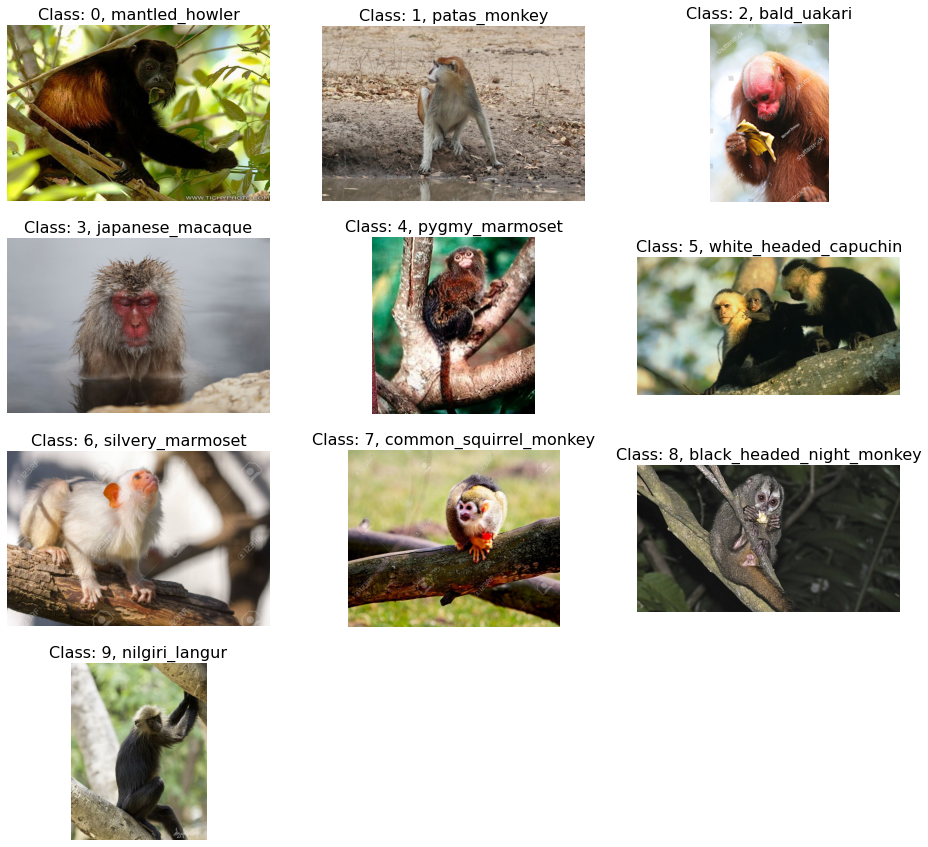
# 데이터 소개 및 EDA

이 데이터는 10종류의 원숭이들 이미지로 이루어져 있으며 이 10 종류의 원숭이를 구분하고자 한다. 데이터의 형태는 다음과 같다. 데이터 개수에 약간의 차이는 있어도 불균형하지는 않다. 하지만 전체적으로 데이터 수가 적은 편이다.



Figure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Label | Latin Name | Common Name | Train Images | Validation Images |
| n0 | alouatta\_palliata | mantled\_howler | 131 | 26 |
| n1 | erythrocebus\_patas | patas\_monkey | 139 | 28 |
| n2 | cacajao\_calvus | bald\_uakari | 137 | 27 |
| n3 | macaca\_fuscata | japanese\_macaque | 152 | 30 |
| n4 | cebuella\_pygmea | pygmy\_marmoset | 131 | 26 |
| n5 | cebus\_capucinus | white\_headed\_capuchin | 141 | 28 |
| n6 | mico\_argentatus | silvery\_marmoset | 132 | 26 |
| n7 | saimiri\_sciureus | common\_squirrel\_monkey | 142 | 28 |
| n8 | aotus\_nigriceps | black\_headed\_night\_monkey | 133 | 27 |
| n9 | trachypithecus\_johnii | nilgiri\_langur | 132 | 26 |
| Total |  |  | 1370(1098) | 272 |

