


A cluster of overlapping, semi-transparent geometric shapes in shades of blue, green, and red, located in the top-left corner of the slide.

# Animals-10

Multiclass Classification  
한성대학교 1971336 김태민

A cluster of overlapping, semi-transparent geometric shapes in shades of light gray, located in the bottom-right corner of the slide.



# 목차

-Data sets

-문제 정의

-코드 설명

-이론 설명

-문제점 및 해결 방안



# Data sets(Animals-10)

10개의 클래스를 가진 컬러 이미지 **26,179** Files

- ✓ Cane (4863 files)
- ✓ Cavallo (2623 files)
- ✓ Elefante (1446 files)
- ✓ Farfalla (2112 files)
- ✓ Gallina (3098 files)
- ✓ Gatto (1668 files)
- ✓ Mucca (1866 files)
- ✓ Pecora (1820 files)
- ✓ Ragno (4821 files)
- ✓ Scoiattolo (1862 files)



# Data sets

## 이미지 특성

각 클래스 별로 수천개의 이미지가 있다.  
파일은 각 **jpeg, jpg, png**로 구성 되어있다.

Raw image를 **Train**과 **test**로 파일 수정  
Test파일과 train파일의 클래스는 각 동일 test파일에는 각 클래스별 이미지 **500**개씩 담겨있다.  
이미지 별 크기는 각기 다릅니다.

기존

이름	수정된 날짜	유형
cane	2022-06-14 오후 11:39	파일 폴더
cavallo	2022-06-14 오후 11:39	파일 폴더
elefante	2022-06-14 오후 11:39	파일 폴더
farfalla	2022-06-14 오후 11:40	파일 폴더
gallina	2022-06-14 오후 11:41	파일 폴더
gatto	2022-06-14 오후 11:41	파일 폴더
mucca	2022-06-14 오후 11:42	파일 폴더
pecora	2022-06-14 오후 11:42	파일 폴더
ragno	2022-06-14 오후 11:43	파일 폴더
scotatolo	2022-06-14 오후 11:43	파일 폴더

수정 후

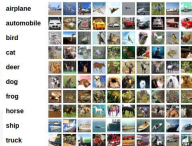
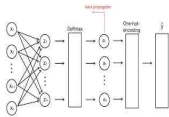
이름	수정된 날짜	유형	크기
raw-img	2022-06-14 오후 11:35	파일 폴더	
test	2022-06-14 오후 11:38	파일 폴더	
train	2022-06-14 오후 11:38	파일 폴더	
translate	2019-12-12 오후 8:30	python file	165

이름	수정된 날짜	유형
cane	2022-06-14 오후 11:39	파일 폴더
cavallo	2022-06-14 오후 11:39	파일 폴더
elefante	2022-06-14 오후 11:39	파일 폴더
farfalla	2022-06-14 오후 11:40	파일 폴더
gallina	2022-06-14 오후 11:41	파일 폴더
gatto	2022-06-14 오후 11:41	파일 폴더
mucca	2022-06-14 오후 11:42	파일 폴더
pecora	2022-06-14 오후 11:42	파일 폴더
ragno	2022-06-14 오후 11:43	파일 폴더
scotatolo	2022-06-14 오후 11:43	파일 폴더

# 문제 정의

## Multiclass Classification



해결하고자 하는 문제는 다중 분류입니다.

총 10개의 클래스를 분류하는 문제로 정의하겠습니다.

기법은 딥러닝 기법을 이용하며 쓰이는 주요 알고리즘은  
이미지 전처리를 위한 `imagedata_generator`

이미지를 효과적으로 처리하기 위한 `Conv2d`, `Max_pooling`  
안정화를 위한 `Batch Normalization`

옵티마이저는 `Adam`을 사용합니다.

과대적합을 방지하기 위해 `Dropout`을 사용합니다.

최종 출력층에서는 `softmax`를 이용해 각 클래스별의 값을 출력하여  
선택합니다.

클래스는

[cane, cavallo, elefante, farfalle, gallina, gatto, mucca, pecora, ragno, scoi  
attolo]가 있습니다.

# Code

## Multiclass Classification(직접 구현)

```
In [1]: import keras
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers
from keras.layers import Dense, Flatten, Input
from keras.models import Model, Sequential
```

필요한 모듈들을 import 합니다.

기본적인 keras,numpy,tensorflow,matplotlib를 import 합니다.

이미지 전처리를 위한 ImageDataGenerator import 합니다.

기본적인 layers구성 및 딥러닝을 사용하기위해 각종 딥러닝 계층 들을 import합니다.

# Code

## Multiclass Classification

```
In [3]: train_datagen = ImageDataGenerator( rescale = 1.0/255. )  
        test_datagen = ImageDataGenerator( rescale = 1.0/255. )  
        train_dir = 'D:\Animals-10\train'  
        test_dir = 'D:\Animals-10\test'
```

ImageDataGenerator 객체를 생성합니다. 매개 변수로 rescale = 1.0/255 하여 0~255RGB계수들을 0~1로 축소시킵니다.

Train데이터 셋과 test데이터셋의 경로를 지정하기 위해 각 파일의 경로를 지정해 줍니다

이것은 나중에 객체에 담겨질 예정입니다.

# Code

## Multiclass Classification

```
In [4]: train_generator = train_datagen.flow_from_directory(train_dir,
                                                         batch_size=20,
                                                         class_mode='categorical',
                                                         target_size=(150, 150))
test_generator = test_datagen.flow_from_directory(test_dir,
                                                  batch_size=20,
                                                  class_mode='categorical',
                                                  target_size=(150, 150))
```

Found 21179 images belonging to 10 classes.  
Found 5000 images belonging to 10 classes.

다시 한번 두 객체를 `flow_from_directory`를 이용하여 연결 해줍니다.  
각 객체의 첫번째 매개변수는 파일 경로를 입력 해줍니다.  
배치 사이즈로는 한번에 반환하는 개수 입니다.

`Class_mode`는 `binary`랑 `categorical`이 있는데 이진분류가 아니므로 `categorical`을 입력 해줍니다.

`Target_size`는 반환할 이미지의 크기 입니다. 컬러이므로  
(150,150,3)의 사이즈가 자동적으로 20개씩 반환됩니다.

찾은 개수로 `test,train` 각각 21179개, 5000개의 이미지와 10개의 클래스를 찾았습니다.



# Code

## Multiclass Classification

```
In [5]: model = keras.Sequential()  
model.add(keras.Input(shape=(150,150,3)))  
model.add(keras.layers.Conv2D(256,kernel_size=3,activation='relu',padding='same',input_shape=(150,150,3)))  
model.add(keras.layers.MaxPooling2D(2))  
model.add(keras.layers.BatchNormalization())  
model.add(keras.layers.Conv2D(128,kernel_size=3,activation='relu',padding='same'))  
model.add(keras.layers.MaxPooling2D(2))  
model.add(keras.layers.BatchNormalization())  
model.add(keras.layers.Conv2D(64,kernel_size=3,activation='relu',padding='same'))  
model.add(keras.layers.MaxPooling2D(2))  
model.add(keras.layers.BatchNormalization())  
model.add(keras.layers.Conv2D(32,kernel_size=3,activation='relu',padding='same'))  
model.add(keras.layers.MaxPooling2D(2))  
model.add(keras.layers.BatchNormalization())  
model.add(keras.layers.Conv2D(16,kernel_size=3,activation='relu',padding='same'))  
model.add(keras.layers.MaxPooling2D(2))  
model.add(keras.layers.BatchNormalization())  
model.add(keras.layers.Flatten())  
model.add(keras.layers.Dense(100,activation='relu'))  
model.add(keras.layers.Dropout(0.4))  
model.add(keras.layers.Dense(10,activation='softmax'))
```

우리가 학습시킬 모델 입니다.

