유방암 조직 이미지를 이용한

침윤성 유관암 분류

숙탯 2기 딥러닝 프로젝트

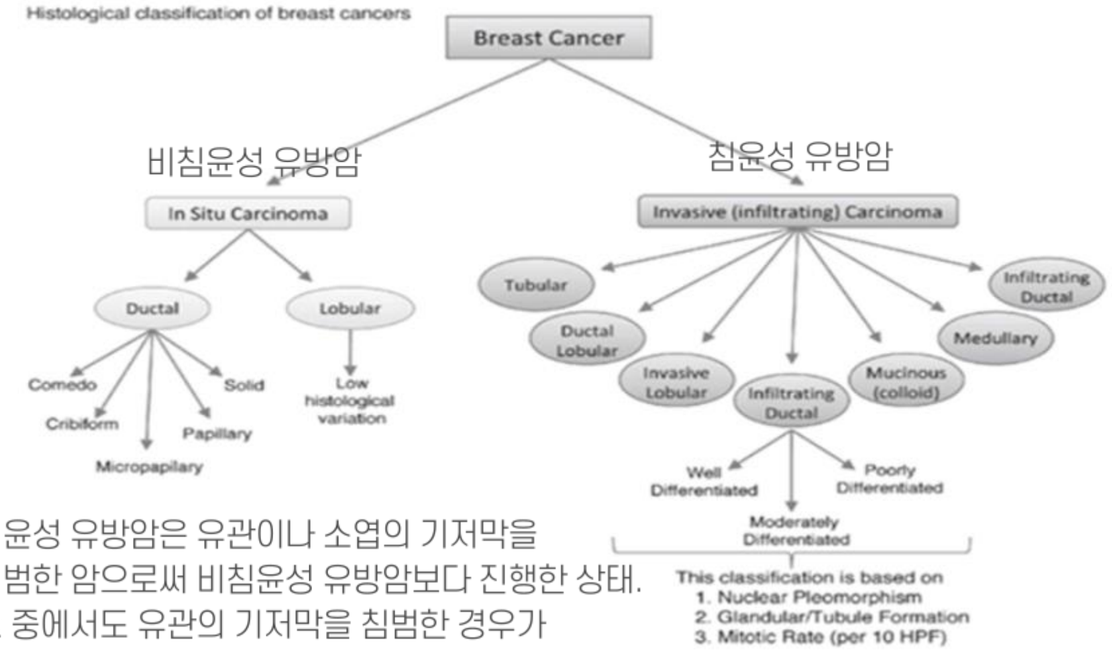
숙탯 2기 김윤진, 김희경, 전소연

프로젝트 소개

1

프로젝트 목표

IDC(Invasive Ductal Carcinoma,
침윤성 유관암)가 조직에 포함되어 있는지
여부 예측



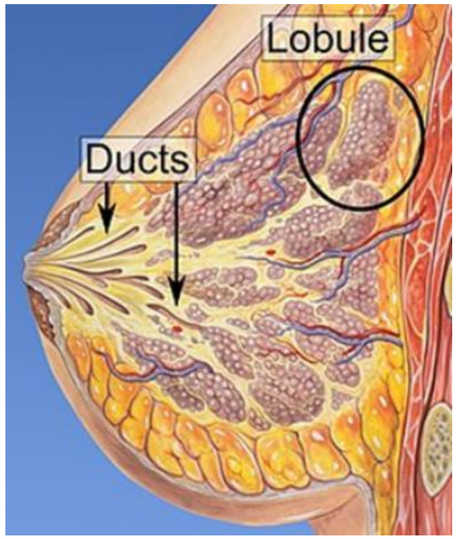
침윤성 유방암은 유관이나 소엽의 기저막을
침범한 암으로써 비침윤성 유방암보다 진행한 상태.
그 중에서도 유관의 기저막을 침범한 경우가
침윤성 유관암.

→ 3가지 가능성 : 건강한 조직, IDC, 유방암의 또 다른 하위 유형

2

Dataset 정보

40배율로 스캔한 유방암 표본의
162개 조직 이미지에서
50 x 50 사이즈 패치 277,524개 추출

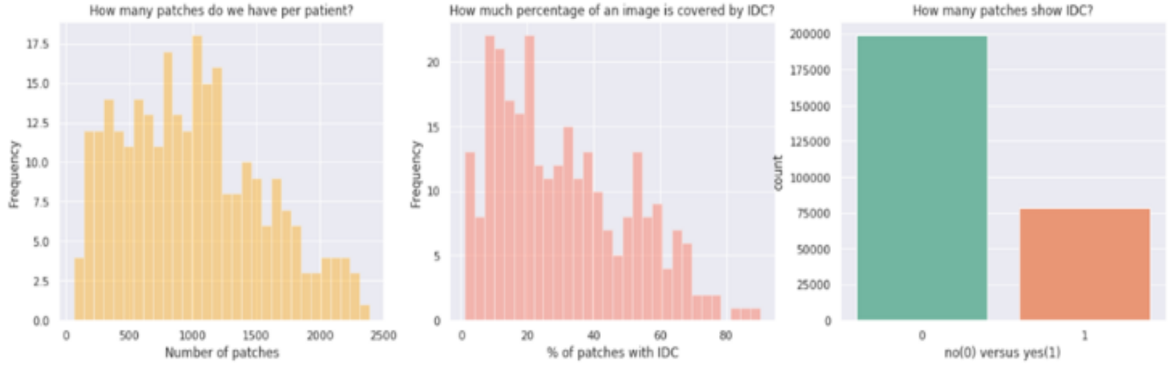


3

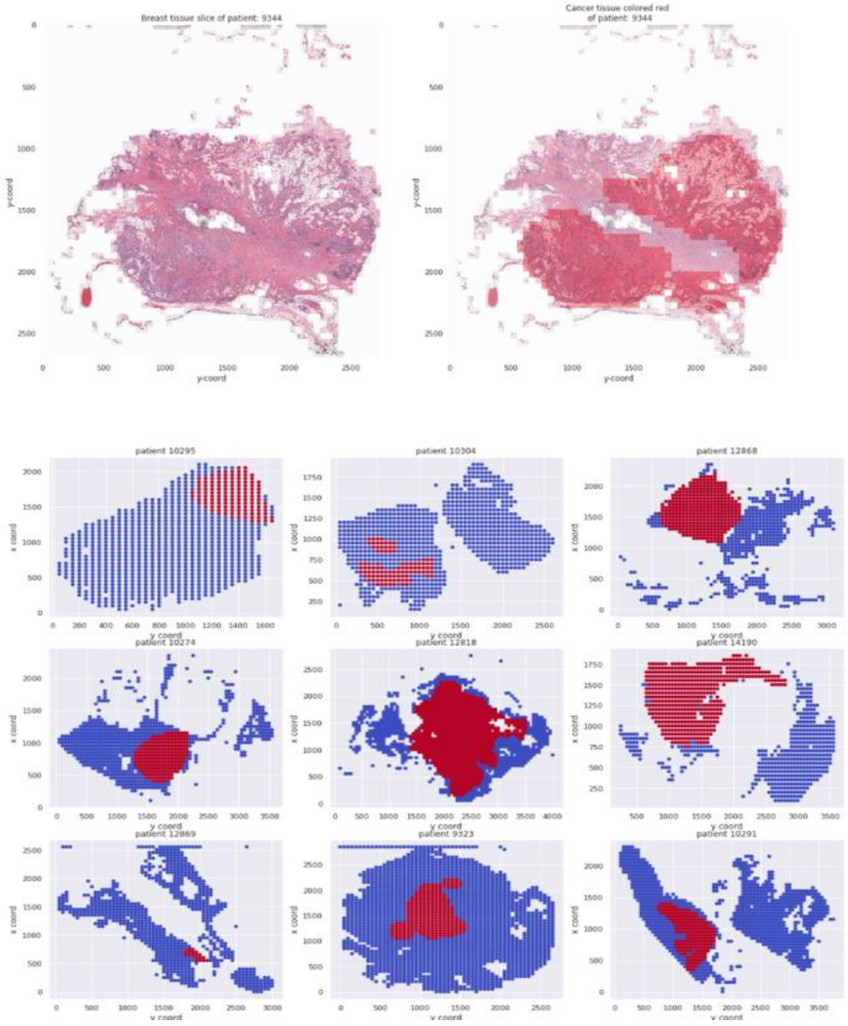
평가 지표

F1 score

EDA



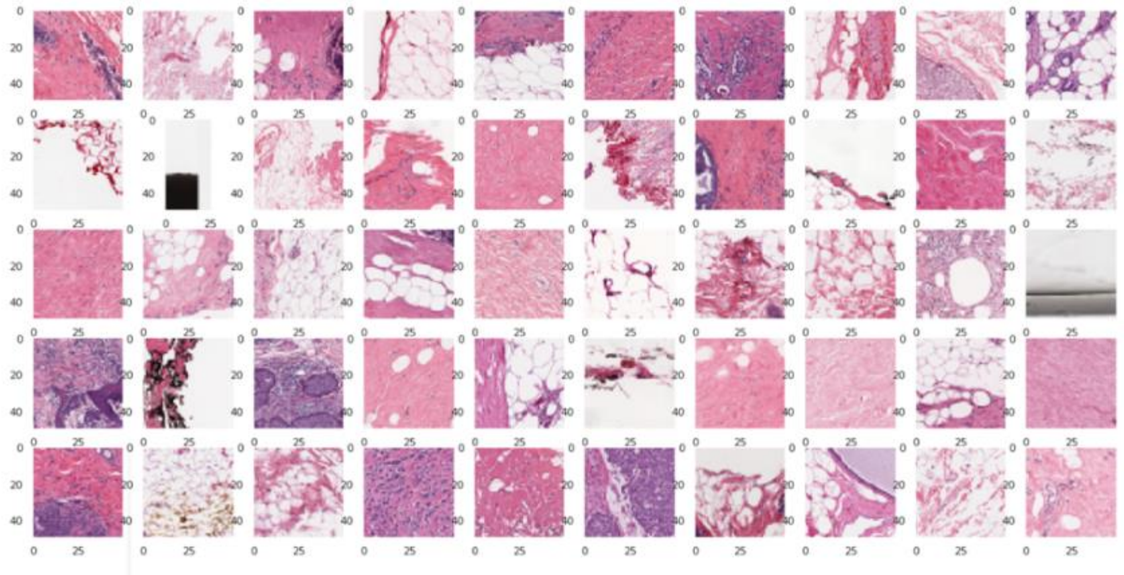
1. 환자당 이미지 패치의 수는 매우 다양함
2. 일부 환자는 IDC를 나타내는 패치가 80% 이상. 즉, 조직이 암으로 가득 차거나 일부만 IDC 암인 조직 조각으로 이루어져 있음.
3. IDC 클래스와 IDC가 없는 클래스는 불균형.