# 기본 CNN

먼저 가장 기본적인 CNN모델로 데이터를 학습시켜 보았다. 채널을 순차적으로 늘려 마지막에는MLP Dense를 100을 주어 학습시켰다. 과적합이 다소 발생하였으나 Valid accuracy는 기본적으로 70%를 넘기지 못하였다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1st | 2nd Dense 1000추가 | 3rd | 4th | 5th |
| Conv2D | 32-64-128-256 | 32-64-128-256 | 32-64-128-256 | 32-64-128-256 | 32-64-128-256 |
| Kernel | 3 | 3 | 3 | 2 | 2 |
| Maxpooling | 2-2-2 | 2-2-2 | 2-2-2 | 3-3-3 | 2-3-2 |
| Dropout | 0 | 0 | 0.5 | 0.3 | 0.3 |
| Train Acc | 0.8202 | 0.9219 | 0.6006 | 0.5361 | 0.7377 |
| Valid Acc | 0.6641 | 0.6367 | 0.4688 | 0.5742 | 0.6250 |

# VGG16 이전학습

## VGG16

VGG16은 imagenet이라는 데이터를 학습시킨 모델이지만, 입력층에 가까운 convolution층은 이미지의 엣지 등 기초적인 이미지의 특성을 추출한다는 사실에 입각해, 먼저 학습되어있는 vgg모델의 모수를 그대로 써보았다. 먼저 VGG16을 미세조정없이 이전학습을 했을 때의 결과는 다음과 같다. 과적합이 계속 발생하였으나 최종적으로는 77%의 valid accuracy정도로 과적합을 어느정도 줄일 수 있었다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| VGG16 | 1st | 2nd | 3rd | 4th | 5th | 6th |
| Dense | 10 | 256-10 | 10 | 256-10 | 256-10 | 256-10 |
| Dropout | 0 | 0.7 | 0.5 | 0.5 | 0.4-0.4 | 0.6 |
| Train Acc | 1 | 0.5518 | 0.9824 | 0.9067 | 0.9325 | 0.7994 |
| Valid Acc | 0.7978 | 0.7463 | 0.7831 | 0.7831 | 0.7757 | 0.7757 |

## VGG16 미세조정

Vgg16에서 출력층에 가까운 은닉층은 학습하여 모수를 조정하였다. 첫번째 시도에서 Optimizer를 RMSprop으로 하자 정확도가 매우 낮아져, 이후로는 adam으로 진행하였다. 또한 Vgg16의 block5\_conv1부터 학습했을 때, 여러 번의 튜닝 후에도 정확도가 잘 나오지 않아 block\_conv3부터 학습하였다. L1, L2규제화와 Dropout을 적절히 사용하자 과적합이 어느정도 해결할 수 있었으며 vgg16 이전학습에서 가장 좋은 모델을 얻을 수 있었다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| VGG 16 | 1st[[1]](#footnote-1) | 2nd | 3rd | 4th | … | 16th |
| Dense | 256->10 | 256->10 | 256->10 | 10 | **…** | 256->10 |
| Dropout | 0.6 | 0.6 | 0.8, 0.9 | 0.8 | **…** | 0.8, 0.8 |
| L1,L2 |  |  |  |  |  | 0.05, 0.01 |
| Train Acc | 0.3430 | 0.9963 | 0.6922 | 0.9338 | **…** | 0.8623 |
| Valid Acc | 0.1344 | 0.8419 | 0.7353 | 0.8199 | **…** | 0.8235 |

# 자료 증대

데이터를 살펴보았을 때 전체 수가 2천개가 안되는 숫자로 아주 적음을 알 수 있었다. 위의 모델들의 성능을 높이기 위해 다음과 같은 자료증대 옵션을 사용하였으며, 그 결과로는 밑의 그림과 같은 자료가 생성되었다. 다만 결론부터 말하자면 이 데이터에서는 자료증대를 했을 때 이전의 결과와 비슷하거나 더 좋지 않은 결과를 보여주었다.



