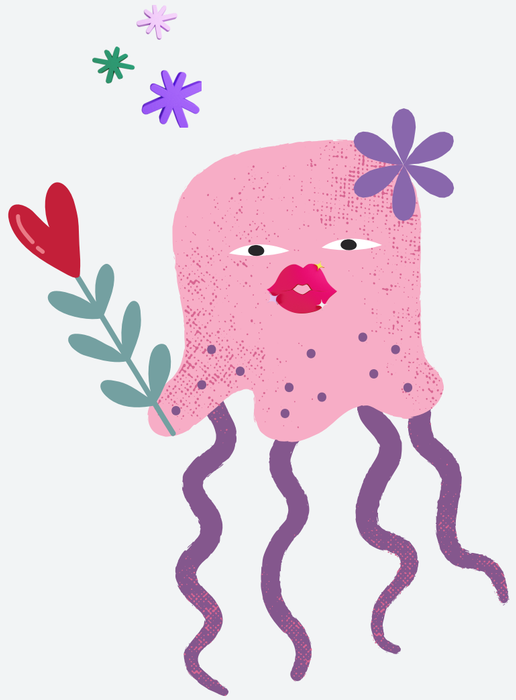
PROJECT
JELLYFISH

해파리 감별사 제 3팀

CONTENT



ABOUT US

JELLYFISH 최강훈

JELLYFISH 홍채림

JELLYFISH 이상원

JELLYFISH 윤혜윤

JELLYFISH

STRATEGIES

적합한 로스와
메트릭을 사용하여
훈련을 진행한다.

STRATEGY N°1

두 가지 이상의
비교 모델을
사용한다.

STRATEGY N°2

훈련 결과,
모델의 상용화를
분석해본다.

STRATEGY N°3

JELLYFISH

DATA PREPROCESSING

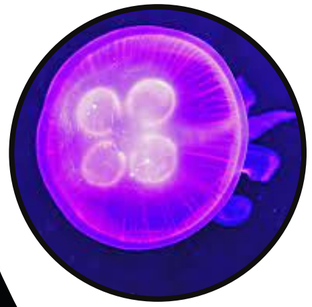
SAMPLES

Barrel jellyfish ★★★★

영국 해역에서 발견되는 가장 큰 해파리로 지름이 90cm까지 자랄 수 있습니다. 순간이동을 하므로 저격총을 이용해 포획해야 합니다.

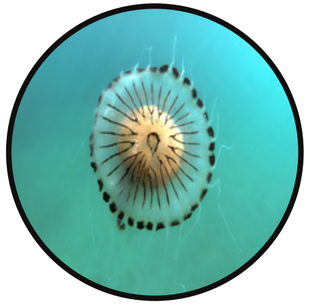
Mauve stinger ★★★

긴 촉수를 가졌으며 혹 같은 구조물을 가진 작은 해파리입니다. 야광 능력과 함께 아주 강한 독을 지녔습니다.



Moon jellyfish ★

반투명한 몸통 넘어로 4개의 특징적인 무늬를 가진 해파리입니다. 높은 비율의 지방산을 가지고 있기 때문에 다양한 포식자들의 먹이가 되곤 합니다.



Compass jellyfish ★★★★

몸통의 갈색 모양이 나침반을 닮아 이름지어졌습니다.
촉수가 무척 강하며 다른 해파리까지 잡아먹습니다.

Lion’s mane jellyfish ★★★★★

세계에서 가장 큰 해파리로, 몸통은 2미터까지 자라며
촉수는 30미터에 다다릅니다.
아주 무서운 맹독을 지녔습니다.

Blue jellyfish ★★

지름이 30cm까지 자랄 수 있는 큰 해파리입니다.
촉수로 플랑크톤과 작은 물고기를 잡아먹습니다.

Tuberculata jellyfish ★

지중해에 사는 튀긴 달걀을 닮은 무해한 해파리입니다.
달걀튀김 해파리에서는 항암 작용을 하는 추출물을 얻을 수 있습니다.

JELLYFISH

데이터 불러오기

```
image_train_data = '/aiffel/aiffel/jellyfish/Train_Test_Valid/Train'
train= pd.DataFrame(os.listdir(image_train_data ),columns=['train'])
train.head()
```

	train
0	blue_jellyfish
1	lions_mane_jellyfish
2	mauve_stinger_jellyfish
3	compass_jellyfish
4	teuberculata_jellyfish

```
image_test_data = '/aiffel/aiffel/jellyfish/Train_Test_Valid/test'
test = pd.DataFrame(os.listdir(image_test_data ),columns=['test'])
test.head()
```

```
files = [i for i in glob.glob(image_train_data + "//*//*")]
np.random.shuffle(files)
labels = [os.path.dirname(i).split("/")[-1] for i in files]
data = zip(files, labels)
dataframe = pd.DataFrame(data, columns = ["Image", "Label"])
dataframe
```

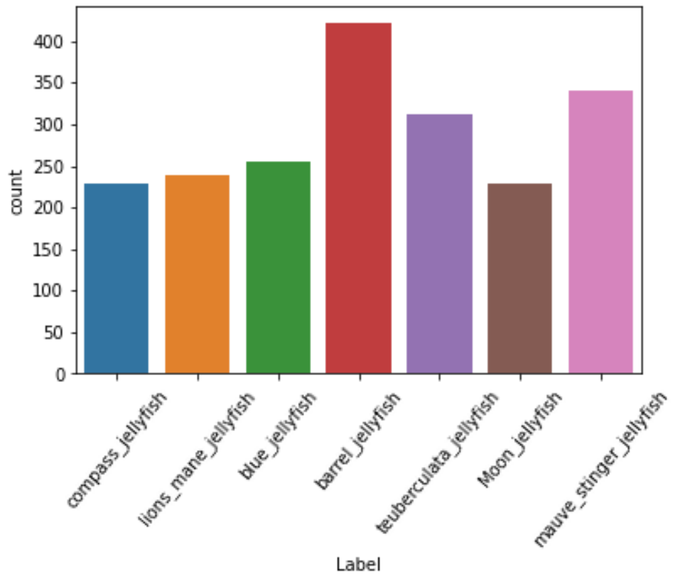
	Image	Label
0	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	compass_jellyfish
1	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	lions_mane_jellyfish
2	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	lions_mane_jellyfish
3	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	blue_jellyfish
4	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	barrel_jellyfish
...	...	...
2022	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	barrel_jellyfish
2023	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	compass_jellyfish
2024	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	blue_jellyfish
2025	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	blue_jellyfish
2026	/aiffel/aiffel/jellyfish/Train_Test_Valid/Trai...	teuberculata_jellyfish

2027 rows × 2 columns

JELLYFISH

데이터 분리 및 라벨링

```
sns.countplot(x = dataframe["Label"])
plt.xticks(rotation = 50);
```




```
batch_size = 32

target_size = (224,224)
train= tf.keras.preprocessing.image_dataset_from_directory(
    image_train_data,
    validation_split=0.2,
    subset="training",
    seed=123,  # 동일한 난수 시드 사용
    image_size=target_size,
    batch_size=batch_size,
)

valid= tf.keras.preprocessing.image_dataset_from_directory(
    image_train_data,
    validation_split=0.2,
    subset="validation",
    seed=123,  # 동일한 난수 시드 사용
    image_size=target_size,
    batch_size=batch_size,
)

test = tf.keras.preprocessing.image_dataset_from_directory(
    image_test_data,
    validation_split=None,
    image_size=target_size,
    batch_size=batch_size,
)
```

```
Found 2025 files belonging to 7 classes.
Using 1620 files for training.
Found 2025 files belonging to 7 classes.
Using 405 files for validation.
Found 210 files belonging to 7 classes.
```

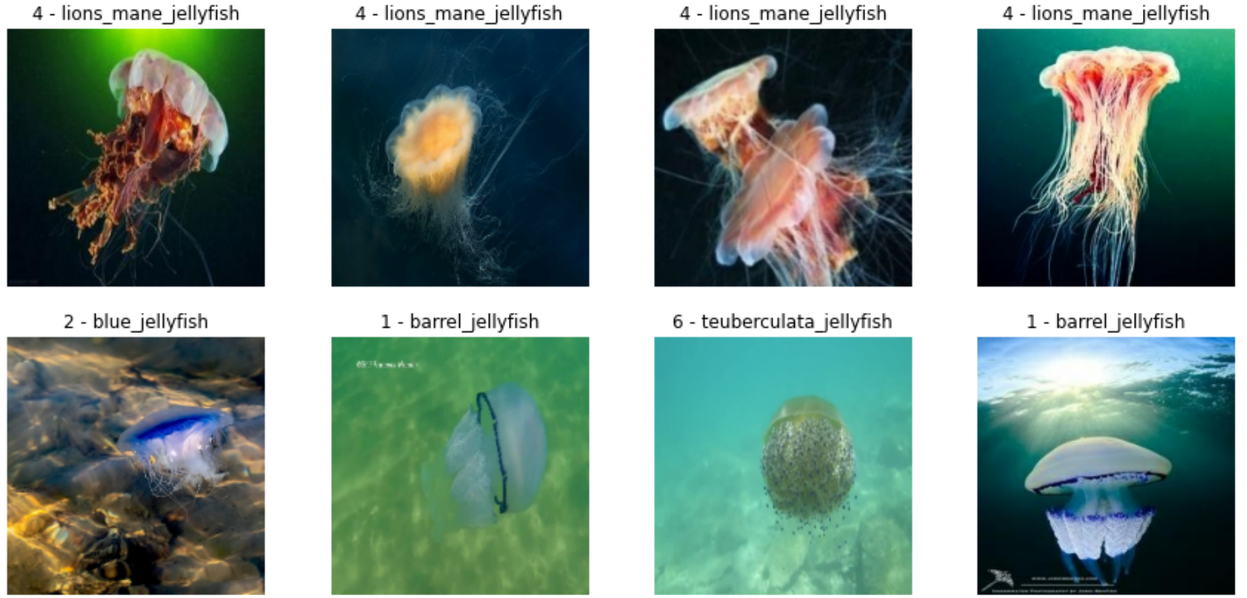
```
class_names = train.class_names
for idx, name in enumerate(class_names):
    print(f"라벨{idx}에 해당하는 클래스 : {name}")
```

```
라벨0에 해당하는 클래스 : Moon_jellyfish
라벨1에 해당하는 클래스 : barrel_jellyfish
라벨2에 해당하는 클래스 : blue_jellyfish
라벨3에 해당하는 클래스 : compass_jellyfish
라벨4에 해당하는 클래스 : lions_mane_jellyfish
라벨5에 해당하는 클래스 : mauve_stinger_jellyfish
라벨6에 해당하는 클래스 : teuberculata_jellyfish
```

JELLYFISH

이미지 라벨링 시각화

```
plt.figure(figsize=(15, 30))
for images, labels in train.take(1):
    for i in range(8):
        ax = plt.subplot(8, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(f"{labels[i]} - {class_names[labels[i]]}")
        plt.axis("off")
```




```
for images, labels in train:
    print(images.shape)  # 이미지 데이터의 쉐입을 출력합니다.
    print(labels.shape)  # 라벨 데이터의 쉐입을 출력합니다.
    break  # 첫 번째 배치만 확인합니다.
```

```
(32, 224, 224, 3)
(32,)
```

JELLYFISH

이미지 데이터 증강

```
from tensorflow.keras import layers

# Rescaling 레이어 생성
normalization_layer = layers.Rescaling(1./255)

# 훈련, 검증, 테스트 데이터셋에 정규화 레이어 적용
train = train.map(lambda x, y: (normalization_layer(x), y))
valid = valid.map(lambda x, y: (normalization_layer(x), y))
test = test.map(lambda x, y: (normalization_layer(x), y))
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.3,
    horizontal_flip=True,
    vertical_flip=True
)
datagen1 = ImageDataGenerator()

# Convert MapDataset to NumPy arrays
train_data_numpy = train.unbatch().as_numpy_iterator()
valid_data_numpy = valid.unbatch().as_numpy_iterator()

# Extract images and labels
train_images, train_labels = zip(*train_data_numpy)
valid_images, valid_labels = zip(*valid_data_numpy)

# Convert to NumPy arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)
valid_images = np.array(valid_images)
valid_labels = np.array(valid_labels)

# Apply data augmentation
augmented_train = datagen.flow(train_images, train_labels, batch_size=20)
augmented_val = datagen1.flow(valid_images, valid_labels, batch_size=20)
```

```
datagen2 = ImageDataGenerator()

# Convert MapDataset to NumPy arrays
test_data_numpy = test.unbatch().as_numpy_iterator()

# Extract images and labels
test_images, test_labels = zip(*test_data_numpy)

# Convert to NumPy arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)


# Apply data augmentation
test_generator = datagen2.flow(test_images, test_labels, batch_size = 20)
```

