

# jellyfish classification project

A whimsical illustration of a jellyfish with a large, translucent, bell-shaped body and long, flowing tentacles. The jellyfish is holding a green glass bottle of Soju (Korean distilled spirit) in its tentacles. The bottle is tilted, and a stream of liquid is pouring out from the neck. The background is a deep blue ocean with bubbles and stylized, glowing jellyfish-like shapes. The text "jellyfish classification project" is written in a bold, yellow, sans-serif font across the top.

팀명 : 소주한잔

팀장 : 박해극

팀원 : 조필선, 정인호, 조세창



# 목차

1. 데이터 EDA
2. 데이터 전처리
3. 모델 서치
4. 성능향상 및 모델링
5. Trial and Error
6. 평가 분석
7. 회고







# Exploratory Data Analysis

- 폴더 구조 및 이미지 사이즈
- 중복 파일 탐색
- 이미지가 아닌 파일 탐색
- 클래스별 이미지 갯수 확인
- 클래스별 이미지 시각화
- Dataset 전체 이미지 사이즈 파악
  - 사이즈가 다르기 때문에 리사이징
- 데이터 imbalance 파악
- 손상된 이미지 파일 파악

```
└─ Moon_jellyfish
   * Image sizes: (1724, 2541, 102, 170, 1701, 471)
   Train Test Val: └─ Moon_jellyfish
                     * 150 images
                     Train_Test_Valid
```

중복 파일 탐색

이미지 데이터 결측치 검사

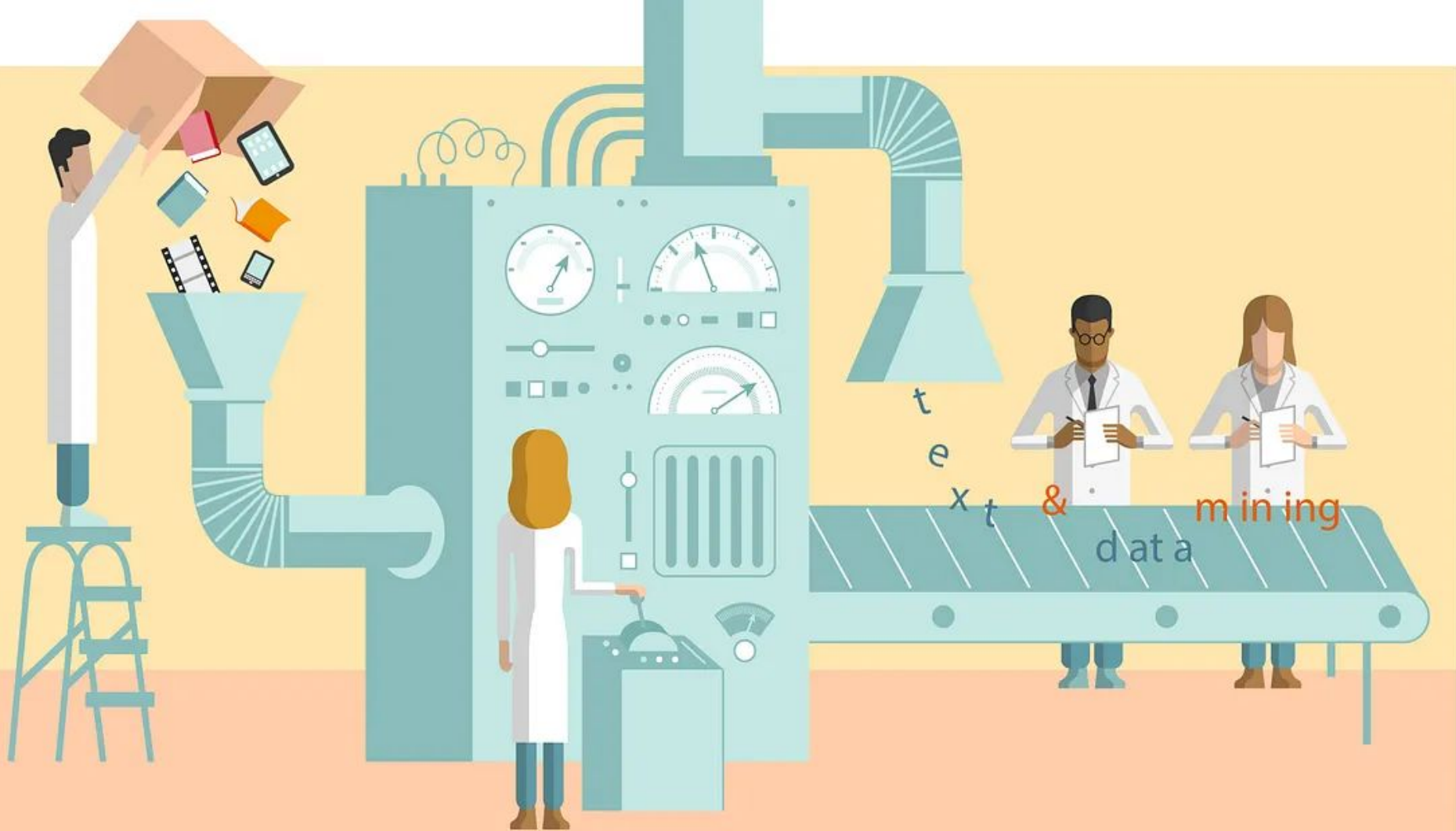
```
[ ] import os
    from PIL import Image

    dataset_path = 'dataset_path' # 데이터셋 경로
    image_extensions = ['.png', '.jpg', '.jpeg'] # 인식할 이미지 확장자 목록
    missing_files = []

    # 데이터셋의 모든 파일에 대해 반복
    for root, dirs, files in os.walk(dataset_path):
        for file in files:
            if any(file.endswith(ext) for ext in image_extensions):
                file_path = os.path.join(root, file)
                try:
                    # 이미지 로드 시도
                    with Image.open(file_path) as img:
                        img.verify() # 이미지 파일 손상 여부 확인
                except (IOError, SyntaxError) as e:
                    # 파일 로드 실패 시 목록에 추가
                    missing_files.append(file_path)

    # 결측치(손상된 파일) 목록 확인 및 출력
    if missing_files:
        print(f"읽을 수 없거나 손상된 파일: {missing_files}") #이미지에 문제가 있으면 경로/파일명 출력하도록
    else:
        print("읽을 수 없거나 손상된 파일이 없습니다.")
```