Figure

|  |
| --- |
| train\_datagen=ImageDataGenerator(rescale=1./255, rotation\_range=40,  width\_shift\_range=0.1,                                  height\_shift\_range=0.1,  shear\_range=0.2, zoom\_range=0.2,                                   horizontal\_flip=True,  fill\_mode='nearest') |

## 

## VGG16 이전학습

1000장 정도의 자료를 더 생성하여 진행하였으며 오히려 더 안 좋은 성능을 보였다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| VGG 미세조정X | 1st 1000추가 | 2nd 1000추가 | 3rd 1000 추가 | 4th 1000 추가 | 5th 1000 추가 |
| Conv2D | 256-10 | 512-10 | 512-256 | 128 | 128 |
| Dropout | 0.2 | 0.2 | 0.3-0.4 | 0.1 | 0.15 |
| Train Acc | 0.7645 | 0.8156 | 0.6877 | 0.7734 | 0.7678 |
| Valid Acc | 0.7390 | 0.7096 | 0.7059 | 0.7206 | 0.7022 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| VGG 미세조정O | 1st 1000추가 | 2nd 1000추가 | 3rd 1000 추가 | 4th 1000 추가 | 5th 1000 추가 |
| Conv2D | 256-10 | 256-10 | 512 | 128 | 128 |
| Dropout | 0.2 | 0.4-0.4 | 0.2 | 0.1 | 0.3 |
| Train Acc | 0.6900 | 0.6511 | 0.7708 | 0.8032 | 0.7952 |
| Valid Acc | 0.6544 | 0.7022 | 0.6985 | 0.7390 | 0.7353 |

## VGG16 with 미세조정

미세조정한 모델 또한 자료증대와 이전학습을 함께한 모델보다는 나아졌지만, 자료증대를 하지 않고 미세조정한 모델보다는 좋지 않은 결과를 보였다.

# 잔차연결

잔차연결은 초기의 은닉층 정보손실을 방지하기 위해 초기 은닉층을 후반부 은닉층에 직접 연결하는 기법이다. 위의 VGG 이전학습에서 80%이상의 정확도를 보이며 좋은 결과를 얻을 수 있었으므로 잔차연결을 시도하였고 그 결과는 다음과 같았다.

|  |  |  |  |
| --- | --- | --- | --- |
| Resid Connect | 1st | 2nd | 3rd |
| Conv2D | 256-10 | 1024-256-10 | 10 |
| Dropout | 0.6 – 0.6 | 0.7-0.7 | 0.7 |
| L1, L2 |  | 0.05 – 0.01 |  |
| Train Acc | 0.1199 | 1 | 0.9822 |
| Valid Acc | 0.1103 | 0.3409 | 0.3068 |

첫번째 시도에서 아주 낮은 정확도를 보였다. 이후 조금 해결하기 위해 사진의 밝기를 조절하는등의 pre-data processing 과정을 거쳐 두 번째, 세 번째 모델에 적용하였으나 과적합이 심각하였고 세번째 모델 이후 과적합 해결을 위해 node를 추가하자 램이 부족하여 중간에 멈춰버리는 현상이 일어났다. VGG 이전학습에서의 잔차연결은 너무 무거워 과적합 문제가 심각하여 데이터에 맞지 않다고 결론 내렸다.

