

반려동물 안구 및 피부질환 데이터톤 소개



1. 대회 규칙

Overview – 행사 소개

- 대회 주제: 반려동물 안구 및 피부 질환 진단 AI모델 개발
 - 목표: 본 대회는 제공된 반려동물 안구 질환(4종) + 피부 증상(2종) 학습용 데이터와 Baseline Code를 활용하여 안구 질환과 피부 증상에 대해 가장 높은 성능의 AI 모델을 만드는 것이 목표
 - 대회 방식
 1. 참가팀은 제공된 반려동물 안구질환(4종)과 피부질환(2종) 데이터셋을 사용하여 **안구질환 진단 AI 모델과 피부 질환 진단 AI 모델을 모두 개발**합니다.
 2. 대회 종료 시 **개발된 AI 모델 5종** (안구질환 진단 AI 모델 4종, 피부질환 진단 AI 모델 1종)과 규정된 양식의 **결과 요약지**를 이용하여 모델 설명 및 자체 성능 평가 결과를 제출합니다.
(결과 요약지는 대회 Github 에서 다운로드 가능합니다 <https://github.com/DatathonInfo/MISOChallenge-animal>)
 3. 제출한 AI 모델을 이용하여 **주최측에서 테스트셋으로 성능 평가를 실시**하고, 안구질환 진단 AI 모델 및 피부질환 진단 AI 모델의 성능 및 우수성을 평가합니다.
 4. 안구질환 진단 AI 모델 점수(4개 질환) + 피부질환 진단 AI 모델 점수(2개 질환)으로 종합평가하여 **총 600점 만점에 고득점자 순으로 대상(1팀), 최우수상(1팀), 우수상(1팀)을 선정**하여 시상이 진행됩니다.
- ※ 동점자 발생 시에는 제출이 빠른 순으로 순위 선정

Overview – 행사 소개

- 지원 사항

원활한 학습을 위해 각 참가팀 별 네이버클라우드 GPU서버가 제공됩니다.

(Nvidia Tesla P40 (2GPU), GPU 메모리 48GB, vCPU8개, 메모리 60GB, 디스크 100GB SSD)

- GPU 서버에 세팅된 OS 및 개발 환경 정보

- CentOS 7.8
- Anaconda3-5.3.1
- Python 3.7
- CUDA 10.0
- cuDNN 7.6.0

※ 참가팀이 다수인 관계로 별도의 세팅 지원은 어려우니 참고 부탁드립니다.

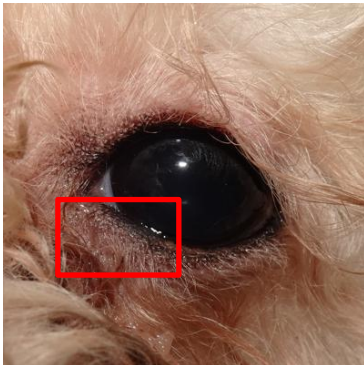
- 유의 사항

- 본 서버 이용에 있어 서버 환경 설정 시 Kernel 업데이트를 할 경우 정상적인 이용이 어려울 수 있으므로 Kernel 업데이트를 진행하지 않기를 권장드립니다.
- 또한, Kernel 업데이트로 인한 팀별 서버 다운 및 복구 시간 소요에 대해서는 별도의 복구는 지원되지 않으니 각별히 유의해주시기 바랍니다.