읽을 수 없거나 손상된 파일이 없습니다.



# 데이터 전처리

- 이미지 리사이징 : **EDA**에서 확인한 각기 다른 이미지 데이터의 사이즈를 동일하게 맞추기 위해서 진행하였음.
- 이미지 정규화 : 연산속도 및 정확도를 높이기 위해서 진행하였음.
- 데이터 증강 : 데이터셋의 갯수가 **900**개 밖에 안되서 분류를 하기에는 턱없이 부족하여 데이터 **augmantation**(회전, 대조, 밝기)을 진행하였음.
- 훈련 데이터와 테스트 데이터 모두 동일하게 진행하였음.
- **Train\_Test\_Valid**폴더의 사진이 학습 이미지와 같아 평가하기에 좋지않다고 판단하여 테스트 데이터는 외부 이미지 데이터를 사용하였음.

```
def load_and_preprocess_image(image_path):  
    image = tf.io.read_file(image_path)  
    image = tf.image.decode_jpeg(image, channels=3)  
    image = tf.image.resize(image, [180, 180])  
    image /= 255.0  
    return image
```

```
class ImageAugment:  
    def __init__(self):  
        pass  
  
    def rotate(self, image):  
        rotated_img = tf.image.rot90(image)  
        return rotated_img  
  
    def zoom(self, image, size=(224, 224)):  
        # 이미지 중앙을 기준으로 확대 후 원본 크기로 조정  
        zoomed = tf.image.central_crop(image, 0.8)  
        zoom_img = tf.image.resize(zoomed, size)  
        return zoom_img  
  
    def adjust_contrast(self, image):  
        contrast_img = tf.image.random_contrast(image, lower=0.8, upper=1.2)  
        return contrast_img  
  
    def adjust_brightness(self, image):  
        brightened_img = tf.image.random_brightness(image, max_delta = 0.2)  
        return brightened_img  
  
    def flip_left_right(self, image):  
        flipped_img = tf.image.flip_left_right(image)  
        return flipped_img  
  
    def adjust_saturation(self, image):  
        sat_img = tf.image.random_saturation(image, 5, 10)  
        return sat_img
```