JELLYFISH

증강한 데이터 이미지 시각화

```python
# 시각화를 위한 코드
plt.figure(figsize=(20, 10))

# augmented_train에서 데이터를 가져와 시각화
num_batches = 2  # 시각화할 배치의 수
images_per_batch = 5  # 각 배치에서 시각화할 이미지 수

for i in range(num_batches):
    augmented_images, augmented_labels = augmented_train.next()  # 다음 증강된 데이터 배치를 가져옴
    for j in range(images_per_batch):
        plt.subplot(num_batches, images_per_batch, i * images_per_batch + j + 1)
        plt.imshow((augmented_images[j] * 255).astype('uint8'), cmap='gray', aspect='auto')
        plt.title(class_names[augmented_labels[j]])
        plt.axis('off')

plt.show()
```

JELLYFISH

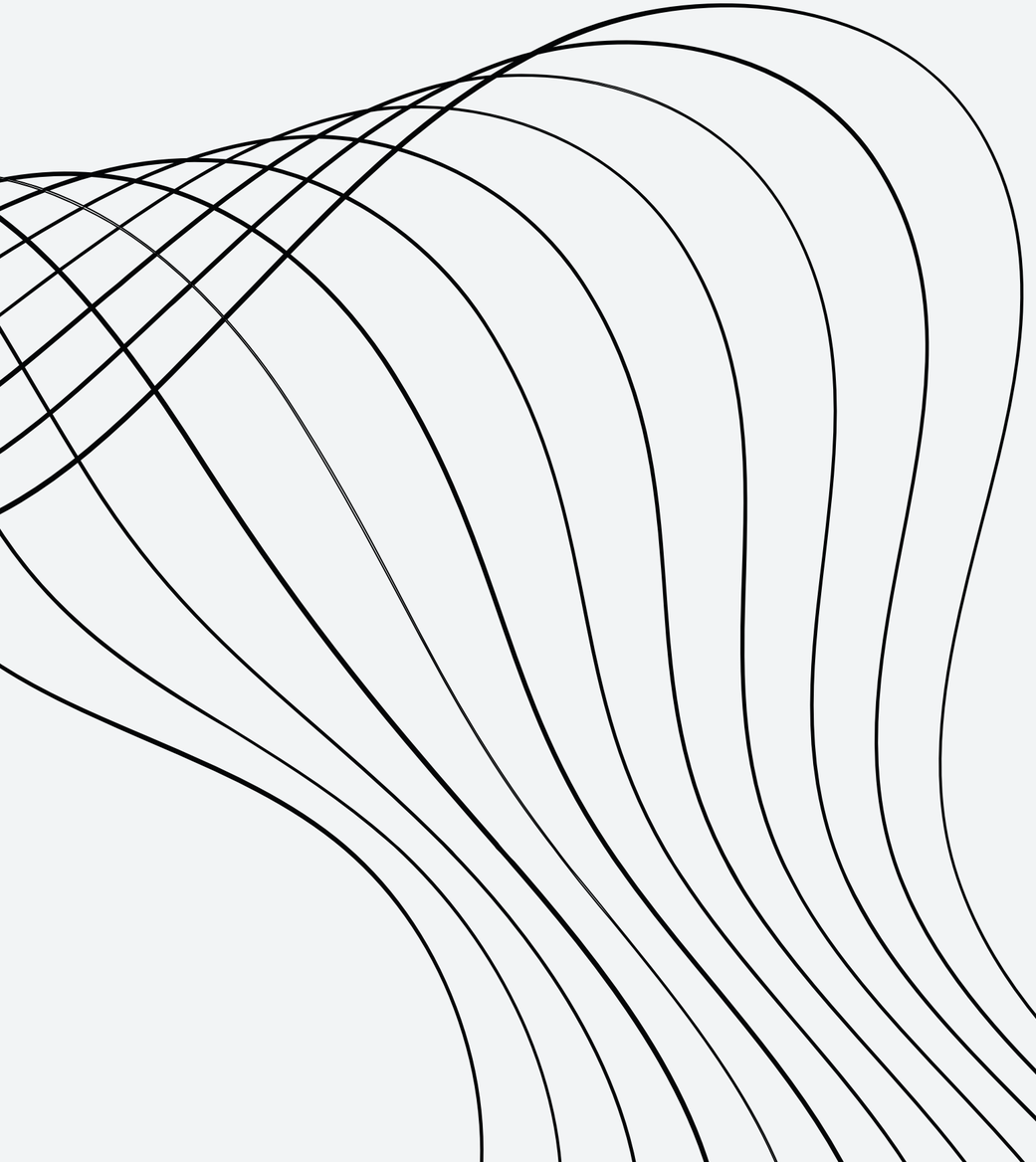
MODEL TRAIN

혼동 행렬 코드

```
y_pred_probs = model.predict(valid_images)
y_pred_labels = np.argmax(y_pred_probs, axis=1)
y_true = valid_labels

conf_matrix = confusion_matrix(y_true, y_pred_labels)

# 혼동 행렬을 정규화
conf_matrix_norm = conf_matrix.astype('float') / conf_matrix.sum(axis=1)[:, np.newaxis]

# 데이터에서 클래스 이름 가져오기
classes = ["0", "1", "2", "3", "4", "5", "6"] # Replace with your actual class names

# 히트맵 생성
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_norm, annot=True, fmt=".2f", cmap="Blues", xticklabels=classes, yticklabels=classes)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

JELLYFISH

우리가 만든 CNN 모델



Conv: 16, 3

Conv: 32, 3

Dropout: 0.25

Conv: 64, 3

Dropout: 0.25

	Type	# Parameters	Output Shape
	InputLayer	0	,224,224,3
	Conv2D	448	None, 224, 224, 16
	Activation	0	None, 224, 224, 16
	Conv2D	4640	None, 224, 224, 32
	Activation	0	None, 224, 224, 32
	MaxPooling2[	0	None, 112, 112, 32
	Dropout	0	None, 112, 112, 32
	Conv2D	18496	None, 112, 112, 64
	Activation	0	None, 112, 112, 64
	MaxPooling2[	0	None, 56, 56, 64
	Dropout	0	None, 56, 56, 64
	Flatten	0	None, 200704
	Dense	25690240	None, 128
	Activation	0	None, 128
	Dense	4128	None, 32
	Activation	0	None, 32
	Dense	231	None, 7

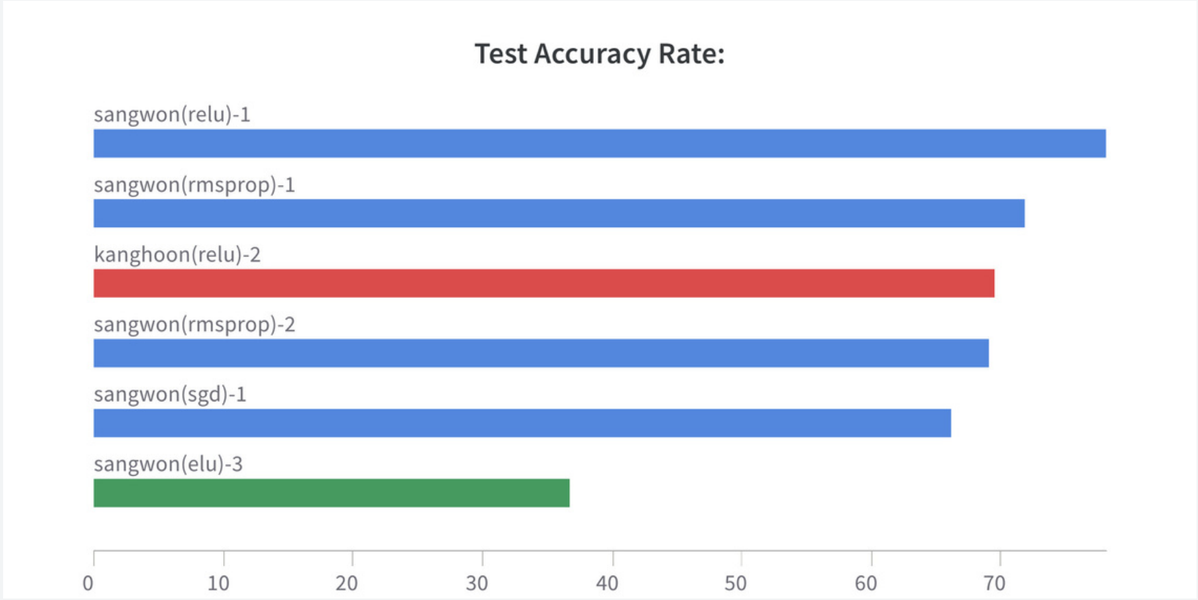
JELLYFISH

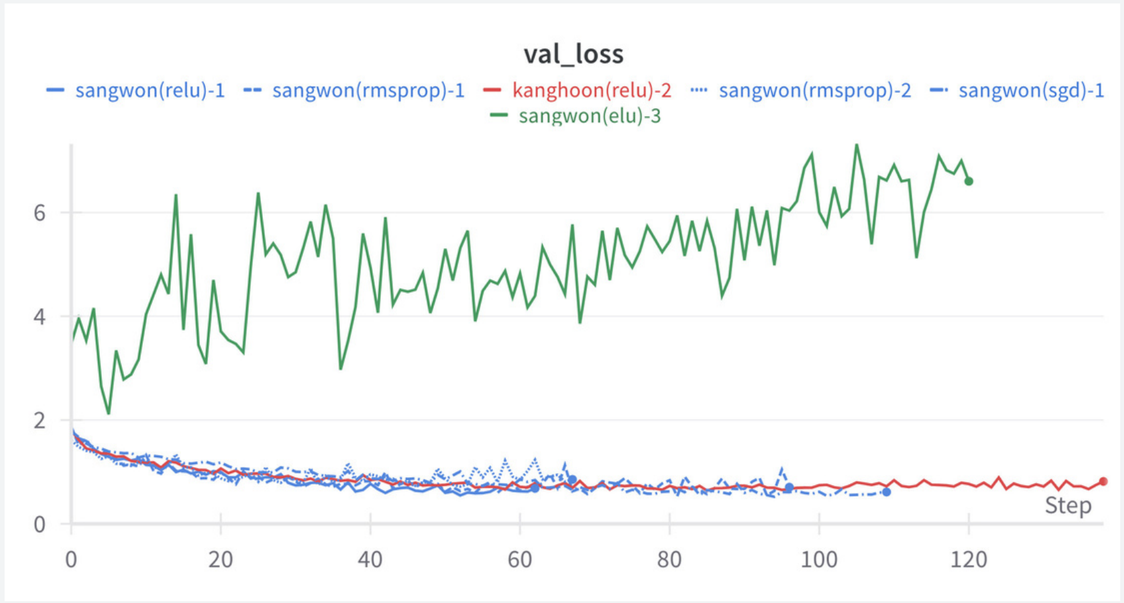
JELLYFISH

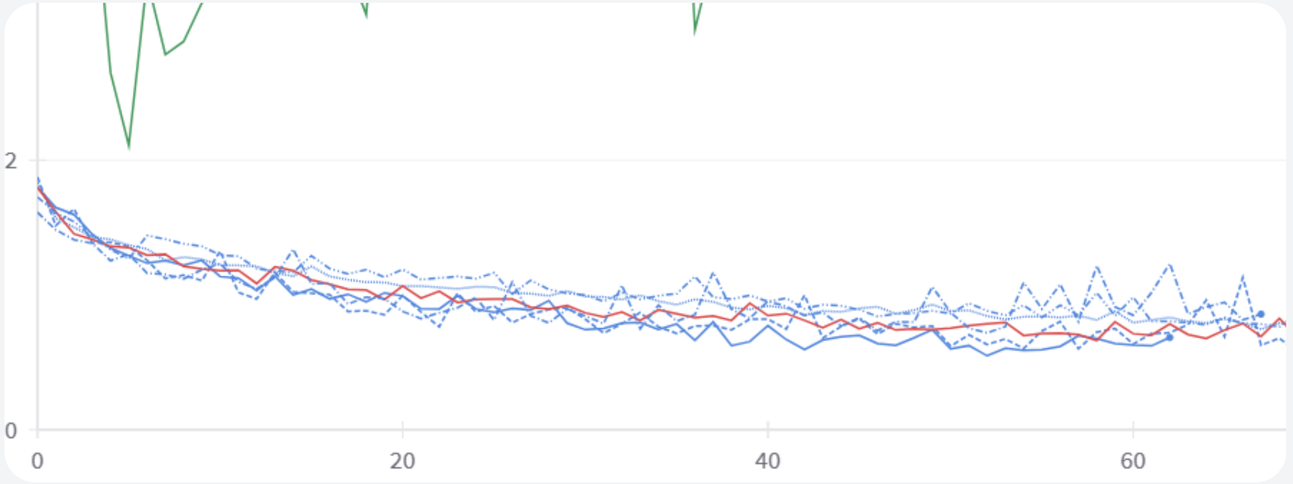
CNN 모델 학습 결과

Name (6 visualized)	optimizer	activation	learning_rate	epoch	loss	val_loss	accuracy	val_accuracy	Test Accuracy
sangwon(relu)-1	adam	relu	0.0007294	62	0.387	0.6841	0.8679	0.8025	78.1
sangwon(rmsprop)-1	rmsprop	relu	0.0007294	96	0.3311	0.7072	0.9006	0.842	71.9
kanghoon(relu)-2	adam	relu	0.0002418	138	0.1897	0.8184	0.9346	0.8123	69.52
sangwon(rmsprop)-2	rmsprop	relu	0.0009458	67	0.4592	0.8577	0.85	0.7605	69.05
sangwon(sgd)-1	sgd	relu	0.01	109	0.2707	0.6144	0.9062	0.8099	66.19
sangwon(elu)-3	adam	elu	0.0001807	120	0.3412	6.602	0.8821	0.3802	36.67

CNN 학습 결과



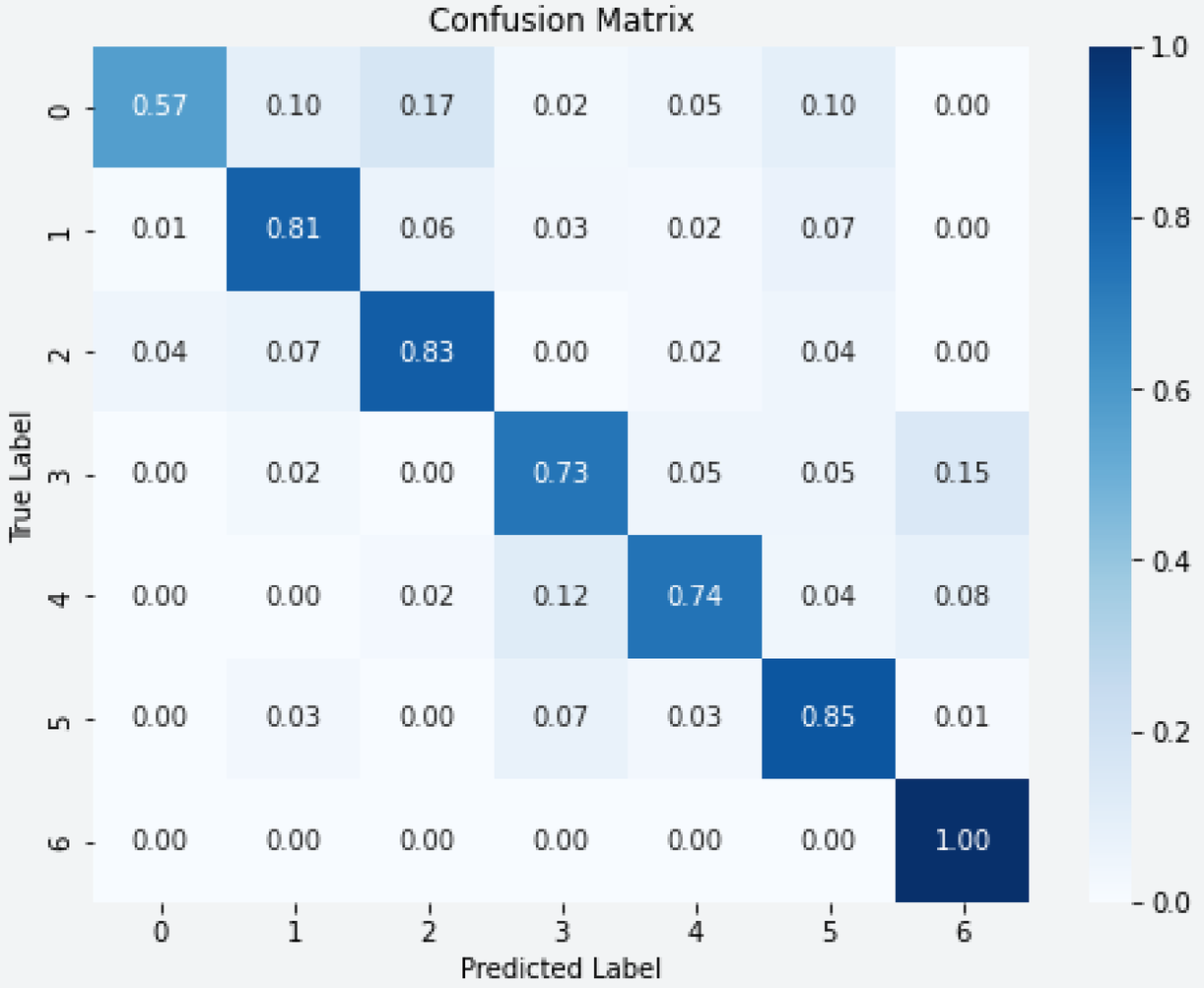




JELLYFISH

CNN 1차 혼동 행렬

