



해파리 Classification!

# 아이펠 DLthon 코어 : JellyBob

---



# [DLthon] Jellyfish Image Classification

난이도: ★★★★☆

장르: 데이터, EDA, CV, 분류

## 목차

- 팀 구성 및 역할
- 프로젝트 수행 절차 및 방법
  - 팀 목표, 일정, Task 관리
- 프로젝트 수행 결과
  1. 데이터 준비하기
    - 데이터 살펴보기 : 특징, 개수 등
    - 데이터 전처리
    - 데이터 시각화
  2. 모델 구현 및 훈련
    - W&B 활용
  3. 모델 평가 및 베이스모델 선정
    - 결과 확인(시각화) 및 분석
  4. 성능 향상 시도
  5. 제품화 시 응용분야 및 모델 분석
- 회고





# [DLthon] Jellyfish Image Classification

## 개요

- 딥러닝 모델을 이용하여 해파리 이미지를 받아 class를 분류한다.
- 🔑 모델 성능을 높이는 방법을 습득하고 다양한 모델을 사용해보는 경험을 쌓는다.

## 팀 구성

- 팀명 : **JellyBob** (Jellyfish는 우리의 밥이다!!)
- 이슬, 김양희, 이승환, 전민규

## 역할 분담

- 데이터 준비~모델 성능 향상 : 개별적으로 모두 실행 후 Best case 조합하여 1개의 모델 완성
- 깃허브 노트북 자료 : 이승환, 전민규
- 발표 자료 (PPT) : 김양희, 이슬



# [DLthon] Jellyfish Image Classification

## 팀 목표

데이터 준비, 모델 구현, 모델 성능 향상의 전반적인 과정  
다뤄보기

모델 성능 향상 및 결과 도출을 위해 다양한 시도해보기

팀원 개별 역량 성장  
(2개월 간의 학습 내용 종합, 복습)

협업 및 커뮤니케이션 능력

새로운 협업 툴(W&B)을 빠르게 익히고 활용하는 능력

## 일정 관리

### 2024.01.10 (수)

- 10:30~11am : 프로젝트 일정, 주요 Task, 역할 논의
- 11am~3pm : 데이터 파악 및 EDA (노드 2-1, 2-2)
- 3~6pm : 각자 모델 구현 및 학습해보기

### 2024.01.11 (목)

- 오전 : 개별 모델을 1개의 모델로 조합
- 오후 : 모델 성능 개선, 발표 자료 준비 및 제출



# [DLthon] Jellyfish Image Classification

## Task 관리 툴

### Notion

- 전체 Tasks 관리
- 정보 및 아이디어 공유

The screenshot shows a Notion project titled "[DLthon] JellyBob". The "Project INFO" section contains a "Goal" (Jellyfish 모델을 이용하여 해파리 이미지를 받아 class를 분류) and a "Rubric" (평가 기준). The "TASKS" section is a table with columns Date, A细腻な内容, and task, listing tasks like "데이터 파악", "데이터 내용 공유", "EDA 방법 고민", etc. The "To-do" section shows a list of tasks categorized by status: "Completed" (3), "In progress" (3), and "Done" (3), including items like "EDA 방법 참고자료", "Google Drive/Colab", and "Data 파악".

### Google Drive, Deep Note

- 코드 공유 및 업데이트

The screenshot shows a Google Drive interface with a folder named "DLthon" containing various Jupyter notebooks and files. To the right, a Deep Note notebook titled "jellyfish" is open, showing Python code for visualizing a dataset. The code includes imports, dataset loading, and plotting. Below the code, several images of different jellyfish species are displayed, each labeled with its name and dimensions (e.g., label 0 (224, 224, 3)).



# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

#### - 디렉토리 구조

AIFFEL

Files    Running    Clusters

Select items to perform actions on them.

Upload    New   

<input type="checkbox"/> 0	<input type="button" value="▼"/>	Name	Last Modified	File size
/ aiffel / jellyfish				
<input type="checkbox"/> .. seconds ago				
<input type="checkbox"/> barrel_jellyfish a day ago				
<input type="checkbox"/> blue_jellyfish a day ago				
<input type="checkbox"/> compass_jellyfish a day ago				
<input type="checkbox"/> lions_mane_jellyfish a day ago				
<input type="checkbox"/> mauve_stinger_jellyfish a day ago				
<input type="checkbox"/> Moon_jellyfish a day ago				
<input type="checkbox"/> Train_Test_Valid a day ago				
<input type="checkbox"/> jellyfish-types.zip 3 months ago 26.8 MB				



# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- 각 이미지 수 : 총 900개
  - 각 150개 \* 6종
  - 레이블 균형
- 데이터셋 Tran\_valid\_test 폴더
  - train\_data : 900장 (각 해파리 폴더별 150장)
  - val\_data : 39장
  - test\_data : 40장

결론

“데이터셋 자체가 부족함!”

시도

- 데이터 추가
- 데이터 증강 시도
- **train, validation dataset 재분배**

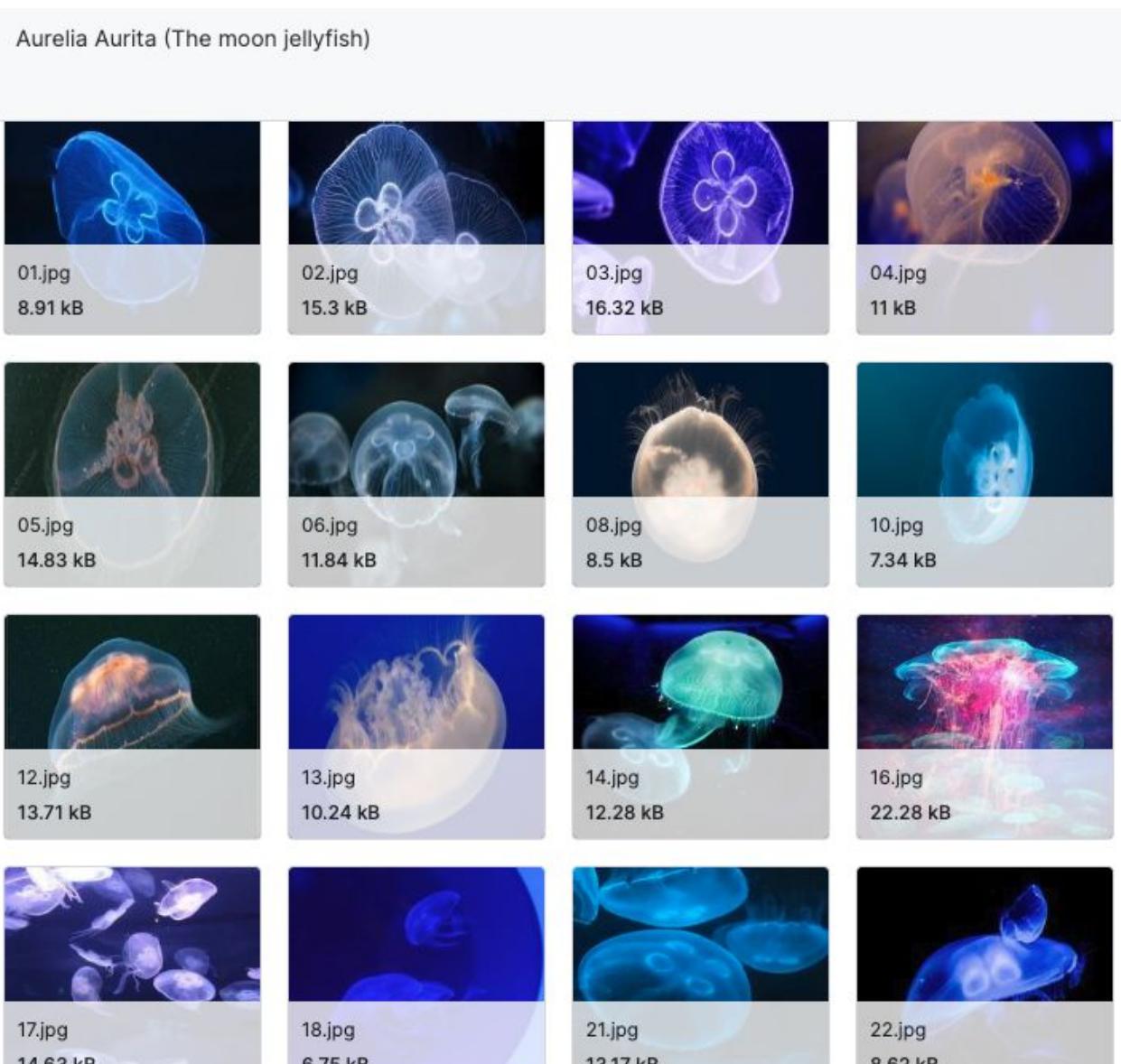


# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- 생김새
  - Moon jellyfish



4개의 꽃잎같은 문양이  
둥근 부분에 있는 것이 특징



# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- 생김새
  - Barrel jellyfish

Rhizostoma pulmo (Barrel Jellyfish)



반 투명하고 머리 하단에 파란색 고리로 되어있다. 촉수가 대체로 두껍다.



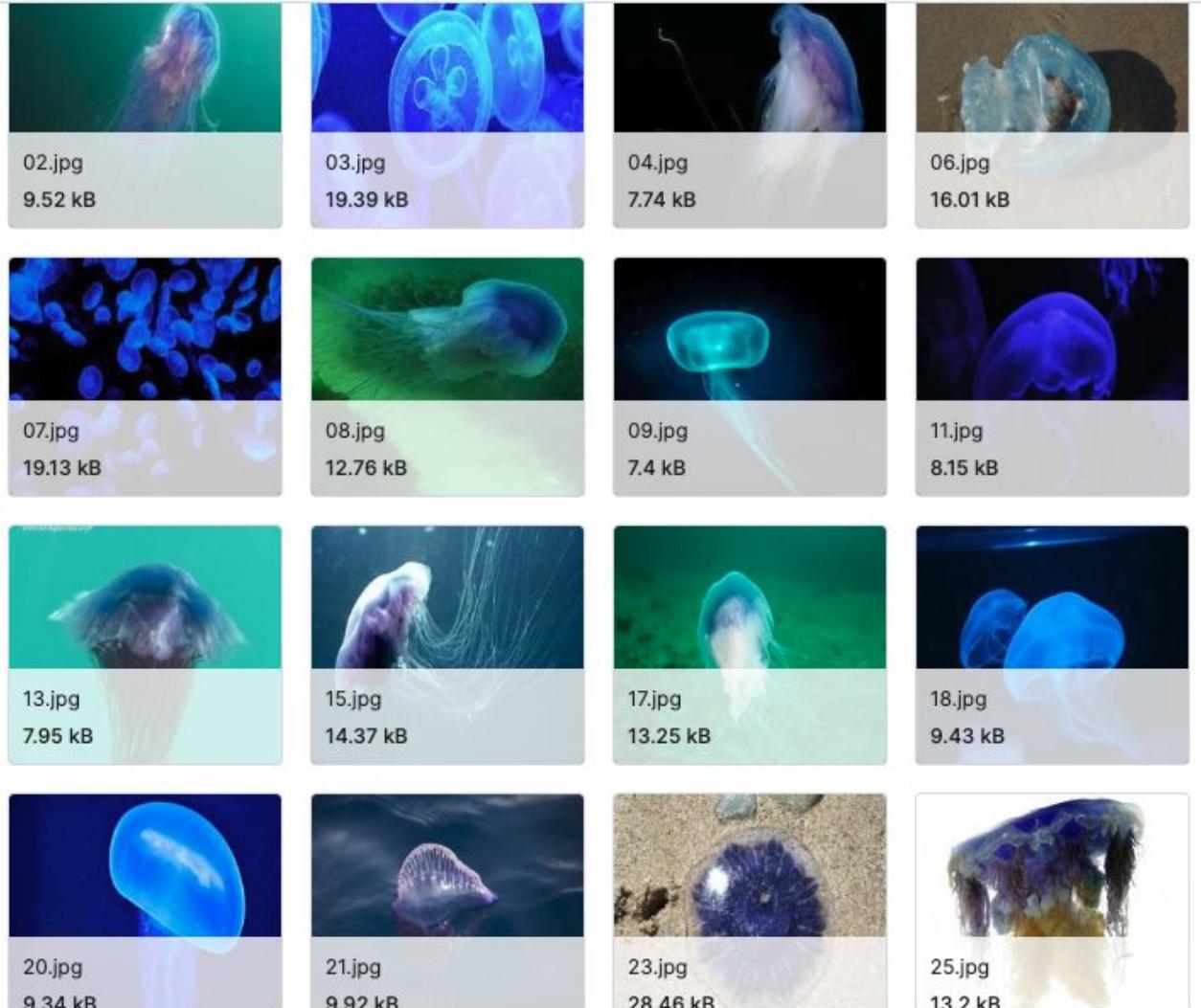
# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- 생김새
  - Blue jellyfish

Cyanea lamarckii (Blue Jellyfish)



머리가 투명하며 대체적으로  
파란색을 띠는 거 같다. 촉수도  
여러 개가 있다.