EDA

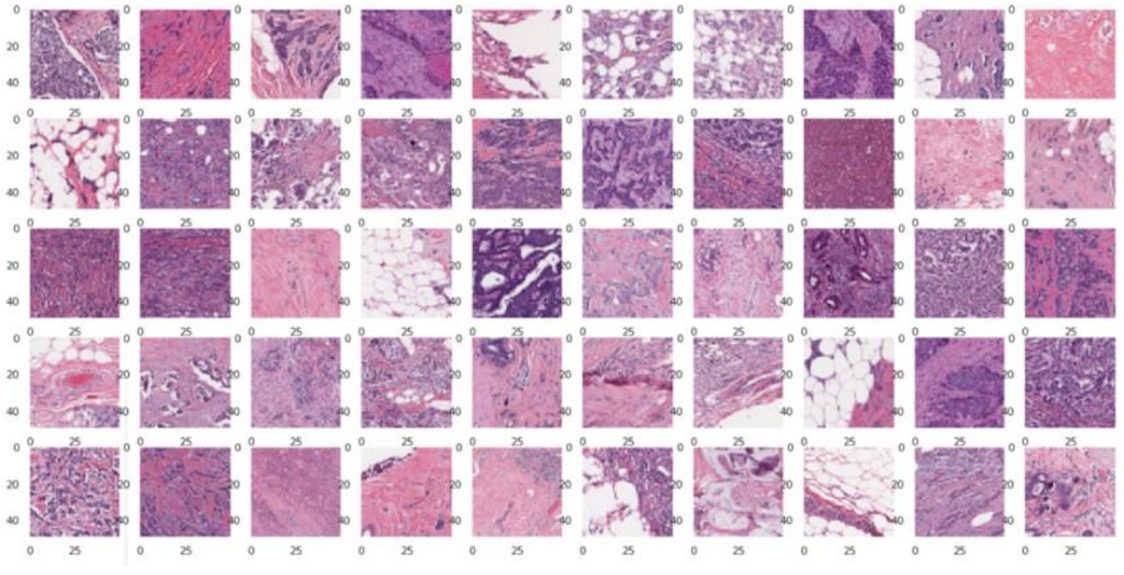
Healthy patches

Show hidden code



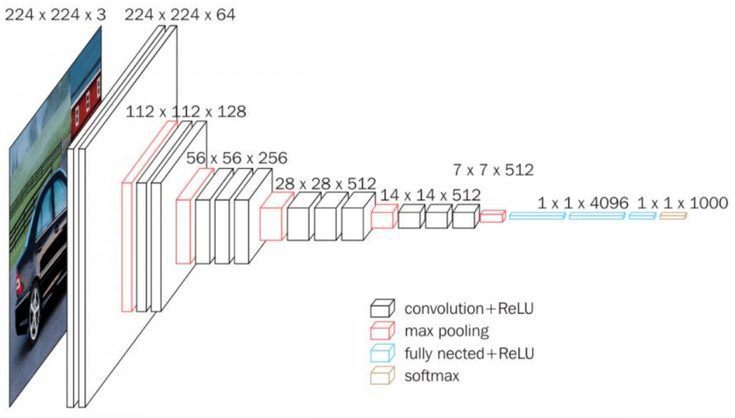
Cancer patches

Show hidden code



VGGNet16

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



모델설명

-3*3짜리 conv2D 2개와 2*2 MaxPool2D를 하나의 층으로 만들어 총 5개의 layer을 만듦. (activation function은 ReLU function을 사용함)

-batch size는 128, num_class는 2(IDC이다/ 아니다),

-학습반복 수(epoch)는 12번으로 설정함.

세부조정

Epoch를 12로 했을 때는 정확도가 50%를 웃돌았음

-> epoch 30으로 하니까 정확도가 점점 좋아짐

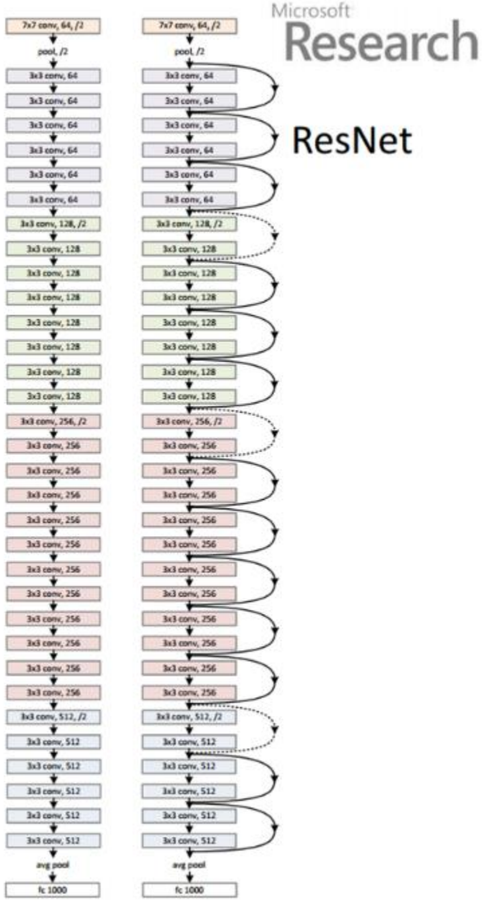
->epoch 50까지 늘림 정확도 76%~75% 까지 올라감

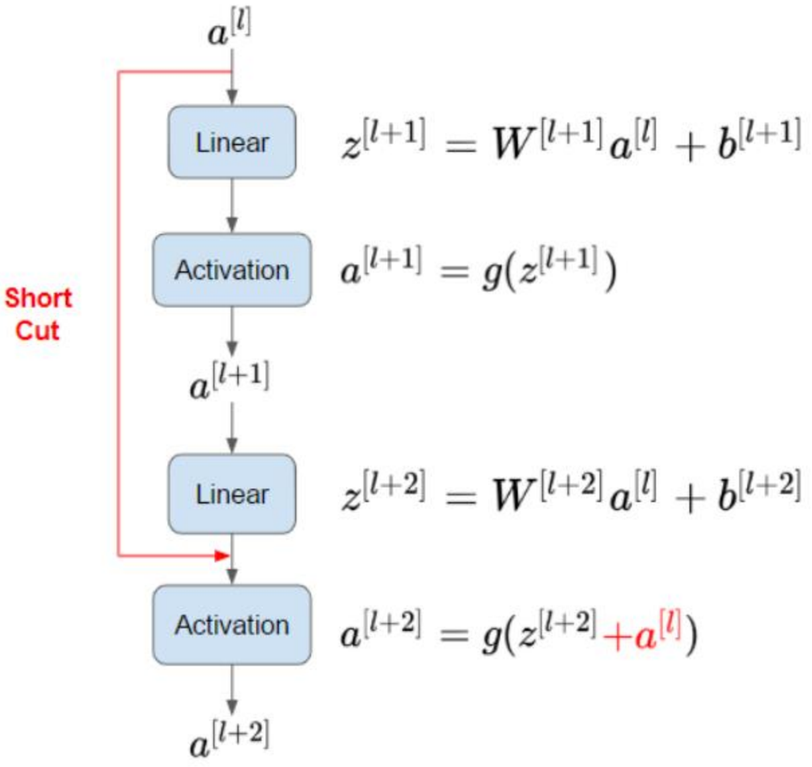
eras CNN #1A - accuracy: 0.7639639377593994

	precision	recall	f1-score	support
IDC(-)	0.73	0.85	0.78	563
IDC(+)	0.81	0.68	0.74	547
accuracy			0.76	1110
macro avg	0.77	0.76	0.76	1110
eighted avg	0.77	0.76	0.76	1110

모델 정확도 76%

ResNet - 모델 소개





https://www.youtube.com/watch?v=ZILIbUvp5lk&feature=youtu.be

- ResNet 에서는 스킵 연결로 경사 소실/폭발 문제를 해결함으로써 아주 깊은 신경망도 학습 가능

- 해당 층의 계산된 값과, 이전 층의 입력값을 더해서 함께 활성화 함수를 거쳐 비선형성을 적용. 이전 층의 정보를 더 깊은 층까지 전달하기 위함.