```
라벨0에 해당하는 클래스 : Moon_jellyfish
라벨1에 해당하는 클래스 : barrel_jellyfish
라벨2에 해당하는 클래스 : blue_jellyfish
라벨3에 해당하는 클래스 : compass_jellyfish
라벨4에 해당하는 클래스 : lions_mane_jellyfish
라벨5에 해당하는 클래스 : mauve_stinger_jellyfish
라벨6에 해당하는 클래스 : teuberculata_jellyfish
```



CNN 2차(Conv2D 1x1 추가)

Conv: 16, 3

Conv: 32, 3

Conv: 64, 1

Activation

Conv: 64, 3

	Type	# Parameters	Output Shape
●	InputLayer	0	,224,224,3
●	Conv2D	448	None, 224, 224, 16
●	Activation	0	None, 224, 224, 16
●	Conv2D	4640	None, 224, 224, 32
●	Activation	0	None, 224, 224, 32
●	MaxPooling2[	0	None, 112, 112, 32
●	Dropout	0	None, 112, 112, 32
●	Conv2D	2112	None, 112, 112, 64
●	Activation	0	None, 112, 112, 64
●	Conv2D	36928	None, 112, 112, 64
●	Activation	0	None, 112, 112, 64
●	MaxPooling2[	0	None, 56, 56, 64
●	Dropout	0	None, 56, 56, 64
●	Flatten	0	None, 200704
●	Dense	25690240	None, 128
●	Activation	0	None, 128
●	Dense	4128	None, 32
●	Activation	0	None, 32
●	Dense	231	None, 7

JELLYFISH

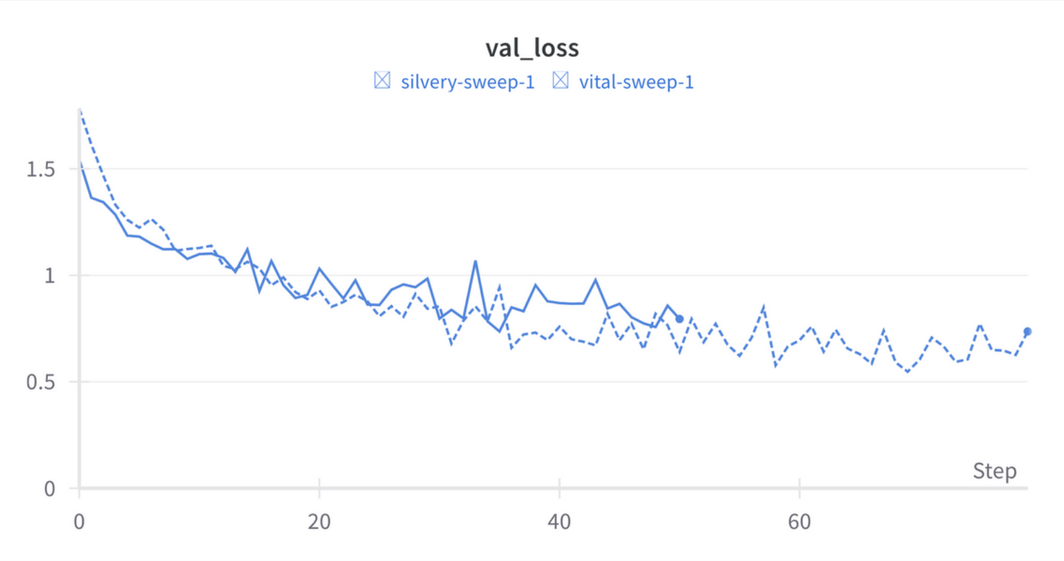
CNN 모델 2차 학습 결과

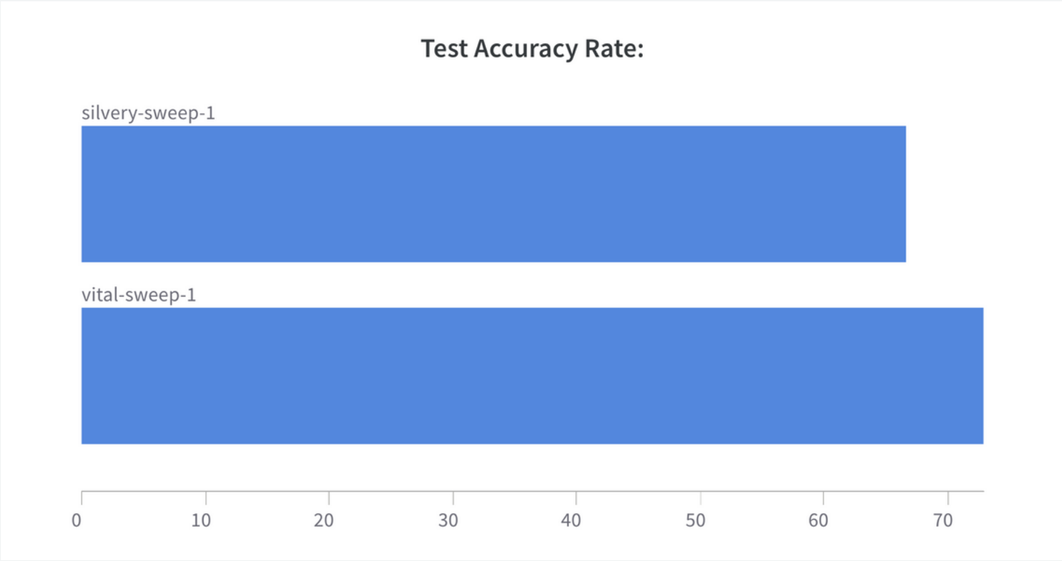
JELLYFISH

Name (2 visualized)	optimizer	activation	learning_rate	epoch	loss	val_loss	accuracy	val_accuracy	Test Accuracy
vital-sweep-1	adam	relu	0.0007294	79	0.3143	0.7367	0.8926	0.7877	72.86
silvery-sweep-1	adam	relu	0.001	50	0.4007	0.7951	0.8556	0.8	66.67

CNN 2차 학습 결과