Model = keras.Sequential()로 모델 객체를 생성해줍니다.

Keras.input으로 입력시킬 사이즈를 정합니다.

이미지의 전처리로 (150,150,3)이므로 동일하게 해줍니다.



# Code

## Multiclass Classification

모델에 대해 계속 설명하겠습니다. (기능에 대한 세부 설명은 이론 설명에 하겠습니다.)

```
Conv2D(256, kernel_size=3, activation='relu', padding='same', input_shape =  
(150, 150, 3))
```

컨볼루션 레이어를 생성해줍니다. 256개의 필터를 생성합니다. 커널 사이즈는 (3,3)으로 설정해줍니다. 활성화 함수로는 relu 함수를 쓰고 입력과 출력이 동일하게 padding을 same으로 둡니다. 우리가 여기서 주로 학습시키는 것은 필터를 학습시킨다고 생각해도 됩니다.

```
MaxPooling2D(2)
```

MaxPooling2D 레이어를 생성합니다. (2,2)의 크기로 Pooling을 진행했기 때문에 크기는 절반으로 줄어듭니다. (150,150) -> (75,75) Pooling 같은 경우에는 학습시킬 가중치가 없습니다. 도장을 찍으면서 각 픽셀의 최대 값을 뽑아냅니다.

번외로 AveragePooling2D가 있습니다.



# Code

## Multiclass Classification

BatchNormalization()입니다. 매개변수는 없습니다.

입력과 출력은 같으며 안정적인 학습을 위해 추가하였습니다.

BatchNormalization같은 경우는 이론 설명 때 집중적으로 살펴보겠습니다.

Conv2D의 필터의 개수만 조정하면서 위의 코드를 반복합니다.

```
model.add(keras.layers.Conv2D(256, kernel_size=3, activation='relu', padding='same', input_shape=(150, 150, 3)))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(128, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(16, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.BatchNormalization())
```



# Code

## Multiclass Classification

```
model.add(keras.layers.Flatten())  
model.add(keras.layers.Dense(100, activation='relu'))  
model.add(keras.layers.Dropout(0.4))  
model.add(keras.layers.Dense(10, activation='softmax'))
```

모델의 마지막 부분입니다.

Flatten()함수는 현재의 2d 이미지를 일렬로 펼칩니다.


Dense()는 100개의 뉴런을 만듭니다 활성화 함수로 relu를 사용합니다.

이로써 마치 100차원으로 압축되는 효과가 발생합니다.

Dropout은 train 데이터의 과대 적합을 방지하기 위해 넣었습니다.

0.4는 입력의 약 40%를 드롭아웃시켜 과대적합을 방지합니다.

마지막으로 10개의 뉴런을 softmax함수를 통해 10개의 클래스를 예측하도록 만듭니다.



# Code

## Multiclass Classification

```
In [6]: model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 150, 150, 256)	7168
max_pooling2d (MaxPooling2D)	(None, 75, 75, 256)	0
batch_normalization (Batch Normalization)	(None, 75, 75, 256)	1024
conv2d_1 (Conv2D)	(None, 75, 75, 128)	295040
max_pooling2d_1 (MaxPooling2D)	(None, 37, 37, 128)	0
batch_normalization_1 (Batch Normalization)	(None, 37, 37, 128)	512
conv2d_2 (Conv2D)	(None, 37, 37, 64)	75792
max_pooling2d_2 (MaxPooling2D)	(None, 18, 18, 64)	0
batch_normalization_2 (Batch Normalization)	(None, 18, 18, 64)	256
conv2d_3 (Conv2D)	(None, 18, 18, 32)	18464
max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 32)	0
batch_normalization_3 (Batch Normalization)	(None, 9, 9, 32)	128
conv2d_4 (Conv2D)	(None, 9, 9, 16)	4624
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 16)	0
batch_normalization_4 (Batch Normalization)	(None, 4, 4, 16)	64
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 100)	25700
dropout (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 10)	1010
Total params: 427,762		
Trainable params: 426,750		
Non-trainable params: 992		

Model.summary()를 통해 우리가 만든 모델을

볼 수가 있습니다. 모델에 대해 좀 더 집중적으로 파헤쳐 보겠습니다.

첫번째 파라미터 같은 경우는 Conv2d아까 우리가 학습시켜야 하는 필터입니다. 필터는 (3,3)이며 컬러이미지이므로 depth는 3입니다. 따라서 총 ((3x3)x256)+256개의 파라미터가 존재합니다. 2번째 Conv2d는 ((3x3)x256x128)+128이므로 295040개의 파라미터 입니다.

Max\_pooling은 학습시킬 파라미터가 없기 때문에 0입니다. 하지만 이미지의 크기가 절반으로 감소하기때문에 (150,150)->(75,75)로 이미지가 축소된것을 확인할 수 있습니다.

Batch\_normalization 층은 파라미터가 필터개수 x 4지만 각 필터의 감마 벡터를 학습하기 때문에 실질적으로는 512라고 볼 수 있다.

# Code

## Multiclass Classification

```
In [6]: model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 150, 150, 256)	7168
max_pooling2d (MaxPooling2D)	(None, 75, 75, 256)	0
batch_normalization (Batch Normalization)	(None, 75, 75, 256)	1024
conv2d_1 (Conv2D)	(None, 75, 75, 128)	256040
max_pooling2d_1 (MaxPooling2D)	(None, 37, 37, 128)	0
batch_normalization_1 (Batch Normalization)	(None, 37, 37, 128)	512
conv2d_2 (Conv2D)	(None, 37, 37, 64)	73792
max_pooling2d_2 (MaxPooling2D)	(None, 18, 18, 64)	0
batch_normalization_2 (Batch Normalization)	(None, 18, 18, 64)	256
conv2d_3 (Conv2D)	(None, 18, 18, 32)	18464
max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 32)	0
batch_normalization_3 (Batch Normalization)	(None, 9, 9, 32)	128
conv2d_4 (Conv2D)	(None, 9, 9, 16)	4624
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 16)	0
batch_normalization_4 (Batch Normalization)	(None, 4, 4, 16)	64
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 100)	25700
dropout (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 10)	1010
Total params: 427,782		
Trainable params: 426,790		
Non-trainable params: 992		

최종적으로 flatten층을 지나게 되면 256개의 뉴런이 나오며

Dense층을 통해 파라미터는  $(256 \times 100) + 100$ 이므로 25700개가 됩니다. 이후 dropout을 통과하고 마지막 출력층은  $(100 \times 10) + 10$ 이므로 1010개의 파라미터를 가지게 됩니다.

최종 출력에 총 파라미터는 다 더한 427,782개이며  
훈련가능한 파라미터는 426,790개 훈련하지 않는 파라미터는 992개입니다.