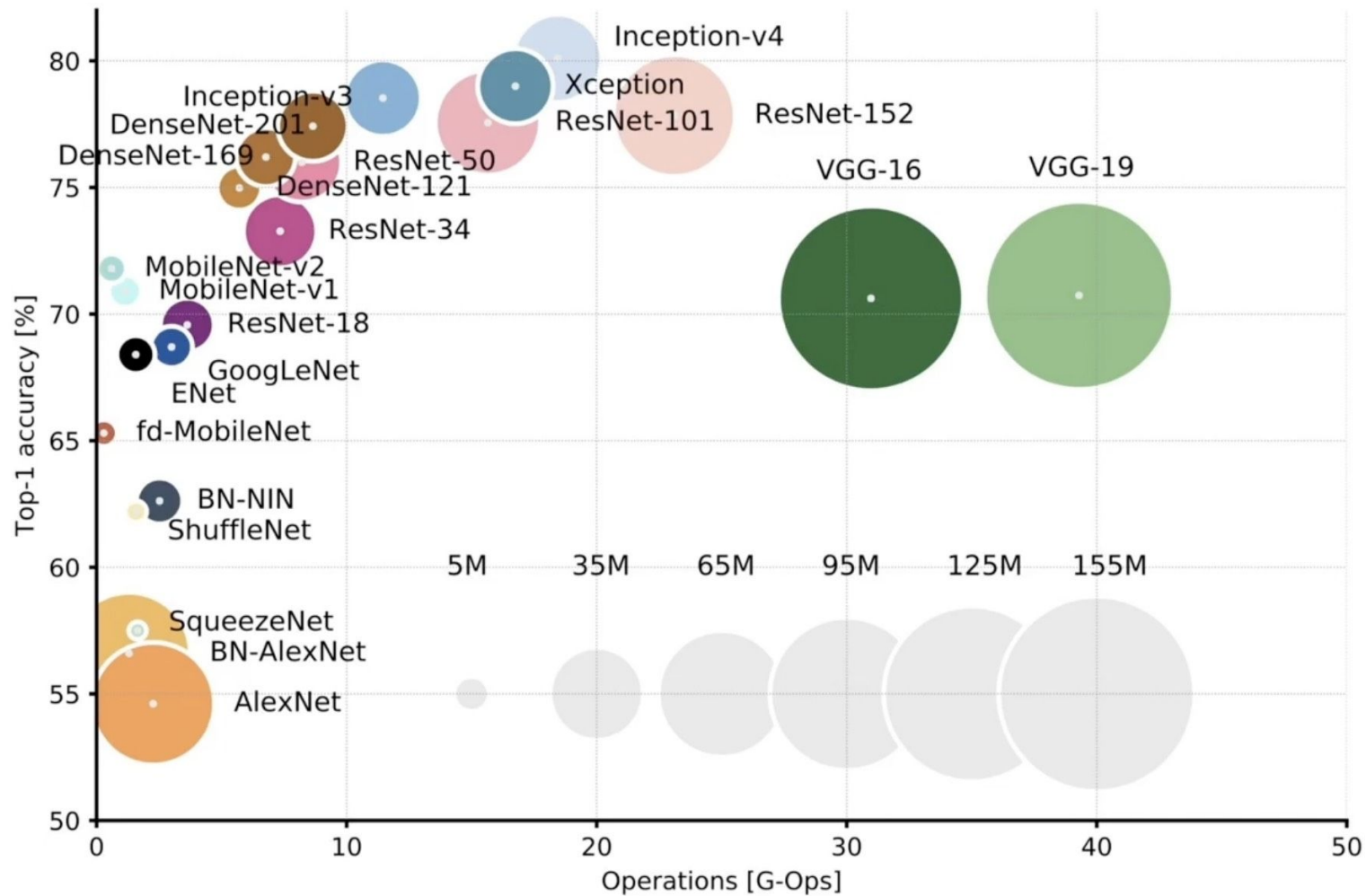
searching...

A 3D rendered image featuring the word "searching..." in a bold, black, sans-serif font. The text is set against a white rectangular background that is slightly recessed into a light blue surface. A magnifying glass with a black handle and a silver-colored frame is positioned over the word, with its lens centered on the letters "a" and "r". The magnifying glass is angled downwards from the right. The overall composition is clean and modern, with soft shadows and a perspective that gives it a three-dimensional feel.



# 모델 서치

- Xception
  - 꽃 분류에서 테스트 accuracy(95%) 와 loss(약 0.15)
- VGG16
  - VGG16 total params: 14,731,454
  - Xception total params: 20,894,454
- MobileNet
  - MobileNet Total params: 3,525,182



**Trial  
&**

**Error**





# Trial and Error

- 오버피팅 방지
  - Xception:
    - 테스트 평가와 훈련 및 검증 metric과의 gap
    - L2 regularizers 와 Dropout
- 모델 구조 변경 해보기
  - MobileNet
  - VGG16 과 Xception은 FC layer 만 변경
- 정규화 진행
- 전처리 부분에서 코드를 간결화
- MobileNet validation, train accuracy, loss 그래프 간격차이
  - 최적화 진행
  - l2 , dropout, adam, RMSprop, BatchNormalization
  - 마지막 convolution 에서 stride 변경
  - Early Stop 적용

# Xception 오버피팅 방지

```
# convolution 위에 레이어 쌓기
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(16, kernel_regularizer=regularizers.l2(l2_factor)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dropout(0.5), # Dropout 추가
    layers.Dense(6, kernel_regularizer=regularizers.l2(l2_factor)),
    layers.BatchNormalization(),
    layers.Activation('softmax')])

model.summary()
```