# [DLthon] Jellyfish Image Classification

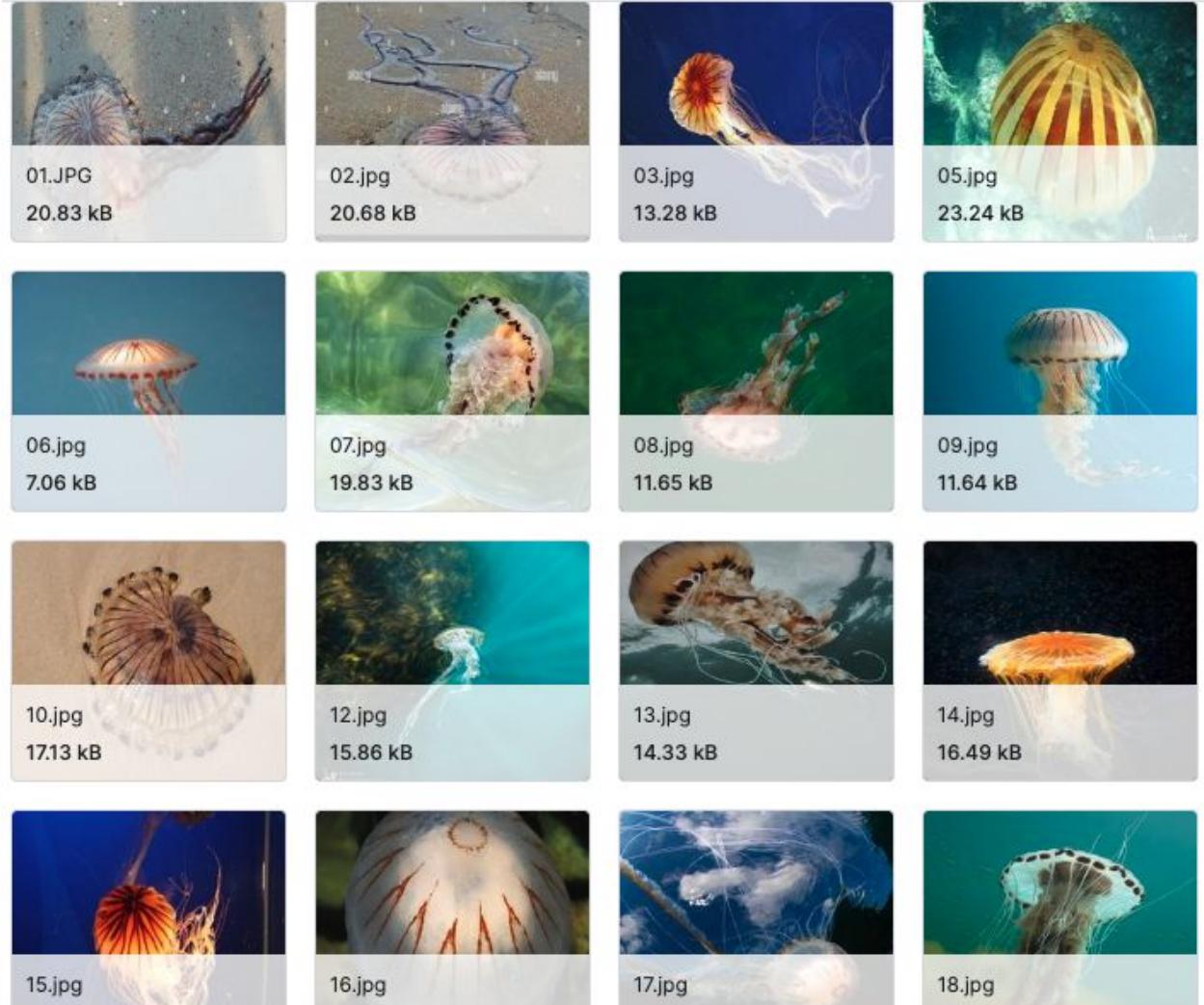
## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- 생김새
  - Compass jellyfish

머리가 투명하고 뾰족한 무늬를 가지고 있으며 머리 하단부에 붉은 점들이 고르게 분포되어 있다.  
족수가 대체로 길며 옆으로는 가느다란 족수들이 여러 개가 있다

Chrysaora hysoscella (compass jellyfish)



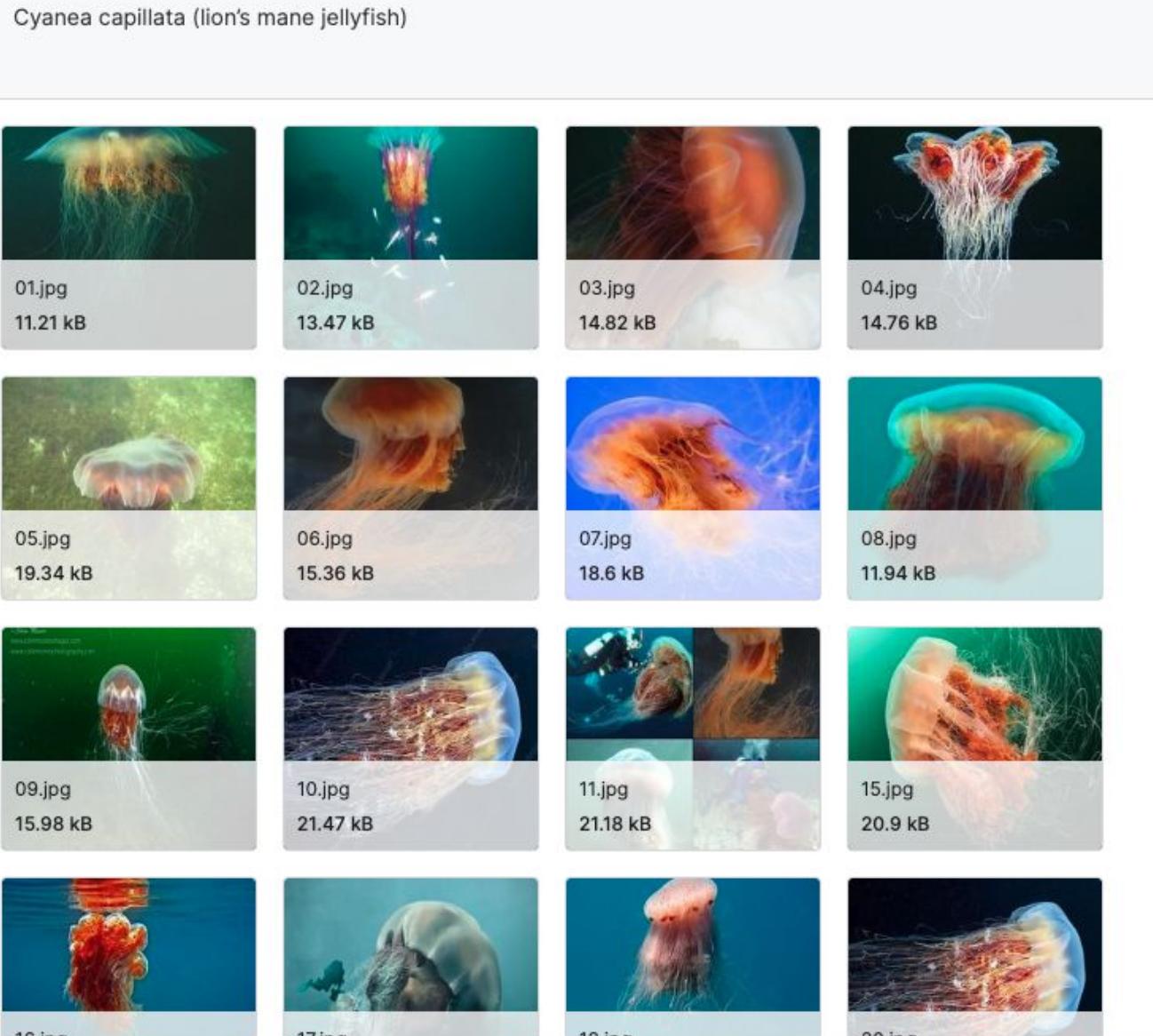


# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- 생김새
  - Lion's mane jellyfish



머리가 투명하고 색은 붉은 색 계열을 띠고 있다. 촉수가 사자 갈기 마냥 많은 것이 특징이다.



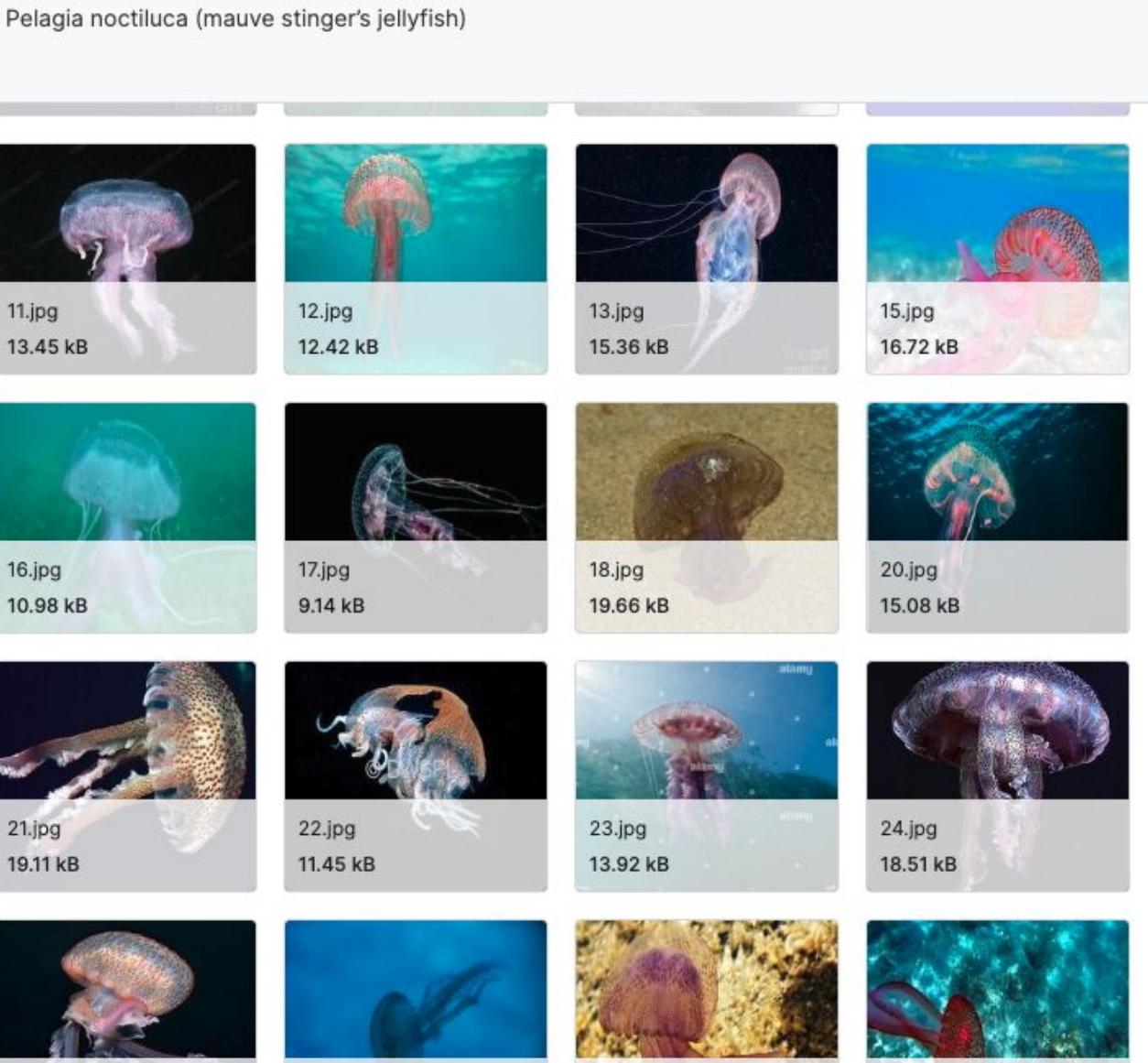
# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- 생김새
  - Mauve stinger's jellyfish

머리가 투명하며 점박이 무늬를 가지고 있다. 메인 촉수 (두께감 있음) 옆에 서브 촉수(얇고 길다)들이 달려있다.





# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- 과학적 탐구

	먹이	크기(유체직경기준)	유체(bell)색상	독성여부	인간에게 치명적 유해여부
Moon jellyfish:	플랑크톤, 연체동물	최대40cm	투명	0	x
Barrel jellyfish:	플랑크톤, 작은 물고기	최대90cm	푸른빛을 띠는 회색에서 흰색	0	x
Blue jellyfish:	플랑크톤, 작은 물고기	최대30cm	투명하거나 약간 푸른빛, 짙은 보라빛	0	x
Compass jellyfish:	플랑크톤, 작은 물고기	최대60cm	유체는 푸르지만 주변에 갈색띠가 형성	0	x
Lion's mane jellyfish	플랑크톤, 작은 물고기	최대2.5m	연한 갈색, 진한 적갈색	0	0
Mauve stinger's jellyfish:	작은 해파리, 바다멍게	보통10cm	자주색에서 분홍색	0	0



# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

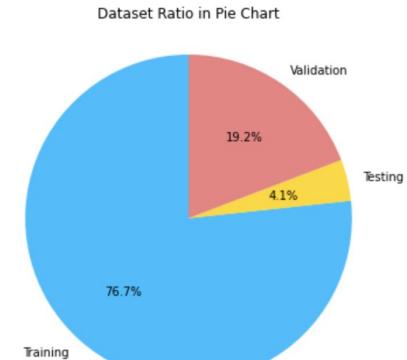
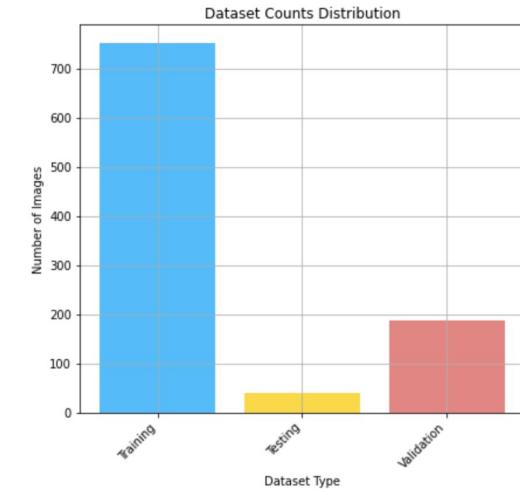
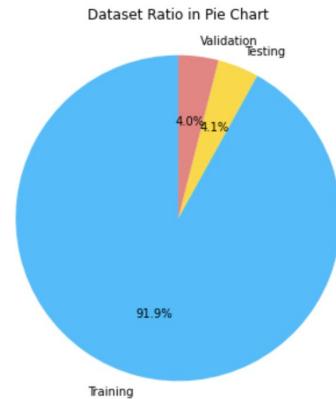
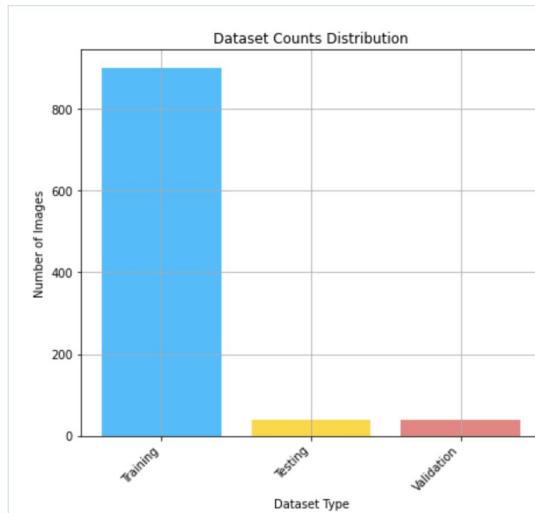
### 1) 데이터 살펴보기

train, valid, test 비율 등 시각화해보니

**validation data 비중이 4%로 매우 적음**

👉 **train, validation data 합친 뒤**

👉 **train : validation = 8:2 로 맞춤**





# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- 미니배치로 작업하기 위한 **tf.data 인스턴스**를 만들기
  - GPU의 성능을 더 효과적으로 활용하여 모델 학습, 속도 향상을 위해
  - 다양한 형태로 변형하여 활용하기 위해
- 데이터셋 형태로 변형해줍니다.

```
1 # tf.data instance 생성
2 train_list_ds = tf.data.Dataset.from_tensor_slices(train_filenames)
3 val_list_ds = tf.data.Dataset.from_tensor_slices(val_filenames)
```

```
1 # Train 데이터셋, validation 데이터셋 개수 확인
2 TRAIN_IMG_COUNT = tf.data.experimental.cardinality(train_list_ds).numpy()
3 print(f"Training images count: {TRAIN_IMG_COUNT}")
4
5 VAL_IMG_COUNT = tf.data.experimental.cardinality(val_list_ds).numpy()
6 print(f"Validating images count: {VAL_IMG_COUNT}")
```

```
Training images count: 751
Validating images count: 188
```



# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 1) 데이터 살펴보기

- **Data Labeling**

Class	barrel jellyfish	blue jellyfish	compass jellyfish	lions mane jellyfish	mauve stinger jellyfish	Moon jellyfish
Label	0	1	2	3	4	5

```
1 # 데이터 레이블 생성
2 def get_label(file_path):
3     parts = tf.strings.split(file_path, os.path.sep)
4
5     if parts[-2] == "barrel_jellyfish":
6         return 0 # barrel_jellyfish이면 라벨 0
7
8     elif parts[-2] == "blue_jellyfish":
9         return 1 # blue_jellyfish이면 라벨 1
10
11    elif parts[-2] == "compass_jellyfish":
12        return 2 # compass_jellyfish이면 라벨 2
13
14    elif parts[-2] == "lions_mane_jellyfish":
15        return 3 # lions_mane_jellyfish이면 라벨 3
16
17    elif parts[-2] == "mauve_stinger_jellyfish":
18        return 4 # mauve_stinger_jellyfish이면 라벨 4
19
20    else :
21        return 5 # Moon_jellyfish이면 라벨 5
```

>> Data Labeling 하는 함수 구현



# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 2) 데이터 전처리

Resize :

224\*224\*3으로 이미지 사이즈 통일

```
1 # 이미지를 알맞은 형식으로 바꿉니다.  
2 def decode_img(img):  
3     img = tf.image.decode_jpeg(img, channels=3) # 이미지를 uint8 tensor로 수정  
4     img = tf.image.convert_image_dtype(img, tf.float32) # float32 타입으로 수정  
5     img = tf.image.resize(img, IMAGE_SIZE) # 이미지 사이즈를 IMAGE_SIZE로 수정  
6     return img  
7  
8 # 이미지 파일의 경로를 입력하면 이미지와 라벨을 읽어옵니다.  
9 def process_path(file_path):  
10    label = get_label(file_path) # 라벨 검출  
11    img = tf.io.read_file(file_path) # 이미지 읽기  
12    img = decode_img(img) # 이미지를 알맞은 형식으로 수정  
13    return img, label
```



# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 2) 데이터 전처리

train, validation datasets 생성

이미지 리사이즈 결과, 레이블 확인

```
1 # train 데이터셋과 validation 데이터셋을 생성
2 train_ds = train_list_ds.map(process_path, num_parallel_calls=AUTOTUNE)
3 val_ds = val_list_ds.map(process_path, num_parallel_calls=AUTOTUNE)
```

```
1 # 이미지 리사이즈 결과, 레이블 확인
2 for image, label in train_ds.take(1): # 하나의 데이터만 가져온다
3     print("Image shape: ", image.numpy().shape)
4     print("Image type: ", image.dtype)
5     print("Label: ", label.numpy())
6
```

```
Image shape: (224, 224, 3)
Image type: <dtype: 'float32'>
Label: 0
```

```
1 # 데이터셋 샘플 시각화
2
3 def show_dataset_samples(dataset, num_samples=15):
4     plt.figure(figsize=(15, 15))
5
6     for idx, (img, label) in enumerate(dataset.take(num_samples)):
7         plt.subplot(3, 5, idx + 1)
8         # img = (img * 255) # 파일 정규화 했을 경우 두 코드 사용 : 원래 픽셀값으로 돌려줌
9         # img = tf.image.convert_image_dtype(img, tf.float32)
10        plt.imshow(img)
11        plt.title(f'label {label}\n{img.shape}')
12        plt.axis('off')
13
14 # Usage example with your training dataset
15 show_dataset_samples(train_ds)
16 plt.show() # Display the figure
17
```





# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 2) 데이터 전처리 - 데이터 증강

데이터셋이 매우 작아 좌우 반전, 채도 조정, 90도 단위의 회전을 적용한 데이터 증강을

실행

```
1 # augmentation function
2 def augment(img, label):
3     # Randomly flip left or right
4     img = tf.image.random_flip_left_right(img)
5
6     img = tf.image.random_saturation(img, lower=0.5, upper=1.5)
7
8     img = tf.image.rot90(img, tf.random.uniform(shape=[], minval=0, maxval=4, dtype=tf.int32))
9
10    return img, label
```

```
1 def prepare_for_training(ds, shuffle_buffer_size=256):
2     ds = ds.map(augment, num_parallel_calls=AUTOTUNE)
3
4     ds = ds.shuffle(buffer_size=shuffle_buffer_size)
5
6     ds = ds.repeat()
7     ds = ds.batch(BATCH_SIZE)
8     ds = ds.prefetch(buffer_size=AUTOTUNE)
9
10    return ds
```



# [DLthon] Jellyfish Image Classification

## 1. 데이터 준비하기

### 2) 데이터 전처리 - 배치 데이터셋 만들기

데이터 전처리 완료 후 train, validation, test batch data 만들기

```
1 # train, val 배치데이터 만들기
2 train_dat = prepare_for_training(train_ds)
3 val_dat = prepare_for_training(val_ds)
```

```
1 # test dataset 만들기
2 test_list_ds = tf.data.Dataset.list_files(TEST_PATH)
3 TEST_IMAGE_COUNT = tf.data.experimental.cardinality(test_list_ds).numpy()
4 test_ds = test_list_ds.map(process_path, num_parallel_calls=AUTOTUNE)
5 test_dat = test_ds.batch(BATCH_SIZE)
6
7 print(TEST_IMAGE_COUNT)
```



# [DLthon] Jellyfish Image Classification

## 2. 모델 구현 및 훈련

- wandb 활용

```
1 # wandb 모델 훈련 함수
2
3 def train(model, config, train_batches, val_batches, test_batches, CLASS_NAMES):
4     wandb.init(config=config)
5     config = wandb.config
6
7     if config.optimizer == 'rmsprop':
8         optimizer = keras.optimizers.RMSprop(learning_rate=config.learning_rate)
9
10    elif config.optimizer == 'adam':
11        optimizer = keras.optimizers.Adam(learning_rate=config.learning_rate)
12
13    elif config.optimizer == 'sgd':
14        optimizer = keras.optimizers.SGD(learning_rate=config.learning_rate)
15
16    else:
17        raise ValueError(f"Unsupported optimizer: {config.optimizer}")
18
19    # using gpu
20    with tf.device('/GPU:0'):
21        model.compile(optimizer=optimizer, loss=config.loss, metrics=config.metrics)
```

```
1 # index로 준비
2 CLASS_NAMES = [0, 1, 2, 3, 4, 5]
3
4 # run the sweep
5 wandb.agent(sweep_id,
6             function=lambda: train(cnn_model, cnn_config, train_batch, val_batch, test_batch, CLASS_NAMES),
7             count=5)
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

'BATCH\_SIZE,  
'BATCH\_SIZE,  
'BATCH\_SIZE,  
'CLASS\_NAMES,  
'predictions=10,  
'input\_type="images")])  
  
test\_loss, test\_accuracy = model.evaluate(test\_batches, verbose=2)  
  
wandb.log({"Test Accuracy Rate": round(test\_accuracy \* 100, 2),  
 "Test Error Rate": round((1 - test\_accuracy) \* 100, 2)})



# [DLthon] Jellyfish Image Classification

## 2. 모델 구현 및 훈련

- 1단계 : 4명 모두 개별적으로 여러 모델을 사용해보며 훈련 및 테스트

```
# 모델 구현
진행

def model(pool_size=7, num_classes=6):

    inputs = keras.layers.Input(shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3))

    x = keras.layers.SeparableConv2D(64, kernel_size=(3,3), strides=1, padding='same')(inputs)
    x = tf.keras.layers.experimental.preprocessing.Normalization()(x)
    x = keras.layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(x)

    x = keras.layers.Conv2D(64, kernel_size=(3,3), strides=1, padding='same')(x)
    x = tf.keras.layers.experimental.preprocessing.Normalization()(x)
    x = keras.layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(x)

    x = keras.layers.Conv2D(128, kernel_size=(3,3), strides=1, padding='same')(x)
    x = tf.keras.layers.experimental.preprocessing.Normalization()(x)
    x = keras.layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(x)

    x = keras.layers.Conv2D(256, kernel_size=(3,3), strides=1, padding='same')(x)
    x = tf.keras.layers.experimental.preprocessing.Normalization()(x)
    x = keras.layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(x)

    x = keras.layers.Conv2D(128, kernel_size=(3,3), strides=1, padding='same')(x)
    x = tf.keras.layers.experimental.preprocessing.Normalization()(x)
    x = keras.layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(x)

    x = keras.layers.Conv2D(64, kernel_size=(3,3), strides=1, padding='same')(x)
    x = tf.keras.layers.experimental.preprocessing.Normalization()(x)
    x = keras.layers.GlobalAveragePooling2D()(x)

    outputs = keras.layers.Dense(num_classes, activation='softmax')

    return keras.models.Model(inputs=inputs, outputs=outputs)

# 모델 작업 - 함수화
from tensorflow.keras.regularizers import l2
def build_model():
    model = keras.models.Sequential()
    model.add(keras.layers.Conv2D(16, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal', input_shape=(256, 256, 3), kernel_regularizer=l2(0.01)))
    model.add(keras.layers.MaxPooling2D(2, 2))
    model.add(keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.MaxPooling2D(2, 2))
    model.add(keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.MaxPooling2D(2, 2))
    model.add(keras.layers.Conv2D(64, (1, 1), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(256, activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.Dropout(0.4))
    model.add(keras.layers.Dense(6, activation='softmax'))

    return model

김양희
```



# [DLthon] Jellyfish Image Classification

## 2. 모델 구현 및 훈련

- 1단계 : 4명 모두 개별적으로 여러 모델을 사용해보며 훈련 및 테스트  
진행

```
# Shallow ResNet 구현
...
num_classes : 분류해야하는 클래스 수
...

class shallow_ResNet(Model):
    def __init__(self, num_classes, **kwargs):
        super().__init__(**kwargs)