- 이전 층의 입력값을 더해 다시 활성화 함수에 넣는 부분까지를 잔여 블록이라고 부르기에 ResNet(Residual Network)

모델 학습

ResNet50 - 전이 학습 (imagenet 가중치 사용, 분류기만 재설정)

ResNet50 최상층 분류기

```
avg_pool (GlobalAveragePooling2D)   (None, 2048)

predictions (Dense)                 (None, 1000)
```

이미지를 1000개의 클래스로 분류

▶▶

예측 목적에 맞게 재설정한 분류기 (기존 분류기 사용 X)

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 2, 2, 2048)	23587712
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2097408
activation (Activation)	(None, 256)	0
batch_normalization (BatchNormalization)	(None, 256)	1024
dense_1 (Dense)	(None, 2)	514
activation_1 (Activation)	(None, 2)	0

이미지를 2개의 클래스로 분류

▶▶

성능

training : 0.9928

validation : 0.6964

test: 0.7324

Inception V3

모델의 크기가 증가하면 연산량이 증가하게 되어 제한된 메모리에서 이를 돌릴 시 단점으로 작용

따라서 convolution 분해를 활용해서 연산량이 최소화 되는 방향으로 모델의 크기를 키우는데 집중

Inception V3

- 5*5 convolution, 7*7 convolution 을 3*3 convolution으로 분해
- 3*3 convolution을 더 작은 convolution으로 분해
- 보조 분류기 사용
- pooling layer과 conv layer을 병렬로 사용

모델 학습

Inception V3

5*5 convolution, 7*7 convolution 을 3*3 convolution으로 분해

5x5 convolution -> 25번 연산
3x3 convolution 2번 -> 18번 연산
↓
연산량 18/25배

5*5보다 3*3을 두번 사용하는게 더 정확도가 올라감

*3x3 convolution보다 큰 filter는 언제든지 3x3 convolution으로 분해하여 사용하는것이 좋다

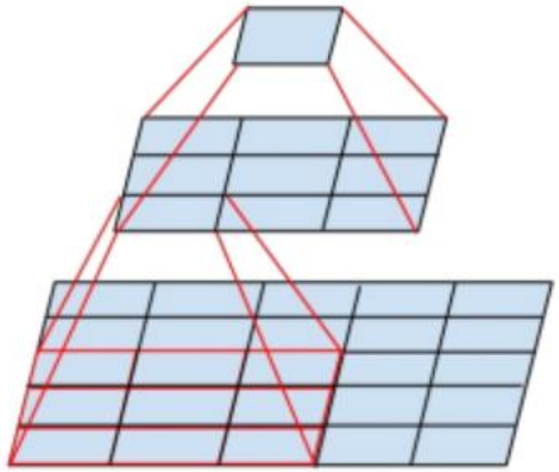


Figure 1. Mini-network replacing the 5 × 5 convolutions.

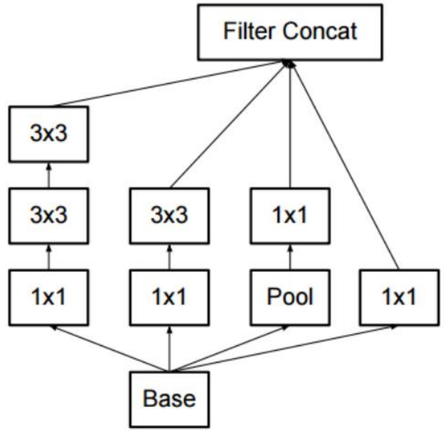


Figure 5. Inception modules where each 5 × 5 convolution is replaced by two 3 × 3 convolution, as suggested by principle 3 of Section 2.

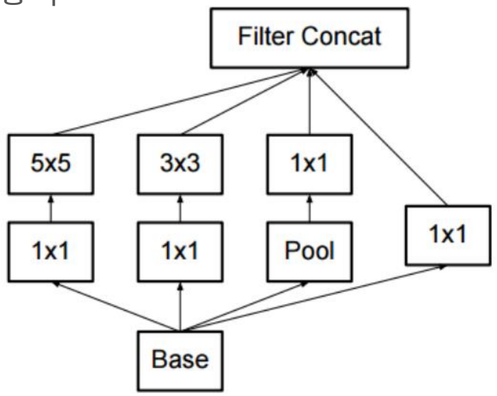


Figure 4. Original Inception module as described in [20].

Inception V3

3*3 convolution을 더 작은 convolution으로 분해

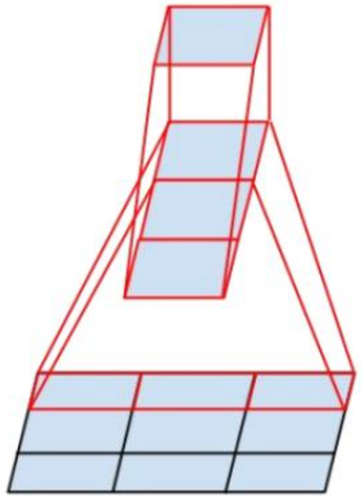


Figure 3. Mini-network replacing the 3×3 convolutions. The lower layer of this network consists of a 3×1 convolution with 3 output units.

3*3 convolution을 더 작은 convolution으로 분해

↓

1) 2*2 or n*1비대칭 convolution으로 분해.
2) 3*3을 3*1또는 1*3으로 분해

↓

3x1 or 1x3의 경우: 33%의 연산량 절감 효과.

2 x2 or nx1의 경우: 11%의 연산량 감소 효과

Inception V3

효율적인 그리드 축소:
pooling layer과 conv layer을 병렬로 사용

기존 CNN

사이즈를 줄이기 위해 pooling 연산
Representational bottlenet 을 피하기 위해
필터 수 증가

↓

연산량을 감소 시켜 주지만 표현력도
감소시킨다.

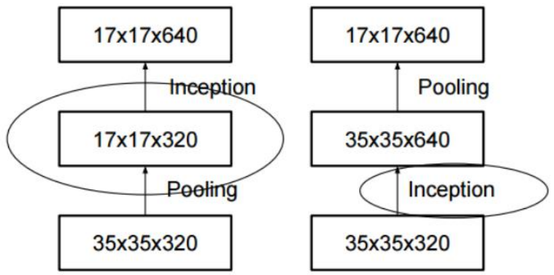


Figure 9. Two alternative ways of reducing the grid size. The solution on the left violates the principle 1 of not introducing an representational bottleneck from Section 2. The version on the right is 3 times more expensive computationally.

Figure out

Inception V3

pooling layer과 conv layer을
병렬로 사용한다. 그리고 이 둘을 연결한다.

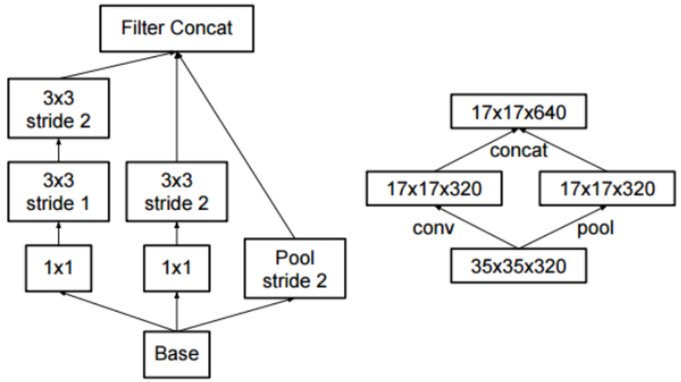


Figure 10. Inception module that reduces the grid-size while expands the filter banks. It is both cheap and avoids the representational bottleneck as is suggested by principle 1. The diagram on the right represents the same solution but from the perspective of grid sizes rather than the operations.

Inception V3

Inception V2

- 5*5 convolution, 7*7 convolution 을 3*3 convolution으로 분해
- 보조 분류기 사용
- pooling layer과 conv layer을 병렬로 사용
- 3*3 convolution을 더 작은 convolution으로 분해

+ ? Label smoothing =

0과 1로만 이루어져 있는 변수들을
0.02 /0.92처럼 값을 조금씩 깎아서 계산 하는 것!

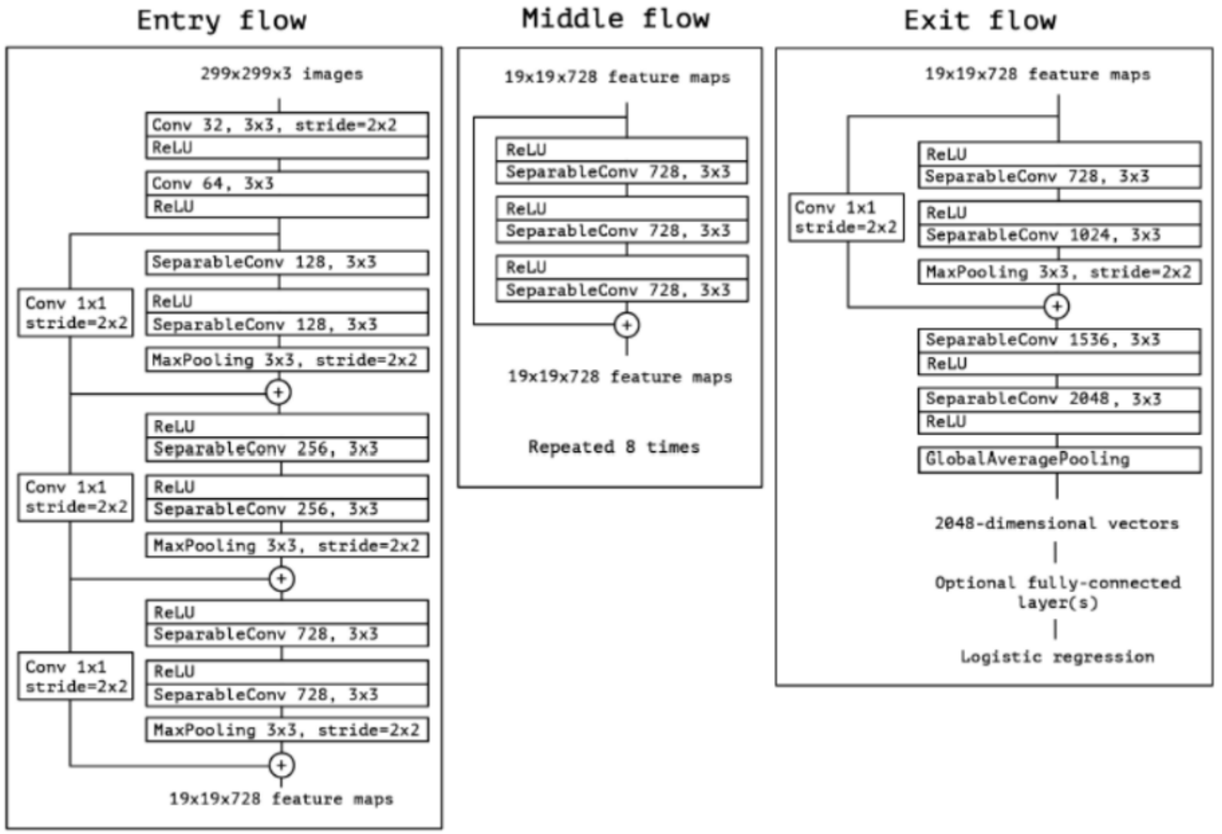
Inception V3

모델 학습

Xception

eXtreme Inception

- depthwise separable convolution



- Xception (BreakHis 사전학습)