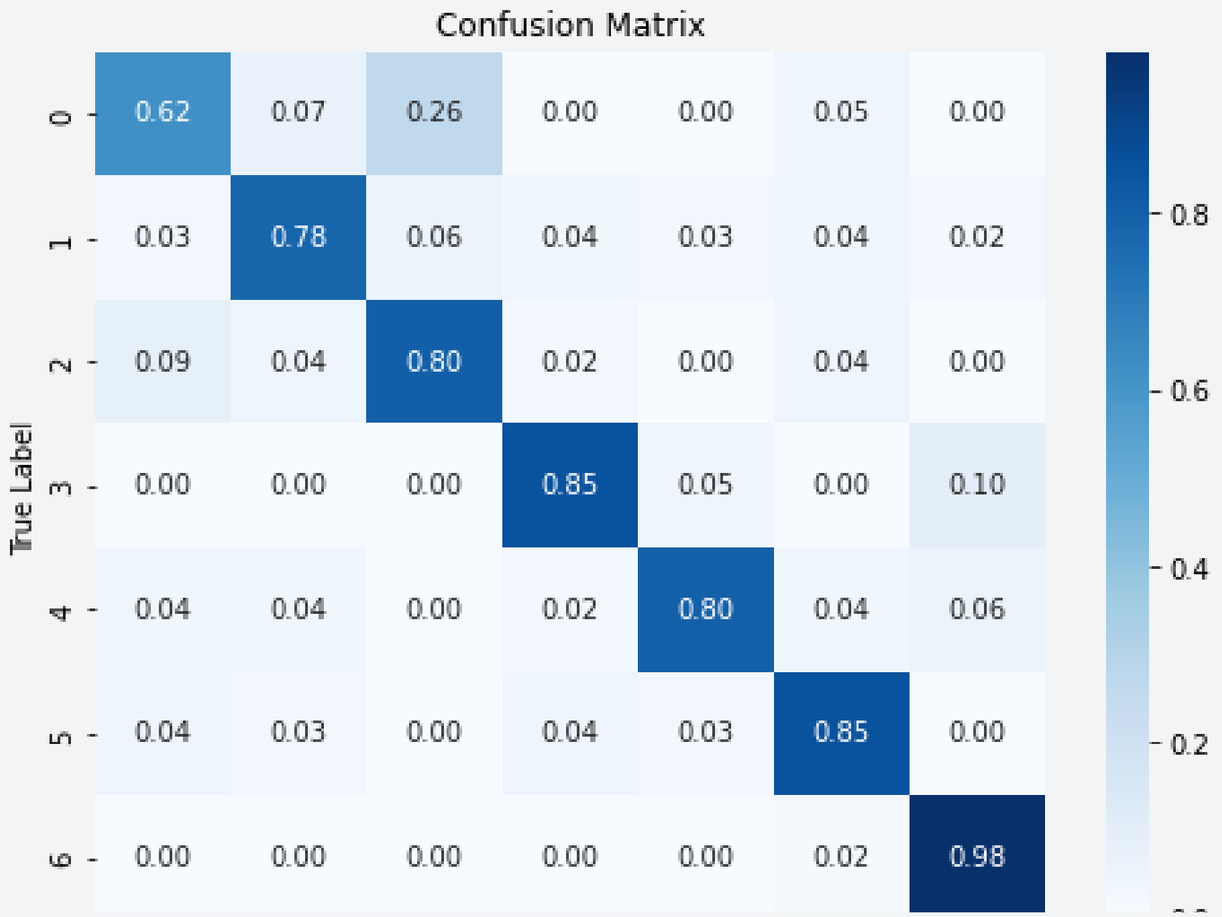
JELLYFISH





CNN 2차 혼동 행렬

라벨0에 해당하는 클래스 : Moon_jellyfish
라벨1에 해당하는 클래스 : barrel_jellyfish
라벨2에 해당하는 클래스 : blue_jellyfish
라벨3에 해당하는 클래스 : compass_jellyfish
라벨4에 해당하는 클래스 : lions_mane_jellyfish
라벨5에 해당하는 클래스 : mauve_stinger_jellyfish
라벨6에 해당하는 클래스 : teuberculata_jellyfish



MobileNetV2

	Name	Type	# Parameters	Output Shape
●	mobilenetv2_1.00_224_input	InputLayer	0	,224,224,3
●	mobilenetv2_1.00_224	Functional	2257984	None, 7, 7, 1280
●	global_average_pooling2d	GlobalAveragePooling2D	0	None, 1280
●	dense	Dense	655872	None, 512
●	dense_1	Dense	3591	None, 7

JELLYFISH

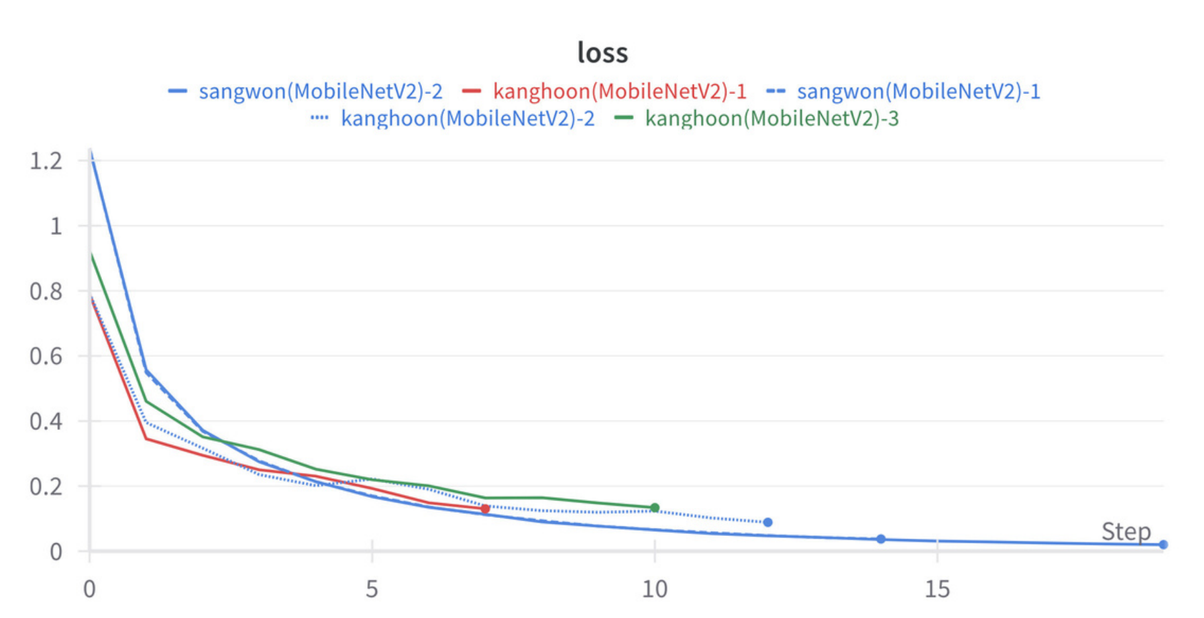
MobileNetV2 학습 결과

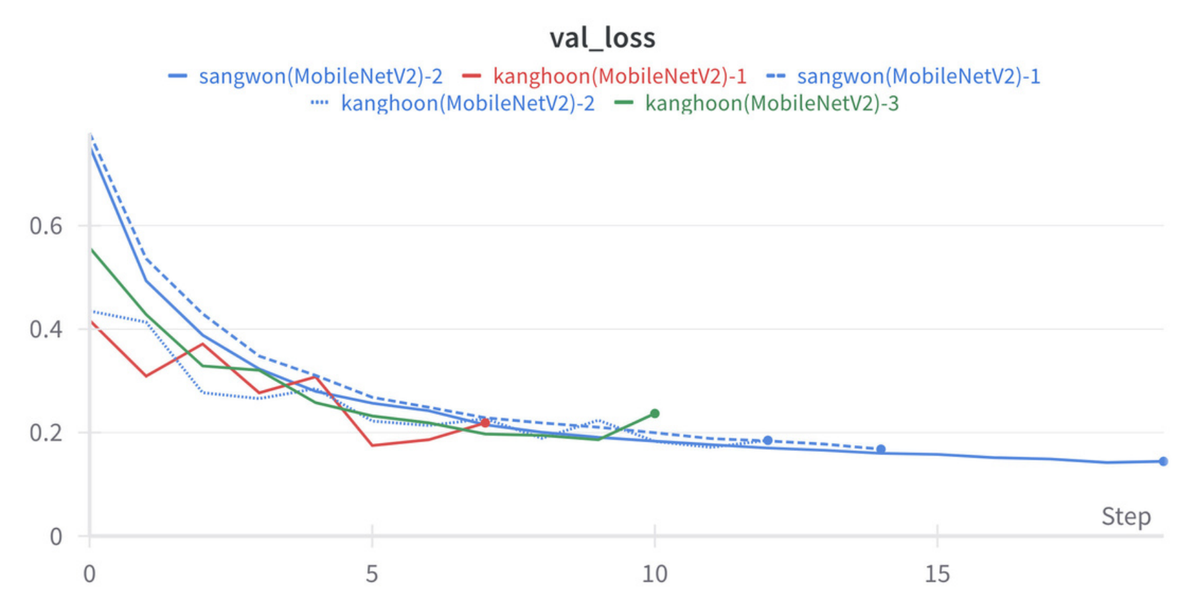
Name (5 visualized)	optimizer	activation	learning_rate	epoch	loss	val_loss	accuracy	val_accuracy	Test Accuracy
sangwon(MobileNetV2)-2	adam	relu	0.0001	19	0.02032	0.1444	1	0.963	88.57
kanghoon(MobileNetV2)-1	adam	relu	0.0007409	7	0.1304	0.2187	0.9562	0.9309	88.1
sangwon(MobileNetV2)-1	adam	relu	0.0001	14	0.03757	0.168	0.9994	0.9556	87.14
kanghoon(MobileNetV2)-2	adam	relu	0.0004393	12	0.0888	0.1851	0.9741	0.9481	87.14
kanghoon(MobileNetV2)-3	adam	relu	0.0002207	10	0.1338	0.2369	0.9549	0.921	87.14

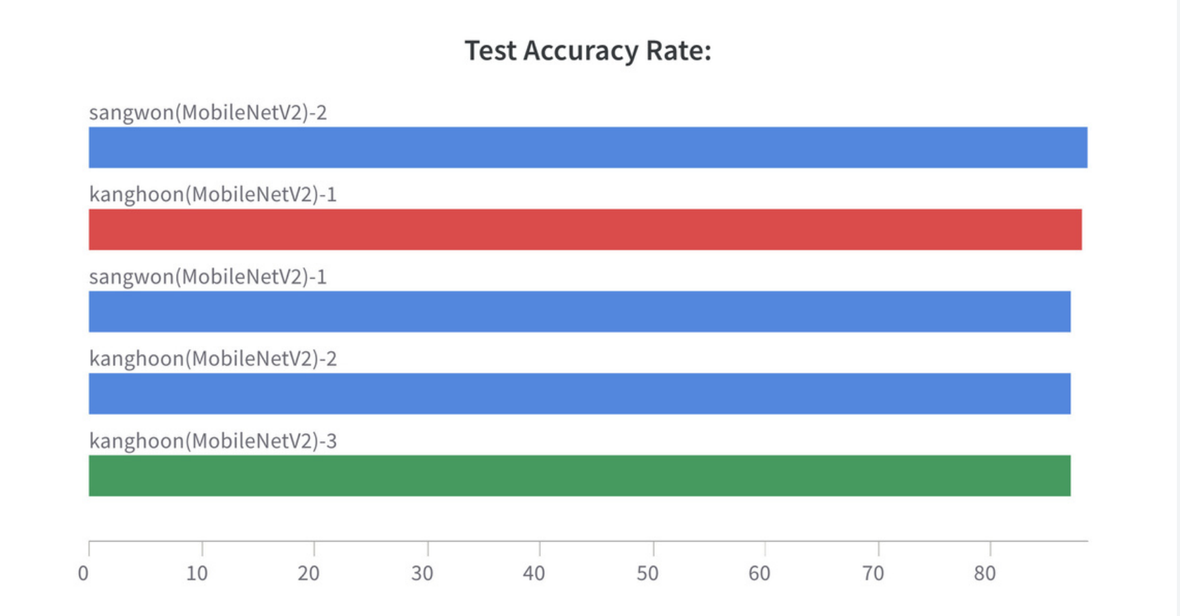
JELLYFISH

MobileNetV2 학습 결과

JELLYFISH

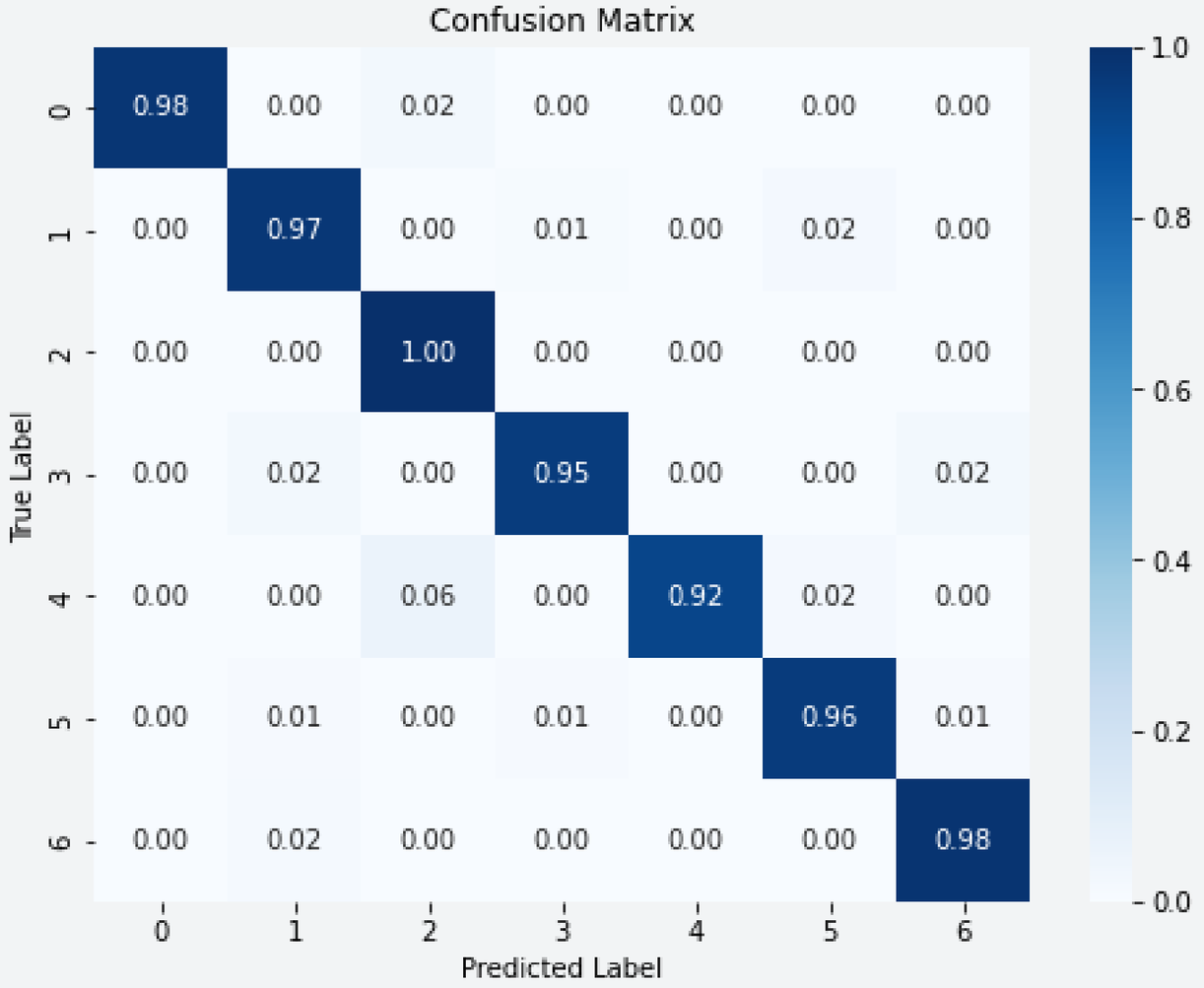






MobileNet 혼동 행렬