# ResNet – 50

ResNet은 2015년에 ILSVRC에서 우승을 차지한, 마이크로소프트에서 개발한 알고리즘이다. Goolgenet이라고 불리는 Inception(v1)에 비해 7배나 깊다. 층을 더 깊게 주면 성능이 좋아지지만, 초기 은닉층의 정보를 손실하는 것을 방지하기 위해 잔차연결을 접목시킨 것이다(Skip connection과 residual connection은 같은 의미로 쓰인다). 앞서 vgg16을 수동으로 잔차연결을 하였을 때 아주 좋지 않은 성능을 보였으나, 미리 학습된 ResNet을 이용하면 어떨지 알아보고자 한다. 이 과정에서, ImageDataGenerator(rescale=1./255)로 설정하였을때, 0.1정도의 정확도가 나오는 것을 확인한 후, 두번째 시도부터 이를 preprocessing\_function = preprocess\_input으로 설정을 바꿔주었다. 그러자 훨씬 좋은 결과를 얻을 수 있었다. 앞선 VGG모델에서도 ImageDataGenerator (rescale=1./255)을 preprocessing\_function으로 바꿔보았으나 큰 차이는 없었다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ResNet | 1st | 2nd | 3rd | 4th | 5th |
| Conv2D | 512-512 | 512-512 | 512-256 | 512-256 | 512-256 |
| Dropout | 0.5-0.5 | 0.5-0.5 | 0.6-0.5 | 0.6-0.55 | 0.65-0.55 |
| Rescale | 1.255 | Preprocess | Preprocess | Preprocess | Preprocess |
| Train Acc | 0.1374 | 0.9740 | 0.9519 | 0.9507 | 0.9390 |
| Valid Acc | 0.1107 | 0.9265 | 0.9449 | 0.9228 | 0.8676 |

# 

# 병렬형 inception V3

## InceptionV3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| InceptionV3 | 1st | 2nd | 3rd | 4th | 5th |
| Conv2D | 512-10 | 1024-512-10 | 256 | 1024-512 | 1024-512 |
| Dropout | 0.6 | 0.6 | 0.6 | 0.6 | 0.5 |
| L1, L2 | - | - | - | 0.01/0.001 | 0.001 / 0.001 |
| Train Acc | 0.7634 | 0.5555 | 0.8382 | 0.6562 | 0.6599 |
| Valid Acc | 0.7610 | 0.6875 | 0.7610 | 0.7022 | 0.6691 |

## 

## InceptionV3 미세조정

|  |  |  |
| --- | --- | --- |
| InceptionV3 | 1st | 2nd |
| Conv2D | 512-10 | 1024 |
| Dropout | 0.6 | 0.7 |
| L1, L2 | - | - |
| Train Acc | 0.7052 | 0.7357 |
| Valid Acc | 0.7243 | 0.7316 |

뒤에서 9번째 블럭까지 학습할 수 있도록 설정하였고, 다음과 같이 과적합 문제없이 잘 적합되었다. 다만, VGG16 이전학습이 inceptionV3보다 좋은 결과를 보여주었다.

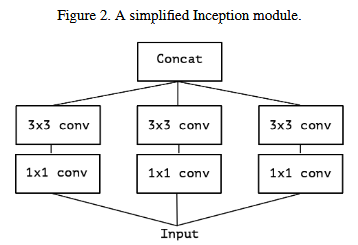
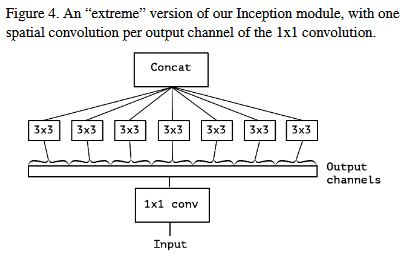
위의 inceptionV3에서 ImageDataGenerator(rescale=1./255)를 앞선 ResNet과 같이preprocessing \_function으로 바꿔 주었으나 큰 차이는 없었다.

# Xception

InceptionV1에 기초를 두고 나온 Xception을 적용해보았다. Inception이 channel의 correlation과spatial correlation을 분리시키고 1x1 컨볼루션을 활용해 연산량을 줄여 깊은 모델을 만드는 것이었다면, Xception은 이 둘을 완전히 분리시킨다.

Figure 4를 보면, input에 대해 1x1 컨볼루션을 거친 후, 모든 채널을 분리시켜 채널 당 3x3 컨볼루션을 실행한다. 이렇게 하면 channel, spatial이 완전히 분리가 된다[[2]](#footnote-2). 이러한 Xception을 적용시키자 지금까지의 결과 중 가장 좋은 정확도를 얻을 수 있었다.