↳ 최소 71X71 이미지만 사용 가능

→ 50X50→71X71 resize

: 사전학습한 층 154개 중 하위 100개 층을 고정하고 미세조정 (epoch : 21)

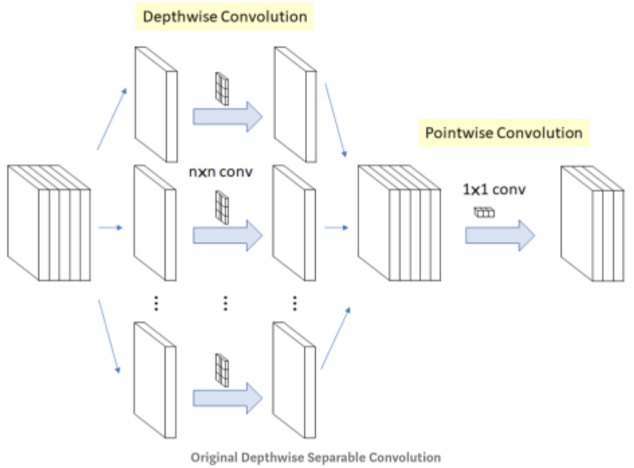
Train	Valid
0.9992	0.7838

모델 학습

MobileNet V2

(MobileNetV2: **Inverted Residuals** and **Linear Bottlenecks**)

- depthwise separable convolution



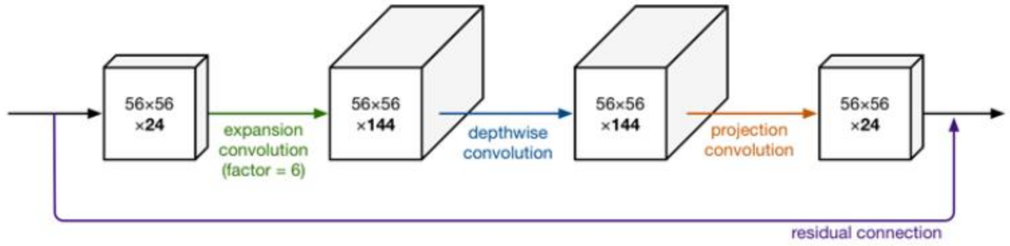
depthwise convolution : 채널별로 분리하여 각 채널의 각각의 커널로 convolution

pointwise convolution : 1x1 conv, 출력의 채널을 바꿀 수 있음

ex) 3x3x3x3=81 → 3x3x1x3+1x1x3x3=27+9=36

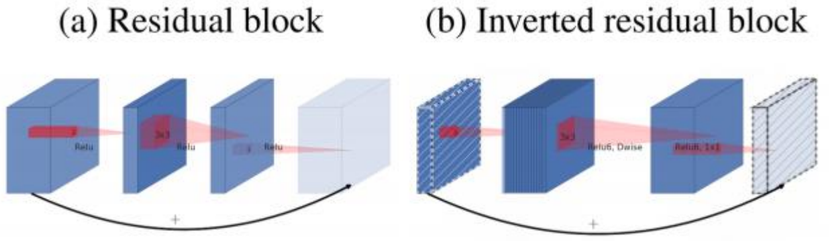
⇒ 파라미터 수를 줄일 수 있음

- Linear Bottlenecks



- 차원수가 충분히 큰 공간에서 ReLU 함수를 사용하면 그만큼 정보 손실율이 낮아짐
- 저차원의 데이터를 expansion 층을 통해 확장
  → depthwise convolution 연산
  → projection(activation function X)

- Inverted residual block



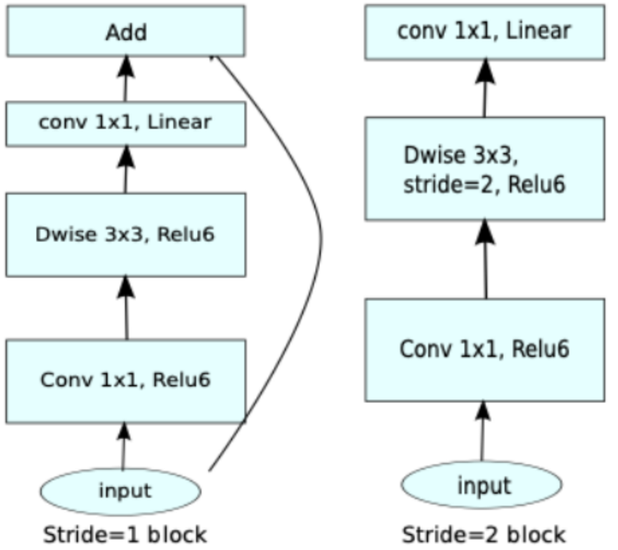
- ReLU6 사용
  x>0인 구간에서 x
  x>6인 구간에서 6

MobileNet V2

(MobileNetV2: **Inverted Residuals** and **Linear Bottlenecks**)

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

t : block 내부에서 팽창시킬 배수
c : 채널 수
n : blottleneck의 반복수
s : stride



(d) Mobilenet V2

모델 학습

MobileNet V2

(MobileNetV2: **Inverted Residuals** and **Linear Bottlenecks**)

모델 성능

- MobileNetV2 (사전학습 X)

Train	Valid
0.9775	0.7342

- MobileNetV2 (Imagenet 사전학습)

: 사전학습한 층 154개 중 하위 50개 층을 고정하고 미세조정 (epoch : 10)

Train	Valid
0.9569	0.8041

- MobileNetV2 (BreakHis 사전학습)

- 사전학습 층 모두 사용, 분류기만 재설정

Train	Valid	Test
0.7360	0.7658	0.7378

- 사전학습한 층 154개 중 하위 100개 층을 고정하고 미세조정 (epoch 66에서 중단)

Train	Valid	Test
0.7968	0.8108	0.7649

↱ 채택 모델

Cancernet - 모델 소개

모델 구조

- 3* 3 컨볼루션 필터를 사용.
- SeparableConv2D ⇒ Relu ⇒ Maxpooling2D 블록을 필터 수를 늘려가며(32, 64, 128개) 3번 반복.
- 배치 정규화와 드롭아웃 적용되어있음.
- sequential API를 사용하여 전체 CancerNet을 패키징

SeperableConv2D 원리

- 깊이별 분리 합성곱 층 (depthwise separable~)
- depthwise convolution을 먼저 수행한 후 각 채널을 1개의 채널로 압축할 수 있는 추가 convoluton을 진행.

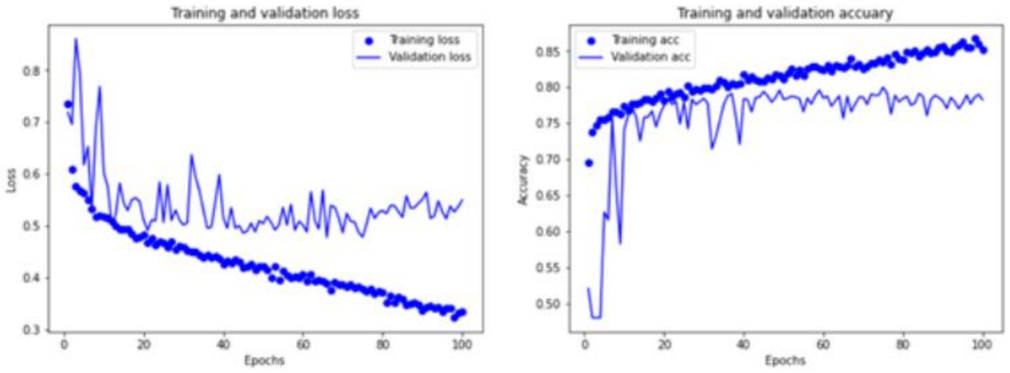
cf) Depthwise Convolution

- 채널별로 각각 feature map 얻은 뒤 얻은 feature map 들을 합쳐 하나의 feature map을 얻는다.
- 즉 채널 방향의 convolution은 하지 않고 공간 방향의 convolution만 진행한다.

