 높고 로스가 낮을수록 최적난이도가 낮다고 한다.

- Batch size 가 클수록 계산하기 위해 더 많은 데이터를 사용하므로 최적의 해에 더 잘 들어맞을 가능성이 높다. 다만 데이터셋이 평평한 구조일 경우 기울기의 절대값이 매우 낮아지고 최소점이 아닌 극소점으로 빠질 수 있다.

2. 일반화 성능 : test data 의 정확도를 의미한다. Batch size 가 커지면 loss function 이 두꺼워 저서 성능이 급격하게 변할 수 있다. Keskar et al(2017)의 논문에 따르면 batch size 가 32-512 가 가장 좋은 batch size 라고 한다. 그래서 실전에서는 메모리가 버티는 가장 큰 batch size(512 이하)로 돌리면 정확도가 높다고 한다. 참고로 you et al(2017)에 따르면 resnet50 은 batch size 가 32 일 때 가장 일반화 성능이 좋았다고 한다.

2. 기본 파라미터

ResNet50V2에서 기본 parameter를 이미지 크기는 (224,224,3), optimizer은 sgd+momentum, 이때 momentum은 0.5, batch size는 64, epoch은 10번, validation split은 0.2, callback 함수는 modelcheckpoint, imagenet layer 수를 0.5배로 해 놓고 최적의 batch size를 찾으려고 한다.

3. Batch size 변경

우리는 batch size 16, 32, 64, 128 을 가지고 돌려보았다.

	16	32	64	128
Top1 정확도	74%	76.04%	76.77%	75.97%
Top5 정확도	93.47%	94.62%	95.43%	92.58%

표 3-D-2-3

위 표는 batch size 에 따른 test 데이터의 top1 과 top5 정확도이다. Batch size 가 64 일 때 가장 높은 값을 가지고 있다. Keskar et al 의 논문처럼 batch size 가 32-512 중 하나인 64 에서 최적값을 보였다. 비록 32 가 아니었지만 버전 2 로 바뀌면서 일부 변형돼서 그렇게 된 것으로 추측된다.

3-D-2.4 imagenet 가중치 조절

1. Fine tuning

기존의 학습이 되어져 있는 모델을 기반으로 모델의 parameter 를 미세하게 조정하여 이미 학습된 weight 로부터 학습을 업데이트하는 방법

- 목적 : transfer learning 한 데이터를 가지고 올 때 정교한 파라미터를 만들기 위해 사용한다.

우리는 모델의 가중치를 fine tuning 을 해 보았다. 그러나 imagenet 데이터를 가지고 학습해서 나온 weight 값이 CHIFAR 100 의 데이터와 100% 맞는다고 볼 수 없고 fine tuning 을 할 경우 시간은 엄청 걸리지만 정확도면에서 별로 상승되지 않았다. 그래서 fine tuning 을 전부하는 것이 아니라 layer 에 일부만 고정시켜가면서 데이터를 학습해 보았다.