Figure



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Xception[[3]](#footnote-3) | 1st | 2nd | 3rd | 4th | … | 7th |
| Conv2D | 512 | 1024 | 1024 | 1024 | **…** | 1024 |
| Dropout | 0.5 | 0.7 | 0.7 | 0.3 | **…** | 0.6 |
| L1, L2 | - | - | 0.05 – 0.01 | 0.001/0.001 |  | 0.001 / 0.001 |
| Train Acc | 0.9929 | 0.9828 | 0.8684 | 0.9818 | **…** | 0.9737 |
| Valid Acc | 0.9455 | 0.9364 | 0.9273 | 0.9273 | **…** | 0.9273 |

# 

# Auto Encoder & VAE

## Auto Encoder

## Denoising AE

오토인코더는 입력자료를 저차원으로 축약한 잠재변수를 입력변수로 넣지만 정보의 손실이 없도록 한다. VAE와는 다르게 X를 그대로 재현하려는 데에 목적이 있다. AutoEncoder와 Denoise AutoEncoder를 실행하였을 때 결과는 다음과 같다. 원본 데이터와 비교했을 때 상대적으로 두루뭉실한 색과 배경 정도만을 인식해내고 원숭이라는 개체의 특성까지는 잡아내지 못하여 생성된 결과는 원숭이라는 것을 알아볼 수 없는 결과를 보였다.

## VAE



Figure 5 Denoising AE

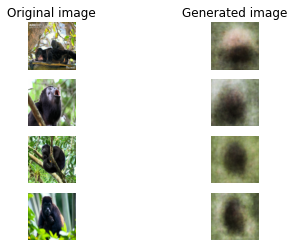
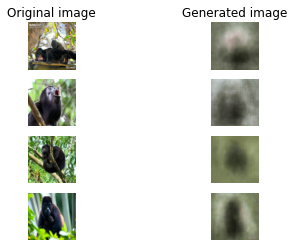
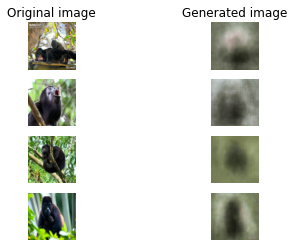
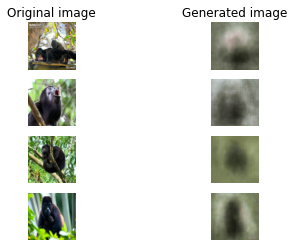
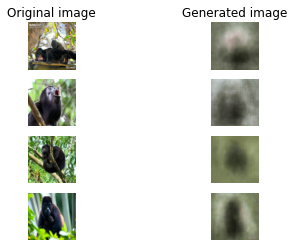
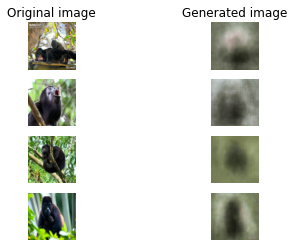
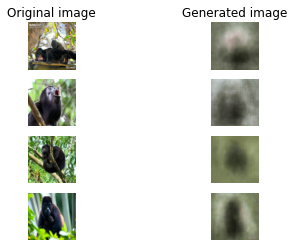
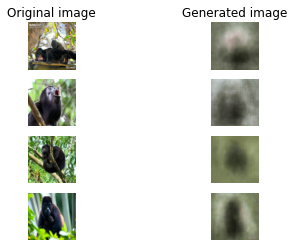


Figure 4 AutoEncoder

마찬가지로 원숭이라는 사실을 알 수 있을 정도의 그림을 생성하지는 못하였다. VAE는 Auto Encoder와 다르게 X를 그대로 재현하는 것이 아닌 비슷한 특성을 가진 새로운 X를 만들어내는 것이 목적이므로 조금 다를 수 있다는 점을 고려할 수 있다. 다만 그 점을 고려한다고 하더라도 전혀 알아볼 수 없는 결과를 얻었다.