# MobileNet Trial and Error

```
from tensorflow.keras.layers import Conv2D, BatchNormalization, ReLU
# convolution 마지막 3개만 unfreeze, 그리고 customizing
x = mobilenet_model.input

# 우선 마지막 3번째 전까지는 레이어 추가
for layer in mobilenet_model.layers[:-3]:
    x = layer(x)

x = Conv2D(filters=1024, kernel_size=(1, 1), strides=(2, 2), padding='same')(x)
x = BatchNormalization()(x)
x = ReLU()(x)
```



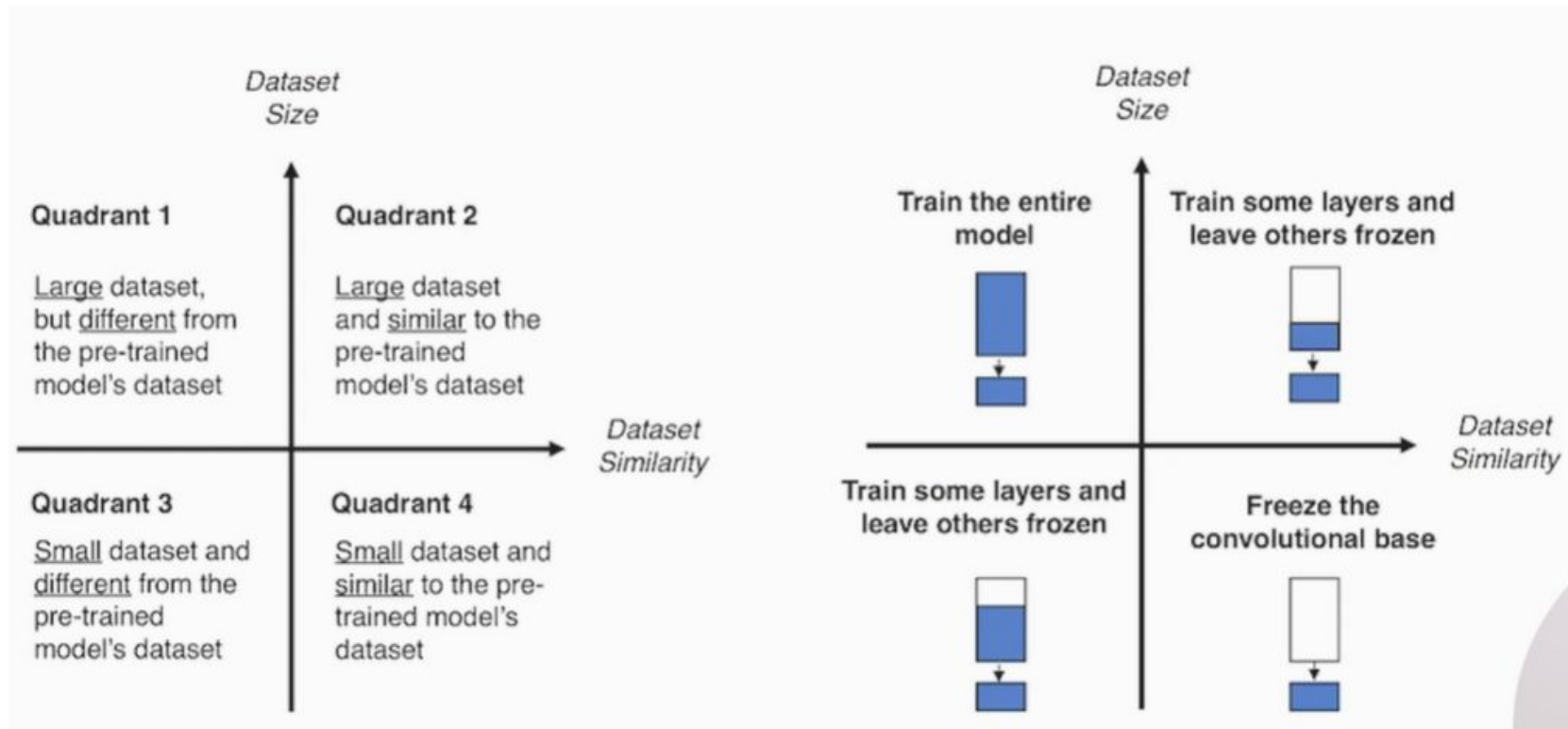
# MobileNet 오버피팅 방지

```
# convolution 위에 레이어 쌓기
# kernel_regularizer=regularizers.l2(0.1)
x = layers.Flatten()(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(32, kernel_regularizer=regularizers.l2(0.01))(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dropout(0.2)(x) # Dropout 추가, 0.25 혹은 0.2
x = layers.Dense(6)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('softmax')(x)

new_model = Model(inputs=mobilenet_model.input, outputs=x)

new_model.summary()
```