        # conv1레이어는 직접 만들고
        self.conv_1 = SeparableConv2D(16, (7, 7), strides=2,
                                    padding="same", kernel_initializer="he_normal")
        self.init_bn = BatchNormalization()
        self.pool_2 = MaxPool2D(pool_size=(2, 2), strides=2, padding="same")
        self.res_1_1 = ResidualBlock(16)
        self.res_1_2 = ResidualBlock(16)
        self.res_2_1 = ResidualBlock(32, down_sample=True) # 논문을 보면 여기에서 stride를 1, 1x2로 두었지만, tf.keras.layers.MaxPool2D()를 사용하기 때문에 strides=2로 두었습니다.
        self.res_2_2 = ResidualBlock(32)
        #
        self.res_3_1 = ResidualBlock(256, down_sample=True)
        self.res_3_2 = ResidualBlock(256)
        #
        self.res_4_1 = ResidualBlock(512, down_sample=True)
        self.res_4_2 = ResidualBlock(512)
        self.avg_pool = GlobalAveragePooling2D()
        self.flat = Flatten()
        self.fc = Dense(num_classes, activation="softmax")

    def call(self, inputs):
        out = self.conv_1(inputs)
        out = self.init_bn(out)
        out = tf.nn.relu(out)
        out = self.pool_2(out)
```

0|슬

## modified cnn

```
# build model = convolution block + dense block 합치기
def build_model():
    model = tf.keras.Sequential([
        tf.keras.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3)),

        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.MaxPool2D(),

        conv_block(32),
        tf.keras.layers.Dropout(0.5),

        tf.keras.layers.Flatten(),
        dense_block(32, 0.5),

        tf.keras.layers.Dense(6, activation='softmax')
    ])

    return model
```



# [DLthon] Jellyfish Image Classification

## 2. 모델 구현 및 훈련

- 1단계 : 4명 모두 개별적으로 여러 모델을 사용해보며 훈련 및 테스트 진행

```
# 모델 만들어보기
import keras
model=keras.models.Sequential()
model.add(keras.layers.Conv2D(64, (3,3), padding='same', activation='relu', input_shape=(224,224,3)))
model.add(keras.layers.Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(keras.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(keras.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(keras.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(32, activation='relu'))
model.add(keras.layers.Dense(6, activation='softmax'))

print('Model에 추가된 Layer 개수: ', len(model.layers))

model.summary()
```

이승환

VGG



# [DLthon] Jellyfish Image Classification

## 2. 모델 구현 및 훈련

- 1단계 : 4명 모두 개별적으로 여러 모델을 사용해보며 훈련 및 테스트 진행

```
#resnet모델만들기
import tensorflow as tf
from tensorflow.keras.layers import Input, AveragePooling2D, Flatten, Activation
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
def ResNet18(classes, input_shape):
    inputs = Input(shape=input_shape)
    conv1 = Conv2D(64, (7,7), strides=2, padding='same', activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.01))(inputs)
    max_pooling = MaxPooling2D(3, strides=2, padding='same')(conv1)

    conv2_x = ResidualBlock(max_pooling, 64, kernel_size=(3,3), kernel_initializer='he_normal', kernel_regularizer=l2(0.01), downsample=False)
    conv2_x = ResidualBlock(conv2_x, 64, kernel_size=(3,3), kernel_initializer='he_normal', kernel_regularizer=l2(0.01), downsample=False)

    conv3_x = ResidualBlock(conv2_x, 128, kernel_size=(3,3), kernel_initializer='he_normal', kernel_regularizer=l2(0.01), downsample=True)
    conv3_x = ResidualBlock(conv3_x, 128, kernel_size=(3,3), kernel_initializer='he_normal', kernel_regularizer=l2(0.01), downsample=False)

    conv4_x = ResidualBlock(conv3_x, 256, kernel_size=(3,3), kernel_initializer='he_normal', kernel_regularizer=l2(0.01), downsample=True)
    conv4_x = ResidualBlock(conv4_x, 256, kernel_size=(3,3), kernel_initializer='he_normal', kernel_regularizer=l2(0.01), downsample=False)

    conv5_x = ResidualBlock(conv4_x, 512, kernel_size=(3,3), kernel_initializer='he_normal', kernel_regularizer=l2(0.01), downsample=True)
    conv5_x = ResidualBlock(conv5_x, 512, kernel_size=(3,3), kernel_initializer='he_normal', kernel_regularizer=l2(0.01), downsample=False)

    x = GlobalAveragePooling2D()(conv5_x)
    x = Flatten()(x)
    x = Dense(64, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.01))(x)
    x = Dense(32, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.01))(x)
    out_layer = Dense(units=classes, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=out_layer)
    return model
```

전민규



# [DLthon] Jellyfish Image Classification

## 3. 모델 평가 및 베이스모델 선정

4명의 개인 기록들 중  
가장 성능이 좋았던 모델을  
**베이스 모델**로 선정

```
# 모델 작업
import keras

model=keras.models.Sequential()
model.add(keras.layers.Conv2D(16, (2,2), padding='same', activation='relu',
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(32, (2,2), padding='same', activation='relu',
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(64, (2,2), padding='same', activation='relu',
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(16, activation='relu', kernel_initializer='he_n
model.add(keras.layers.Dense(6, activation='softmax'))