1st

2nd

3rd

4th

5th

6th

Figure 6 VAE

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| VAE | 1st | 2nd | 3rd | 4th | 5th | 6th |
| Conv2D | 32-64 | 32-64 | 32-64 | 32-64 | 32-64 | 32-64 |
| Latent | 3 | 10 | 30 | 50 | 15 | 8 |
| L1, L2 |  |  |  | 0.001 / 0.001 |  | 0.001 / 0.001 |
| Train loss | 1279.3226 | 1173.1994 | 1269.2293 | 1567.3230 | 1292.1348 | 1567.3230 |
| Valid loss | 1283.4971 | 1197.3910 | 1261.7750 | 1558.7056 | 1280.1620 | 1558.7056 |

# 결론

위에서의 방법론을 전부 종합해 보면, ResNet이 가장 과적합의 문제가 적고 좋은 정확도를 보여주었으며 Xception의 경우 과적합이 다소 발생하였지만 가장 빠른 수렴속도를 보여주었다. 또한, 자료증대에 있어서는 크게 성능 향상에 영향이 없었다.

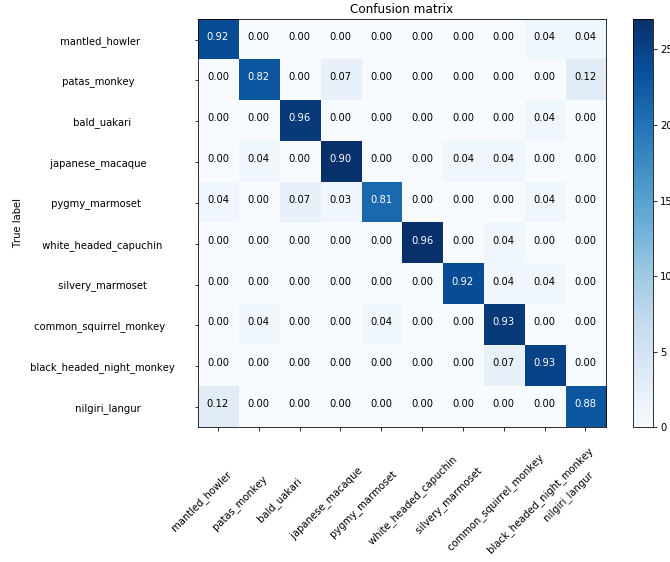


Figure 9 Confusion Matrix

Figure 7 mathled\_howler

Figure 8 nilgiri\_langur

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CNN | VGG16 | VGG16  미세조정 | 자료증대 VGG16 | 자료증대VGG16  미세조정 | ResNet | Inception | Inception  미세조정 | Xception |
| Conv2D | 32-64-128-256 | 256-10 | 256-10 | 128 | 128 | 512-256 | 512-10 | 1024 | 1024 |
| Dropout | 2 | 0.6 | 0.8, 0.8 | 0.1 | 0.3 | 0.6-0.5 | 0.6 | 0.7 | 0.6 |
| L1, L2 | 2-3-2 |  | 0.05, 0.01 |  |  |  |  | - | 0.001 / 0.001 |
| Train acc | 0.7377 | 0.7994 | 0.8623 | 0.7734 | 0.7952 | 0.9519 | 0.7634 | 0.7357 | 0.9737 |
| Valid acc | 0.6250 | 0.7757 | 0.8235 | 0.7206 | 0.7353 | 0.9449 | 0.7610 | 0.7316 | 0.9273 |

Xception의 Confusion matrix를 살펴보았을 때, mantled\_howler과 nilgiri\_langur가 약간의 혼동이 있었을 뿐 대체로 잘 분류되었음을 확인 할 수 있다.

1. VGG16의 block5\_conv1부터 학습 [↑](#footnote-ref-1)
2. Xception: Deep Learning with Depthwise Separable Convolutions [↑](#footnote-ref-2)
3. Epoch=30 [↑](#footnote-ref-3)