# Transfer Learning Type 결정 과정





# Accuracy & Precision



**Accurate**  
but , not precise



**Precise**  
but , not accurate



**Accurate**  
and **Precise**



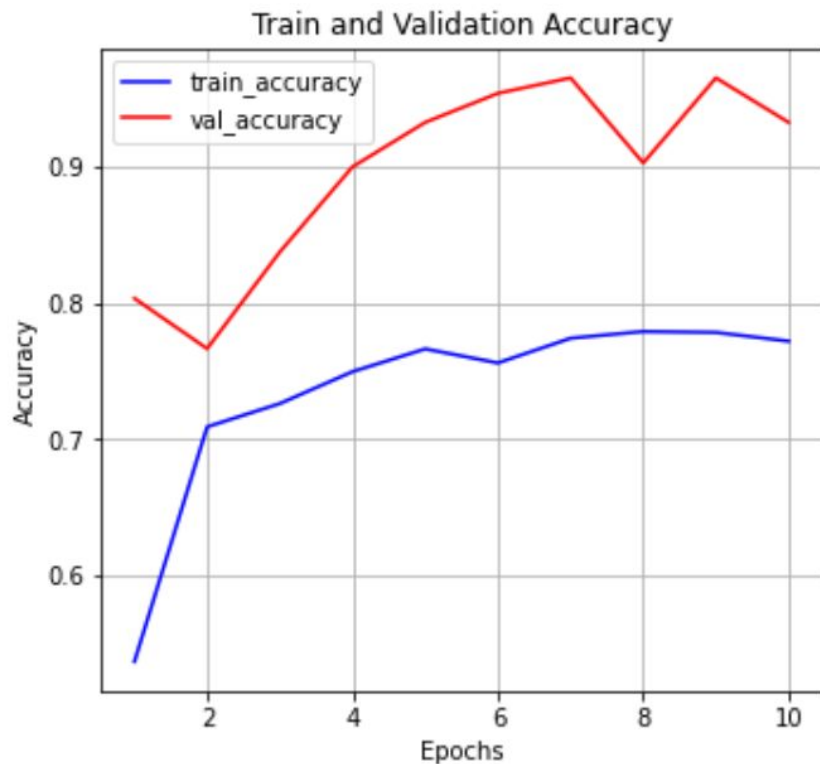
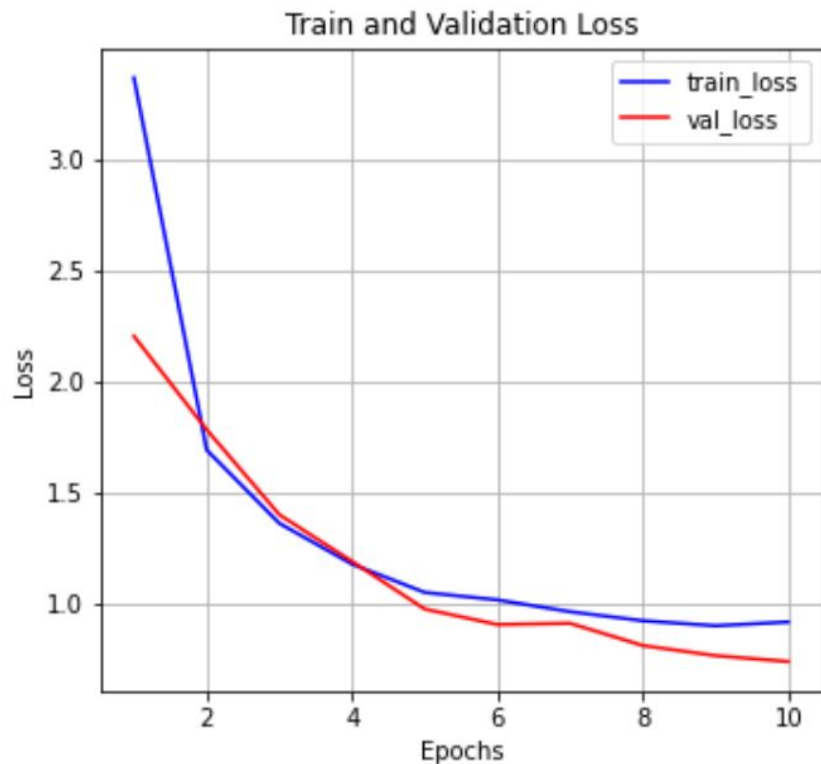
# 평가 분석