Cancernet - 최적화 알고리즘 적용

01. Adagrad

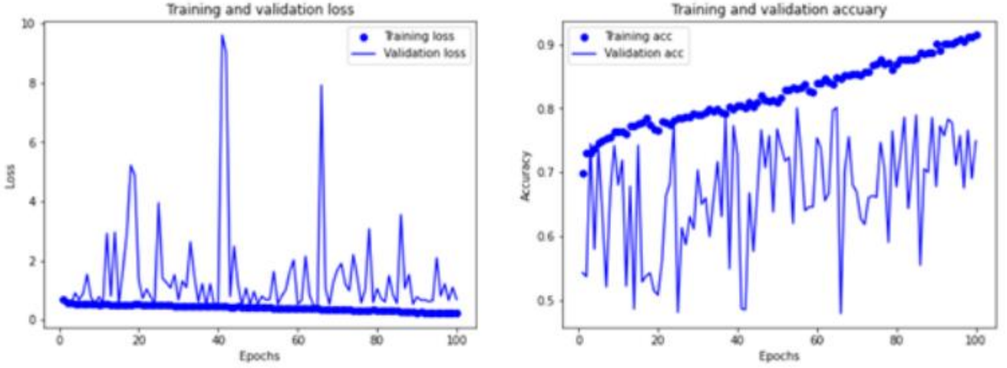
학습을 진행하면서 학습률을 점차 줄여가는 방식. 개별 매개변수 각각에 대해 학습률을 조정하며 학습. 갈수록 보폭을 줄여 세밀하게 탐색하는 셈.



validation accuracy 0.75 정도로 학습

02. Momentum

iteration에 따라 방향이 반대로 계속 바뀌는 경로를 지나온 경우 해당 방향으로 update하는 변화량이 점차 작아지고, iteration에 따라 방향이 계속 유지되면 가속을 붙여주는 방식. 관성을 따라 진행.

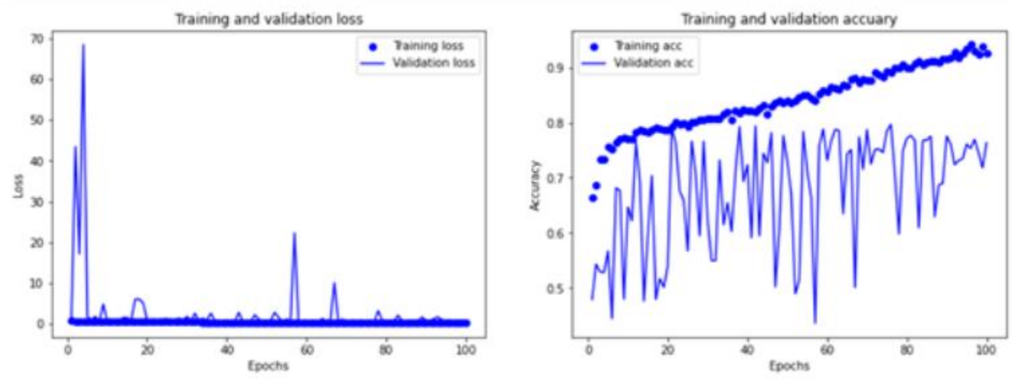


validation accuracy 변동이 심해서 학습이 잘 이뤄지는 것 같지 않음

Cancernet - 최적화 알고리즘 적용

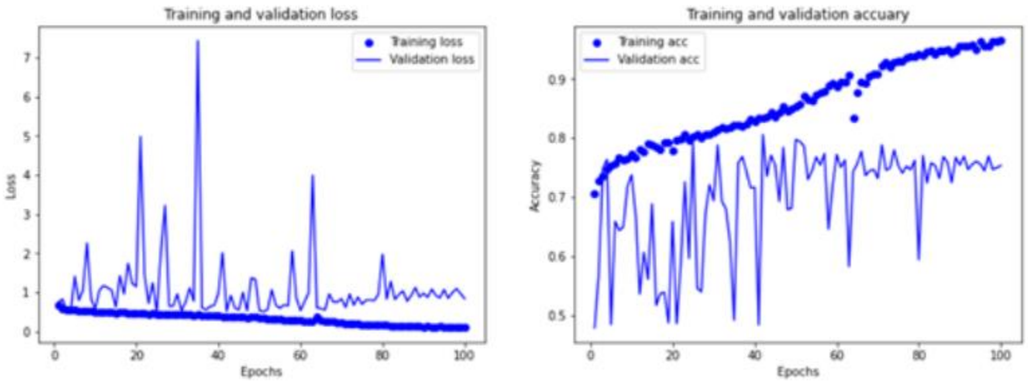
03. RMSProp

Gradient의 방향을 이용하지 않고 그 크기만을 이용해 업데이트 해주고자 하는 각 parameter에 대한 학습 속도를 조절한다.
Adagrad에 비하여 이전 맥락을 고려해 학습률을 조절.



04. Adam

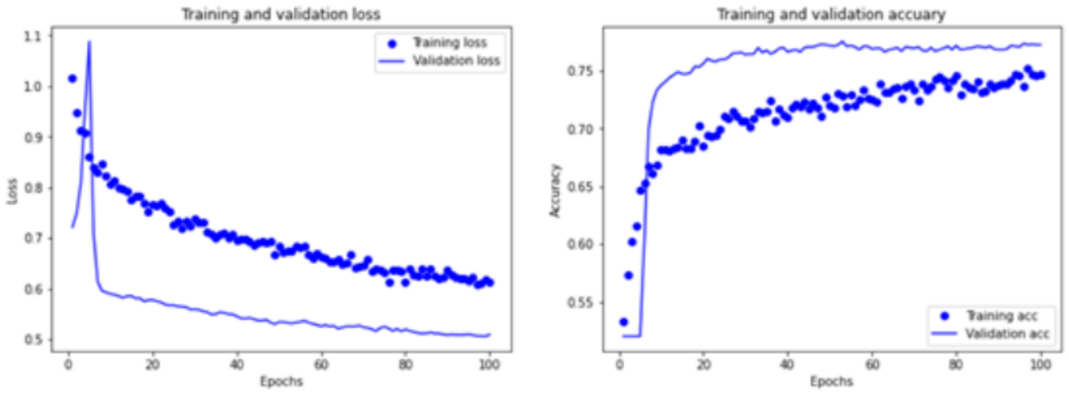
RMSProp과 Momentum의 융합. gradient descent를 진행하는 방향도, 스텝의 크기도 적당하게 택하고자 하는 알고리즘.



Cancernet - 최적화 알고리즘 적용

05. Adadelta

iteration마다 learning rate가 작아지는 Adagrad의 문제점을
해결하고자 고안된 알고리즘.



validation accuracy 0.78 정도로 학습

결론

Momentum, RMSProp, Adam 최적화 알고리즘은 accuracy 변동이 심한 것으로 보아 학습이 제대로 이뤄지지 않는다고 판단,

Adagrad와 Adadelta를 최적화 알고리즘으로 선택.

Cancernet - 네트워크 정규화

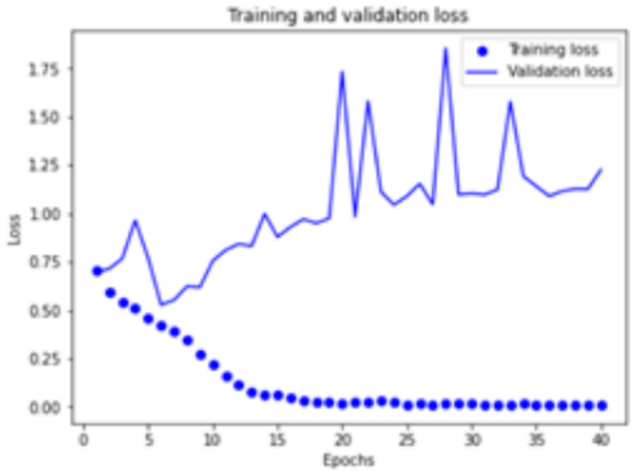
- 과대적합 방지 목적

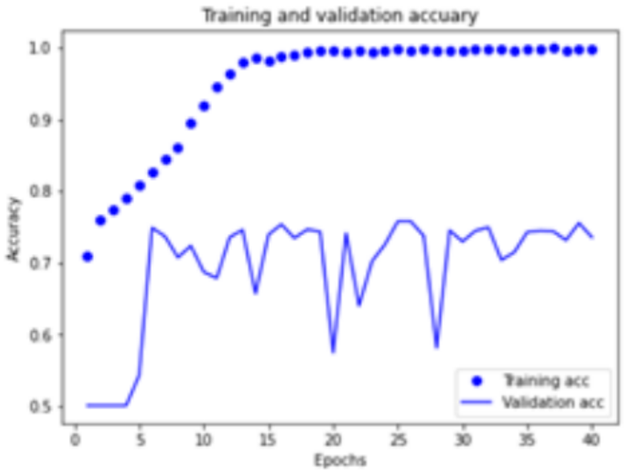
01. L2 정규화

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

$$w \to w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} w$$
$$= \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}.$$

- 손실함수에 L2 정규화 항을 추가
- 람다 값을 크게 만들수록 가중치 행렬 W를 0에 가깝게 설정할 수 있음. 그 결과 간단하고 작은 신경망이 되기에 과대적합이 덜 일어남.





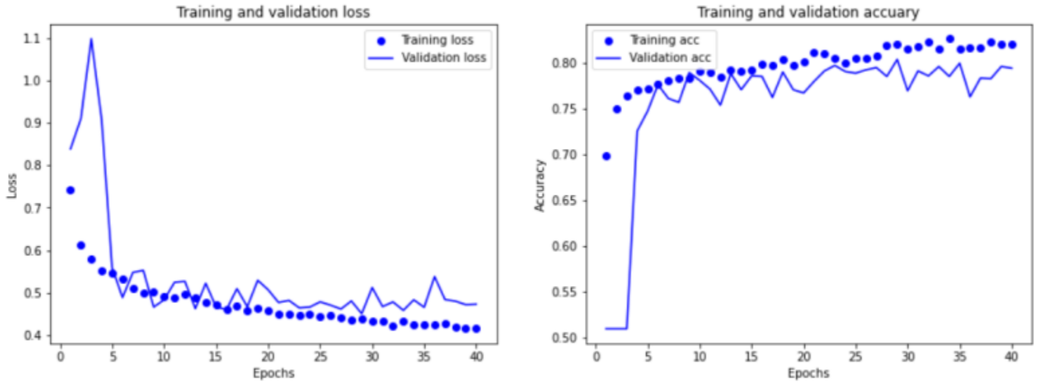
-Validation loss 와 validation accuracy를 보았을 때 L2 정규화는 Cancernet에 적합하지 않은 것으로 보임.