2. 반려동물 안구질환 데이터톤 소개

- 목표

- 전체 데이터는 주요 반려동물 안구질환 4종(안검내반, 유루증, 백내장, 결막염)으로 구성
- 본 대회는 주어진 데이터와 Baseline Code를 활용하여 가장 높은 성능의 진단 AI 모델을 만드는 것이 목표



<안검내반증>



<유루증>

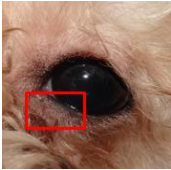





<백내장>



<결막염>

- 질환 정의

질환명	정의
 안검내반증	눈꺼풀테가 눈 쪽으로 말려들어가 눈썹이 눈을 찌르는 질환. 노령화로 인한 '퇴행성(노인성) 안검내반'과 만성 염증으로 인한 '반흔성안검내반'이 있음
 유루증	눈물흘림증(유루증)이란 코로 통하는 배출로인 눈물길이 좁아지고 막혀서 눈물 배출 기능이 저하되고, 이에 따라 눈물이 코로 배출되지 못하고 밖으로 흘러 넘치는 질환
 백내장	안구의 수정체가 혼탁해지는 질환
 결막염	안구와 안검을 결합하는 결막에 염증

- 데이터 정의(1) – 이미지 데이터

질환명	중증도	정의
 안검내반증	유	안검내반증 진단이 나온 경우
	무	정상 진단이 나온 경우
 유루증	유	유루증 진단이 나온 경우
	무	정상 진단이 나온 경우
 백내장	무	정상 진단이 나온 경우
	하	백내장 진단이 나옴 중증도가 약한 경우
	상	백내장 진단이 나옴 중증도가 강한 경우
 결막염	유	결막염 진단이 나온 경우
	무	정상 진단이 나온 경우

Overview

- 데이터 정의(2) – Json
: 1개의 이미지 데이터당 1개의 Json 데이터

※ JSON 구성 요소

```
{
  "images": {
    "meta": {
      "file_name": "D_d5c63201-2cd0-11ec-8402-0a7404972c70.png",
      "date_time": "16005",
      "device": "스마트폰",
      "gender": 1,
      "age": 5,
      "breed": "말티즈",
      "medical_type": 1,
      "width_height": [
        4032,
        3024
      ],
      "eye_position": "오른쪽눈",
      "image_resolution": [
        72,
        72
      ]
    }
  }
}
```

구분	항목명	타입	필수여부	설명	범위
1	Meta	Object		이미지 정보	
1-1	filename	String	Y	파일명	
1-2	date_time	String	Y	촬영 일시	
1-3	device	String	Y	촬영장비	
1-4	gender	Number	Y	성별	
1-5	age	Number	Y	나이	
1-6	breed	String	Y	품종	
1-7	medical_type	Boolean	Y	진료 형태	
1-8	width_height	String	Y	이미지 사이즈	
1-9	eye_position	String	Y	촬영는 방향	R, L
1-10	image_resolution	String	Y	이미지 해상도	

Overview

- 데이터 규모 및 형식(1)

질환명		중증도	총 데이터 수
	안검내반증	유	약 1,000 개
		무	약 1,000 개
	유루증	유	약 1,000 개
		무	약 1,000 개
	백내장	무	약 1,000 개
		하	약 1,000 개
		상	약 1,000 개
	결막염	유	약 1,000 개
		무	약 1,000 개

→ 반려동물 안구질환 4종에 대해 각각 약 1,000개, 총 9,023 개의 이미지 데이터 및 라벨 정보가 담긴 Json 데이터 제공

Overview

- 데이터 규모 및 형식(2)
 - 데이터 형식
 - jpg, png 형식의 이미지 데이터와 Json 데이터
 - 데이터 규모



Train Data
80%



Test Data
10%



Validation Data
10%

평가시 사용

Baseline Code - Import_data

```
class Import_data:
    def __init__(self, train_path):
        self.train_path = train_path
        self.test_path = val_path

    def train(self):
        train_datagen = ImageDataGenerator(rescale=1. / 255,
                                           featurewise_std_normalization=True,
                                           zoom_range=0.2,
                                           channel_shift_range=0.1,
                                           rotation_range=20,
                                           width_shift_range=0.2,
                                           height_shift_range=0.2,
                                           horizontal_flip=True
                                           )

        train_generator = train_datagen.flow_from_directory(
            self.train_path,
            target_size=(224, 224),
            batch_size=8
        )
        val_generator = train_datagen.flow_from_directory(
            self.test_path,
            target_size=(224, 224),
            batch_size=8
        )

        return train_generator, val_generator
```