- Prediction 과 True Label 비교하여 Accuracy 기준으로 평가
  - 이미지시각화
  - 혼동 행렬을 통해 시각화

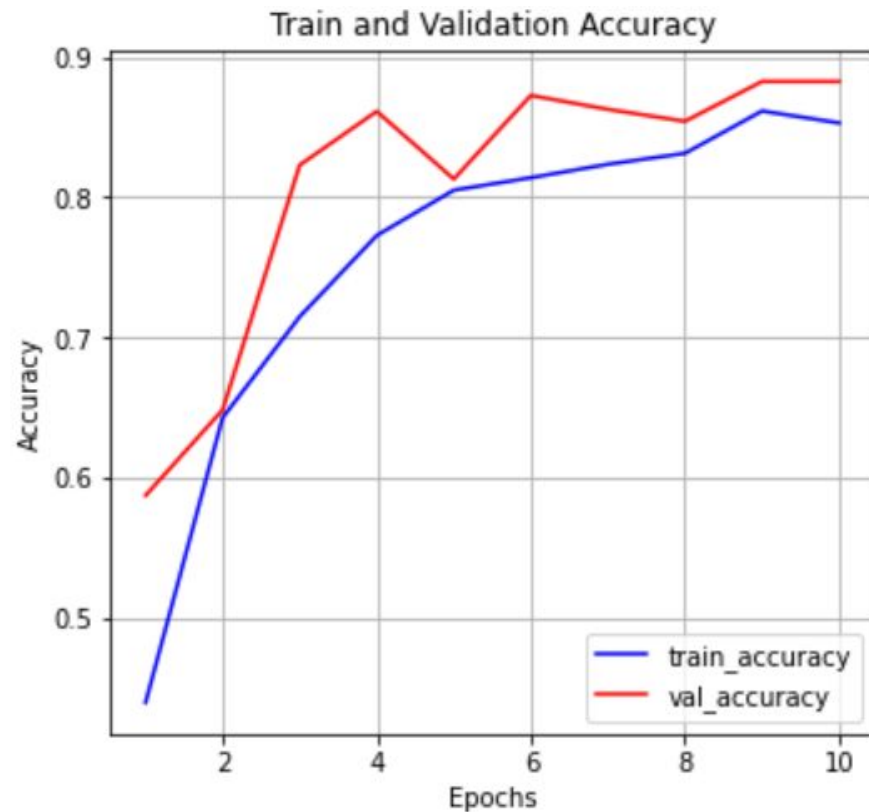
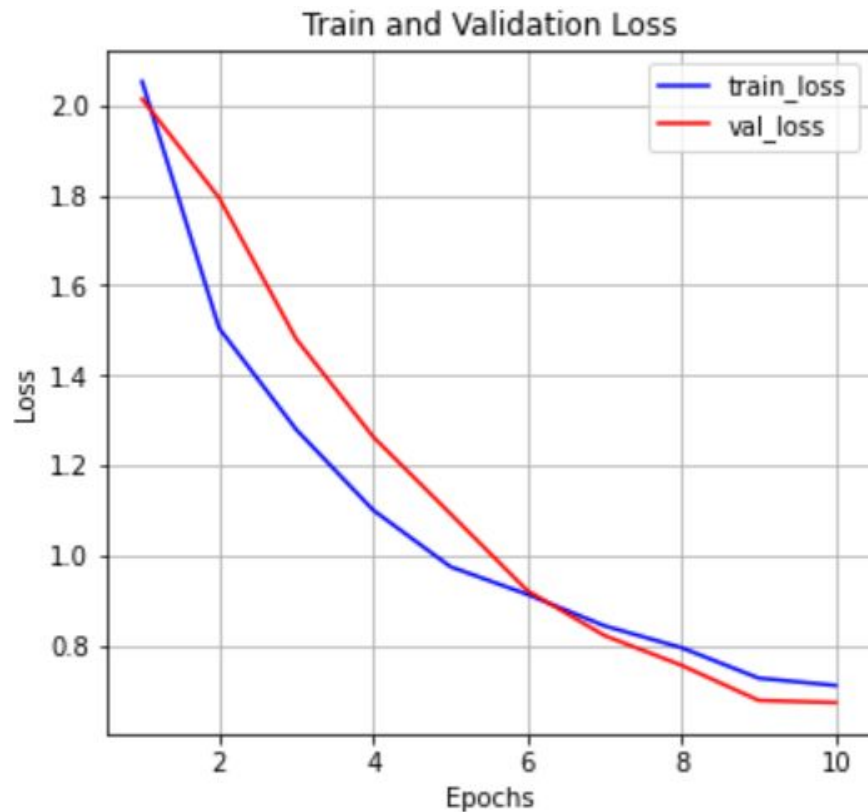
## 모델별 Test Accuracy, Loss