Cancernet - 네트워크 정규화

02. 드롭아웃

- 신경망의 각각의 층에 대해 노드를 삭제할 확률을 설정하고,
  삭제할 노드를 랜덤으로 선정한 후
  삭제된 노드에 들어가는 링크와 나가는 링크를 모두 삭제.

- 그 결과 더 작고 간소화된 네트워크가 만들어지고
  이 작아진 네트워크로 훈련을 진행하게 되므로
  과대적합이 일어나는 것을 줄일 수 있음.



결론

드롭아웃을 적용한 경우 유의미한 accuracy를
얻을 수 있고 모델이 제대로 학습하는 것으로
보이므로 네트워크 정규화 기법으로 드롭아웃 채택.

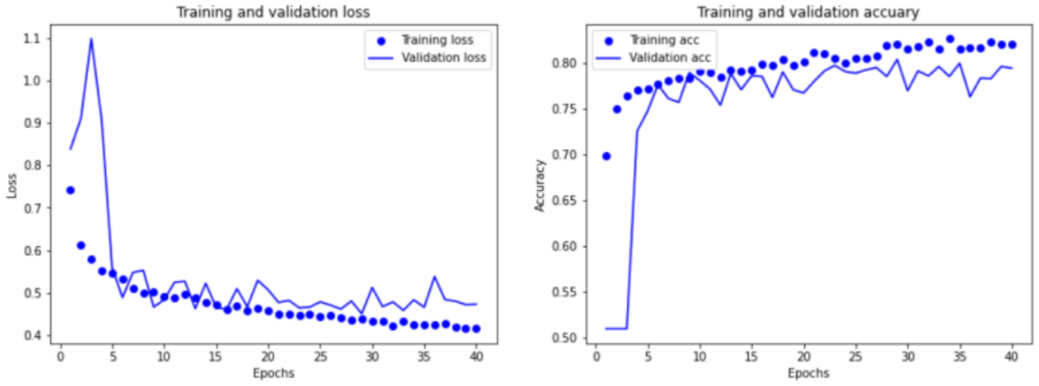
* L2 정규화와 드롭아웃의 차이점
- L2 정규화는 개별 가중치 각각에 적용되나 드롭아웃은 그렇지 않음.
- 컴퓨터 비전에는 대체로 드롭아웃이 적합함

Cancernet - 네트워크 정규화

02. 드롭아웃

- 신경망의 각각의 층에 대해 노드를 삭제할 확률을 설정하고, 삭제할 노드를 랜덤으로 선정한 후 삭제된 노드에 들어가는 링크와 나가는 링크를 모두 삭제.

- 그 결과 더 작고 간소화된 네트워크가 만들어지고 이 작아진 네트워크로 훈련을 진행하게 되므로 과대적합이 일어나는 것을 줄일 수 있음.



결론

드롭아웃을 적용한 경우 유의미한 accuracy를 얻을 수 있고 모델이 제대로 학습하는 것으로 보이므로 네트워크 정규화 기법으로 드롭아웃 채택.

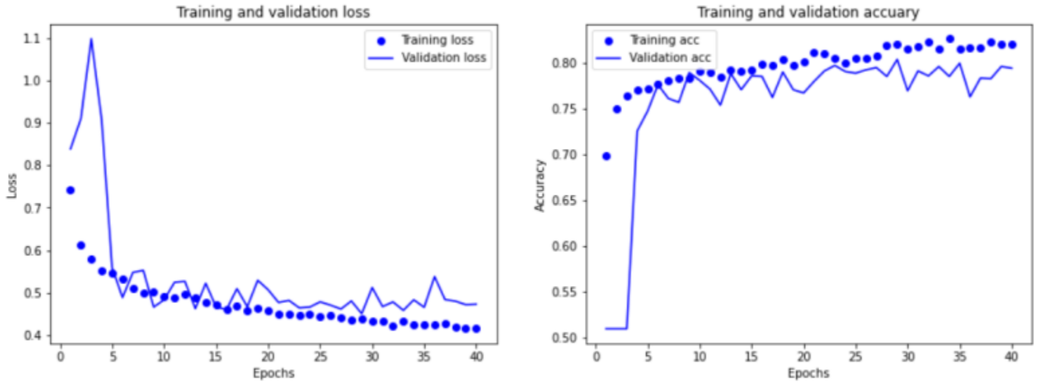
* L2 정규화와 드롭아웃의 차이점
- L2 정규화는 개별 가중치 각각에 적용되나 드롭아웃은 그렇지 않음.
- 컴퓨터 비전에는 대체로 드롭아웃이 적합함

Cancernet - 네트워크 정규화

02. 드롭아웃

- 신경망의 각각의 층에 대해 노드를 삭제할 확률을 설정하고, 삭제할 노드를 랜덤으로 선정한 후 삭제된 노드에 들어가는 링크와 나가는 링크를 모두 삭제.

- 그 결과 더 작고 간소화된 네트워크가 만들어지고 이 작아진 네트워크로 훈련을 진행하게 되므로 과대적합이 일어나는 것을 줄일 수 있음.



결론

드롭아웃을 적용한 경우 유의미한 accuracy를 얻을 수 있고 모델이 제대로 학습하는 것으로 보이므로 네트워크 정규화 기법으로 드롭아웃 채택.

* L2 정규화와 드롭아웃의 차이점
- L2 정규화는 개별 가중치 각각에 적용되나 드롭아웃은 그렇지 않음.
- 컴퓨터 비전에는 대체로 드롭아웃이 적합함

Cancernet - 네트워크 크기, 아키텍쳐 조정

- train accuracy 향상 목적으로 시도하였음.

기존 모델에서
SeperableConv2D 층을
네 번 쌓은 블록을
하나 더 추가한 모델

기존 모델에서
SeperableConv2D를
Conv2D로 바꾼 모델

SeperableConv2D를
Conv2D로 바꾸고
블록이 네 개인 모델

- 모델 1과 기존 모델 성능 동일
- 학습 시간은 기존 모델이 더 빠름
- 따라서 **네트워크 크기나 구조는**
  변동 없이 기존 모델을 사용하기로 함.

Cancernet - 하이퍼파라미터 튜닝

미니 배치 크기와 **learning rate** 값을 랜덤하게 탐색.

무작위 접근의 타당성

튜닝 시 우선순위에 대해 관행은 존재할지라도, 특정 모델에서의 하이퍼 파라미터의 우선순위를 정확히 파악하기는 어렵고, grid search 방식의 경우 더 중요도가 높은 하이퍼파라미터의 몇 가지 값에 의해 사실상 결정되므로 탐색하는 경우의 수가 적기 때문.

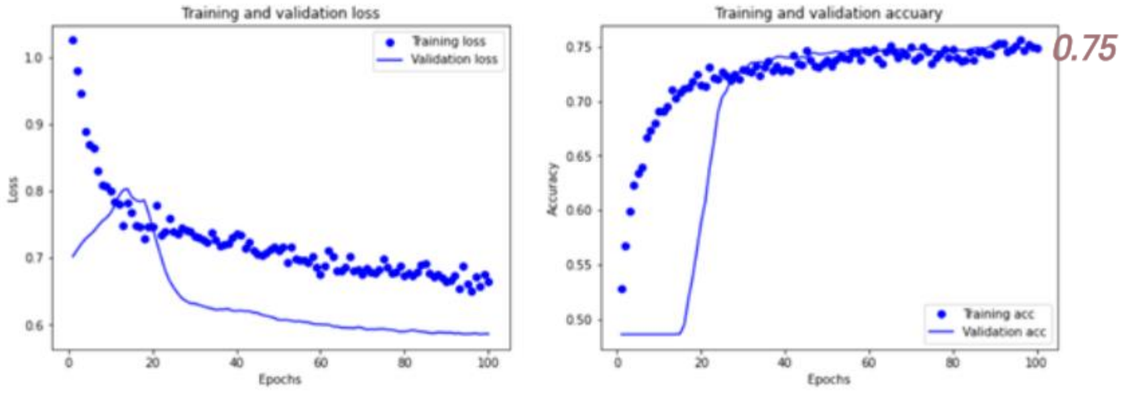
최종 모델

⇒ F1 score : 0.7887

Case 1

optimizer : Adadelta
batch size = 128
learning rate = 0.003962703907659129

Test accuracy : 0.7252

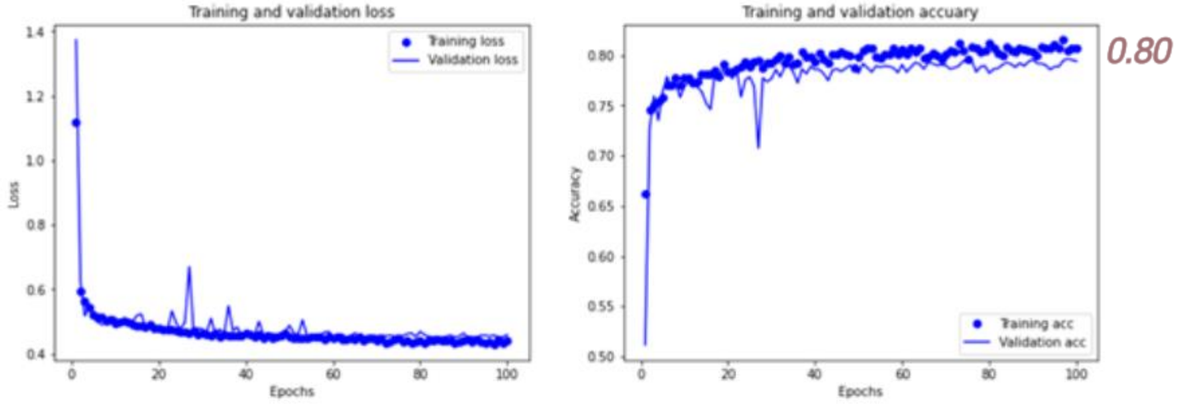


0.75

Case 2

optimizer : Adagrad
batch size = 64
learning rate = 0.028213040859089513

Test accuracy : 0.7955



0.80

Cancernet -기타시도

이미지 늘리기

```
img_np = np.array(img_list_np) #리스트를 numpy로 변환
print(img_np.shape)
```

```
(5547, 75, 75, 3)
```