Baseline code - def train

```
class Import_data:
    def __init__(self, train_path):
        self.train_path = train_path
        self.test_path = val_path

    def train(self):
        ① train_datagen = ImageDataGenerator(rescale=1. / 255,
                                            featurewise_std_normalization=True,
                                            zoom_range=0.2,
                                            channel_shift_range=0.1,
                                            rotation_range=20,
                                            width_shift_range=0.2,
                                            height_shift_range=0.2,
                                            horizontal_flip=True
                                            )

        ② train_generator = train_datagen.flow_from_directory(
            self.train_path,
            target_size=(224, 224),
            batch_size=8
        )
        val_generator = train_datagen.flow_from_directory(
            self.test_path,
            target_size=(224, 224),
            batch_size=8
        )

        ③ return train_generator, val_generator
```

- ① ImageDataGenerator: 학습 및 검증 이미지에 적용하고 싶은 전처리를 정의
 - rescale = RGB값을 리스케일링(0~1)
 - Featurewise_std_normalization : 인풋을 각 특성 내에서 데이터셋의 표준편차로 나누기
 - zoom_range : 이미지 확대 및 축소
 - Channel_shift_range : 무작위 채널 이동
 - Rotation_range : 이미지를 회전
 - width_shift_range : 좌우로 이동
 - height_shift_range : 상하로 이동
 - Horizontal_flip : 이미지 좌우 반전
- ② train_datagen.flow_from_directory: 경로 상의 image를 정의한 전처리 규칙 적용, 이미지 사이즈와 배치사이즈 배정
- ③ Return train_generator, val_generator: train_path, val_path에 전처리 규칙을 적용한 데이터 셋을 반환

Baseline code - Class Fine_tunning

```
class Fine_tunning:
    def __init__(self, train_path, model_name, epoch):
        self.data = Import_data(train_path)
        self.train_data, self.val_data = self.data.train()
        self.load_model = Load_model(train_path, model_name)
        # self.multi_gpu = multi_gpu
        self.epoch = epoch
        self.model_name = model_name
        self.train_path = train_path

    def training(self):
        data_name = self.train_path.split('/')
        data_name = data_name[len(data_name)-3]
        optimizer = tf.keras.optimizers.SGD(learning_rate=0.001, decay=1e-5, momentum=0.999, nesterov=True)
        model = self.load_model.build_network()
        save_folder = './model_saved/' + data_name + '/' + self.model_name + '_' + str(self.epoch) + '/'
        if not os.path.exists(save_folder):
            os.makedirs(save_folder)
        check_point = ModelCheckpoint(save_folder + 'model-{epoch:03d}-{acc:03f}-{val_acc:03f}.h5', verbose=1,
                                      monitor='val_acc', save_best_only=True, mode='auto')
        model.compile(loss='categorical_crossentropy',
                      optimizer=optimizer,
                      metrics=['acc'])
        history = model.fit_generator(
            self.train_data,
            steps_per_epoch=self.train_data.samples / self.train_data.batch_size,
            epochs=self.epoch,
            validation_data=self.val_data,
            validation_steps=self.val_data.samples / self.val_data.batch_size,
            callbacks=[check_point],
```