model.summary()
```

## CNN Model

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 224, 224, 16)	208
max_pooling2d_3 (MaxPooling2D)	(None, 112, 112, 16)	0
conv2d_5 (Conv2D)	(None, 112, 112, 32)	2080
max_pooling2d_4 (MaxPooling2D)	(None, 56, 56, 32)	0
conv2d_6 (Conv2D)	(None, 56, 56, 64)	8256
max_pooling2d_5 (MaxPooling2D)	(None, 28, 28, 64)	0
flatten_1 (Flatten)	(None, 50176)	0
dense_2 (Dense)	(None, 16)	802832
dense_3 (Dense)	(None, 6)	102

Total params: 813,478

Trainable params: 813,478

Non-trainable params: 0



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도

### 1. 하이퍼파라미터 튜닝

- 학습률
- 배치사이즈
- optimizer
- kernel size

### 2. 모델 수정

- dense layer node
- dropout
- convolution layer filter 수

# [DLthon] Jellyfish Image Classification

base code

## 4. 성능향상 시도 - 1

- 배치사이즈 :  
**16 ~ 32**
- 커널사이즈 :  
**2x2, 3x3, 5x5**
- model :  
**dense node**  
**dropout**  
**conv layer filter**

#	학습률	배치사이즈	test accuracy(%)	optimizer	kernel size	epochs	모델 구조 변경
base	0.00003	24	65	rmsprop	2, 2	37	
1	0.0001 (고정)	24	67.5	rmsprop	3, 3		
2	0.0001 (고정)	30	77.5	rmsprop	3, 3		
3	0.00009	24	80	rmsprop	3, 3	50	dense 16 → 255 dropout 30%
4	0.00005	24	75	rmsprop	2, 2	21	
5	0.00008	24	77.5	rmsprop	5, 5	29	
6	0.00009	32	77.5	rmsprop	2, 2	40	
7	0.00009	32	75	rmsprop	2, 2	50	filter 16, 32, 32, 32 dence 255→128
8	0.0008	16	70	rmsprop	2, 2	46	
9	0.00004	24	67.5	rmsprop	2, 2	43	dense255→ 128 dropout 20% dense64추가



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 2)

- optimizer :  
**rmsprop**  
**sgd**  
**adam**

- model :  
**dense node**  
**dropout**  
**conv layer filter**

#	학습률	배치사이즈	test accuracy(%)	optimizer	kernel size	epochs	모델 구조 변경
10	0.00002	16	75	rmsprop	2, 2	44	dense128, dropout 20% ,cnn처음 layer16→32
11	0.00007	16	70	rmsprop	2, 2	48	dense 256 dropout 40%
12	0.00078	30	70	rmsprop	3, 3	45	
13	0.0007	16	72.5	rmsprop	2, 2	48	
14	0.00008	24	52.5	sgd	3, 3	24	dence255 dropout30%
15	0.000006	32	67.5	rmsprop	3, 3	34	filter 16, 32, 64, 64
16	0.00005	24	77.5	adam	3, 3	49	dence255 dropout30%
17	0.0002	32	70	rmsprop	3, 3	18	
18	0.00009	24	67.5	rmsprop	3, 3	101	



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - Baseline

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_4 (Conv2D)	(None, 224, 224, 16)	208
<hr/>		
max_pooling2d_3 (MaxPooling2	(None, 112, 112, 16)	0
<hr/>		
conv2d_5 (Conv2D)	(None, 112, 112, 32)	2080
<hr/>		
max_pooling2d_4 (MaxPooling2	(None, 56, 56, 32)	0
<hr/>		
conv2d_6 (Conv2D)	(None, 56, 56, 64)	8256
<hr/>		
max_pooling2d_5 (MaxPooling2	(None, 28, 28, 64)	0
<hr/>		
flatten_1 (Flatten)	(None, 50176)	0
<hr/>		
dense_2 (Dense)	(None, 16)	802832
<hr/>		
dense_3 (Dense)	(None, 6)	102
<hr/>		
Total params: 813,478		
Trainable params: 813,478		
Non-trainable params: 0		

### Run summary:

Test Accuracy Rate: 65.0

Test Error Rate: 35.0

accuracy 0.99059

best\_epoch 19

best\_val\_loss 1.07539

epoch 36

loss 0.05588

val\_accuracy 0.67262

val\_loss 1.25103

### Run history:

Test Accuracy Rate: -

Test Error Rate: -

accuracy

epoch

loss

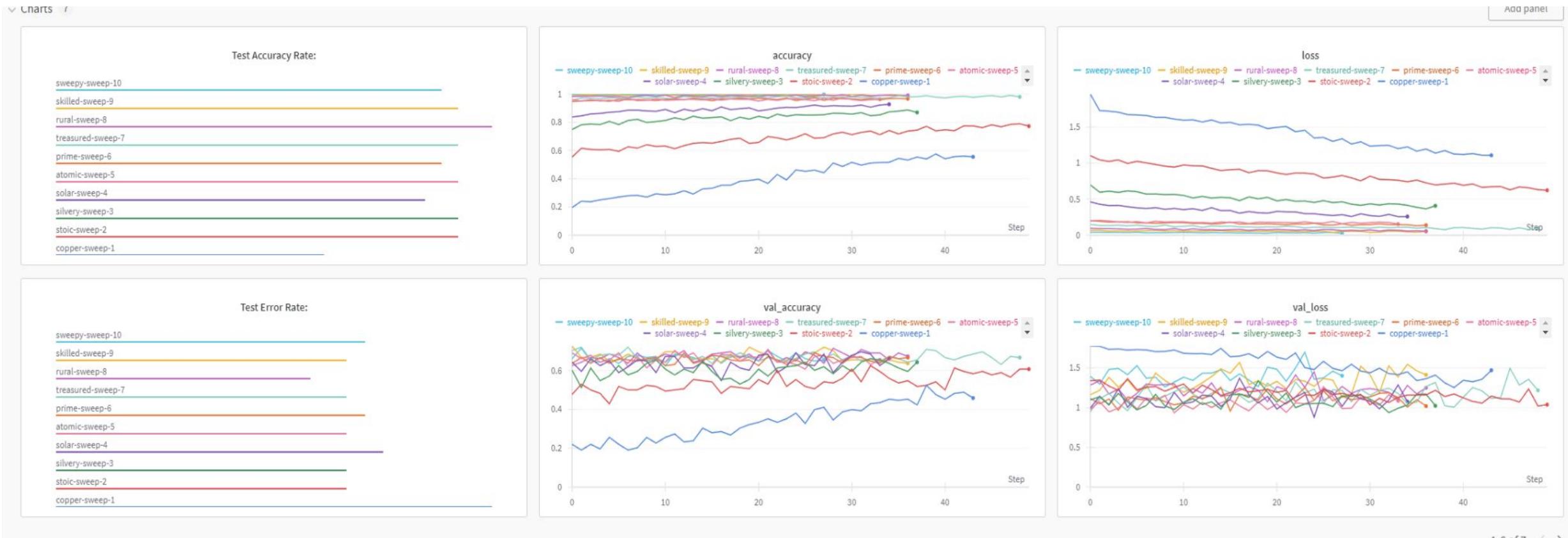
val\_accuracy

val\_loss



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - Baseline





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 3. model : dense 16 → 255 , dropout 30%

Config

Config parameters describe your model's inputs. [Learn more](#)

Key	Value
activation	"relu"
batch_size	24
epoch	51
input	
0	224
1	224
2	3
kernel	
0	3
1	3
learning_rate	0.00009250732138017458
loss	"sparse_categorical_crossentropy"
metrics (1 collapsed)	
optimizer	"rmsprop"

Summary

Summary metrics describe your results. [Learn more](#)

Key	Value
Test Accuracy Rate:	80
Test Error Rate:	20
accuracy	0.9408602118492126
best_epoch	35
best_val_loss	0.8006104230880737
epoch	50
examples	
_type	"images/separated"
captions (10 collapsed)	
count	10
filenames (10 collapsed)	
format	"png"
height	224
width	224
graph	
_type	"graph-file"
path	"media/graph/graph_summary_5e24610b4799336087bc.graph.json"

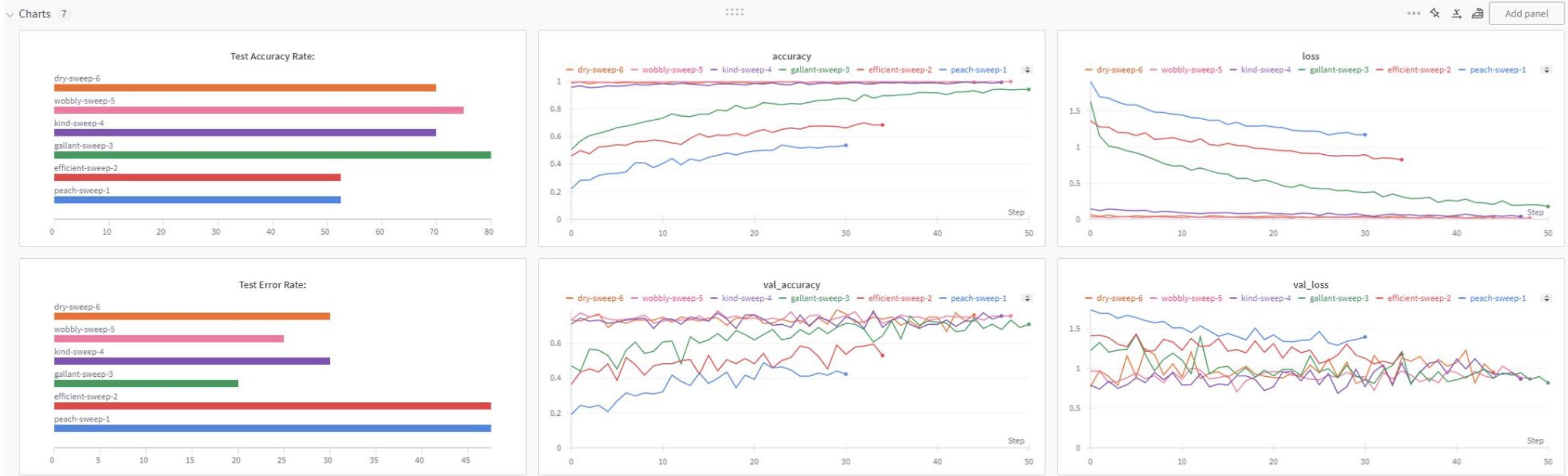
**base와 비교.**  
커널 사이즈 2에서 3으로 변경하고,  
모델 구조 변경함.  
**(dense node 16에서 255로,  
드롭아웃 30% 적용).**  
**test\_accuracy 65 → 80으로 상승!**





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 3. model : dense 16 → 255 , dropout 30%





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 4. kernel size 3x3 → 2x2 : 사이즈 축소

Config

Config parameters describe your model's inputs. [Learn more](#)

Key	Value
activation	"relu"
batch_size	24
epoch	22
input	
0	224
1	224
2	3
kernel	
0	2
1	2
learning_rate	0.00005745773904455136
loss	"sparse_categorical_crossentropy"
metrics (1 collapsed)	
optimizer	"rmsprop"

Summary

Summary metrics describe your results. [Learn more](#)

Key	Value
Test Accuracy Rate:	75
Test Error Rate:	25
accuracy	0.86021506786744
best_epoch	20
best_val_loss	0.8497185707092285
epoch	21
examples (7 collapsed)	
graph (4 collapsed)	
loss	0.395361840724945
val_accuracy	0.6666666865348816
val_loss	0.861005961894989

실험3과 비교.  
커널만 작게 바꿨더니  
**test\_accuracy** 떨어짐  
**test\_accuracy** 80 → 75로 소폭  
하락.



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 4. kernel size 3x3 → 2x2 : 사이즈 축소





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 5. kernel size 3x3 → 5x5 : 사이즈 확대

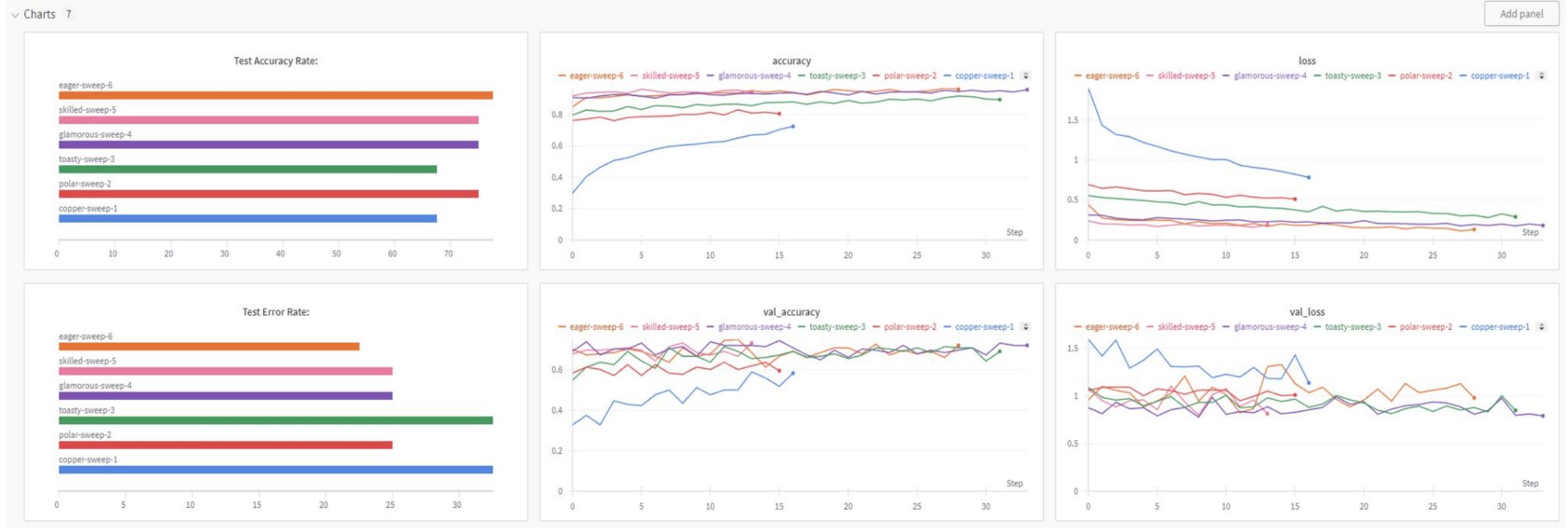
Config		Summary																																																											
Config parameters describe your model's inputs. <a href="#">Learn more</a>		Summary metrics describe your results. <a href="#">Learn more</a>																																																											
<table border="1"><thead><tr><th colspan="2">Search keys</th></tr><tr><th>Key</th><th>Value</th></tr></thead><tbody><tr><td>activation</td><td>"relu"</td></tr><tr><td>batch_size</td><td>24</td></tr><tr><td>epoch</td><td>29</td></tr><tr><td>input</td><td></td></tr><tr><td>0</td><td>224</td></tr><tr><td>1</td><td>224</td></tr><tr><td>2</td><td>3</td></tr><tr><td>kernel</td><td></td></tr><tr><td>0</td><td>5</td></tr><tr><td>1</td><td>5</td></tr><tr><td>learning_rate</td><td>0.00008453112620321092</td></tr><tr><td>loss</td><td>"sparse_categorical_crossentropy"</td></tr><tr><td>metrics</td><td>(1 collapsed)</td></tr><tr><td>optimizer</td><td>"rmsprop"</td></tr></tbody></table>		Search keys		Key	Value	activation	"relu"	batch_size	24	epoch	29	input		0	224	1	224	2	3	kernel		0	5	1	5	learning_rate	0.00008453112620321092	loss	"sparse_categorical_crossentropy"	metrics	(1 collapsed)	optimizer	"rmsprop"	<table border="1"><thead><tr><th colspan="2">Search keys</th></tr><tr><th>Key</th><th>Value</th></tr></thead><tbody><tr><td>Test Accuracy Rate:</td><td>77.5</td></tr><tr><td>Test Error Rate:</td><td>22.5</td></tr><tr><td>accuracy</td><td>0.961021482944488</td></tr><tr><td>best_epoch</td><td>11</td></tr><tr><td>best_val_loss</td><td>0.8220961689949036</td></tr><tr><td>epoch</td><td>28</td></tr><tr><td>examples</td><td>(7 collapsed)</td></tr><tr><td>graph</td><td>(4 collapsed)</td></tr><tr><td>loss</td><td>0.13144072890281677</td></tr><tr><td>val_accuracy</td><td>0.7202380895614624</td></tr><tr><td>val_loss</td><td>0.9765037298202516</td></tr></tbody></table>		Search keys		Key	Value	Test Accuracy Rate:	77.5	Test Error Rate:	22.5	accuracy	0.961021482944488	best_epoch	11	best_val_loss	0.8220961689949036	epoch	28	examples	(7 collapsed)	graph	(4 collapsed)	loss	0.13144072890281677	val_accuracy	0.7202380895614624	val_loss	0.9765037298202516
Search keys																																																													
Key	Value																																																												
activation	"relu"																																																												
batch_size	24																																																												
epoch	29																																																												
input																																																													
0	224																																																												
1	224																																																												
2	3																																																												
kernel																																																													
0	5																																																												
1	5																																																												
learning_rate	0.00008453112620321092																																																												
loss	"sparse_categorical_crossentropy"																																																												
metrics	(1 collapsed)																																																												
optimizer	"rmsprop"																																																												
Search keys																																																													
Key	Value																																																												
Test Accuracy Rate:	77.5																																																												
Test Error Rate:	22.5																																																												
accuracy	0.961021482944488																																																												
best_epoch	11																																																												
best_val_loss	0.8220961689949036																																																												
epoch	28																																																												
examples	(7 collapsed)																																																												
graph	(4 collapsed)																																																												
loss	0.13144072890281677																																																												
val_accuracy	0.7202380895614624																																																												
val_loss	0.9765037298202516																																																												

실험4와 비교.  
커널만 크게 바꿨더니  
**test\_accuracy 75 → 77.5로**  
실험4 대비 소폭 상승, 실험3보다  
낮음.



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 5. kernel size 3x3 → 5x5 : 사이즈 확대





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 6. 배치사이즈 24 → 32로 수정

Search keys	
Key	Value
activation	"relu"
batch_size	32
epoch	40
input	
0	224
1	224
2	3
kernel	
0	2
1	2
learning_rate	0.0000964353414884436
loss	"sparse_categorical_crossentropy"
metrics (1 collapsed)	
optimizer	"rmsprop"

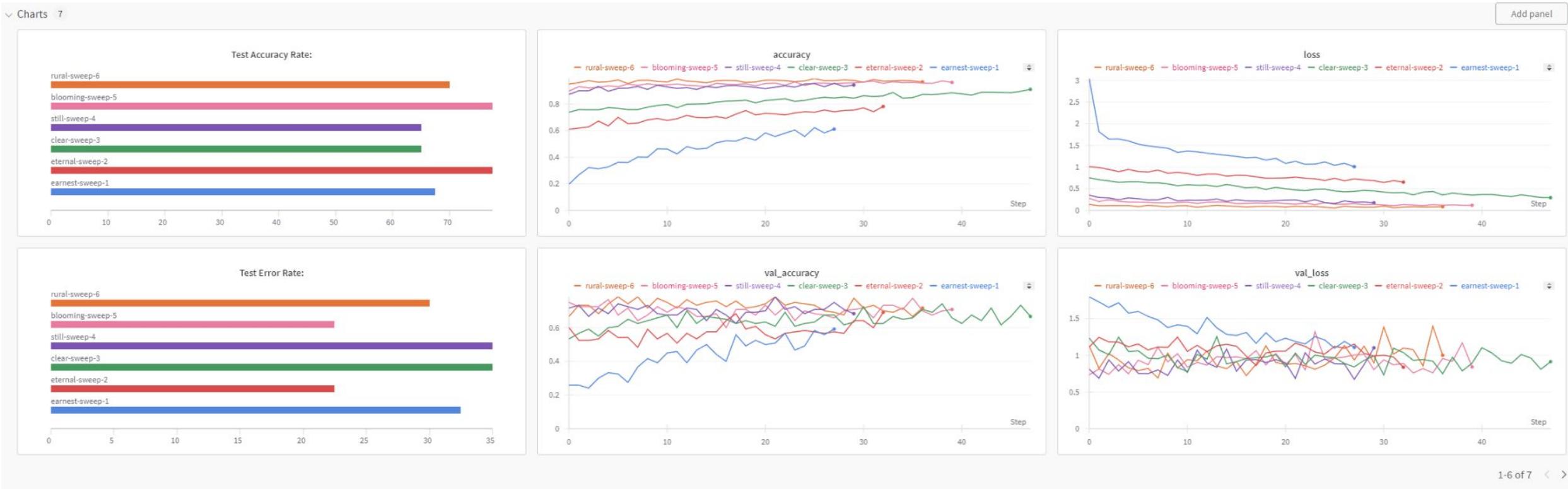
Search keys	
Key	Value
Test Accuracy Rate:	77.5
Test Error Rate:	22.5
accuracy	0.9637681245803832
best_epoch	0
best_val_loss	0.7344850897789001
epoch	39
> examples (7 collapsed)	
> graph (4 collapsed)	
loss	0.119355671107769
val_accuracy	0.7083333134651184
val_loss	0.8414343595504761

실험4와 비교.  
배치사이즈만 늘렸더니  
`test_accuracy` 75 → 77.5로  
실험4 대비 소폭 상승.



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 6. 배치사이즈 24 → 32로 수정





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 7. 실험 6에서 모델 구조 변경 :

**conv filter 16 - 32 - 64 - 64 → 16- 32- 32 - 32 / dense 255 → 128**

Search keys	
Key	Value
activation	"relu"
batch_size	32
epoch	50
input	
0	224
1	224
2	3
kernel	
0	2
1	2
learning_rate	0.00009519943328852636
loss	"sparse_categorical_crossentropy"
> metrics (1 collapsed)	
optimizer	"rmsprop"

Search keys	
Key	Value
Test Accuracy Rate:	75
Test Error Rate:	25
accuracy	0.77173912525177
best_epoch	43
best_val_loss	0.8743914365768433
epoch	49
> examples (7 collapsed)	
> graph (4 collapsed)	
loss	0.6232919096946716
val_accuracy	0.5916666388511658
val_loss	0.9678006768226624

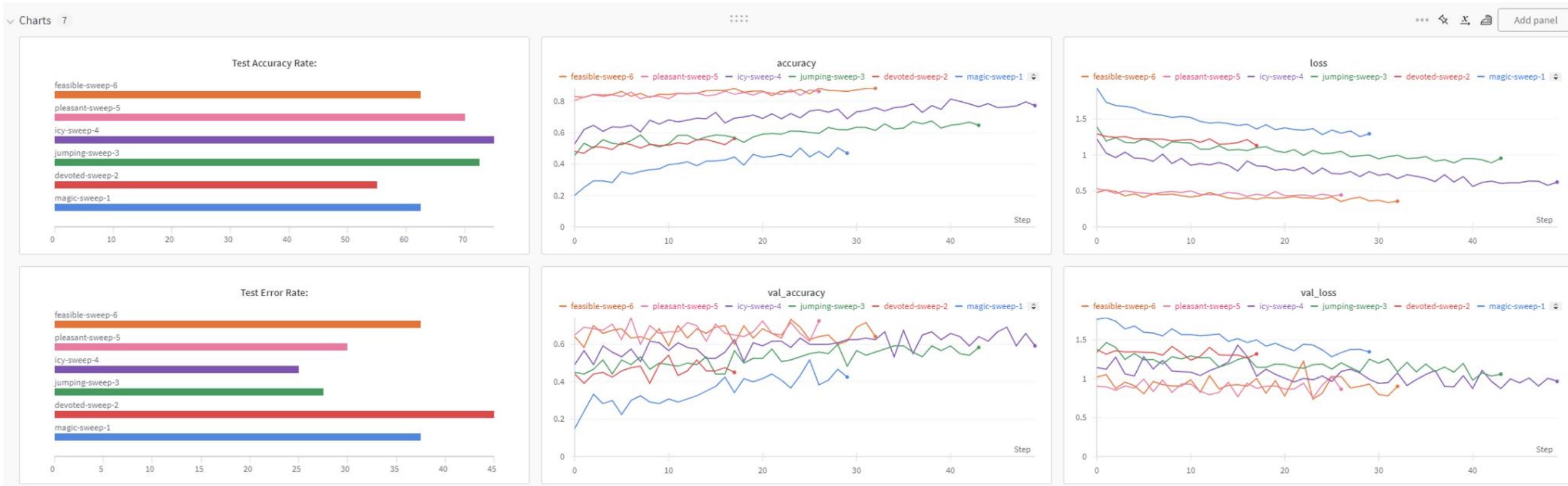
실험6과 비교.  
모델 구조만 변경.  
**conv layers** 필터 수,  
**dense** 노드 수 줄였더니  
**test\_accuracy** 77.5 → 75로 소폭 하락.



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 7. 실험 6에서 모델 구조 변경 :

**conv filter 16 - 32 - 64 - 64 → 16- 32- 32 - 32 / dense 255 → 128**





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 14. 실험 6과 비교 : kernel size 2x2 → 3x3

# 모델 작업 - 할수화