Adam 알고리즘

```
111/111 [==============================] - 4s 34ms/step - loss: 0.7542 - accuracy: 0.6551 - val_loss: 19.1723 - val_accuracy: 0.4802
Epoch 39/40
111/111 [==============================] - 4s 35ms/step - loss: 0.6680 - accuracy: 0.6627 - val_loss: 1507.9822 - val_accuracy: 0.5694
Epoch 40/40
111/111 [==============================] - 4s 34ms/step - loss: 0.6654 - accuracy: 0.6709 - val_loss: 0.6349 - val_accuracy: 0.6748
```

Momentom 알고리즘

```
111/111 [==============================] - 4s 34ms/step - loss: 0.5989 - accuracy: 0.7070 - val_loss: 156.7462 - val_accuracy: 0.7297
Epoch 39/40
111/111 [==============================] - 4s 34ms/step - loss: 0.5874 - accuracy: 0.7143 - val_loss: 16.7607 - val_accuracy: 0.7207
Epoch 40/40
111/111 [==============================] - 4s 33ms/step - loss: 0.5881 - accuracy: 0.7117 - val_loss: 30.6879 - val_accuracy: 0.7225
```

RMS prop 알고리즘

```
111/111 [==============================] - 4s 35ms/step - loss: 0.5451 - accuracy: 0.7456 - val_loss: 3655.4145 - val_accuracy: 0.7378
Epoch 39/40
111/111 [==============================] - 4s 35ms/step - loss: 0.5602 - accuracy: 0.7292 - val_loss: 1751.4178 - val_accuracy: 0.7189
Epoch 40/40
111/111 [==============================] - 4s 36ms/step - loss: 0.5505 - accuracy: 0.7422 - val_loss: 7479.4634 - val_accuracy: 0.7243
```

Cancernet -기타시도

스트라이드 늘리기/ 줄이기

```
# CONV => RELU => POOL
cancernet.add(SeparableConv2D(32, (2, 2), padding="same",
  input_shape=inputShape))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(MaxPooling2D(pool_size=(2, 2)))
cancernet.add(Dropout(0.25))

# (CONV => RELU => POOL) * 2
cancernet.add(SeparableConv2D(64, (2, 2), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(64, (2, 2), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(MaxPooling2D(pool_size=(2, 2)))
cancernet.add(Dropout(0.25))

# (CONV => RELU => POOL) * 3
cancernet.add(SeparableConv2D(128, (2, 2), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(128, (2, 2), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(128, (2, 2), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(MaxPooling2D(pool_size=(2, 2)))
cancernet.add(Dropout(0.25))

# first (and only) set of FC => RELU layers
cancernet.add(Flatten())
cancernet.add(Dense(256))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(Dropout(0.5))

# softmax classifier
cancernet.add(Dense(2))
```

```
# CONV => RELU => POOL
cancernet.add(SeparableConv2D(32, (5, 5), padding="same",
  input_shape=inputShape))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(MaxPooling2D(pool_size=(2, 2)))
cancernet.add(Dropout(0.25))

# (CONV => RELU => POOL) * 2
cancernet.add(SeparableConv2D(64, (5, 5), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(64, (5, 5), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(MaxPooling2D(pool_size=(2, 2)))
cancernet.add(Dropout(0.25))

# (CONV => RELU => POOL) * 3
cancernet.add(SeparableConv2D(128, (5, 5), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(128, (5, 5), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(128, (5, 5), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(MaxPooling2D(pool_size=(2, 2)))
cancernet.add(Dropout(0.25))

# first (and only) set of FC => RELU layers
cancernet.add(Flatten())
cancernet.add(Dense(256))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(Dropout(0.5))

# softmax classifier
cancernet.add(Dense(2))
```

Adam 알고리즘

```
Epoch 39/40
122/122 [==============================] - 2s 14ms/step - loss: 0.4477 - accuracy: 0.7955 - val_loss: 94.2518 - val_accuracy: 0.7790
Epoch 40/40
122/122 [==============================] - 2s 14ms/step - loss: 0.4462 - accuracy: 0.8035 - val_loss: 53.4037 - val_accuracy: 0.7712
```

Momentom 알고리즘

```
Epoch 38/40
122/122 [==============================] - 1s 12ms/step - loss: 0.7311 - accuracy: 0.7063 - val_loss: 1.0853 - val_accuracy: 0.4775
Epoch 39/40
122/122 [==============================] - 2s 13ms/step - loss: 0.8328 - accuracy: 0.6904 - val_loss: 63.3661 - val_accuracy: 0.7550
```

RMS prop 알고리즘

```
122/122 [==============================] - 1s 12ms/step - loss: 0.5287 - accuracy: 0.7715 - val_loss: 0.5685 - val_accuracy: 0.7369
Epoch 39/40
122/122 [==============================] - 1s 12ms/step - loss: 0.5301 - accuracy: 0.7597 - val_loss: 0.5297 - val_accuracy: 0.7592
Epoch 40/40
122/122 [==============================] - 2s 13ms/step - loss: 0.5248 - accuracy: 0.7643 - val_loss: 0.5221 - val_accuracy: 0.7544
```

Adam 알고리즘

```
Epoch 39/40
122/122 [==============================] - 2s 18ms/step - loss: 0.7670 - accuracy: 0.6911 - val_loss: 1.1178 - val_accuracy: 0.5057
Epoch 40/40
122/122 [==============================] - 2s 17ms/step - loss: 0.9225 - accuracy: 0.6726 - val_loss: 0.6339 - val_accuracy: 0.7321
```

Momentom 알고리즘

```
122/122 [==============================] - 2s 17ms/step - loss: 0.5117 - accuracy: 0.7643 - val_loss: 761.3763 - val_accuracy: 0.7844
Epoch 39/40
122/122 [==============================] - 2s 17ms/step - loss: 0.5131 - accuracy: 0.7638 - val_loss: 1448.8942 - val_accuracy: 0.7850
Epoch 40/40
122/122 [==============================] - 2s 17ms/step - loss: 0.5224 - accuracy: 0.7715 - val_loss: 0.5013 - val_accuracy: 0.7760
```

RMS prop 알고리즘

```
Epoch 39/40
122/122 [==============================] - 2s 19ms/step - loss: 0.4543 - accuracy: 0.7973 - val_loss: 18392320.0000 - val_accuracy: 0.7886
Epoch 40/40
122/122 [==============================] - 2s 18ms/step - loss: 0.4587 - accuracy: 0.8006 - val_loss: 295908736.0000 - val_accuracy: 0.7850
```

Cancernet -기타시도

모델 깊게 쌓기

```
cancernet.add(SeparableConv2D(64, (3, 3), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(MaxPooling2D(pool_size=(2, 2)))
cancernet.add(Dropout(0.25))

# (CONV => RELU => POOL) * 3
cancernet.add(SeparableConv2D(128, (3, 3), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(128, (3, 3), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(128, (3, 3), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(MaxPooling2D(pool_size=(2, 2)))
cancernet.add(Dropout(0.25))

# (CONV => RELU => POOL) * 4
cancernet.add(SeparableConv2D(256, (3, 3), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(256, (3, 3), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(SeparableConv2D(256, (3, 3), padding="same"))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(MaxPooling2D(pool_size=(2, 2)))
cancernet.add(Dropout(0.25))

# first (and only) set of FC => RELU layers
cancernet.add(Flatten())
cancernet.add(Dense(512))
cancernet.add(Activation("relu"))
cancernet.add(BatchNormalization())
cancernet.add(Dropout(0.5))

# softmax classifier
```

Adam 알고리즘

```
Epoch 38/40
122/122 [==============================] - 2s 15ms/step - loss: 5.5825 - accuracy: 0.6440 - val_loss: 19755.6934 - val_accuracy: 0.4901
Epoch 39/40
122/122 [==============================] - 2s 16ms/step - loss: 4.1933 - accuracy: 0.6303 - val_loss: 5253.1392 - val_accuracy: 0.5988
Epoch 40/40
122/122 [==============================] - 2s 15ms/step - loss: 1.1418 - accuracy: 0.6765 - val_loss: 0.7470 - val_accuracy: 0.7459
```

Momentom 알고리즘

```
122/122 [==============================] - 2s 16ms/step - loss: 0.6685 - accuracy: 0.6448 - val_loss: 1030328157133078528000.0000 - val_accuracy: 0.6799
Epoch 39/40
122/122 [==============================] - 2s 15ms/step - loss: 0.6445 - accuracy: 0.6579 - val_loss: 4058873034448589291520.0000 - val_accuracy: 0.6847
Epoch 40/40
122/122 [==============================] - 2s 15ms/step - loss: 0.6608 - accuracy: 0.6515 - val_loss: 4423851081437670277120.0000 - val_accuracy: 0.6817
```

RMS prop 알고리즘

```
Epoch 38/40
122/122 [==============================] - 2s 17ms/step - loss: 0.4761 - accuracy: 0.7929 - val_loss: 26847586304.0000 - val_accuracy: 0.7928
Epoch 39/40
122/122 [==============================] - 2s 17ms/step - loss: 0.4751 - accuracy: 0.7849 - val_loss: 1254176352894976.0000 - val_accuracy: 0.7916
Epoch 40/40
122/122 [==============================] - 2s 17ms/step - loss: 0.4824 - accuracy: 0.7839 - val_loss: 22267457662091264.0000 - val_accuracy: 0.7940
```