```
라벨0에 해당하는 클래스 : Moon_jellyfish
라벨1에 해당하는 클래스 : barrel_jellyfish
라벨2에 해당하는 클래스 : blue_jellyfish
라벨3에 해당하는 클래스 : compass_jellyfish
라벨4에 해당하는 클래스 : lions_mane_jellyfish
라벨5에 해당하는 클래스 : mauve_stinger_jellyfish
라벨6에 해당하는 클래스 : teuberculata_jellyfish
```



VGG 16 & 19

	Name	Type	# Parameters	Output Shape
●	vgg19_input	InputLayer	0	,224,224,3
●	vgg19	Functional	20024384	None, 7, 7, 512
●	global_average_pooling2	GlobalAveragePooling2D	0	None, 512
●	dense	Dense	262656	None, 512
●	dense_1	Dense	3591	None, 7

JELLYFISH

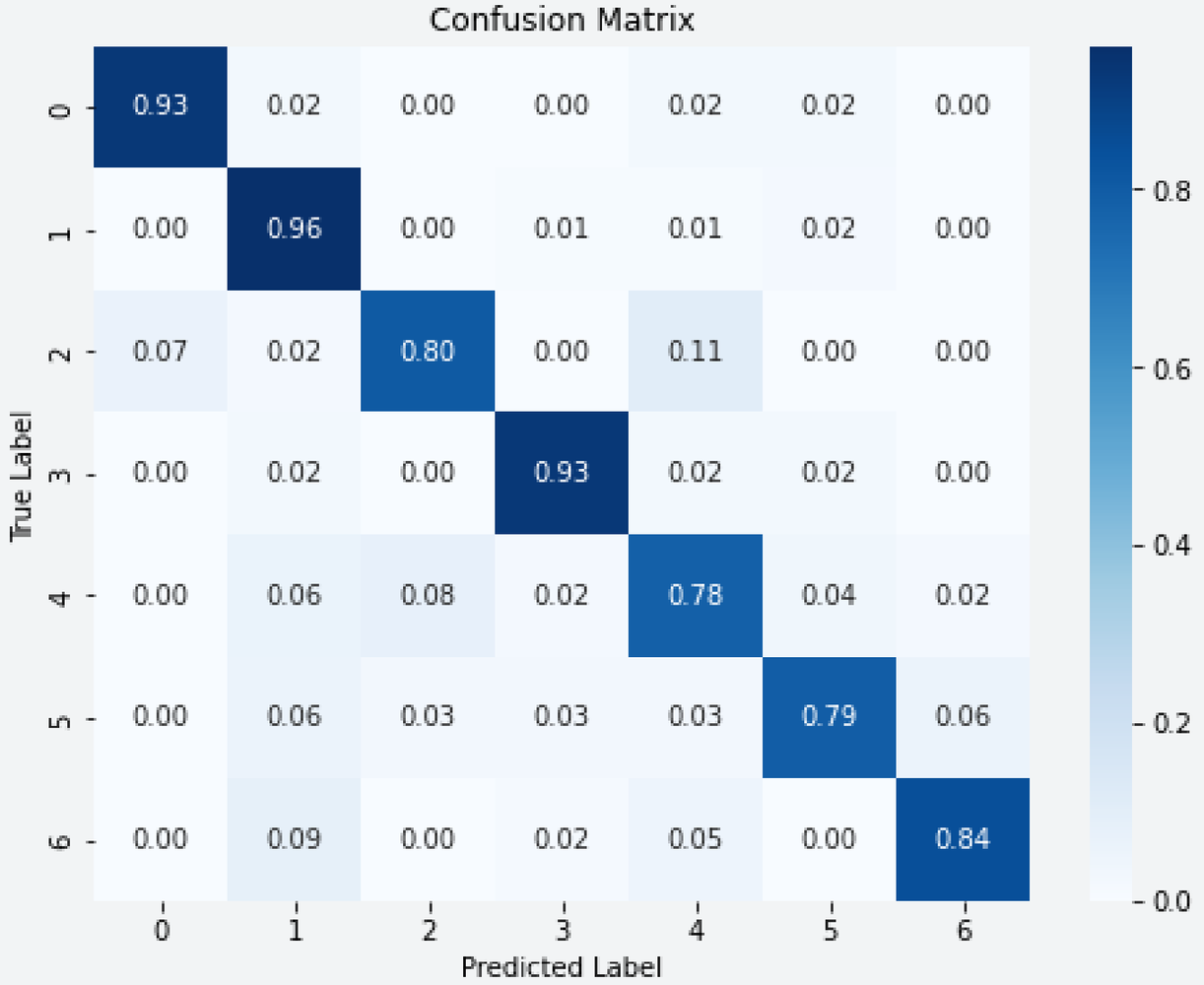
VGG 16 & 19 학습 결과

Name (3 visualized)	optimizer	activation	learning_rate	epoch	loss	val_loss	accuracy	val_accuracy	Test Accuracy
sangwon(VGG19)-2	adam	relu	0.001	39	0.1025	0.3061	0.9778	0.8914	81.43
sangwon(VGG19)-1	adam	relu	0.001	26	0.1853	0.3533	0.95	0.8667	80.48
sangwon(VGG16)	adam	relu	0.001	15	0.2154	0.4457	0.9444	0.8667	83.81

JELLYFISH

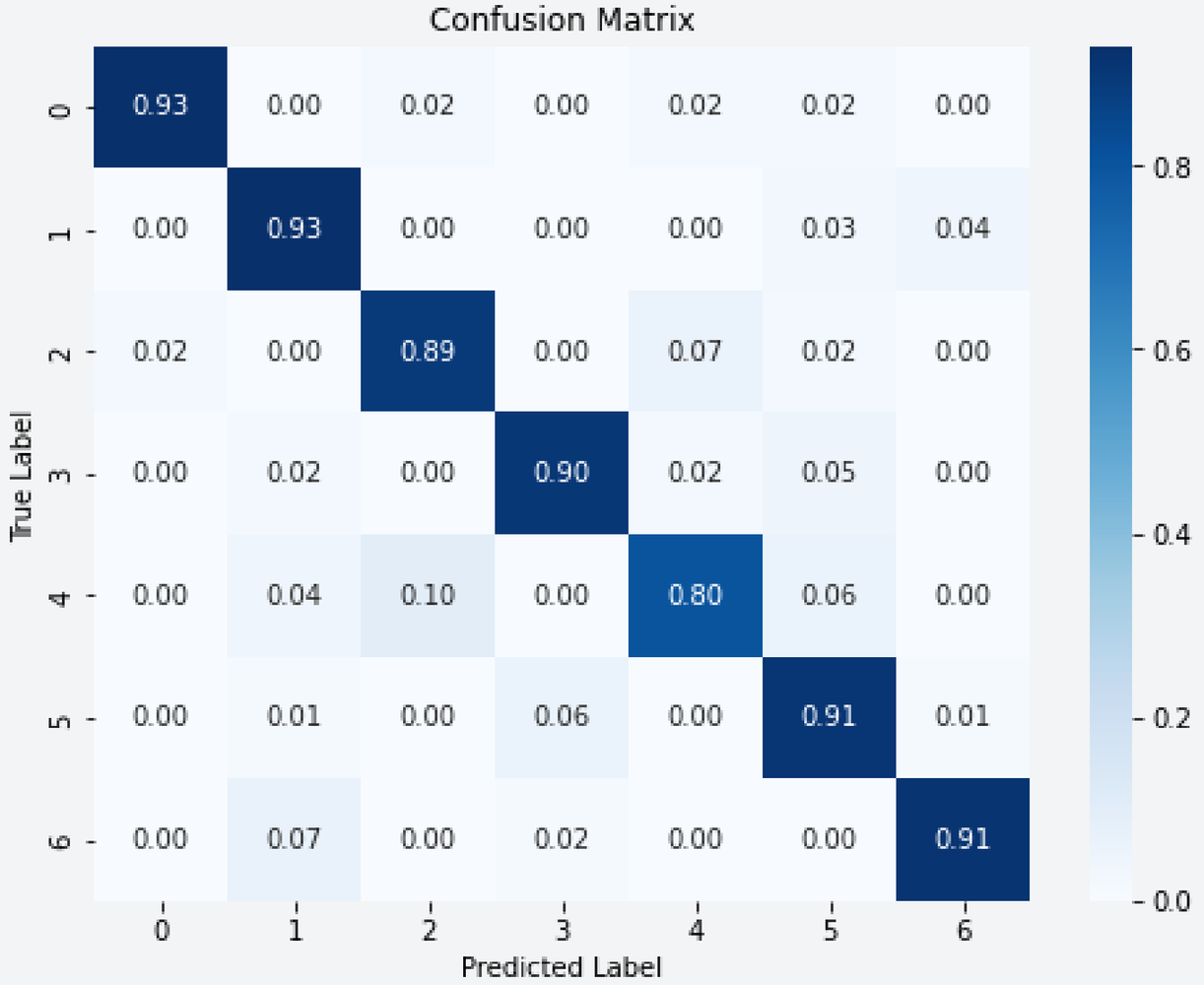
VGG16 혼동 행렬