# Code

## Multiclass Classification

```
In [7]: model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
In [8]: model.fit(train_generator, epochs=10)
```

```
Epoch 1/10
1059/1059 [=====] - 60s 60ms/step - loss: 1.7329 - accuracy: 0.4165
Epoch 2/10
1059/1059 [=====] - 44s 41ms/step - loss: 1.2812 - accuracy: 0.5645
Epoch 3/10
1059/1059 [=====] - 44s 41ms/step - loss: 1.0995 - accuracy: 0.6291
Epoch 4/10
1059/1059 [=====] - 44s 41ms/step - loss: 0.9385 - accuracy: 0.6638
Epoch 5/10
1059/1059 [=====] - 43s 41ms/step - loss: 0.9078 - accuracy: 0.6945
Epoch 6/10
1059/1059 [=====] - 43s 41ms/step - loss: 0.8357 - accuracy: 0.7186
Epoch 7/10
1059/1059 [=====] - 43s 41ms/step - loss: 0.7681 - accuracy: 0.7432
Epoch 8/10
1059/1059 [=====] - 43s 41ms/step - loss: 0.7074 - accuracy: 0.7595
Epoch 9/10
1059/1059 [=====] - 43s 41ms/step - loss: 0.6643 - accuracy: 0.7757
Epoch 10/10
1059/1059 [=====] - 44s 41ms/step - loss: 0.6029 - accuracy: 0.7992
```

```
Out[8]: <keras.callbacks.History at 0x1236980b>
```

이제 모델을 compile 합니다.

로스 함수로 다중 분류에 쓰이는 'categorical\_crossentropy'를 쓰고 옵티마이저는 무슨 옵티마이저를 쓰면 좋을지 모르겠다 그럴땐 Adam을 쓰라는 말이 잇듯이 adam으로 옵티마이저를 만들어 줍니다.

평가지표인 metrics는 정확도를 보기 위해 accuracy를 측정해줍니다.

`Model.fit(train_generator, epochs=10)`

모델을 훈련시킵니다. `Model.fit_generator`을 썼었지만 버전 오류등도 있어 그냥 `fit` 시켜줍니다. Epoch는 반복횟수입니다.

`Train_generator`을 이용해 학습을 진행합니다. 전에 만든 객체를 통해 배치 사이즈를 20개로 지정하여 전체 train 이미지 개수 21179개를 배치사이즈 20으로 나누어 총 1059개를 반환시켜줍니다. 이를 10번 반복하여 훈련합니다.

훈련이 경과함에따라 loss는 감소 accuracy는 증가하는 것을 볼 수 있습니다.

# Code

## Multiclass Classification

```
In [9]: score = model.evaluate_generator(test_generator, steps=100)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))
```

C:\ProgramData\Anaconda3\envs\python37\lib\site-packages\keras\engine\training.py:2006: UserWarning: 'Model.evaluate\_generator' is deprecated and will be removed in a future version. Please use 'Model.evaluate', which supports generators.  
warnings.warn('Model.evaluate\_generator' is deprecated and '

accuracy: 66.40%

훈련이 종료되었으면 `model.evaluate_generator`이라는 전용 스코어 측정을 쓰도록 하겠습니다.

`Train_generator`로 훈련하였으며 이로 인해 `test_generator`을 통해 점수를 매깁니다, `steps=100`은 `test_generator`에서 몇번을 꺼내서 점수를 측정하냐 입니다. 100번으로 일관성있게 점수를 측정합니다.

출력으로 `metrics_name`에 `accuracy`가 담겨있고 나온 `score`는 0번째 배열이 아닌 1번째 배열에 담겨 있습니다.

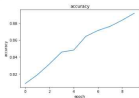
소수점으로 출력이 되기 때문에 `*100`을 통하여 출력시켜 줍니다.



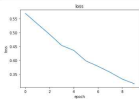
# Code

## Multiclass Classification

```
In [25]: plt.plot(history.history['accuracy'])  
plt.title('accuracy')  
plt.xlabel('epoch')  
plt.ylabel('accuracy')  
plt.show()
```



```
In [27]: plt.plot(history.history['loss'])  
plt.title('loss')  
plt.xlabel('epoch')  
plt.ylabel('loss')  
plt.show()
```



Accuracy 와 loss를 출력해보았습니다.

검증 세트를 사용하지 않아 1개의 선만 그려진 것을 볼 수 있습니다.

Epoch가 증가할수록 accuracy는 증가하고 loss는 감소하는 것을 볼 수 있습니다.

# Code

## Multiclass Classification

```
In [10]: test = model.predict(test_generator, steps=1)
```

```
In [11]: print(test.shape)
print(test[0])
```

```
(20, 10)
[6.7403900e-04 9.7406411e-01 7.3093131e-05 1.2061146e-11 8.2779167e-08
 6.8776105e-12 2.5089904e-02 1.0761617e-04 2.4673196e-12 1.4540831e-13]
```

```
In [15]: word = ['dog', 'horse', 'elefante', 'butterfly', 'chicken', 'cat', 'buffalo', 'sheep', 'soldier', 'squirrel']
#word = {'name': '강아지', 'animal': '말', 'elefante': '코끼리', 'butterfly': '나비', 'gallina': '닭', 'gatto': '고양이', 'mucca': '황소', 'pecora'}

test_generator = list(test_generator[0])
test_generator = np.array(test_generator[0])
print(test_generator.shape)
print(test_generator[0].shape)

test = model.predict(test_generator)

plt.imshow(test_generator[0])
plt.title(word[np.argmax(test[0])])
plt.show()

plt.imshow(test_generator[2])
plt.title(word[np.argmax(test[2])])
plt.show()
```

Model.predict를 통해 모델을 테스트 해봅시다.

Test\_generator, steps=1을 통하여 1개의 배치를 꺼내옵니다.


그 후 새로운 변수로 1개의 배치안에서 0번째와 2번째를 뽑아 matplotlib로 그려줍니다 이와 동일한 이미지를 모델에 넣어 동물과 이름이 매칭되는지 봅니다.





# 이론 설명

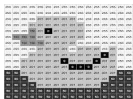
## 목차

- ImageDataGenerator
  - Convolutional Neural Network
  - Max Pooling
  - BatchNormalization
  - Dropout
  - Softmax
  - Adam
  - Categorical **Cross Entropy** Error
- 

# 이론 설명

## ImageDataGenerator

```
tf.keras.preprocessing.image.ImageDataGenerator(  
    featurewise_center=False,  
    samplewise_center=False,  
    featurewise_std_normalization=False,  
    samplewise_std_normalization=False,  
    zca_whitening=False,  
    zca_epsilon=1e-06,  
    rotation_range=0,  
    width_shift_range=0.0,  
    height_shift_range=0.0,  
    brightness_range=None,  
    shear_range=0.0,  
    zoom_range=0.0,  
    channel_shift_range=0.0,  
    fill_mode='nearest',  
    cval=0.0,  
    horizontal_flip=False,  
    vertical_flip=False,  
    rescale=None,  
    preprocessing_function=None,  
    data_format=None,  
    validation_split=0.0,  
    interpolation_order=1,  
    dtype=None  
)
```



ImageDataGenerator은 이미지 전처리를 위한 도구이다.

기본적인 기능

-Rescale

-Horizontal\_flip

-Vertical\_flip

-rotation\_range

위 3가지가 주로 많이 쓰인다.

Rescale = 1.0/255

이미지의 픽셀 값은 0~255의 범위로 정해져 있다.

이를 딥러닝 학습에 맞춰 0~1의 범위로 수정하여 학습시킨다.

아래의 오리는 이미지를 수치화 시킨 것이다. 모든 값을 0~1로 변환시킨다.

# 이론 설명

## ImageDataGenerator

```
tf.keras.preprocessing.image.ImageDataGenerator(  
    featurewise_center=False,  
    samplewise_center=False,  
    featurewise_std_normalization=False,  
    samplewise_std_normalization=False,  
    zca_whitening=False,  
    zca_epsilon=1e-06,  
    rotation_range=0,  
    width_shift_range=0.0,  
    height_shift_range=0.0,  
    brightness_range=None,  
    shear_range=0.0,  
    zoom_range=0.0,  
    channel_shift_range=0.0,  
    fill_mode='nearest',  
    cval=0.0,  
    horizontal_flip=False,  
    vertical_flip=False,  
    rescale=None,  
    preprocessing_function=None,  
    data_format=None,  
    validation_split=0.0,  
    interpolation_order=1,  
    dtype=None  
)
```

### -Horizontal\_flip

이미지를 수평으로 뒤집는다. 매개변수를 True로 지정할시 적용된다.