```
from tensorflow.keras.regularizers import l2
def build_model():
    model = keras.models.Sequential()
    model.add(keras.layers.Conv2D(16, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal', input_shape=(224, 224, 3))
    model.add(keras.layers.MaxPooling2D(2, 2))
    model.add(keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.MaxPooling2D(2, 2))
    model.add(keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.MaxPooling2D(2, 2))
    model.add(keras.layers.Conv2D(64, (1, 1), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(255, activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.Dropout(0.3))
    model.add(keras.layers.Dense(6, activation='softmax'))

    return model
```

실험6(test accuracy 77.5%)과

비교.

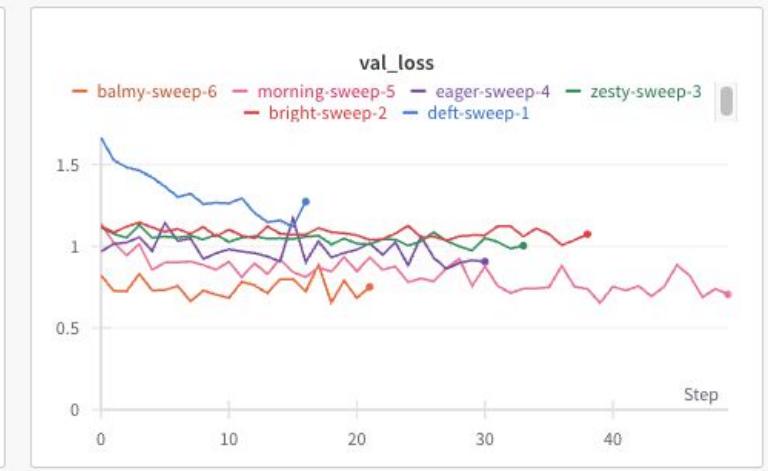
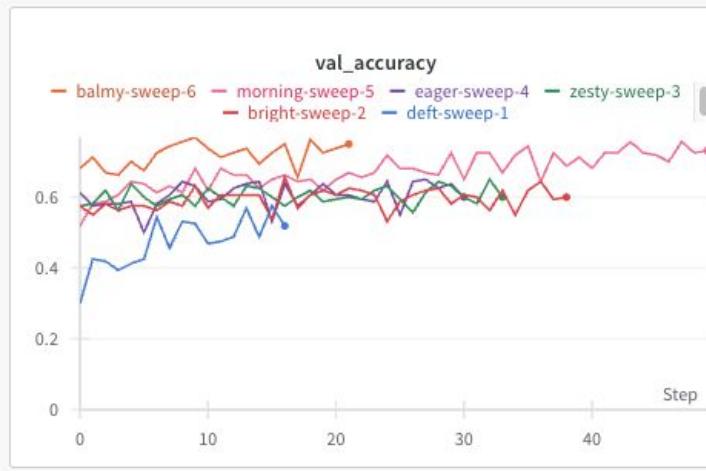
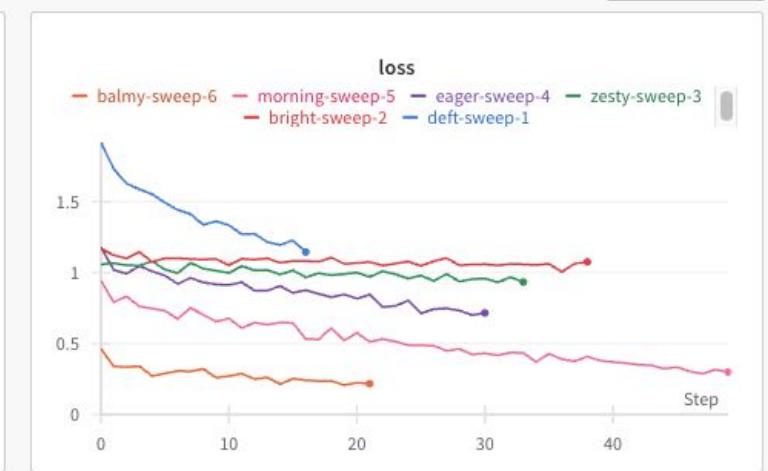
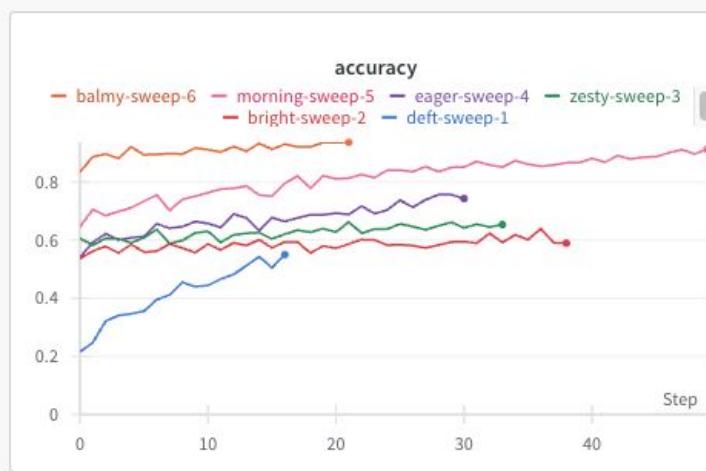
커널 사이즈만 2에서 3으로 변경.

test\_accuracy 67.5%로 대폭 하락.



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 14. 실험 6과 비교 : kernel size 2x2 → 3x3





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 15. 실험 14와 비교 : 학습률 범위 조정