```
라벨0에 해당하는 클래스 : Moon_jellyfish
라벨1에 해당하는 클래스 : barrel_jellyfish
라벨2에 해당하는 클래스 : blue_jellyfish
라벨3에 해당하는 클래스 : compass_jellyfish
라벨4에 해당하는 클래스 : lions_mane_jellyfish
라벨5에 해당하는 클래스 : mauve_stinger_jellyfish
라벨6에 해당하는 클래스 : teuberculata_jellyfish
```



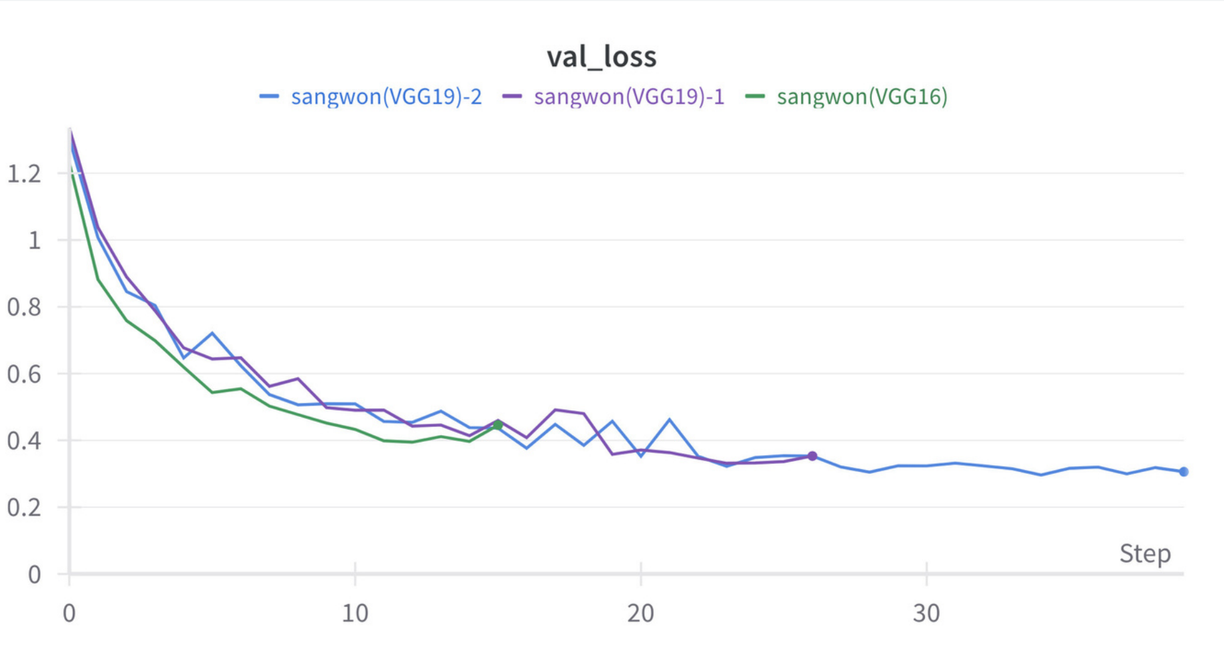
VGG19 혼동 행렬

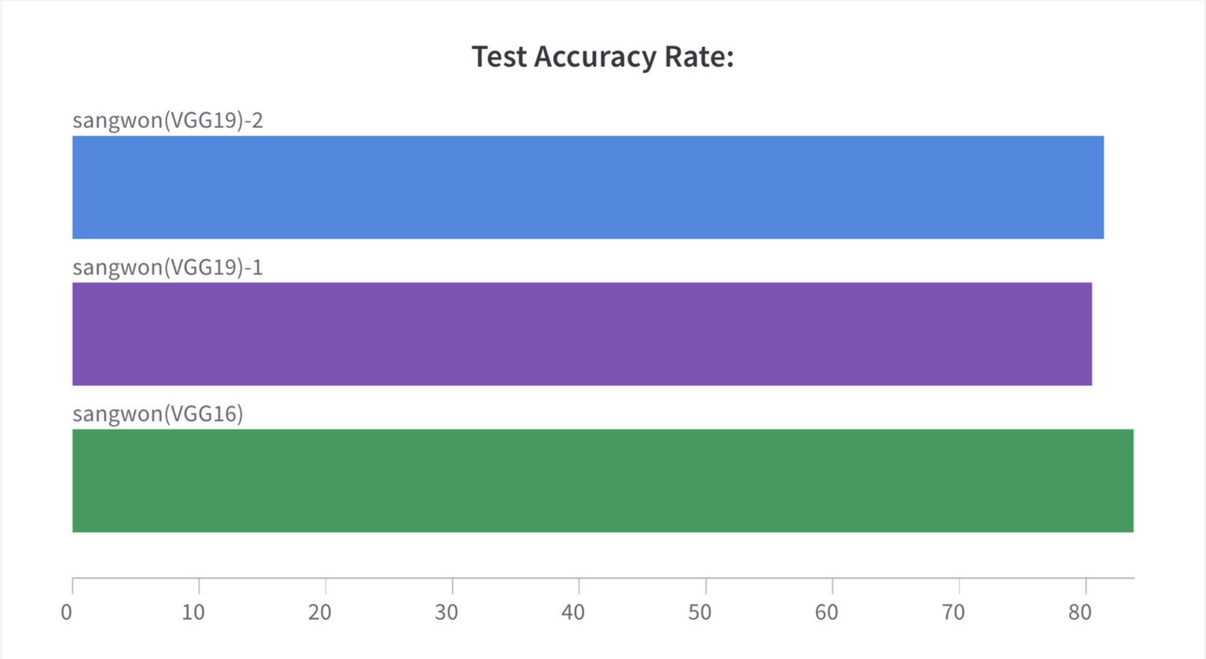
라벨0에 해당하는 클래스 : Moon_jellyfish
라벨1에 해당하는 클래스 : barrel_jellyfish
라벨2에 해당하는 클래스 : blue_jellyfish
라벨3에 해당하는 클래스 : compass_jellyfish
라벨4에 해당하는 클래스 : lions_mane_jellyfish
라벨5에 해당하는 클래스 : mauve_stinger_jellyfish
라벨6에 해당하는 클래스 : teuberculata_jellyfish



VGG 16 & 19 학습 결과

JELLYFISH





GoogleNet

	Name	Type	# Parameters	Output Shape
●	inception_v3_input	InputLayer	0	,224,224,3
●	inception_v3	Functional	21802784	None, 5, 5, 2048
●	global_average_pooling2d	GlobalAveragePooling2D	0	None, 2048
●	dense	Dense	1049088	None, 512
●	dense_1	Dense	3591	None, 7

JELLYFISH

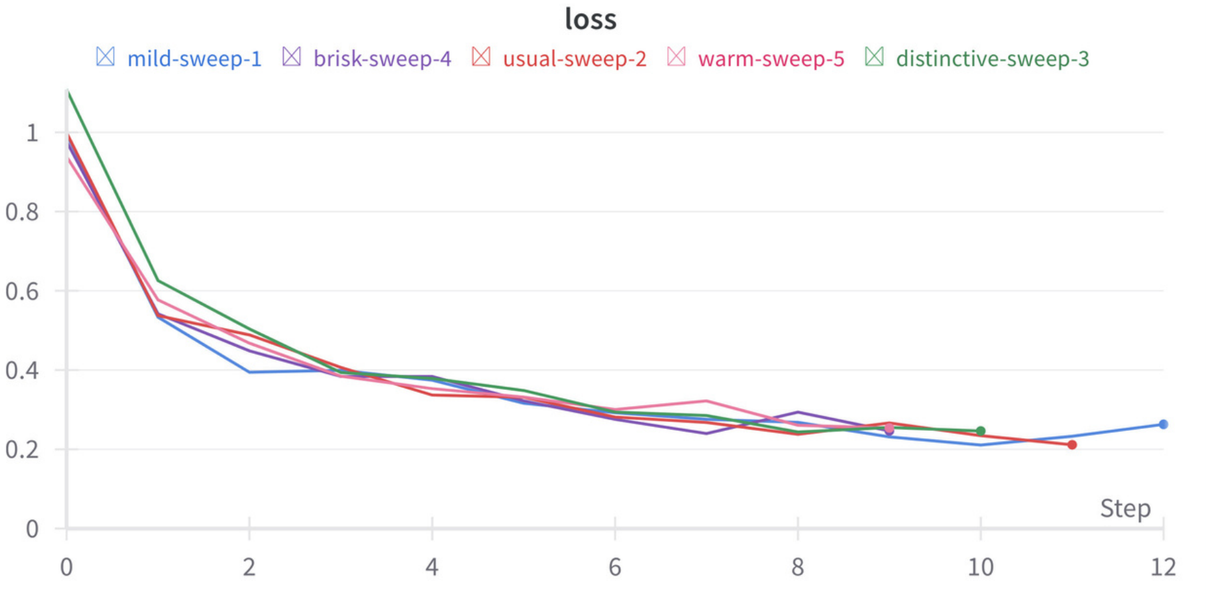
GoogleNet 학습 결과

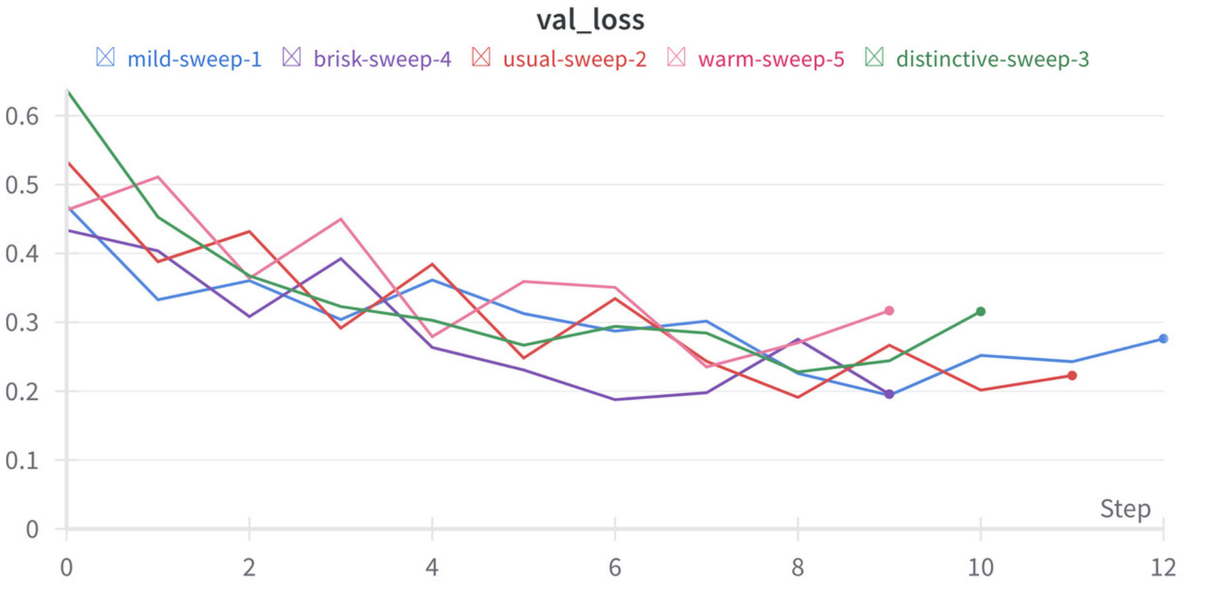
JELLYFISH

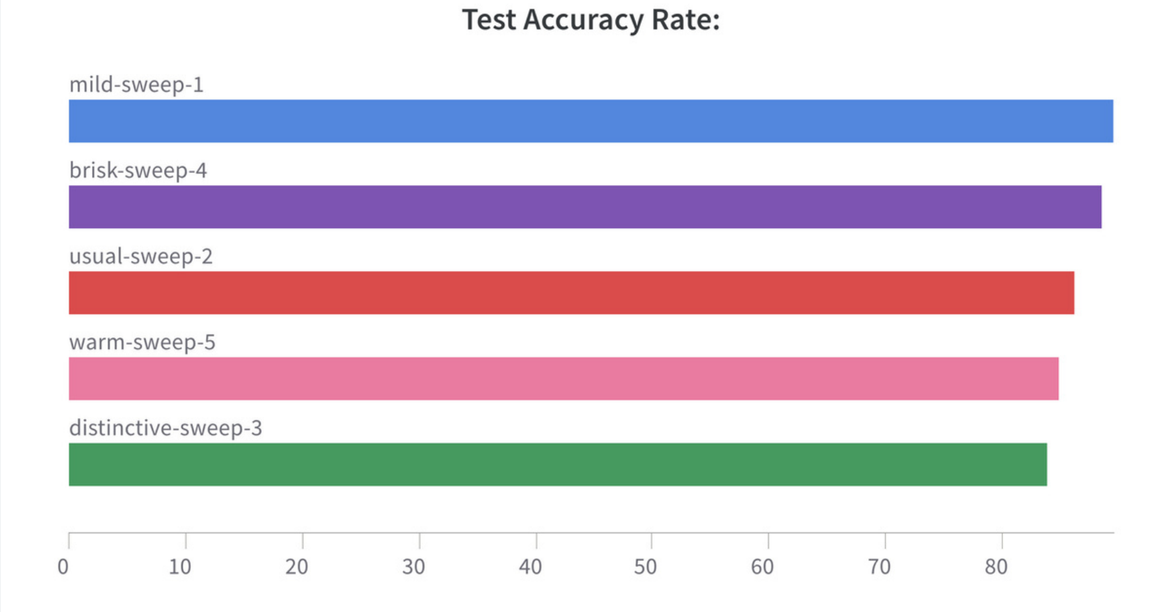
Name (5 visualized)	Runtime	activat	learning_rate	accuracy	epoch	loss	val_accuracy	val_loss	Test Accuracy
GoogleNet_1	8m 47s	relu	0.0004772	0.9123	12	0.263	0.9062	0.2762	89.52
GoogleNet_2	7m 36s	relu	0.0003081	0.921	9	0.2465	0.9284	0.1957	88.57