# 이론 설명

## ImageDataGenerator

```
tf.keras.preprocessing.image.ImageDataGenerator(  
    featurewise_center=False,  
    samplewise_center=False,  
    featurewise_std_normalization=False,  
    samplewise_std_normalization=False,  
    zca_whitening=False,  
    zca_epsilon=1e-06,  
    rotation_range=0,  
    width_shift_range=0.0,  
    height_shift_range=0.0,  
    brightness_range=None,  
    shear_range=0.0,  
    zoom_range=0.0,  
    channel_shift_range=0.0,  
    fill_mode='nearest',  
    cval=0.0,  
    horizontal_flip=False,  
    vertical_flip=False,  
    rescale=None,  
    preprocessing_function=None,  
    data_format=None,  
    validation_split=0.0,  
    interpolation_order=1,  
    dtype=None  
)
```

### -Vertical\_flip

이미지를 수직으로 뒤집는다. 매개변수를 True로 지정할시 적용된다.



# 이론 설명

## ImageDataGenerator

```
tf.keras.preprocessing.image.ImageDataGenerator(  
    featurewise_center=False,  
    samplewise_center=False,  
    featurewise_std_normalization=False,  
    samplewise_std_normalization=False,  
    zca_whitening=False,  
    zca_epsilon=1e-06,  
    rotation_range=0,  
    width_shift_range=0.0,  
    height_shift_range=0.0,  
    brightness_range=None,  
    shear_range=0.0,  
    zoom_range=0.0,  
    channel_shift_range=0.0,  
    fill_mode='nearest',  
    cval=0.0,  
    horizontal_flip=False,  
    vertical_flip=False,  
    rescale=None,  
    preprocessing_function=None,  
    data_format=None,  
    validation_split=0.0,  
    interpolation_order=1,  
    dtype=None  
)
```

-Rotation\_range

이미지의 각도를 변형시킵니다.

매개변수를 도 단위로 지정하여 이미지를 생성시킵니다.





# 이론 설명

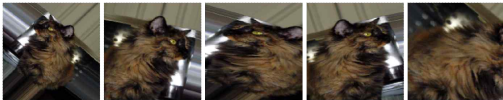
## ImageDataGenerator

```
tf.keras.preprocessing.image.ImageDataGenerator(  
    featurewise_center=False,  
    samplewise_center=False,  
    featurewise_std_normalization=False,  
    samplewise_std_normalization=False,  
    zca_whitening=False,  
    zca_epsilon=1e-06,  
    rotation_range=0,  
    width_shift_range=0.0,  
    height_shift_range=0.0,  
    brightness_range=None,  
    shear_range=0.0,  
    zoom_range=0.0,  
    channel_shift_range=0.0,  
    fill_mode='nearest',  
    cval=0.0,  
    horizontal_flip=False,  
    vertical_flip=False,  
    rescale=None,  
    preprocessing_function=None,  
    data_format=None,  
    validation_split=0.0,  
    interpolation_order=1,  
    dtype=None  
)
```

### -imageDataGenerator

위의 기능과 다른 추가적인 기능을 모았을 때 1개의 이미지로 여러 각기 다른 이미지를 생성 해낼 수 있습니다.

이는 좀 더 모델이 다양하게 학습 할 수 있고 train\_set의 과대 적합을 어느정도 방지 할 수 있다고 생각됩니다.



# 이론 설명

## ImageDataGenerator.flow\_from\_directory

```
In [4]: train_generator = train_datagen.flow_from_directory(train_dir,
                                                            batch_size=20,
                                                            class_mode='categorical',
                                                            target_size=(150, 150))
        test_generator = test_datagen.flow_from_directory(test_dir,
                                                            batch_size=20,
                                                            class_mode='categorical',
                                                            target_size=(150, 150))
```

Found 21179 images belonging to 10 classes.  
Found 5000 images belonging to 10 classes.

-imageDataGenerator.flow\_from\_directory

디렉토리 관련 함수 입니다.

인자는 directory, target\_size, batch\_size, class\_mode을 볼 수 있습니다.

그럼 위에 대해 각각 설명하겠습니다.

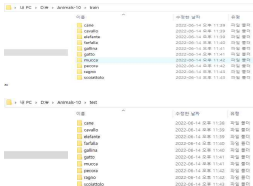
# 이론 설명

## ImageDataGenerator.flow\_from\_directory

```
In [4]: train_generator = train_datagen.flow_from_directory(train_dir,  
                                                         batch_size=20,  
                                                         class_mode='categorical',  
                                                         target_size=(150, 150))  
  
test_generator = test_datagen.flow_from_directory(test_dir,  
                                                  batch_size=20,  
                                                  class_mode='categorical',  
                                                  target_size=(150, 150))
```

Found 21179 images belonging to 10 classes.

Found 5000 images belonging to 10 classes.



이름	수정된 날짜	유형
cat	2022-06-14 오후 11:39	파일 폴더
cavallo	2022-06-14 오후 11:39	파일 폴더
elefante	2022-06-14 오후 11:39	파일 폴더
fariola	2022-06-14 오후 11:40	파일 폴더
galina	2022-06-14 오후 11:41	파일 폴더
gatto	2022-06-14 오후 11:41	파일 폴더
mucca	2022-06-14 오후 11:42	파일 폴더
pecora	2022-06-14 오후 11:42	파일 폴더
ragno	2022-06-14 오후 11:43	파일 폴더
scintobolo	2022-06-14 오후 11:43	파일 폴더

-directory

디렉토리들의 경로를 설정 해줍니다. 파일 경로의 하단에 있는 각 파일들은 클래스가 되며 이미지 개수의 총합을 출력해줍니다.

위의 코드를 실행 시 train은 21179개의 이미지 10개의 클래스, test 5000개의 이미지 10개의 클래스를 찾았다고 알려줍니다.

# 이론 설명

## ImageDataGenerator.flow\_from\_directory

```
In [4]: train_generator = train_datagen.flow_from_directory(train_dir,
                                                            batch_size=20,
                                                            class_mode='categorical',
                                                            target_size=(150, 150))

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  batch_size=20,
                                                  class_mode='categorical',
                                                  target_size=(150, 150))
```

Found 21179 images belonging to 10 classes.  
Found 5000 images belonging to 10 classes.

-batch\_size

만약 우리가 각각의 데이터마다 가중치를 수정하게 되면 어떻게 될까요??  
연산량이 높아져 학습 진행이 어려우며 컴퓨터에 무리가 갈 수 있을 것입니다.  
이를 대체하기 위해 batch\_size가 있습니다.

총 3가지로 배치 사이즈를 분류 할 수 있습니다.

일반적인 확률적 경사 하강법은 훈련세트에서 샘플을 1개씩 꺼내 조금씩 내려갑니다.

두번째 미니배치 경사 하강법으로 일정 개수의 샘플을 꺼내 조금씩 내려갑니다. 위의 코드는 미니배치 입니다.

마지막으로 배치 경사 하강법으로 모든 샘플을 꺼내 경사를 따라 내려갑니다.

이것을 매 에포크 마다 반복하여 수행합니다. 위의 코드는 이미지를 한번에 20개씩 꺼내 내려가는 것 이라고 볼 수 있습니다. 어떻게 보면 함수가 한번 수행되면 20개씩 반환된다고 볼 수 있겠네요!