```
"learning_rate": {
    "min": 0.00001,
    "max": 0.001 }
```

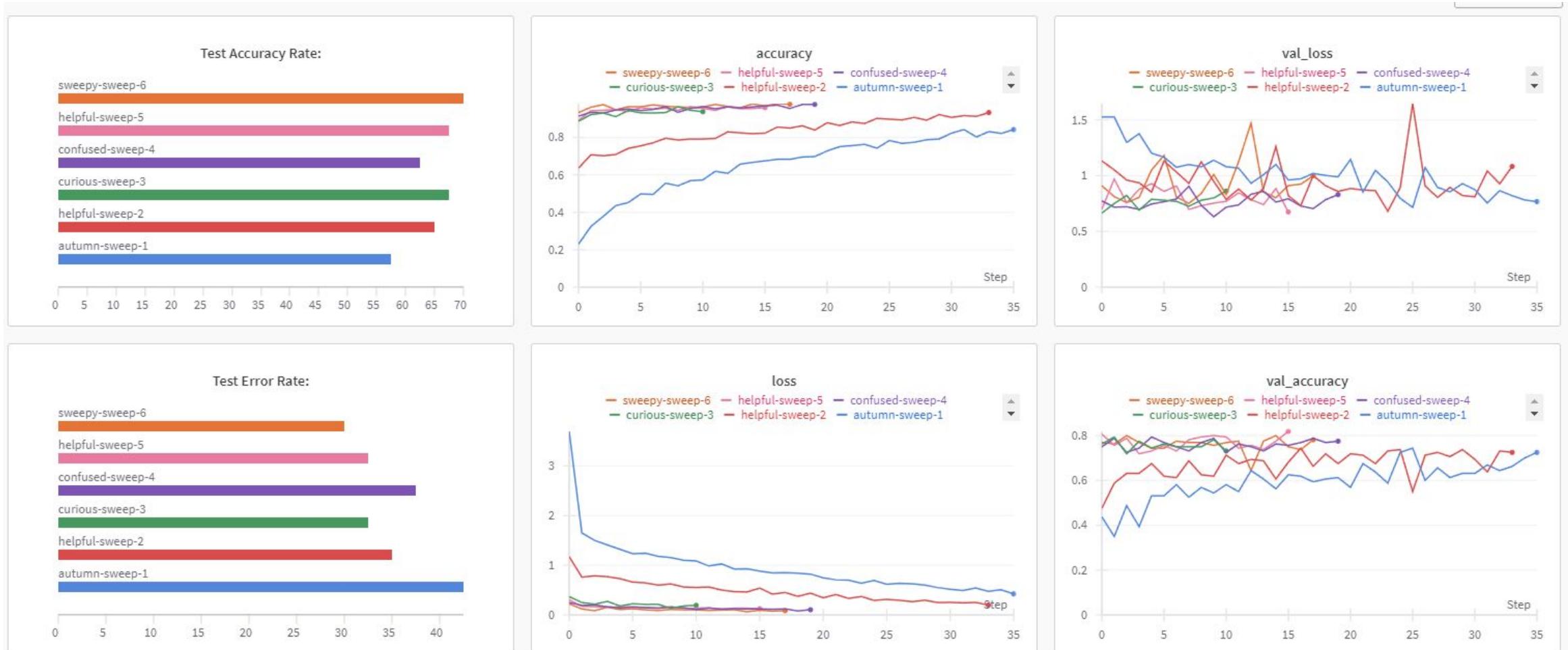
Config		Summary																																																											
Config parameters describe your model's inputs. <a href="#">Learn more</a>		Summary metrics describe your results. <a href="#">Learn more</a>																																																											
Search keys <table border="1"> <thead> <tr><th>Key</th><th>Value</th></tr> </thead> <tbody> <tr><td>activation</td><td>"relu"</td></tr> <tr><td>batch_size</td><td>32</td></tr> <tr><td>epoch</td><td>18</td></tr> <tr><td colspan="2">&gt; input (3 collapsed)</td></tr> <tr><td colspan="2">&gt; kernel (2 collapsed)</td></tr> <tr><td>learning_rate</td><td>0.0002648177074972438</td></tr> <tr><td>loss</td><td>"sparse_categorical_crossentropy"</td></tr> <tr><td colspan="2">&gt; metrics (1 collapsed)</td></tr> <tr><td>optimizer</td><td>"rmsprop"</td></tr> </tbody> </table>		Key	Value	activation	"relu"	batch_size	32	epoch	18	> input (3 collapsed)		> kernel (2 collapsed)		learning_rate	0.0002648177074972438	loss	"sparse_categorical_crossentropy"	> metrics (1 collapsed)		optimizer	"rmsprop"	Search keys <table border="1"> <thead> <tr><th>Key</th><th>Value</th></tr> </thead> <tbody> <tr><td>Test Accuracy Rate:</td><td>70</td></tr> <tr><td>Test Error Rate:</td><td>30</td></tr> <tr><td>accuracy</td><td>0.97826087474823</td></tr> <tr><td>best_epoch</td><td>7</td></tr> <tr><td>best_val_loss</td><td>0.7512744069099426</td></tr> <tr><td>epoch</td><td>17</td></tr> <tr><td colspan="2">&gt; examples</td></tr> <tr><td>_type</td><td>"images/separated"</td></tr> <tr><td colspan="2">&gt; captions (10 collapsed)</td></tr> <tr><td>count</td><td>10</td></tr> <tr><td colspan="2">&gt; filenames (10 collapsed)</td></tr> <tr><td>format</td><td>"png"</td></tr> <tr><td>height</td><td>224</td></tr> <tr><td>width</td><td>224</td></tr> <tr><td colspan="2">&gt; graph</td></tr> <tr><td>_type</td><td>"graph-file"</td></tr> <tr><td>path</td><td>"media/graph/graph_summary_72856936f7f674302bd9.graph.json"</td></tr> <tr><td colspan="2"></td></tr> </tbody> </table>		Key	Value	Test Accuracy Rate:	70	Test Error Rate:	30	accuracy	0.97826087474823	best_epoch	7	best_val_loss	0.7512744069099426	epoch	17	> examples		_type	"images/separated"	> captions (10 collapsed)		count	10	> filenames (10 collapsed)		format	"png"	height	224	width	224	> graph		_type	"graph-file"	path	"media/graph/graph_summary_72856936f7f674302bd9.graph.json"		
Key	Value																																																												
activation	"relu"																																																												
batch_size	32																																																												
epoch	18																																																												
> input (3 collapsed)																																																													
> kernel (2 collapsed)																																																													
learning_rate	0.0002648177074972438																																																												
loss	"sparse_categorical_crossentropy"																																																												
> metrics (1 collapsed)																																																													
optimizer	"rmsprop"																																																												
Key	Value																																																												
Test Accuracy Rate:	70																																																												
Test Error Rate:	30																																																												
accuracy	0.97826087474823																																																												
best_epoch	7																																																												
best_val_loss	0.7512744069099426																																																												
epoch	17																																																												
> examples																																																													
_type	"images/separated"																																																												
> captions (10 collapsed)																																																													
count	10																																																												
> filenames (10 collapsed)																																																													
format	"png"																																																												
height	224																																																												
width	224																																																												
> graph																																																													
_type	"graph-file"																																																												
path	"media/graph/graph_summary_72856936f7f674302bd9.graph.json"																																																												
<b>실험15와 비교했을때 학습률만 조정, test acc상승</b>																																																													



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 15. 실험 14와 비교 : 학습률 범위 조정

"learning\_rate" : {  
    "min" : 0.00001,  
    "max" : 0.001 }





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 16. 실험 3 모델에서 optimizer 변경 : RMSprop →

**SGD**

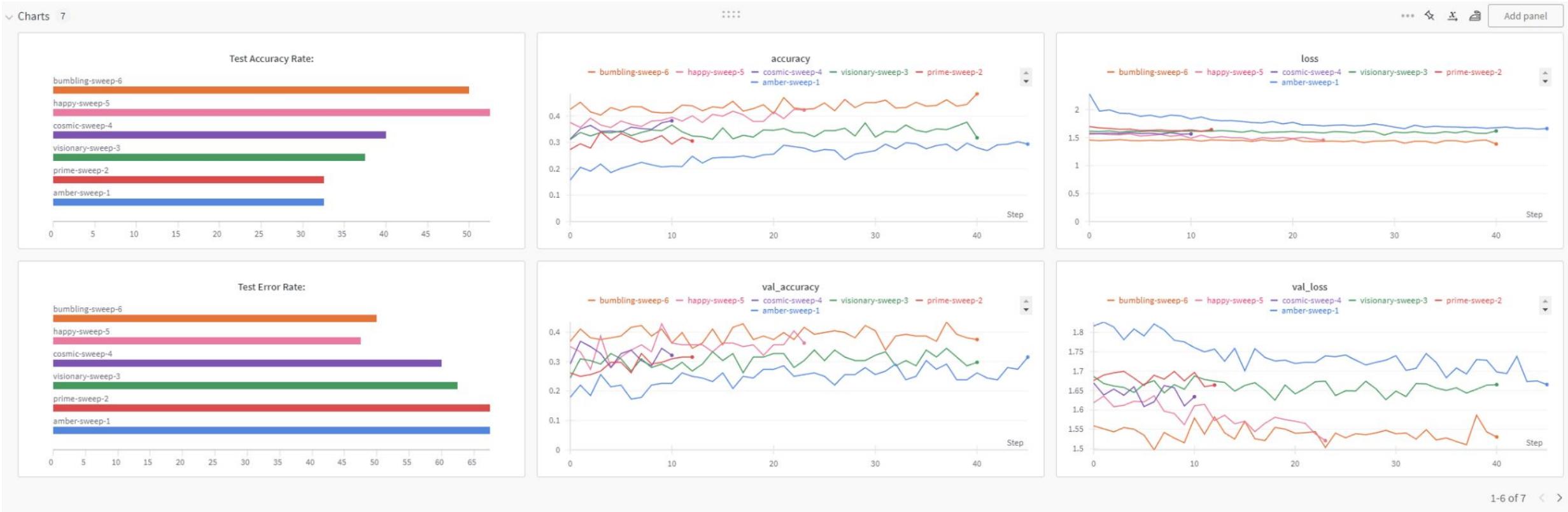
Config		View raw data	Summary		View raw data
<i>Config parameters describe your model's inputs. <a href="#">Learn more</a></i>					
Search keys			Search keys		
Key	Value		Key	Value	
activation	"relu"		Test Accuracy Rate:	52.5	
batch_size	24		Test Error Rate:	47.5	
epoch	24		accuracy	0.4220430254936218	
input			best_epoch	23	
0	224		best_val_loss	1.520728349685669	
1	224		epoch	23	
2	3		> examples (7 collapsed)		
kernel			> graph (4 collapsed)		
0	3		loss	1.4563263654708862	
1	3		val_accuracy	0.363095223903656	
learning_rate	0.00008728212882144466		val_loss	1.520728349685669	
loss	"sparse_categorical_crossentropy"				
> metrics (1 collapsed)					
optimizer	"sgd"				

실험3(test accuracy 80%)과 비교.  
SGD로 변경, test\_accuracy  
52.5%로 대폭 하락.



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 16. 실험 3 모델에서 optimizer 변경 : RMSprop → SGD





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 17. 실험 16과 비교 : optimizer SGD → Adam

Config

View raw data

Config parameters describe your model's inputs. Learn more

Key	Value
activation	"relu"
batch_size	24
epoch	49
input	
0	224
1	224
2	3
kernel	
0	3
1	3
learning_rate	0.00005510630890837101
loss	"sparse_categorical_crossentropy"
metrics (1 collapsed)	
optimizer	"adam"

Summary

View raw data

Summary metrics describe your results. Learn more

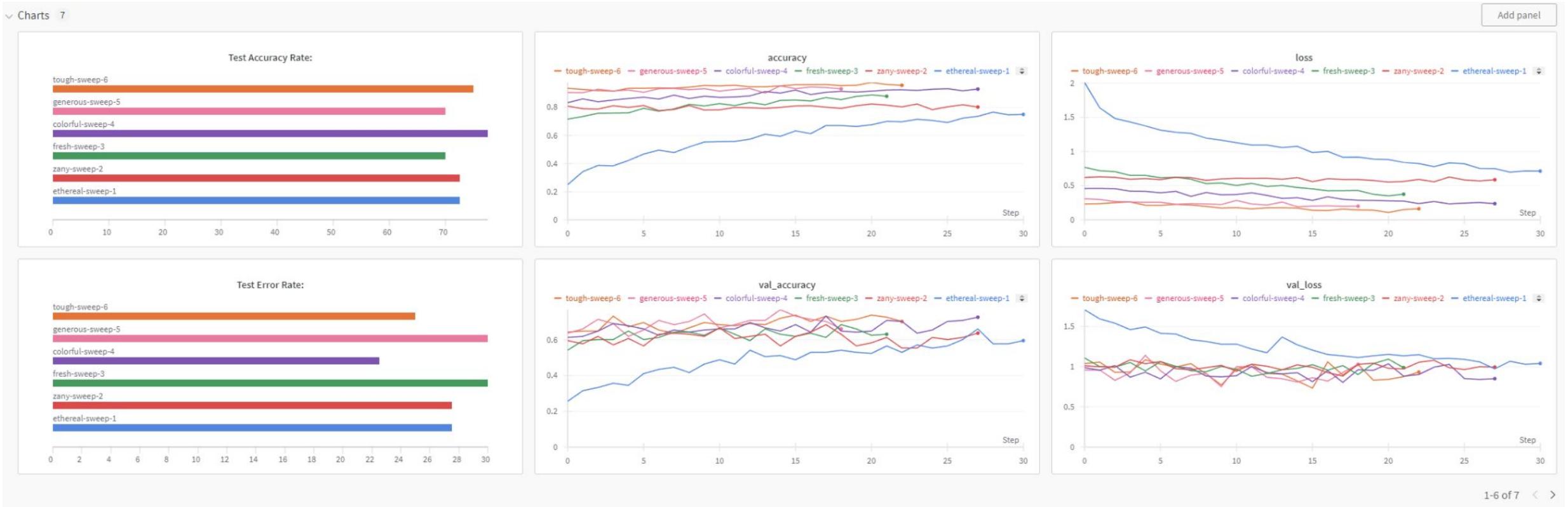
Key	Value
Test Accuracy Rate:	77.5
Test Error Rate:	22.5
accuracy	0.9301075339317322
best_epoch	17
best_val_loss	0.8048309087753296
epoch	27
examples (7 collapsed)	
graph (4 collapsed)	
loss	0.2368229627609253
val_accuracy	0.726190447807312
val_loss	0.8505206108093262

실험14와 비교.  
optimizer adam으로 변경,  
test\_accuracy 77.5%로  
실험3과 유사하지만 80%에는 못  
미침.



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 - 실험 17. 실험 16과 비교 : optimizer SGD → Adam





# [DLthon] Jellyfish Image Classification

epochs 50 → 100

## 4. 성능향상 시도 - 실험 18. Epoch 100 확대

Search keys	
Key	Value
activation	"relu"
batch_size	24
epoch	101
input	
0	224
1	224
2	3
kernel	
0	3
1	3
learning_rate	0.00009
loss	"sparse_categorical_crossentropy"
> metrics (1 collapsed)	
optimizer	"rmsprop"

Search keys	
Key	Value
Test Accuracy Rate:	67.5
Test Error Rate:	32.5
accuracy	0.9556451439857484
best_epoch	90
best_val_loss	0.9252473711967468
epoch	100
> examples (7 collapsed)	
> graph (4 collapsed)	
loss	0.13817526400089264
val_accuracy	0.6726190447807312
val_loss	1.4239022731781006

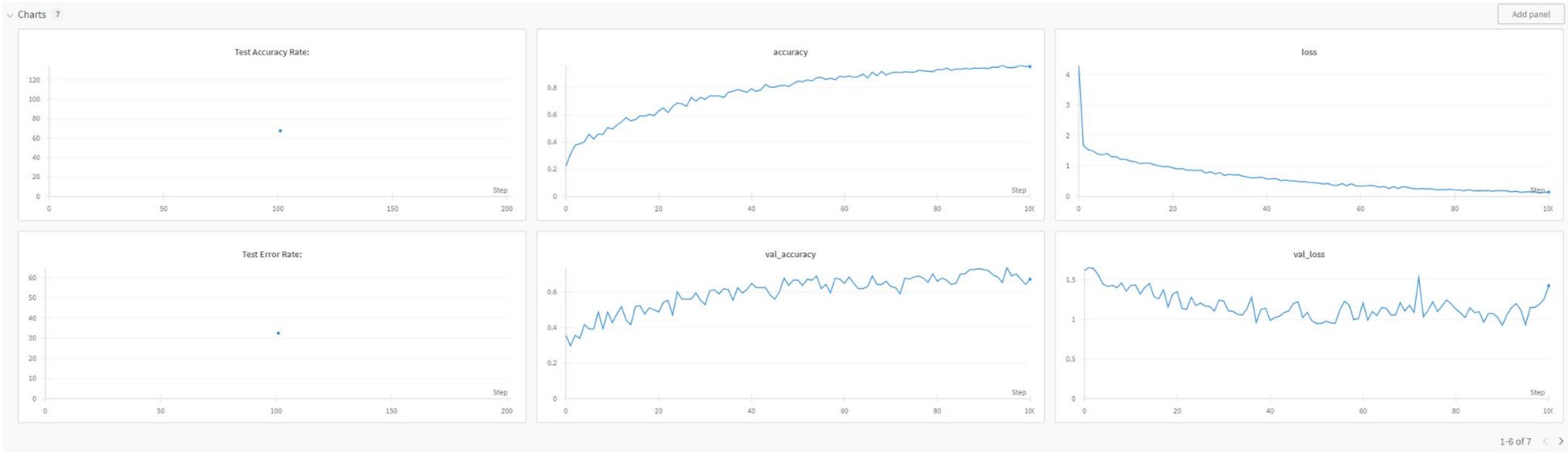
실험3과 비교.  
에폭 범위만 100까지로 늘림.  
**test accuracy**는 67.5%로  
아쉽지만 그래프 모양이 이상적으로  
수렴하는 형태(이상적인 학습의  
형태)가 보임. 얇은 층에 비해  
에폭의 수가 커서 과적합이 발견됨.



# [DLthon] Jellyfish Image Classification

epochs 50 → 100

## 4. 성능향상 시도 - 실험 18. Epoch 100 확대





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 ➡️ ✅ 종합 ✅

### ● **Lesson learned 1. 배치 사이즈와 학습률의 상관관계**

배치사이즈에 따라 한번에 학습하는 양을 나눴기 때문에 배치 사이즈가 커지면 학습률을 더 적게 설정해야 할 것 같아서 증명을 해보기로 했다.

[Case1] 배치사이즈=30으로 설정 시, 학습률=0.00004일 때 test\_accuracy=72.5

[Case2] 배치사이즈=16으로 설정 시, 학습률=0.0008일 때 test\_accuracy=70

[Case3] 배치사이즈=24로 설정 시, 학습률=0.0001일 때 test\_accuracy=67.5

[Case4] 배치사이즈=30으로 설정 시, 학습률=0.0001일 때 test\_accuracy=77.5

- 배치사이즈를 통해서 미니배치 형식으로 트레이닝 시 학습률과 배치사이즈는 반비례하게 설정해야 모델성능이 좋다!!!
- [Case1, 4 비교] 학습률이 작다고 무조건 좋은 accuracy가 나오진 않는다.

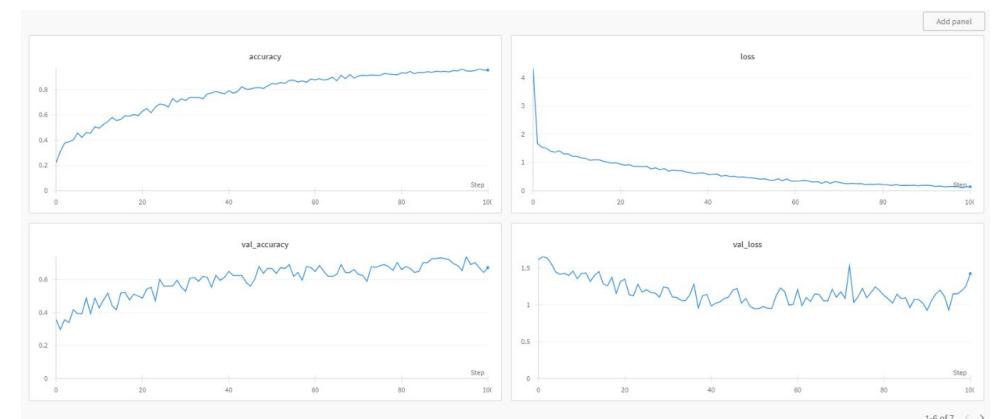


# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 ➡️ ✅ 종합 ✅

- **Lesson learned 2.**

- epoch을 100으로 늘린 경우,  
train / validation accuracy, loss 그래프가 이상적으로 수렴하는 그래프가 나옴
- epoch 값에 비해 모델 layer가 너무 얕아서 과대적합을 의심 : test accuracy 67.5%
- epoch 수를 대폭 늘린 부분은 이상적인 그래프를 얻게 된 좋은 시도였으나
  - 과적합 위험이 있어서 규제기법을 늘리거나 skip connection 등
  - 과적합방지를 위한 추가 기법이 필요





# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 ➡️ ✅ 종합 ✅

- **Lesson learned 3. 최대 성능 : test accuracy 80%**