Baseline code - def training

```
class Fine_tuning:
    def __init__(self, train_path, model_name, epoch):
        self.data = Import_data(train_path)
        self.train_data, self.val_data = self.data.train()
        self.load_model = Load_model(train_path, model_name)
        # self.multi_gpu = multi_gpu
        self.epoch = epoch
        self.model_name = model_name
        self.train_path = train_path

    def training(self):
        ① data_name = self.train_path.split('/')
        data_name = data_name[len(data_name)-3]
        optimizer = tf.keras.optimizers.SGD(learning_rate=0.001, decay=1e-5, momentum=0.999, nesterov=True)
        model = self.load_model.build_network()
        save_folder = './model_saved/' + data_name + '/' + self.model_name + '_' + str(self.epoch) + '/'
        if not os.path.exists(save_folder):
            os.makedirs(save_folder)
        check_point = ModelCheckpoint(save_folder + 'model-{epoch:03d}-{acc:03f}-{val_acc:03f}.h5', verbose=1,
                                     monitor='val_acc', save_best_only=True, mode='auto')
        model.compile(loss='categorical_crossentropy',
                      optimizer=optimizer,
                      metrics=['acc'])
        history = model.fit_generator(
            self.train_data,
            steps_per_epoch=self.train_data.samples / self.train_data.batch_size,
            epochs=self.epoch,
            validation_data=self.val_data,
            validation_steps=self.val_data.samples / self.val_data.batch_size,
            callbacks=[check_point],
```

- ① 데이터 추출 및 학습준비
- Data_name : train_path를 통한 data_name 추출 및 정의
 - tf.keras.optimizers.SGD : 모델 학습의 optimizer(최적화) 방식을 케라스에서 제공하는 SGD(Stochastic Gradient Descent)로 사용
 - Model : 학습할 모델 정의
 - Save_folder : 학습된 모델 결과를 저장할 경로 지정
 - check_point : 모델 학습 결과 중간 저장 설정 (verbose=학습진행과정 표시방법, monitor=중간저장 기준 설정, save_best_only=best값만 저장)

Baseline code - def training

```
self.data = Import_data(train_path)
self.train_data, self.val_data = self.data.train()
self.load_model = Load_model(train_path, model_name)
# self.multi_gpu = multi_gpu
self.epoch = epoch
self.model_name = model_name
self.train_path = train_path

def training(self):
    data_name = self.train_path.split('/')
    data_name = data_name[len(data_name)-3]
    optimizer = tf.keras.optimizers.SGD(learning_rate=0.001, decay=1e-5, momentum=0.999, nesterov=True)
    model = self.load_model.build_network()
    save_folder = './model_saved/' + data_name + '/' + self.model_name + '_' + str(self.epoch) + '/'
    if not os.path.exists(save_folder):
        os.makedirs(save_folder)
    check_point = ModelCheckpoint(save_folder + 'model-{epoch:03d}-{acc:03f}-{val_acc:03f}.h5', verbose=1,
                                  monitor='val_acc', save_best_only=True, mode='auto')
    ① model.compile(loss='categorical_crossentropy',
                    optimizer=optimizer,
                    metrics=['acc'])
    history = model.fit_generator(
        self.train_data,
        steps_per_epoch=self.train_data.samples / self.train_data.batch_size,
        epochs=self.epoch,
        validation_data=self.val_data,
        validation_steps=self.val_data.samples / self.val_data.batch_size,
        callbacks=[check_point],
        verbose=1)
    return history
```

① 모델 세부설정

- **Model.compile** : 모델 학습과정 설정 (loss=손실함수지정, optimizer=학습과정 최적화 방식 설정, metrics=학습과정 모니터링 지표 설정)
- **Model.fit_generator** : 모델 학습 방법 정의 (step_per_epoch = epoch당 사용되는 스텝 수, epochs = 학습 epoch 수, callbacks = 중간 저장 방법 지정, verbose = 학습진행 과정 표시 설정)
- **Return history** : 학습 결과를 반환