Cancernet -기타시도

모델 얕게 쌓기

```
with tf.device('/device:GPU:0'):
  cancernet = Sequential()
  inputShape = (50, 50, 3)

  # CONV => RELU => POOL
  cancernet.add(SeparableConv2D(32, (3, 3), padding="same",
    input_shape=inputShape))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(MaxPooling2D(pool_size=(2, 2)))
  cancernet.add(Dropout(0.25))

  # (CONV => RELU => POOL) * 2
  cancernet.add(SeparableConv2D(64, (3, 3), padding="same"))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(SeparableConv2D(64, (3, 3), padding="same"))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(MaxPooling2D(pool_size=(2, 2)))
  cancernet.add(Dropout(0.25))

  # (CONV => RELU => POOL) * 3
  cancernet.add(SeparableConv2D(128, (3, 3), padding="same"))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(SeparableConv2D(128, (3, 3), padding="same"))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(MaxPooling2D(pool_size=(2, 2)))
  cancernet.add(Dropout(0.25))

  # first (and only) set of FC => RELU layers
  cancernet.add(Flatten())
  cancernet.add(Dense(128))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(Dropout(0.5))

  # softmax classifier
  cancernet.add(Dense(2))
  cancernet.add(Activation("softmax"))
```

```
with tf.device('/device:GPU:0'):
  cancernet = Sequential()
  inputShape = (50, 50, 3)

  # CONV => RELU => POOL
  cancernet.add(SeparableConv2D(32, (3, 3), padding="same",
    input_shape=inputShape))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(MaxPooling2D(pool_size=(2, 2)))
  cancernet.add(Dropout(0.25))

  # (CONV => RELU => POOL) * 2
  cancernet.add(SeparableConv2D(64, (3, 3), padding="same"))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(SeparableConv2D(64, (3, 3), padding="same"))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(MaxPooling2D(pool_size=(2, 2)))
  cancernet.add(Dropout(0.25))

  # first (and only) set of FC => RELU layers
  cancernet.add(Flatten())
  cancernet.add(Dense(64))
  cancernet.add(Activation("relu"))
  cancernet.add(BatchNormalization())
  cancernet.add(Dropout(0.5))

  # softmax classifier
  cancernet.add(Dense(2))
  cancernet.add(Activation("softmax"))
```

Adam 알고리즘

```
122/122 [==============================] - 2s 16ms/step - loss: 0.6099 - accuracy: 0.7146 - val_loss: 133490.0938 - val_accuracy: 0.3970
Epoch 39/40
122/122 [==============================] - 2s 17ms/step - loss: 0.6401 - accuracy: 0.7125 - val_loss: 0.6684 - val_accuracy: 0.7556
Epoch 40/40
122/122 [==============================] - 2s 15ms/step - loss: 0.6470 - accuracy: 0.7210 - val_loss: 19609.6895 - val_accuracy: 0.7592
```

Momentom 알고리즘

```
122/122 [==============================] - 2s 13ms/step - loss: 0.5131 - accuracy: 0.7674 - val_loss: 49.3750 - val_accuracy: 0.7760
Epoch 39/40
122/122 [==============================] - 2s 14ms/step - loss: 0.5098 - accuracy: 0.7679 - val_loss: 17.1214 - val_accuracy: 0.7610
Epoch 40/40
122/122 [==============================] - 2s 14ms/step - loss: 0.5045 - accuracy: 0.7674 - val_loss: 25.4026 - val_accuracy: 0.7688
```

RMS prop 알고리즘

```
122/122 [==============================] - 2s 16ms/step - loss: 0.4269 - accuracy: 0.8199 - val_loss: 9.0367 - val_accuracy: 0.8072
Epoch 39/40
122/122 [==============================] - 2s 15ms/step - loss: 0.4259 - accuracy: 0.8189 - val_loss: 0.4779 - val_accuracy: 0.8066
Epoch 40/40
122/122 [==============================] - 2s 15ms/step - loss: 0.4217 - accuracy: 0.8181 - val_loss: 0.4451 - val_accuracy: 0.8084
```

Adam 알고리즘

```
122/122 [==============================] - 1s 11ms/step - loss: 0.5505 - accuracy: 0.7602 - val_loss: 3.2200 - val_accuracy: 0.5021
Epoch 39/40
122/122 [==============================] - 1s 11ms/step - loss: 0.5880 - accuracy: 0.7437 - val_loss: 1.1562 - val_accuracy: 0.5297
Epoch 40/40
122/122 [==============================] - 1s 11ms/step - loss: 0.5723 - accuracy: 0.7576 - val_loss: 1.6178 - val_accuracy: 0.5105
```

Momentom 알고리즘

```
122/122 [==============================] - 2s 12ms/step - loss: 0.4583 - accuracy: 0.8004 - val_loss: 0.9423 - val_accuracy: 0.7868
Epoch 39/40
122/122 [==============================] - 2s 12ms/step - loss: 0.4644 - accuracy: 0.7996 - val_loss: 1.2216 - val_accuracy: 0.7886
Epoch 40/40
122/122 [==============================] - 2s 13ms/step - loss: 0.4602 - accuracy: 0.8024 - val_loss: 1.0882 - val_accuracy: 0.7988
```

RMS prop 알고리즘

```
Epoch 39/40
122/122 [==============================] - 1s 12ms/step - loss: 0.3174 - accuracy: 0.8635 - val_loss: 0.5432 - val_accuracy: 0.7892
Epoch 40/40
122/122 [==============================] - 2s 13ms/step - loss: 0.3227 - accuracy: 0.8640 - val_loss: 0.5100 - val_accuracy: 0.7862
```

모델 성능 높이기

Cancernet

• 데이터 증식

ImageDataGenerator 사용
rotation_range=20,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True

→ 성능이 증가하기 보다는 변동 폭이 줄어 안정된 모습을 보임

• BreakHis 사전학습

- 베이지안 최적화로 fine_tune_at(고정시킬 사전학습 층 개수)과 learning_rate 결정

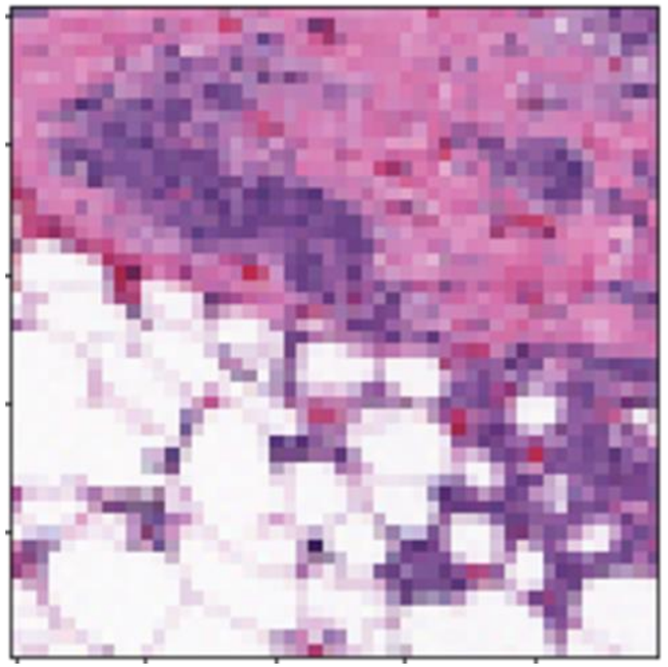
→ fine_tune_at = 5, learning_rate=0.0346 에서 가장 큰 값

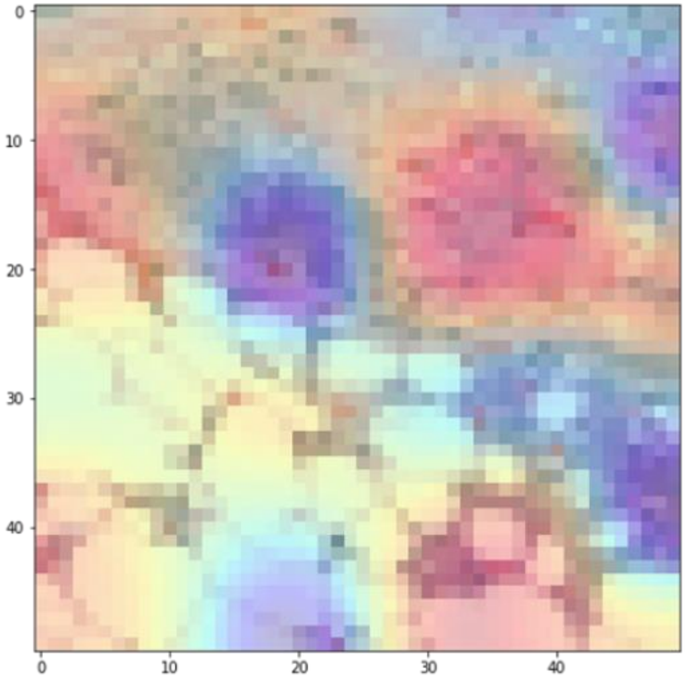
Train	Valid	Test
08211	0.8176	0.7892

모델 시각화

클래스 활성화 맵 시각화

: 특정 출력 클래스에 대해 입력 이미지의 모든 위치에 대해 계산된 2D 점수 그리드





→ 컨브넷의 최종 분류 결정에 기여한 부분일수록 빨간색을 띔

THANK YOU
FOR WATCHING

숙탯 2기 딥러닝 프로젝트
-
유방암 조직 이미지를 이용한
침윤성 유관암 분류