# 이론 설명

## ImageDataGenerator.flow\_from\_directory

```
In [4]: train_generator = train_datagen.flow_from_directory(train_dir,
                                                         batch_size=20,
                                                         class_mode='categorical',
                                                         target_size=(150, 150))
test_generator = test_datagen.flow_from_directory(test_dir,
                                                  batch_size=20,
                                                  class_mode='categorical',
                                                  target_size=(150, 150))
```

Found 21179 images belonging to 10 classes.  
Found 5000 images belonging to 10 classes.



### -class\_mode

Class\_mode는 3개의 인자를 선택 할 수 있습니다. Categorical, sparse, binary입니다. 첫번째 categorical는 멀티 레이블 클래스 이면서 one-hot 인코딩된 형태입니다. 클래스를 각각 one-hot 인코딩 해준다고 볼 수 있습니다.

위의 그림처럼 나머지의 원소는 0이고 각 1개씩만 1인 것을 볼 수 있습니다. 위가 바로 one-hot 인코딩입니다.

컴퓨터에게 데이터의 연속성이 없다는 것을 알려주기 위하여 위를 수행합니다.

두번째는 sparse입니다. 멀티 레이블 클래스 이지만 레이블로 인코딩된 형태 입니다.

세번째는 binary입니다. 이진 분류로 0 아니면 1인 형태를 말해주고 있습니다.

# 이론 설명

## ImageDataGenerator.flow\_from\_directory

```
In [4]: train_generator = train_datagen.flow_from_directory(train_dir,
                                                            batch_size=20,
                                                            class_mode='categorical',
                                                            target_size=(150, 150))

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  batch_size=20,
                                                  class_mode='categorical',
                                                  target_size=(150, 150))
```

Found 21179 images belonging to 10 classes.  
Found 5000 images belonging to 10 classes.

-target\_size

Target\_size는 매우 간편합니다. 우리가 가진 컴퓨터로는 일반적인 고화질의 데이터를 처리하기에는 너무 버겁습니다. 설정 캐글 data\_set이라고해도 평균 (224,224)의 사이즈를 가진 이미지가 일반 적입니다.

하지만 (224,244)도 배치사이즈를 크게 줄이지 않는 이상 우리가 가진 GPU 메모리로는 한계입니다. 메모리 초과시 학습 자체가 진행되지 않습니다.

이를 해결하기 위해 이미지 사이즈를 축소 시킵니다. (224,244)->(150,150)으로 축소시켜 학습에 이용합니다. 정확도는 어느정도 떨어지겠지만 학습을 좀 더 빠르고 그리고 기능이 부족한 GPU라도 학습 시킬 수 있게 합니다.

# 이론 설명

## ImageDataGenerator

```
In [4]: train_generator = train_datagen.flow_from_directory(train_dir,
                                                            batch_size=20,
                                                            class_mode='categorical',
                                                            target_size=(150, 150))
test_generator = test_datagen.flow_from_directory(test_dir,
                                                  batch_size=20,
                                                  class_mode='categorical',
                                                  target_size=(150, 150))
```

Found 21179 images belonging to 10 classes.  
Found 5000 images belonging to 10 classes.

ImageDataGenerator을 종합해 보겠습니다.

경로를 지정하여 하위에 있는 폴더들을 자동적으로 찾아서 분류 해줍니다.

배치사이즈를 지정하여 한번 경사를 내려갈때 얼마만큼의 데이터를 쓰는지 알려주고 반환 합니다.

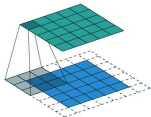
클래스 모드로 우리가 원하는 형태의 분류기법을 수행 할 수 있습니다.

타겟 사이즈로 학습속도를 개선시킬 수 있으며 학습을 가능케 만들어줍니다.

# 이론 설명

## Convolutional Neural Network

```
tf.keras.layers.Conv2D(  
    filters,  
    kernel_size,  
    strides=(1, 1),  
    padding='valid',  
    data_format=None,  
    dilation_rate=(1, 1),  
    groups=1,  
    activation=None,  
    use_bias=True,  
    kernel_initializer='glorot_uniform',  
    bias_initializer='zeros',  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    **kwargs  
)
```

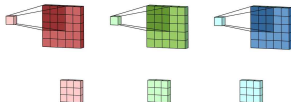


Convolution Neural Network은 이미지 데이터를 좀 더 효과적으로 학습시키기 위해 고안된 방안이다.

기본적인 기능

Filters, kernel\_size, strides, padding, activation이 주로 쓰인다.

우선 컨볼루션 연산이 무엇인지 그리고 위 4가지의 기능에 대해 하나씩 살펴 보겠습니다.





# 이론 설명

## Convolutional Neural Network

0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

1	0	1
0	1	0
1	0	1

1	4	3	4	1
1	2	4	3	3
1	2	3	4	1
1	3	3	1	1
3	3	1	1	0

컨볼루션 연산

컨볼루션 연산이 과연 무엇일까 보기 전에 아래의 GIF를 보면 빨간색과 초록색으로 된 작은 이미지가 전체 이미지를 훑고나서 그걸 바탕으로 새로운 이미지를 만들어 냈습니다.



우린 저 GIF의 Input 이미지를 왼쪽의 원본 사진이라 보고

GIF의 노란색, 빨간색 작은 이미지는 왼쪽의 파란색 이미지이며 이것을 필터로 보겠습니다.

아래 Feature Map은 왼쪽은 새로 만들어진 이미지로 보겠습니다.

왼쪽에 초록색 4는 빨간색과 파란색 이미지가 서로 곱해져서 만든 것입니다. 위를 1차원 배열로 펼쳐보겠습니다.

$[1,0,0,1,1,0,1,1,1] * [1,0,1,0,1,0,1,0,1]$

이를 각 인덱스 에 맞춰 곱한 후 모두 더하면 4라는 결과 값이 나옵니다. 이것이 픽셀의 값입니다. 이러한 행동을 반복하여 새로운 이미지를 생성해냅니다.

# 이론 설명

## Convolutional Neural Network

	2	2	2		
1	1	1	1	1	0
1	1	1	1	1	0
1	1	1	1	1	0
1	1	1	1	1	0
1	1	1	1	1	0
0	0	0	0	0	0

input

-1	0
0	-1

filter



-2	-2	-1
-2	-2	-1
-1	-1	-1

-filter

컨볼루션 연산에 대해 알아보았으니 filter가 무엇인지 알아 보겠습니다. 우리 필터를 도장이라고 부르겠습니다.

원본 이미지에 대하여 (2,2)크기의 도장을 찍습니다.  
이때 도장안의 픽셀은 숫자입니다.

만약 이 도장을 1개만 가지고 있으면 1개의 특성 맵만 나오는 것은 당연합니다. 우리가 컨볼루션 연산을 하는 이유는 무엇입니까?  
이미지의 특성을 잘 뽑아내고 싶은 것 입니다.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved  
Feature

좋은 특성, 확실히 명확한 특성을 많이 뽑아내야 합니다. 그러므로 우리는 1개의 도장만으로는 부족합니다. 도장이 많을 수록 여러가지의 특성 맵이 뽑혀질 것 입니다.

우리는 이 도장의 개수를 filter의 매개변수로 지정이 가능합니다.  
이 도장은 좀더 좋은 특성 맵을 뽑아내기 위해 학습이 가능합니다.

# 이론 설명

## Convolutional Neural Network

3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

-kernel size

커널 사이즈는 매우 간단합니다.

단순히 도장의 크기를 지정해주는 것입니다.