2. 기본 파라미터

ResNet50V2에서 기본 parameter를 이미지 크기는 (224,224,3), optimizer은 sgd+momentum 0.8, 이때 momentum은 0.5, batch size는 64, epoch은 10번, validation split은 0.2, callback 함수는 modelcheckpoint로 해 놓고 imagenet의 layer를 줄여가며 최적의 layer 수를 찾으려고 한다.

```
base_model = tf.keras.applications.ResNet50V2(input_tensor=resized_images,
                                               input_shape=(224,224,3),
                                               include_top=False,
                                               weights='imagenet')

base_model.trainable = True

fine_tune_at = len(base_model.layers)//2

for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

layer 수를 절반으로 줄인 코드

코드 3-D-2.4

3. Layer 수 변경

	0	0.25	0.5	1
Top1 정확도	78.71%	78.01%	76.58%	63.94%
Top5 정확도	95.89%	95.36%	94.85%	87.85%

표 3-D-2.4

위 그래프처럼 imagenet layer 를 없애니 가장 큰 정확도를 얻었다. 최종적으로 ResNet50V2 에서 Optimizer sgd, momentum 0.5, batch size=64, imagenet layer 없애고 계산해서 top1 정확도 85.84, top5 정확도 95.89 를 얻었다.

느낀점

상현 : 이 프로젝트는 어려운 분야라고 생각해왔던 딥 러닝을 조금 더 쉽고 재미있게 공부할 수 있는 좋은 기회였다. 또한 새로운 우리만의 학습 네트워크를 만들어 보기도 하고 기존 잘 만들어진 네트워크를 다루기도 하며 학습하는 과정의 구조와 원리를 제대로 이해할 수 있게 되었다. 다루는 과정속에서 여러 문제가 발생했고 이를 해결하기 위한 새로운 방법들을 찾고 배워가며 조금 더 깊은 이해를 할 수 있게 되었다. 하지만 이와 함께 좋은 성능을 갖는 네트워크를 만드는 것은 매우 복잡하고 어려우며 특정 한계점들이 있다는 것을 알 수 있었다.

예찬 : 이 수업을 처음 시작할 때 별로 깊게 공부하지 않을 것이라고, 혹은 이론만 깊게 공부할 것이라고 생각했다. 하지만 중간고사 이후부터 여태껏 실질적인 코딩을 같이 수업해 인공지능과 딥러닝에 대해 많이 배우게 되었다. 특히 마지막 2주는 로드가 많아 매우 힘들었지만 몇시간에 걸쳐 구해낸 파라미터가 좋은 성능을 보일 때와 몇일 동안 고생하여 구현한 모델이 잘 돌아가는

모습을 볼 땐 정말 신났고 내가 무엇인가 해냈다는 달성감을 느꼈다. 마지막으로 시간이 부족해 모델을 개선하지 못했다는 점이 아쉽다.

학찬 : 처음 이 수업에서 Term project 과제내용 진행한다고 들었을 때에는, 무척이다 고달프고 팀원들에게 정말로 미안할 일만 생길 것이라고 생각했다. 그 정도로 나에게 인공지능과 프로그래밍은 멀게만 느껴지는 새로운 분야였다. 하지만 Term project을 진행하기 전에, 교수님의 딥러닝에 관한 세세한 설명과 조교님의 면밀한 코드분석과 설명을 통해서, 딥러닝의 기초를 다질 수 있었고, 그럼에도 부족한 실력과 지식은 팀원들과 협업하는 과정 속에서 채울 수 있었다. 이번 Term project를 통해서, TV, 뉴스 속에서나 볼 수 있었던 인공지능을 피부로 느낄 수 있는 좋은 체험이 되었고, 앞으로도 계속 인공지능에 관심을 가질 수 있는 좋은 계기를 가질 수 있었다.

환규: 로드량이 많은 과제였다. 수업을 열심히 들어도 정말 많은 것을 찾고 조사하게 만든 과제였다. 그러나 그러던 중 많은 것을 배우고 단지 시험만 본 것이 아니라 직접 만들어 봄으로써 실습 능력이 생겨서 좋았다. 앞으로 어떠한 문제점이 생겼을 때 해결이 되지 않으면 딥러닝이라는 새로운 무기가 생겨서 다른 방식으로 문제를 해결할 수 있을 것 같다. 또한 이정도로 많은 시간과 막대한 정보 리서치와 데이터를 다룬 과제는 처음이었다. 처음에는 정리하지 않고 조사를 해서 저장한 내용을 찾느라 해매었지만 나중에는 정리의 중요성을 깨닫고 데이터와 정보를 어떻게 정리하는지 알게 되었다. 앞으로 이것보다 더 많은 데이터와 리서치가 필요한 프로젝트를 하게 될 때 이번 과제로 얻은 정리하는 방법과 중요성 인식은 큰 도움이 될 것이라고 확신한다.

진행과정

Jetson nano와 webcam 및 실습에 필요한 물건들을 수령한후 바로 Term-project는 시작되었다. 법학도서관 스터디룸을 빌려 프로젝트의 시작을 열려고 했으나 인터넷선을 연결할 수 없어 다른 장소를 물색하게 되었다. 하지만 프로젝트를 시행하기에 마땅한 공간을 찾는 것은 매우 어려운 일이었다. 우리는 보드게임 카페의 방을 잠시 빌려 그곳에서 프로젝트를 시작했고 그날 딥러닝에 관련된 여러가지 정보를 리서치 하였고 Jetson 과 Tensorflow, opencv등의 설치를 했고 jetson nano 환경에서 real time image detection이 구동되는 것까지 확인하였다. 이후 우리는 앞으로 어떻게 할지를 계획하고 해산하였다. 이후 만남은 비대면으로 Discord를 이용하여 진행되었다. zoom이나 보이스톡과 달리 화면 공유가 가능하고 시간적인 한계가 없기 때문에 비대면으로 프로젝트하기 매우 편리하였다. 우리는 각자 맡은 파트를 수행하였고 거의 매일저녁에 모여 진행상황, 개선할 점, 앞으로의 계획 등을 공유하였다. Task 2와 task 1이 끝난 뒤 그 파트를 맡은 조원 3명 중 2명이 상대적으로 양이 많은 task3을 도와서 코드를 같이 돌렸고 나머지 한 명은 그 당시까지

나온 자료를 취합하여 발표자료를 만들기 시작하였다. Task3까지 모두 끝난 후 각자가 맡은 파트의 대본을 발표자에게 전달하였고 발표자는 발표 연습을 나머지 조원은 예상질문을 작성해 발표를 대비하였다.

참고문헌

밑바닥부터 시작하는 딥러닝 - 사이토 고키 저, 개앞맵시 역-190~194, 199. 210~212, 219~220, 227, 240~241p

파이썬과 opencv를 이용한 컴퓨터 비전 학습, 알렉세이 스피쉐보이, 에이콘

출처

https://pjreddie.com/media/files/papers/YOLOv3.pdf

https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b#:~:text=YOLO%20v2%20used%20a%20custom,the%20layers%20downsampled%20the%20input.

https://paperswithcode.com/sota/image-classification-on-cifar-100

https://keras.io/ko/initializers/#randomnormal

https://www.cs.toronto.edu/~kriz/cifar.html

https://metar.tistory.com/3 gradient decent 사진

https://lv99.tistory.com/25

https://wikidocs.net/55580 배치 사이즈 사용 규칙

https://arxiv.org/abs/1609.04836 cornell 대학 keskar et al

https://arxiv.org/abs/1708.03888 cornell 대학 you et al

https://www.cs.toronto.edu/~kriz/cifar.html CIFAR-100 공식정보

https://arxiv.org/pdf/1602.07261.pdf InceptionResnetV2에 관한 논문

https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf Resnet 관한 논문

https://arxiv.org/pdf/1409.1556.pdf%20http://arxiv.org/abs/1409.1556.pdf Vgg16에 관한 논문

http://image-net.org/about-overview imagenet 설명