	Xception	VGG16	MobileNet
Test Accuracy	68%	64%	86%
Test Loss	1.19	1.23	0.66

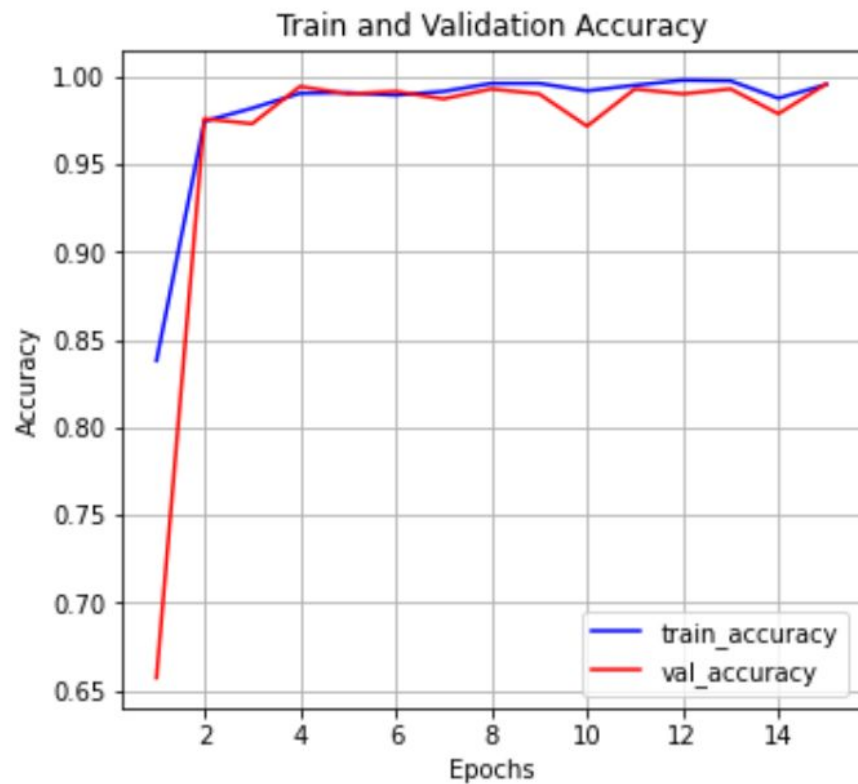
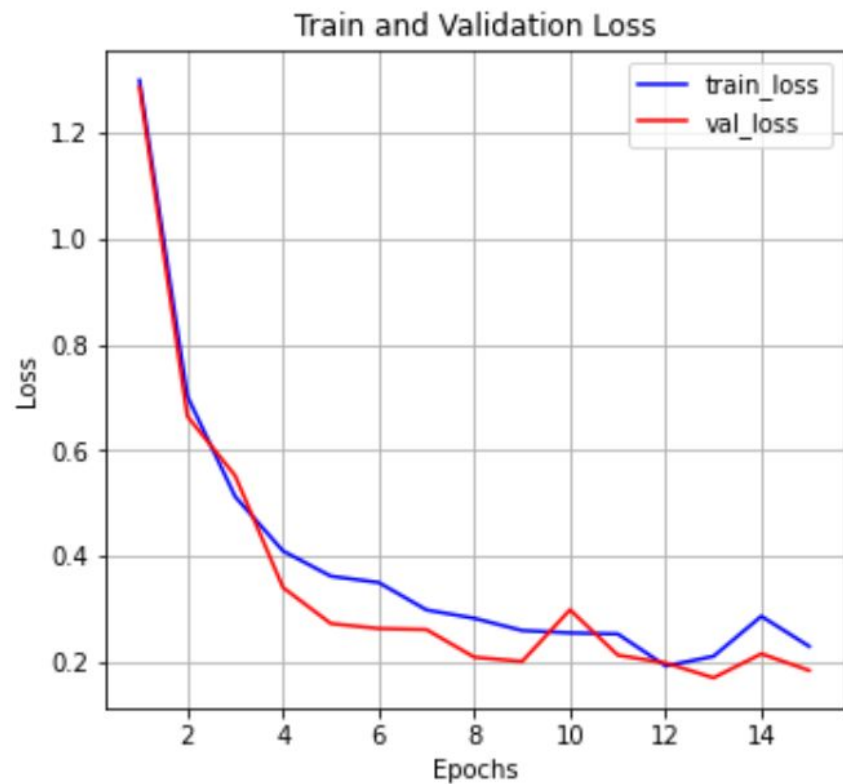
# Xception