GIF는 (3,3)으로 커널 사이즈를 지정해서 적용시킵니다.

커널 사이즈는 곧 도장의 크기 입니다.

3,3이면 곧 9개의 파라미터로 일반 뉴럴 네트워크에서 Dense(9)와 유사하다고 볼 수 있습니다.

# 이론 설명

## Convolutional Neural Network

3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

-strides

Strides는 우리가 필터가 몇칸을 이동시킬지 결정 해줍니다.

위 는 strides를 1로 지정한 것이며 아래는 strides를 2로 지정했다고 볼 수 있습니다.

Strides에 따라 특성맵의 크기가 달라집니다.

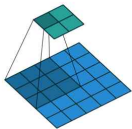
만약 위와 같이 5,5의 크기에 3,3의 필터를 적용 그리고 strides를 1로 지정 시킬 경우

가로로 우리는 3번을 이동할 수 있고 세로도 동일합니다.

그러므로 특성맵은 (3,3)입니다.

아래는 5,5크기를 3,3의 필터로 적용 시키고 strides를 2로 지정했습니다.

이럴 경우 가로로 2번이동 세로도 2번 이동이기 때문에 2,2의 특성맵이 생성됩니다.



# 이론 설명

## Convolutional Neural Network

3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

-padding

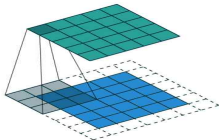
Padding은 특성 맵의 크기를 맞추어 주기 위해 주로 사용됩니다.

만약 우리가 원본 이미지랑 특성맵의 크기를 동일하게 사용하고 싶다고 가정합니다.

하지만 위 처럼 축소되는게 일반적 입니다. 이를 해결하기 위해 우리는 원본 이미지의 테두리에 0을 넣은 값을 임의로 추가합니다. 이럴 경우 원본이미지가 필터에 참여하는 기회가 증가됩니다.

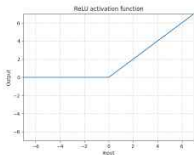
또한 (5,5)의 이미지가 패딩을 거쳐 (7,7)로 확장되어 특성 맵의 크기 또한 (5,5)로 적용 될 수 있습니다.

입력과 출력이 동일한 패딩을 주로 same padding으로 부릅니다.



# 이론 설명

## Convolutional Neural Network



### [ Backpropagation Algorithm ]

1. Error at the output layer

$$\delta^L = \nabla_a C \odot \sigma'(x^L)$$

- $C$  : Cost (Loss)
- $a$  : final output of DNN
- $\sigma(\cdot)$  : activation function

2. Error relationship between two adjacent layers

$$\delta^l = \sigma'(x^l) \odot ((w^{l+1})^T \delta^{l+1})$$

3. Gradient of C in terms of bias

$$\nabla_b C = \delta^l$$

4. Gradient of C in terms of weights

$$\nabla_{w^l} C = \delta^l (a^{l-1})^T$$

(이활성의 오토 인코더)

- activation

활성화 함수 입니다.

여기선 Relu함수를 사용하고 있습니다.

Max(0,x)라고 볼 수 있습니다.

Sigmoid와 tanh가 가지고 있는 Gradiend Vanishing문제를 해결하기 위해 나왔습니다.

위 2개 함수는 미분함수에서 일정 값 이상일 경우 미분 값이 소실되기 때문에 Relu함수를 사용 하였습니다.

입력값이 만약 0보다 작으면 다른 함수와 다르게 그냥 0으로 출력시키버리고 0이상일 경우 입력값 을 그대로 출력시켜

Gradient Vanishing 문제를 어느정도 해결시켜줍니다.

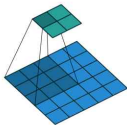
Gradient Vanishing은 Backpropagation으로 학습 시킬 시 가중치와 바이어스의 미분 값이 뒤로 갈 수록 0이 되어 버리기 때문에 가중치가 바뀌는 양이 매우 미미해져 사라져 버리는 문제이다. 출력단과 근처 부분만 바뀌고 뒤쪽은 바뀌지 않는 현상 일어난다.

# 이론 설명

## Convolutional Neural Network

- $O$  : Size of output image
- $I$  : Size of input image
- $K$  : Size of kernels used in the convolution layer
- $N$  : Number of kernels
- $S$  : Stride of the convolution layer
- $P$  : Padding size
- 출력 이미지의 채널 수는 커널의 개수( $N$ )와 같음

$$O = \frac{I - K + 2P}{S} + 1$$



### -Convolutional Neural Network

마지막으로 특성맵의 크기를 계산하는 법을 보겠습니다.

크기 계산은  $(I-K+2P/S)+1$  입니다.

아래 이미지로 계산 할 경우

$(5-3+0/2) + 1 = 2$ 가 나오는 것을 볼 수 있습니다.

컨볼루션에서 만약 컬러 이미지일 경우 입력을

```
model.add(layers.Conv2D(256, kernel_size=3, activation='relu', padding='same', input_shape=(150, 150, 3)))
```

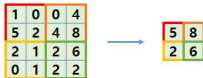
위와 같이 마지막에 3을 넣어주면 됩니다.

3을 넣어주는 이유는 컬러 이미지는 RGB이기 때문에 깊이가 3인 이미지 입니다.

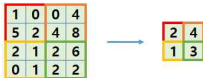
# 이론 설명

## Pooling

Max Pooling



Average Pooling



Pooling은 매우 간단합니다.  
컨볼루션과 유사하지만 가중치가 존재하지 않습니다.  
Pooling은 총 2가지 기법으로 분류됩니다.

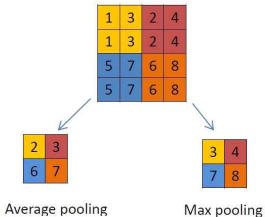
Max Pooling

Average Pooling



# 이론 설명

## Pooling



### -Average Pooling

Average Pooling은 평균 값을 출력합니다.

인자로 filter\_size와 strides 가 있습니다.

주로 2,2로 적용합니다. 아래의 평균 Pooling일 경우

첫번째 값은 2 입니다. 이는  $1+3+1+3/4$ 로 볼 수 있습니다.

나머지도 동일합니다.

### -Max Pooling

Max Pooling은 최대 값을 출력합니다.

인자는 동일하며 주로 2,2로 적용합니다.

1,3,1,3중에 최대값을 뽑아 냅니다.

Pooling을 이용함으로써 주로 얻는 이점은 크기를 절반으로 줄이며 가장 중요한 특성들을 뽑아냅니다. 노이즈를 제거 합니다.

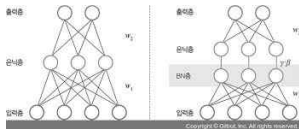
Max를 쓰는 이유는 가장 두드러지는 특성을 찾아내기 때문에 쓰며

Average는 덜 중요한 부분을 포함할 수 있습니다.

장점으로 지역정보를 일반 레이어보다 적게 잃어 위치 정보를 담아 낼 수 있습니다.

# 이론 설명

## Batch Normalization



Batch Normalization의 학습의 안정성과 Gradient Vanishing의 문제 그리고 과적합의 해결을 위해 도입되었습니다.

여기서는 일반적인 Batch Normalization에 대해서는 간략하게 설명하고 CNN에서의 Batch Normalization에 대해 설명하겠습니다.