GoogleNet_3	8m 8s	relu	0.0003115	0.9253	11	0.2115	0.9235	0.2228	86.19
GoogleNet_4	6m 10s	relu	0.000492	0.9235	9	0.254	0.8864	0.317	84.76
GoogleNet_5	8m 33s	relu	0.000147	0.9272	10	0.2463	0.8963	0.3158	83.81

GoogleNet
학습 결과

JELLYFISH

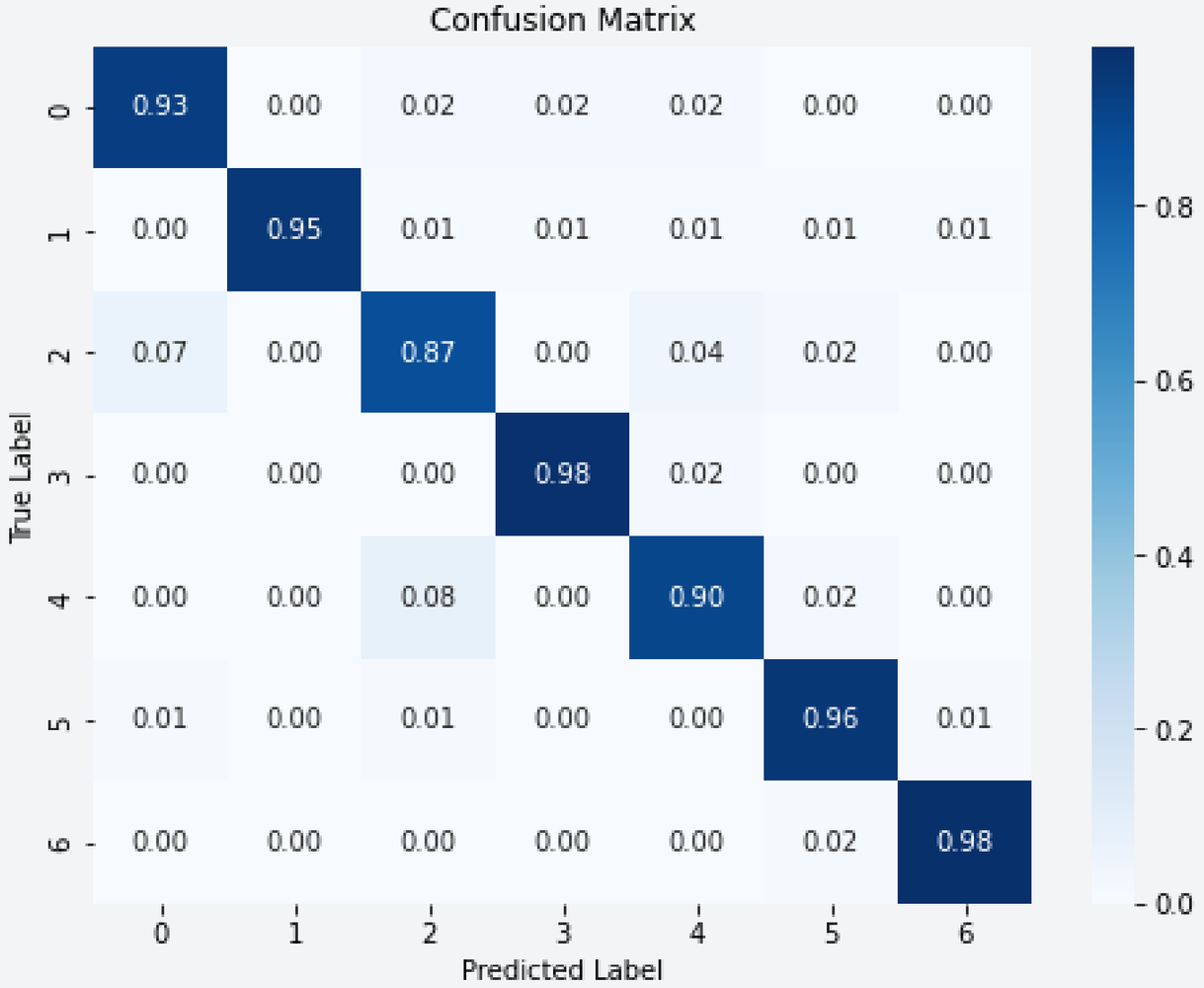






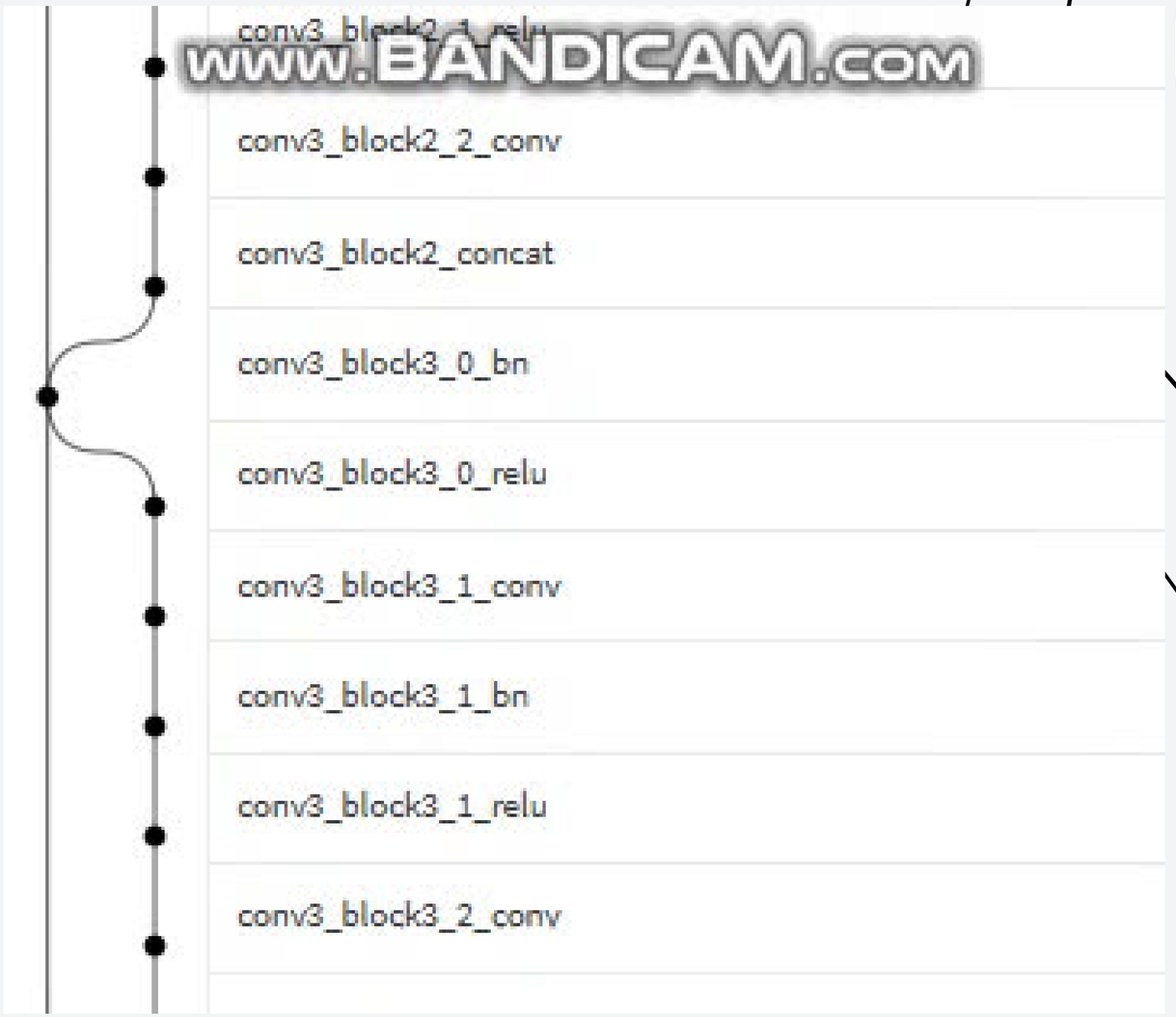
GoogleNet 혼동 행렬

라벨0에 해당하는 클래스 : Moon_jellyfish
라벨1에 해당하는 클래스 : barrel_jellyfish
라벨2에 해당하는 클래스 : blue_jellyfish
라벨3에 해당하는 클래스 : compass_jellyfish
라벨4에 해당하는 클래스 : lions_mane_jellyfish
라벨5에 해당하는 클래스 : mauve_stinger_jellyfish
라벨6에 해당하는 클래스 : teuberculata_jellyfish



DenseNet

JELLYFISH



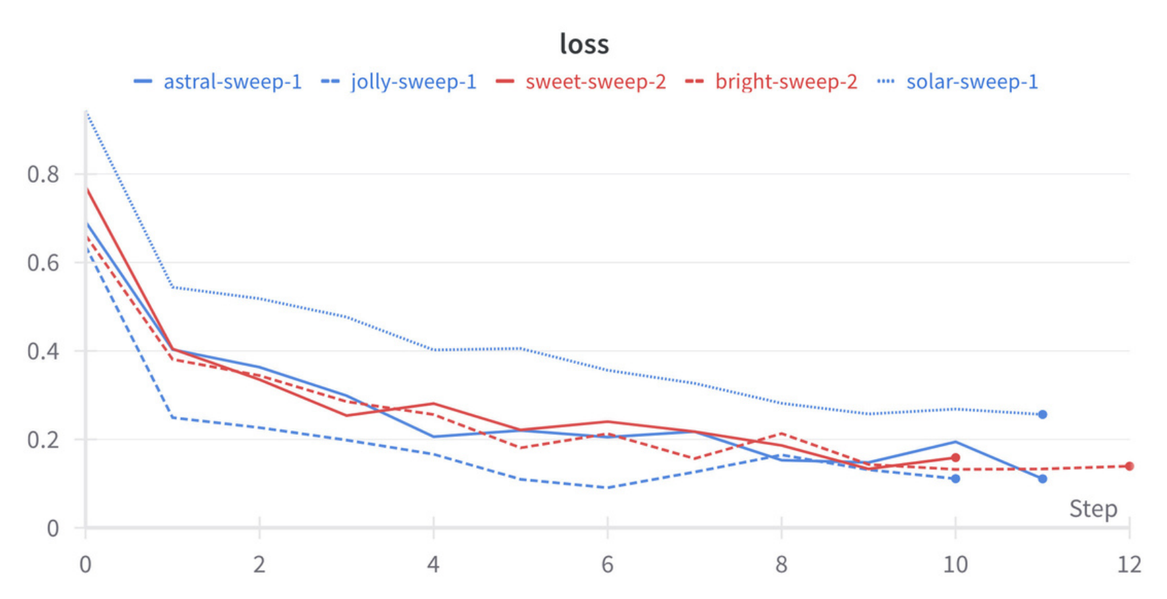
DenseNet 학습 결과

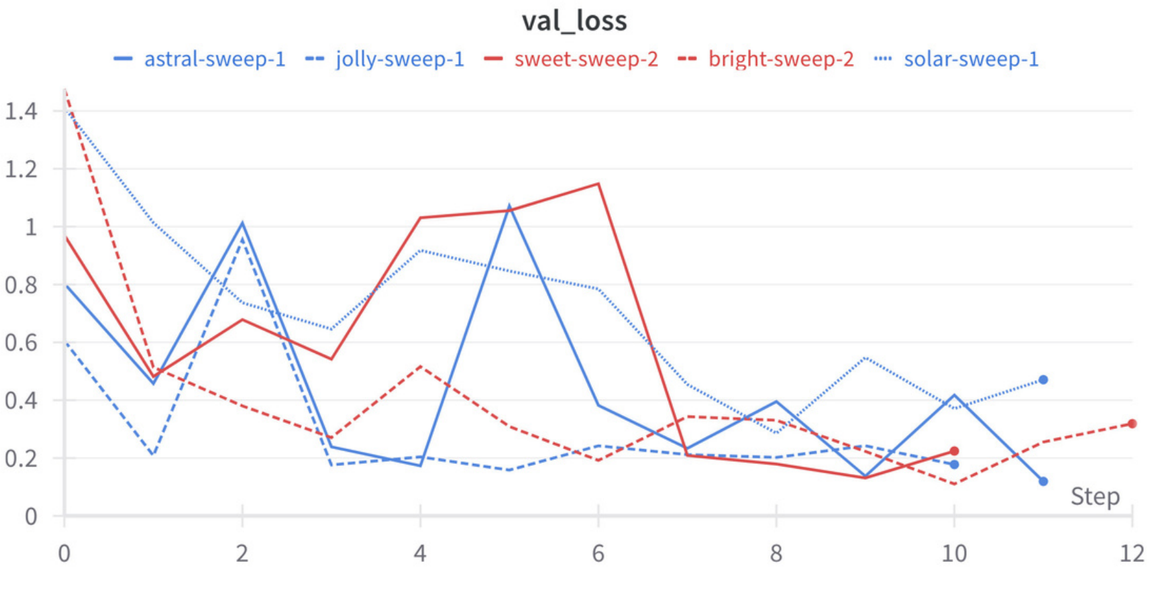
Name (5 visualized)	optimizer	activation	learning_rate	epoch	loss	val_los	accuracy	val_accuracy	Test Accuracy Rate:
astral-sweep-1	adam	tanh	0.0004178	11	0.1108	0.1192	0.9623	0.9556	91.9
jolly-sweep-1	adam	tanh	0.0002703	10	0.1111	0.1776	0.9586	0.9432	90.48
sweet-sweep-2	adam	tanh	0.0004314	10	0.1588	0.2243	0.9469	0.9284	85.24
bright-sweep-2	adam	tanh	0.0004354	12	0.1394	0.3189	0.9525	0.8963	81.43
solar-sweep-1	adam	tanh	0.0007235	11	0.2564	0.471	0.9148	0.8444	80.48

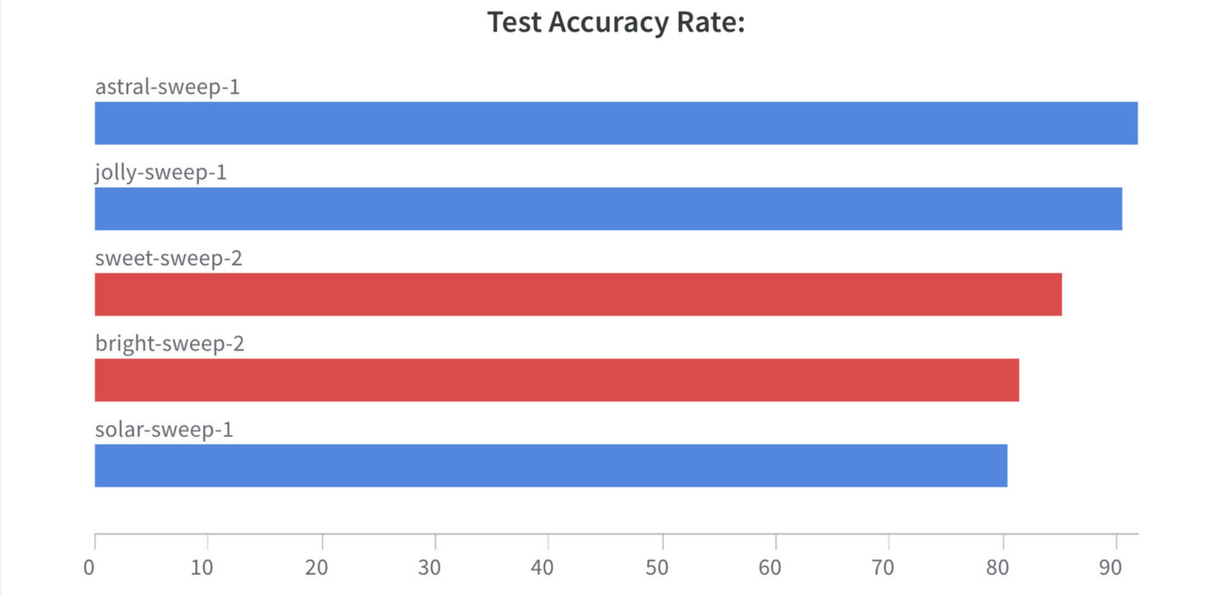
JELLYFISH

DenseNet 학습 결과

JELLYFISH

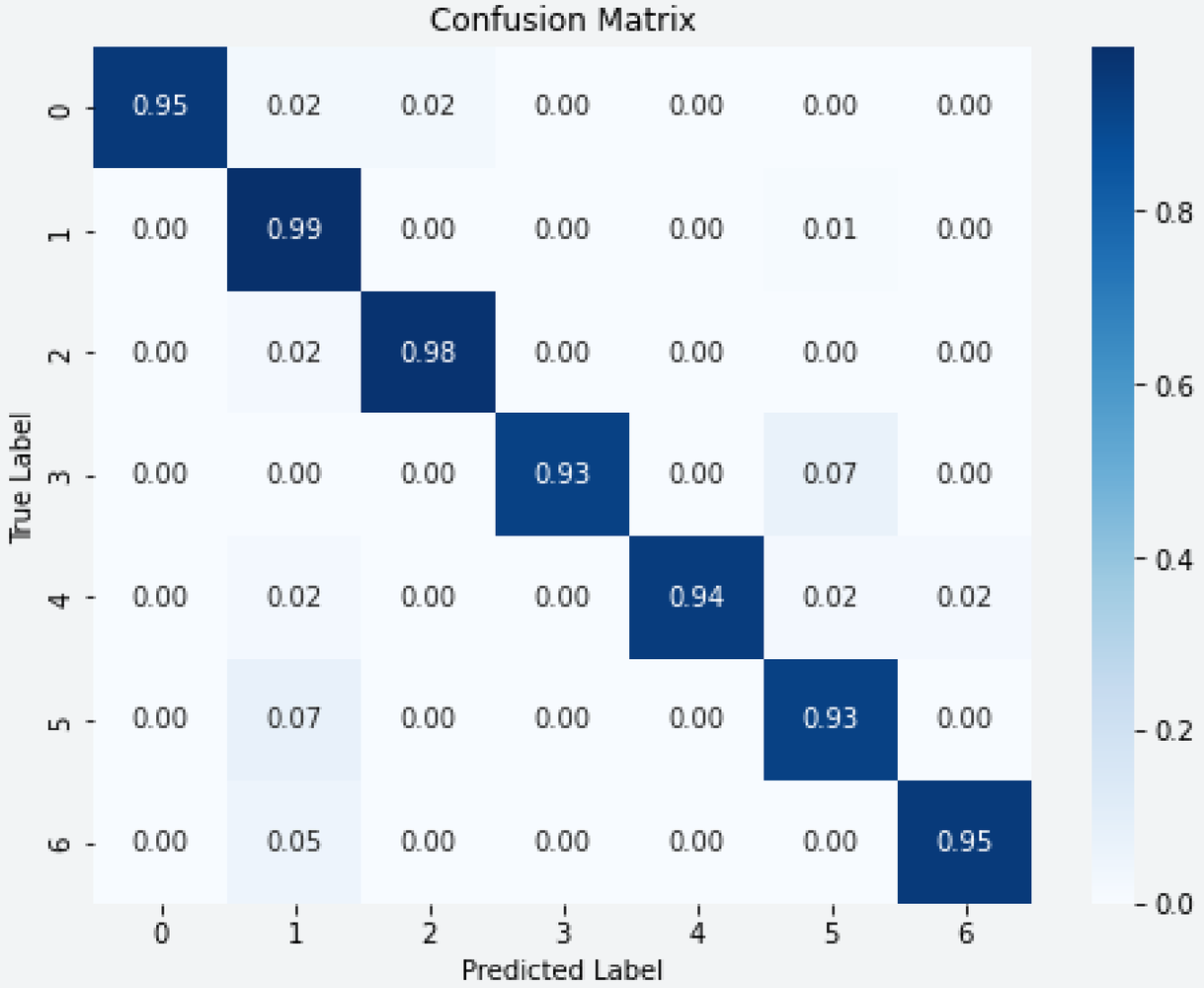






DenseNet 혼동 행렬