# VGG16

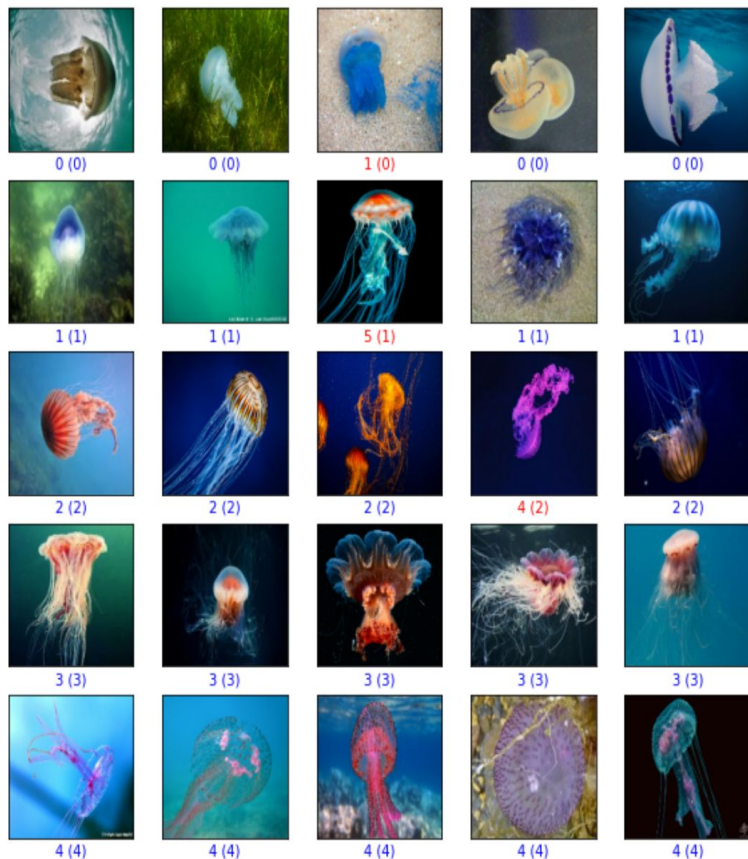


# MobileNet

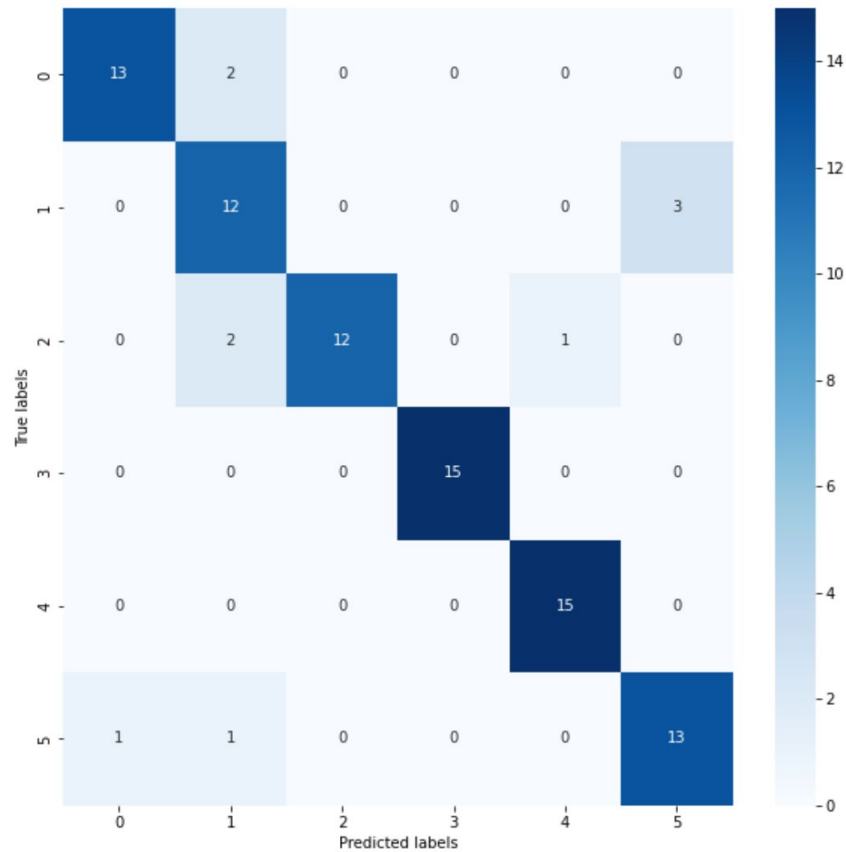




## 이미지별 평가 시각화



## 혼동행렬(Confusion Matrix)



# 회고

- **Learning rate** 스케줄
  - 학습이 더 잘 됐을 것 같다.
  - 그러나 학습 시간 증가로 인해 소요시간 **penalty**
- EDA 에서 **feature** 들을 살펴보기
- 전처리 부분에 있어 **YOLO** 모델을 활용하여 양질의 데이터 추출
  - **object detection** 기술을 활용하여 **bounding box**로 해파리 객체만 추출하려고 했으나 코드 구현 문제로 아쉽게 할 수 없었음.
- 배경을 없애고, 이미지 마스크를 통해 해파리 객체만 추출려고 했으나 구현 문제

# Appendix.

689	Xception	79%	22.855952M	94.5	×
-----	----------	-----	------------	------	---

24	MobileNetV2	72%	3.4M	0.600	✓
----	-------------	-----	------	-------	---

---