# 이론 설명

## Batch Normalization

<b>Input:</b> Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1, \dots, x_m\}$ ; Parameters to be learned: $\gamma, \beta$	
<b>Output:</b> $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$	
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$	// mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$	// mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$	// normalize
$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$	// scale and shift

Algorithm 1: Batch Normalizing Transform, applied to activation  $x$  over a mini-batch.

$$\frac{\partial \mathcal{L}}{\partial y_i} = \frac{\partial \mathcal{L}}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial x_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \mathcal{L}}{\partial \mu_{\mathcal{B}}} = \left( \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial x_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^m x_i - 2(\mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial x_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \mathcal{L}}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial y_i} \cdot \hat{x}_i$$

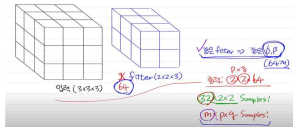
$$\frac{\partial \mathcal{L}}{\partial \beta} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial y_i}$$

자 우선 CNN의 Batch Normalization을 알기전에 일반적인 Batch Normalization에 대해 알아보자.  
미니 배치로 32개씩 들어온다고 가정 했을 때 입력 값은 랜덤 할 것이다. 이것은 어떤 분포를 따를 것이다.  
어떤 한 노드를 통과할 때 값들의 평균과 분산을 구한다. 평균은  $\mathbf{M}(\mathbf{b})$  분산은  $\sigma(\mathbf{b})$  라고 지정한다.  
그리고 이것을 Normalize 해준다 각 샘플  $\mathbf{x}(i) - \mathbf{M}(\mathbf{b}) / \sigma(\mathbf{b})$ 를 해주어 정규화 해준다. 그럴 경우 평균이 0  
분산이 1인 값이 나온다. 이를  $X(i)$ 라고 한다. 이후 마지막으로  $\Gamma X(i) + \beta = y(i)$ 를 해준다. 이때  $\Gamma, \beta$ 를 학습 가  
능하게 만들어준다. 이를 각 노드마다 적용시킨다. 이게 왼쪽에 있는 그림이다.

오른쪽은  $\Gamma, \beta$ 에 대한 Backpropatation 수식이다.  $\Gamma, \beta$ 는 Backpropatation을 통해 학습이 진행된다.

# 이론 설명

## Batch Normalization



CNN에서의 BatchNormalization이다.

중요한 것은 같은 filter은 같은  $\gamma, \beta$ 를 공유한다. 그럼 필터의 평균과 분산은 어떻게 구할까?

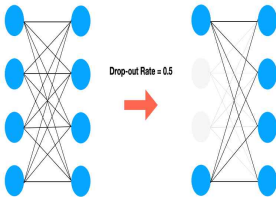
1개의 필터가 32개의 미니배치를 통과한다 할 경우 (5,5)사이즈의 이미지에 (3,3)의 필터를 적용 시키면 한번씩 이동하면서 찍어낼 것이다. 이때 찍을때 마다의 전부를 다 더 해서 평균과 분산을 구한다.

그럼 그게 즉 필터의 평균과 분산이다.

그럼 32개의 필터를 쓸 경우 32개의  $\gamma, \beta$  32개가 나올 것이다.

# 이론 설명

## Drop out



<https://heytech.tistory.com/>

### -Drop out

Drop out은 말 그대로 드롭 아웃이다.

주로 사용되는 목적은 과대적합을 방지하기 위함이다.

만약 어느 한 개의 뉴런이 출력에 대해 과대하게 영향을 미친다고 했을 경우 학습을 진행할 때 해당 가중치가 크게 설정되어 나머지 뉴런,특성에 대해 가중치가 제대로 학습되지 않습니다.

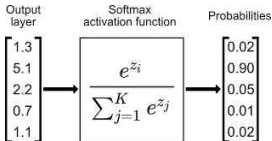
이를 방지하기 위해 Drop out을 적용하는데 옆의 그림은 출력의 절반을 랜덤으로 막습니다. 그럼 나머지 정보만을 가지고 출력을 하기 때문에 과대적합을 방지합니다.

테스트 시에는  $h(n) = a(\alpha W(n)h(n-1)+b(n))$ 입니다 이때

A는 기존 학습 시 꺼진 비율만큼 곱해주어 scaling를 적용 시켜 같은 scale를 가지도록 합니다.

# 이론 설명

## Softmax



### -Softmax

다중분류의 최종층에 씁니다.

Softmax의 출력은 항상 0에서 1.0 사이에 실수입니다.

모든 합은 1이며 마치 확률 값처럼 표현됩니다.

로스로는 주로 cross entropy를 씁니다.

아래는 강의 교안의  $i=j$  일때랑  $i \neq j$  일때의 미분이다.

코드로 구현 시 NaN이 발생 할 수 있으므로 입력 값의 최대값을 빼준다.

#### ■ $i=j$ 일 때,

$$p_i = \frac{e^{z_i}}{\sum_{k=1}^3 e^{z_k}}$$

$$\frac{\partial p_i}{\partial z_i} = \frac{\partial}{\partial z_i} \left( \frac{e^{z_i}}{\sum_{k=1}^3 e^{z_k}} \right) = \frac{e^{z_i} \left( \sum_{k=1}^3 e^{z_k} - e^{z_i} \right)}{\left( \sum_{k=1}^3 e^{z_k} \right)^2} = \frac{e^{z_i}}{\sum_{k=1}^3 e^{z_k}} \left( 1 - \frac{e^{z_i}}{\sum_{k=1}^3 e^{z_k}} \right) = p_i (1 - p_i)$$

#### ■ $i \neq j$ 일 때,

$$p_i = \frac{e^{z_i}}{\sum_{k=1}^3 e^{z_k}}$$

$$\frac{\partial p_i}{\partial z_j} = \frac{\partial}{\partial z_j} \left( \frac{e^{z_i}}{\sum_{k=1}^3 e^{z_k}} \right) = -\frac{e^{z_i} e^{z_j}}{\left( \sum_{k=1}^3 e^{z_k} \right)^2} = -p_i p_j$$

# 이론 설명

## Adam

Momentum

$$w_{t+1} = w_t - \alpha m_t$$

$$m_t = \beta m_{t-1} + (1 - \beta) \left[ \frac{\partial L}{\partial w_t} \right]$$

$$v_1 \leftarrow -\eta \frac{\partial L}{\partial w_1} = -0.5 \ (\eta=0.1)$$

$$W \leftarrow W + v_1 = W - 0.5$$

$$v_2 \leftarrow \alpha v_1 - \eta \frac{\partial L}{\partial w_2} = -0.45 - 0.3 = -0.75 \ (\alpha = 0.9)$$

$$W \leftarrow W + v_2 = W - 0.75$$

AdaGrad

$$h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

RMSP

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t + \epsilon)^{1/2}} * \left[ \frac{\partial L}{\partial w_t} \right]$$

$$v_t = \beta v_{t-1} + (1 - \beta) * \left[ \frac{\partial L}{\partial w_t} \right]^2$$

-Adam

Adam 옵티마이저는 경사하강법을 위한 알고리즘이다.

기본적인 옵티마이저 같은 경우는 강의에서 다루어 본 적 있으므로 간단하게 설명하고 Adam에 대해 설명하겠습니다.

옵티마이저는 경사하강법인데 어떻게 내려갈까? 관건입니다.

일반적으로는 함수의 기울기를 구해 경사의 반대 방향으로 조금씩 이동 하는 것입니다.

Adam은 기본적으로 Momentum과 RMSP의 알고리즘의 조합입니다.

옆의 식은 모멘텀의 식입니다. Momentum은 알고리즘에 운동량을 추가한 것이라고 볼 수 있습니다. 일반적인 GD보다 더욱 증가하는 것을 볼 수 있습니다.

RMSP는 AdaGrad를 개선한 알고리즘입니다. 제곱 기울기의 누적 합 대신 지수 평균 이동을 사용합니다.