```
# 모델 작업 - 함수화
def build_model():
    model = keras.models.Sequential()
    model.add(keras.layers.Conv2D(16, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal', input_shape=(224, 224, 3)))
    model.add(keras.layers.MaxPooling2D(2, 2))
    model.add(keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.MaxPooling2D(2, 2))
    model.add(keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.MaxPooling2D(2, 2))
    model.add(keras.layers.Conv2D(64, (1, 1), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(255, activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.Dropout(0.3))
    model.add(keras.layers.Dense(6, activation='softmax'))

    return model
```

- 배치사이즈 : 24, optimizer : RMSprop, kernel size : 3x3, Dense node 255



## [DLthon] Jellyfish Image Classification

### 4. 성능향상 시도 ➡️ ✅ 종합 ✅

- **Lesson learned 4. 향후 시도해볼만한 사항**

- 데이터 전처리

- 데이터 수가 900여개로 현저히 부족한 한계 극복

- ➡️ 데이터 추가 : 새로운 이미지 수집

- 일반화 성능 향상(강건성)을 위해

- ➡️ 데이터 증강 시도 : 색상 조정(밝기), 노이즈 추가(랜덤노이즈추가),  
기하학적 변형(이미지 기울임, 비틀기등)



# [DLthon] Jellyfish Image Classification

## 4. 성능향상 시도 ➡️ ✅ 종합 ✅

- **Lesson learned 4. 향후 시도해볼만한 사항**

- **모델 구현 및 학습**

- W&B는 일정 범위 내에서 최적의 파라미터 조합을 찾아줌.  
실험하고자 하는 하이퍼파라미터 외의 조건(학습률, 에폭 등)이 범위 내에서 변경됨.  
(파라미터 범위는 고정하여 동일한 조건에서 시도하더라도)  
👉 실험하고자 하는 특정 하이퍼파라미터만 변경하는 방식. (다른 조건 완전 Fix)
    - padding을 사용하지 않거나 횟수 줄이기  
👉 모델 층이 얕으므로 특성 추출에 방해되지 않도록.
    - k-fold 교차검증방법 활용  
👉 적은 datasets으로 인한 과적합 방지(데이터 활용 극대화), 일반화 성능 향상 목적.
    - 앞선 개선사항을 적용하여 모델을 깊게 쌓아보기, 다양한 모델 구조 적용해보기

## 5. 제품화 시 응용분야 및 모델 분석

- 모델 특징 : 데이터셋이 적은 분류 모델 (데이터를 구하기 어렵고, 모델의 종이 얇은 특징을 가진다.)
- 응용 분야 :
  - 해파리 사전 수매 작업으로 어업 피해 방지 및 어획량 확보
    - 피해 실사례 뉴스 : <https://v.daum.net/v/20220814214351423>
    - 해파리 중 유해종을 분류하여 해파리 피해로 인한 물고기 폐사를 사전에 방지
    - 개선사항 : 바닷속에서 Object Detection이 가능한 로봇 제작 기술 접목, 유해종 라벨링 추가, 해파리 처리 방식 및 도구 디벨롭(해파리 특성 활용 방향) 등
  - 수매 작업이란?  
해양수산부 해파리 피해 방지 매뉴얼에 따라 수매선박에서 해파리를 분쇄해 바다에 흘려보내는 것





# [DLthon] Jellyfish Image Classification

## 회고 이슬

keep:

- 팀원간의 소통이 잘 되어 프로젝트를 진행하는데 큰 어려움이 없었습니다. 최강의 팀플레이를 했다고 자신..!
- 기존에 함수화해둔 시각화나 데이터셋을 만드는 코드를 다시 사용했고, `wandb`로 모델학습을 시키는 부분도 함수화해서 사용하는 시도를 해보았습니다.

problem

- 짧은 시간내에 `wandb`라는 새로운 툴을 이용해서 프로젝트의 결과를 시각화 하는게 쉽지 않았습니다. 다행히 `wandb` 툴 사용을 포기하기 직전 사용방식에 대한 이해가 조금씩 되어서 다행이었지만, 여전히 `wandb`의 시각데이터를 읽고 해석하는게 쉽지 않습니다.
- 백지상태의 노트북에 코드를 채워가는게 정말 어렵다는걸 다시 깨달았습니다. 데이터 폴더에서부터 데이터를 꺼내오고 시각화 하는 과정이 노드의 베이스코드를 따라가면서 작업하던 때와 다르게 낯설고 어려웠습니다.
- 여전히 데이터셋을 `tf.instance`로 만들었을때 데이터증강이나 `kfold` 등 익숙한 데이터셋을 사용할때와는 다르게 에러해결이 어려웠습니다.

try:

- 주말에는 꼭 캐글 노트북의 데이터를 가져와서 이번 프로젝트에서 진행했던 방식처럼 코드를 만들어보고 노트북을 정답지 삼아 공부를 해볼수 있을것 같습니다. 단순히 캐글 노트북 필사를 해야겠다고 생각만 했는데, 구체적으로 시도해봐야겠습니다.
- `tf instance`를 활용한 노트북 자료를 찾고 공부해보려고합니다.



# [DLthon] Jellyfish Image Classification

회고      김양희

## keep

- 지금까지의 학습 내용을 폭넓게 적용해보았습니다.  
(데이터 특징에 따른 전처리 방향성, 함수화하면 편리한 코드, 모델 구현 및 향상 방법 등)
- 오류를 맞닥뜨리는 부담감이 예전에 비해 완화되어 해결한 경우가 소폭 늘어났습니다.
- 팀원 간 소통이 원활하여 자유로운 아이디어 공유 및 시도가 가능했고, 역할 분담과 주요 과업 진행이 빠르게 이루어졌습니다.

## problem

- W&B 코드, 모델 구현, GPU 활용하는 코드를 함께 적용하는 과정에서 몇 가지 오류가 반복되는 무한 오류에 빠졌습니다.
- 짧은 시간 안에 W&B라는 새로운 툴을 빠르게 익히고 활용하는 것이 쉽지는 않았습니다.
- 모델 성능 향상 자체가 쉽지 않았고, 다양한 하이퍼파라미터와 모델 구조를 적용해보자는 못했습니다.

## try:

- 해결하지 못한 오류 해결해보기
- 데이터셋 보강하는 방법 찾아보고 적용해보기
- 모델 성능 향상시켜보기 (+Case 별로 적절한 하이퍼파라미터, 모델 구조 등을 확립하기)
- Callbacks&WandB 익히며 활용해보기



# [DLthon] Jellyfish Image Classification

## 회고 이승환

### keep

- 다양한 파라미터, 모델구조 등을 여러방면으로 학습을 시켰다.
- 노드에서 배운 내용들을 적용할 수 있었다.
- 팀원분들 간 소통에 큰 어려움없이 문제를 해결하고 학습 진행이 잘 되었다.

### problem

- 코드를 구현하면서 그래프 시각화 하는 부분이 많이 부족했다.
- 전처리 하는 과정에서 어려움이 있었다.
- WandB를 사용하는 과정에 이해에 어려움이 있었다.
- 다양한 파라미터를 정리하고 실행하는 부분이 부족했다.

### try

- 그래프 시각화 부분을 다양한 플랫폼을 활용하여 배워야겠다.
- 코드 전처리하는 과정을 노드를 다시 학습하며 배워야겠다.
- WandB 사용하는 부분도 추가적인 학습을 해야겠다.
- 다양한 파라미터를 적용하기 전 변경할 자료들을 정리 후 차근차근 적용하면서 결과 값을 얻어야겠다.



# [DLthon] Jellyfish Image Classification

## 회고 전민규

### keep

- 이번 DLthon을 진행하면서 팀원들과 의사소통하며 같이 발전하고 새로운 시도들을 통해 배워갔던거 같습니다.
- 혼자라면 시간내에 시도하지 못했을거 같은 실험들을 팀원들과 역할분담을 진행하며 같이 소통하여 많고 다양한 시도들을 진행할수 있었습니다.
- 각자 데이터에 대한 이해가 다른만큼 여러가지방면에서 데이터셋을 분석하고 처리할수 있었습니다.
- 오류가 많이 발생했지만 다양한 시각과 팀원들과의 소통으로 잘 해결했던거 같습니다.
- w&b툴을 처음 써봤고 이해가 안갔지만 팀원들과 상의후 방법들을 찾아보았더니 w&b툴에대한 이해와 활용도가 높아진거 같습니다.
- 팀프로젝트는 처음이라 어떻게 시도해야할지 막막했지만 팀원들과 원활한 의사소통으로 빠르게 작업을 시작할수 있었던거같다.
- 오류나 다양한 문제를 직면했을때 팀원들과의 시도를 통해 다양한 해결방법과 시각으로 다가갈수 있어서 너무 좋았고 실력향상에 도움이 많이된것같다.

### problem

- w&b툴을 처음쓰다보니 사용법과 다양한 옵션들 그리고 활용하는 방안을 생각하고 실행하는것에 시간을 많이쓰고 오류가 많았던거 같습니다.
- 처음에 다양한 모델의 구조를 각각 시도해보는과정에서 resnet18을 사용해보았는데 데이터셋이 적어서 그런지 성능이 생각보다 너무 좋지않게 나왔습니다.
- 데이터셋이 현저히 적어 어떻게해야 모델의 성능을 높여야 할지 고민하는과정과 다양한 시도들에 대해 shape문제, w&b와 활용하는 방법등 다양한 문제가 생겼습니다.



# [DLthon] Jellyfish Image Classification

회고

전민규

try

- 데이터셋이 적기때문에 모델의 구조를 얕게쌓았고 그로인해 과적합이 많이 발생했던것같아, 더많은 데이터셋을 가져오거나 과적합을 방지할수있는 다양한 방법들을 추가로 시도해보아야겠다.
- cnn구조뿐만아니라 다양한 네트워크의 장점들을 활용하여 다른모델의 구조로 변경하여 시도해봐야겠다.
- 에폭을 100으로 대폭향상시켰더니 그래프자체는 이상적인 모양으로 수렴하는것을 알수있었다. 에폭을 늘리는대신에 과적합이 발생하여 testacc는 떨어졌지만 그래프가 이상적으로나와 다양한 과적합 규제방법을통해 에폭을 늘렸을때의 testacc를 높이는 방법을 시도해보아야겠다.
- w&b툴을 사용하는방법이 아직 완전히 익숙하지는 않은거같지만 활용시 좋은 성능을 비교할수있는 툴이기때문에 자세히 더 알아보아야겠다.



# 감사합니다!