```
라벨0에 해당하는 클래스 : Moon_jellyfish
라벨1에 해당하는 클래스 : barrel_jellyfish
라벨2에 해당하는 클래스 : blue_jellyfish
라벨3에 해당하는 클래스 : compass_jellyfish
라벨4에 해당하는 클래스 : lions_mane_jellyfish
라벨5에 해당하는 클래스 : mauve_stinger_jellyfish
라벨6에 해당하는 클래스 : teuberculata_jellyfish
```



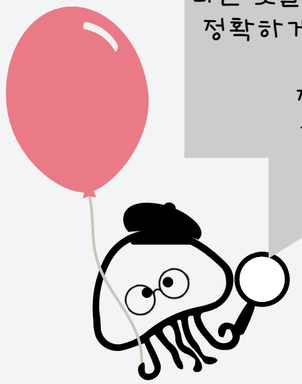
BUSINESS MODEL

스쿠버 다이빙,
해양 구조대,
민간 어부

연구기관,
해파리 의약품
제조업체

JELLYFISH

데이터 전처리를 하는 부분에서 매우 많은 시간이 걸렸습니다.
데이터를 수집하는 것 부터 불러오는 것만으로도 하루가 걸릴 정도로
기초적인 코딩실력이 부족하다는 것을 느꼈고, 팀원들과 같이 코드를 작성
하면서 이야기를 나누는 부분에서 많은 학습을 했던 것 같습니다.
또한, 불균형한 데이터와 학습률, 모델 구조마다 정확도가 매우 많이 차이
나는 것을 느꼈습니다. 전이학습을 무턱대고 쓴다기 보다는 그 모델의 구조를
정확하게 이해를 하고 작성해야 한다는 것을 뼈저리게 알게되었습니다.
생물, 사물 부분에서는 전이학습된 모델구조를 사용하면
정말 빠르게 학습이되고, 좋은 결과를 도출 해낼수 있어서
다양한 비즈니스 모델을 만드는데 도움이 될것 같습니다.



JELLYFISH

밤샘 작업은 너무 너무 힘들다 해파리냉채 올 여름 오기 전까지 안먹을테야...
데이터 전처리가 모델에 정말 큰 영향을 끼친다는 것을 느꼈고, 모델을 만들때
데이터에 대한 이해도가 높은 것이 중요하다고 생각했습니다.

KEEP: 전처리에 많은 시간을 할애한 것.
PROBLEM: 전처리 이후에 모델을 만들고 프로젝트를 잘 마무리는 것까지의 시간 분배의 어려움.
TRY: 시간 분배를 좀 더 잘하고 데이터 전처리를 익숙하게 처리하도록 코드 필사해보기

다같이 회고 한마디

JELLYFISH

데이터 전처리부터 모델 구성 및 학습까지,

모든 과정에 팀원들의 고생과 노력이 있었기에

부족하지만, 지금과 같은 결과를 만들 수 있었습니다.

인공지능을 만드는 작업은 매우 인간적인 일이라는 것을 느꼈습니다.

JELLYFISH

전처리를 잘 하는 것부터가 모델 훈련의 시작이라는 것을 느꼈고
사람의 미세한 튜닝 능력이 정말 중요하구나...도 느꼈습니다.

이제는 해파리를 봐도
기계 입장에서 어떤 것이 유사하고 어떤 것으로 분류하게 될 지
다른 각도로 보게 됩니다. ^^

JELLYFISH

REFERENCE

JELLYFISH

데이터 수집

WIKIMEDIA

ADOVE STOCK

GOOGLE

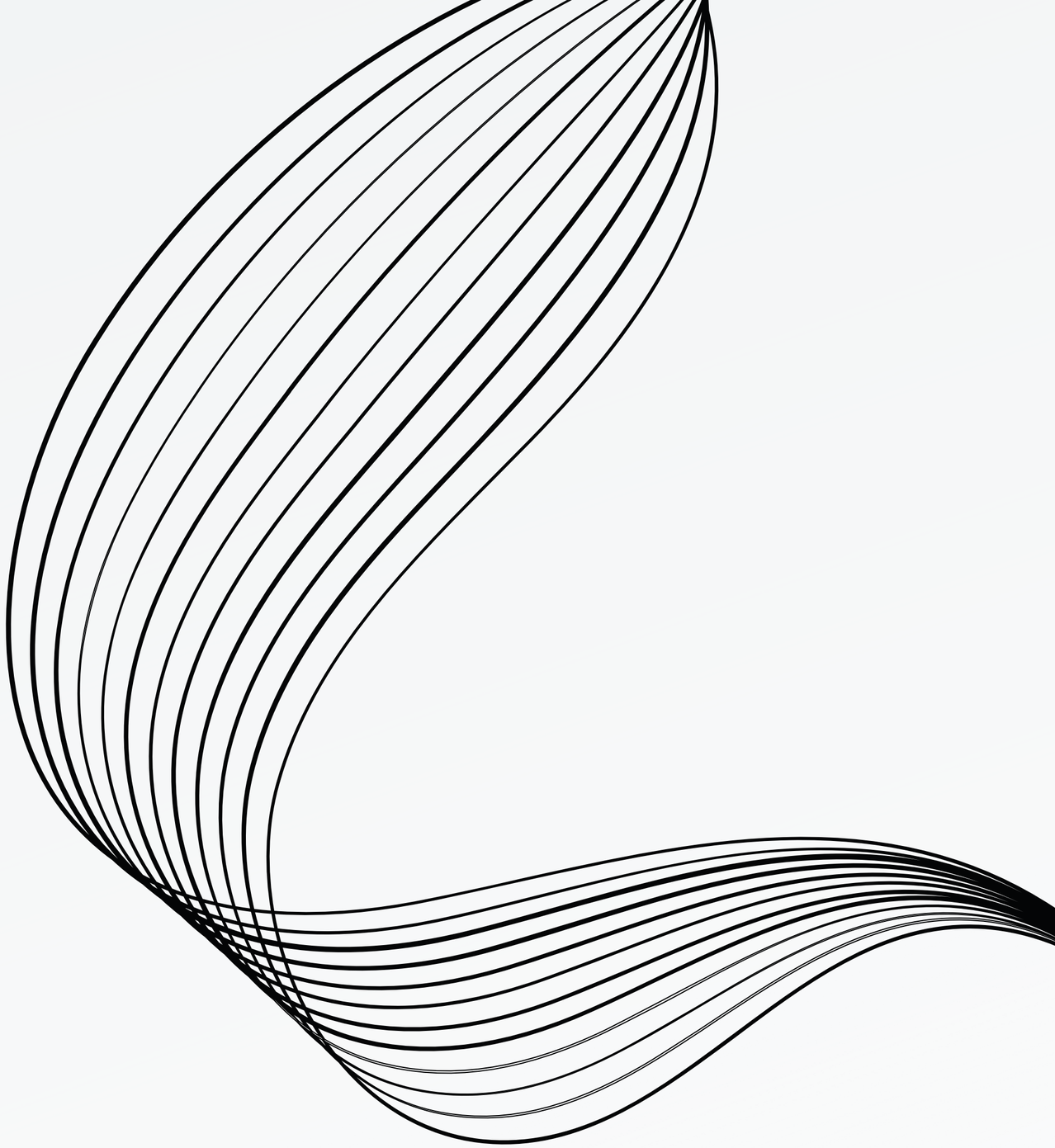
비즈니스 모델

"해파리와의 전쟁" 우리나라 동해와 서해에 급증하고 있는 독성 해파리 떼

꿈의 '치매 치료제'...해파리가 열쇠될까?

THANK'S FOR WATCHING

Q&A



JELLYFISH