# 이론 설명

## Adam

$$v_{dw} = 0, S_{dw} = 0, v_{db} = 0, S_{db} = 0$$

$$\begin{aligned} v_{dw} &= \beta_1 v_{dw} + (1 - \beta_1) dW \\ v_{db} &= \beta_1 v_{db} + (1 - \beta_1) db \end{aligned}$$

$$\begin{aligned} S_{dw} &= \beta_2 S_{dw} + (1 - \beta_2) dW^2 \\ S_{db} &= \beta_2 S_{db} + (1 - \beta_2) db^2 \end{aligned}$$

$$\begin{aligned} v_{dw}^{biascorr} &= v_{dw} / (1 - \beta_1^t) \\ v_{db}^{biascorr} &= v_{db} / (1 - \beta_1^t) \\ S_{dw}^{biascorr} &= S_{dw} / (1 - \beta_2^t) \\ S_{db}^{biascorr} &= S_{db} / (1 - \beta_2^t) \end{aligned}$$

$$\begin{aligned} W &= W - \alpha v_{dw}^{biascorr} / \sqrt{S_{dw}^{biascorr} + \epsilon} \\ b &= b - \alpha v_{db}^{biascorr} / \sqrt{S_{db}^{biascorr} + \epsilon} \end{aligned}$$

-Adam

Adam알고리즘에 더 살펴보겠습니다.

Momentum,RMSP에 사용한 V,S를 지정해줍니다.

그후 Momentum에서 나온 Bias correction 해줍니다.

Bias correction 은 기존 v(t)에서 1-  $\beta(t)$ 를 나누어 주는 것입니다.

마지막으로 두개의 가중치 업데이트 방식을 사용 하여 가중치를 업데이트 합니다.

Adam의 하이퍼 파라미터

$\alpha$  : learning rate

$\beta_1$  : 1차 moment, 대부분 0.9 ( $dW$ 의 지수 가중 평균 계산)

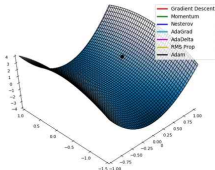
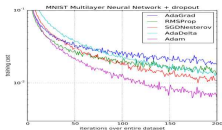
$\beta_2$  : 2차 moment, 논문에서는 0.99 ( $dW^2$ 과  $db^2$ 의 지수 가중 평균 계산)

$\epsilon$  : 논문에서는  $0.10^{-8}$



# 이론 설명

## Adam



### -Adam

옵티마이저 별 교육 비용에 대한 성능비교

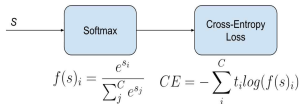
옵티마이저별 수렴 GIF를 보면 항상 Adam이 좋은것은 아니다.  
각 모델과 목적에 맞춰 옵티마이저를 잘 선택하여야 한다.

여기서는 가장 안정적이며 효과가 좋은 Adam 옵티마이저를 선택하였습니다.



# 이론 설명

## - Categorical Cross Entropy Error



### -Categorical Cross Entropy Error

Categorical Cross Entropy Error는 다중 클래스 분류에서도 원 핫 벡터인 경우 사용되는 로스 함수이다. Softmax loss라고도 불린다. 확률모델 시점으로 봤을 때 두 확률 분포의 간이 차이를 줄이는게 목적이다.

실제 값가 예측 값에 대해 각 클래스 별로 오차를 계산하고 오차를 모두 더한다.

공식은 아래와 같다.

$$L = -\frac{1}{N} \sum_{j=1}^N \sum_{i=1}^C t_{ij} \log(y_{ij}) \quad \dots (\text{categorical crossentropy})$$

$$L = -(1\log 1 + 0\log 0 + 0\log 0 + 0\log 0 + 0\log 0) = 0$$

$$L = -(1\log 0 + 0\log 1 + 0\log 0 + 0\log 0 + 0\log 0) = \infty$$

공식을 살펴보면  $C$ 는 클래스의 개수이다.

만약 정답을 맞췄을 경우

아래와 같이 로스는 0이 된다.

만약 실제값은  $[1, 0, 0, 0]$  예측값은  $[0, 1, 0, 0]$ 일 경우 손실은 양의 무한대가 나옵니다. 실제로 한쪽으로 1이 나오는 현상은 드물지만 높은 로스를 가지는 것을 알 수 있습니다.

이렇듯 원-핫 벡터일 경우에 로스는 Categorical Cross Entropy Error가 적합하다는 것을 볼 수 있습니다.

만약 원-핫 벡터가 아닌 라벨일 경우는 sparse categorical crossentropy가 쓰입니다.

# 최종 이론 설명

## - Animals-10

### -Animals-10

이미지 데이터를 imagedatagenerator로 이미지 전처리를 시켜 배치 사이즈를 20으로 잡아줍니다.

위에 설명한 기능들 Conv2d과 maxpooling,batch\_normalization을 이용하여 순서대로 총 4개의 층을 통과시킨다.

Flatten층을 이용하여 일렬로 펼쳐준다.

뉴런 100개를 만들어 통과시켜 100차원으로 축소를 시키고 이에 40%를 Dropout으로 떨어뜨려 준다.

다시 뉴런 10개를 softmax함수를 통과시켜 원 핫벡터로 출력을 내보낸다.

모델 컴파일은 로스로 원핫벡터이기 때문에 categorical\_crossentropy를 사용하며 옵티마이저로는 adam을 사용한다. 부가적인 설명으로 accuracy를 사용한다. Epoch는 10으로 합니다 이를 통하여 훈련시킨다.

Test\_generator에서 20\*100개의 샘플을 뽑아내어 평균 accuracy를 측정한다.

모델의 accuracy와 loss를 차트로 확인한다.

20개의 이미지를 Test\_generator에서 뽑아내어 배열로 만든 후 각각 모델에 통과시켜 레이블과 사진이 맞는지 비교한다.

20\*5개의 이미지에 대한 모델 예측 값을 확인한다.



# 문제점 및 해결방안

## - 문제점 및 해결 방안

### -문제점

- 검증 세트가 존재 하지 않음
- test세트의 점수가 높지 않음
- 원본 이미지를 다운 받았을때 손으로 파일을 분류해야함
- 컬러 이미지라 고가의 GPU사용을 요함
- 랜덤 시드를 설정해서 고정된 출력을 내놓게 한다.
- 처음엔 훈련시 `fit_generator`을 사용했지만 신버전이라서 호환 오류가 발생하여 `fit`으로 교체했습니다.

### -해결 방안

- 검증 세트에 대한 파일을 새로 만들어서 적용한다.
  - test 세트의 점수를 높이기 위해 이미지의 배치 크기를 늘린다.
  - 이미지를 (150,150)으로 축소 했다. 이미지를 모든 이미지의 최저 값으로 맞추어 최대한 큰 품질을 유지하게 만든다.
  - epoch를 늘린다. 너무 늘리면 과대적합이 발생 할 수 있다.
  - 현재 컨볼루션을 4번 진행한다. 필터의 개수를 늘리거나 컨볼루션을 더욱 더 증가시킨다.
  - 파일을 자동적으로 분류하게 하는 알고리즘이나 파일 컨트롤 하는 프로그램을 찾아낸다.
  - image net을 학습한 가중치를 받아 전이 학습을 시킨다.
  - 코랩을 이용하는 방안도 있지만 현재로서는 가장 딥러닝에 적합하고 가성비가 좋은 GPU를 구입한다. GPU의 램 크기에 절대적이기 때문이다.
  - 랜덤 시드를 만들어 설정한다.
- 