Baseline code - def save_accuracy

```
def save_accuracy(self, history):  
    data_name = self.train_path.split('/')  
    data_name = data_name[len(data_name)-3]  
    save_folder = './model_saved/' + data_name + '/' + self.model_name + '_' + str(self.epoch) + '/'  
    acc = history.history['acc']  
    val_acc = history.history['val_acc']  
    loss = history.history['loss']  
    val_loss = history.history['val_loss']  
    epochs = range(len(acc))  
    epoch_list = list(epochs)  
  
    df = pd.DataFrame({'epoch': epoch_list, 'train_accuracy': acc, 'validation_accuracy': val_acc},  
                      columns=['epoch', 'train_accuracy', 'validation_accuracy'])  
    df_save_path = save_folder + 'accuracy.csv'  
    df.to_csv(df_save_path, index=False, encoding='euc-kr')  
  
    plt.plot(epochs, acc, 'b', label='Training acc')  
    plt.plot(epochs, val_acc, 'r', label='Validation acc')  
    plt.title('Training and validation accuracy')  
    plt.legend()  
    save_path = save_folder + 'accuracy.png'  
    plt.savefig(save_path)  
    plt.cla()
```

① 학습결과 저장

- Save_folder = 모델 저장 경로 설정
- acc = 학습 정확도
- val_acc = 검증데이터 정확도
- loss = 학습 손실함수
- val_loss = 검증 손실함수
- df = 학습 결과 리포트 csv 저장을 위한 데이터 프레임 형성
- Plt.plot = epochs 당 학습 정확도 및 검증 정확도 그래프 생성

Baseline code - def save_accuracy

```
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
save_path = save_folder + 'loss.png'
plt.savefig(save_path)
plt.cla()

name_list = os.listdir(save_folder)
h5_list = []
for name in name_list:
    if '.h5' in name:
        h5_list.append(name)
h5_list.sort()
h5_list = [save_folder + name for name in h5_list]
for path in h5_list[:len(h5_list) - 1]:
    os.remove(path)
K.clear_session()
```

① 학습결과 저장

- `Plt.plot` = epochs 당 학습 손실함수 및 검증 손실함수 그래프 생성
- `h5_list` = 중간 저장된 학습 결과 모델들의 리스트 저장
- `os.remove(path)` = 중간 저장된 결과들을 제거하고 마지막 결과(가장 좋은)만 저장

Baseline code - def save_accuracy

①

```
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
save_path = save_folder + 'loss.png'
plt.savefig(save_path)
plt.cla()

name_list = os.listdir(save_folder)
h5_list = []
for name in name_list:
    if '.h5' in name:
        h5_list.append(name)
h5_list.sort()
h5_list = [save_folder + name for name in h5_list]
for path in h5_list[:len(h5_list) - 1]:
    os.remove(path)
K.clear_session()
```

① 학습결과 저장

- Plt.plot = epochs 당 학습 손실함수 및 검증 손실함수 그래프 생성
- h5_list = 중간 저장된 학습 결과 모델들의 리스트 저장
- os.remove(path) = 중간 저장된 결과들을 제거하고 마지막 결과(가장 좋은)만 저장

Baseline code - Keras_train.py run

①

```
# from ops import *

train_path = '/home/ubuntu/Dataset/hackathon/Cataract_circle/train/'
val_path = '/home/ubuntu/Dataset/hackathon/Cataract_circle/val/'
model_name = 'resnet_v1_50'
epoch = 100

if __name__ == '__main__':
    fine_tunning = Fine_tunning(train_path=train_path,
                                model_name=model_name,
                                epoch=epoch)
    history = fine_tunning.training()
    fine_tunning.save_accuracy(history)
```

① 학습 실행

- train_path = 학습이미지 경로
- val_path = 검증 이미지 경로
- model_name = 사용할 모델명
- epoch = 학습횟수

- 평가기준

- Accuracy 점수: 제공하지 않은 별도 검증 데이터를 이용한 모델의 Accuracy 측정

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- 제출 파일

- keras_test.py : 모델 등 변경사항에 따른 있더라도 Base line code를 참조하여 수정된 파일, Test Data를 활용하여 검증 필요
- 기타 소스코드
- accuracy.csv : Accuracy 출력 값
- accuracy.png : Accuracy 출력 그래프
- loss.png : Loss 출력 그래프
- model-xxx-xxx-xxx.h5 : 출력 모델
- 또는, 상기 파일 및 기타 소스코드에 대한 인스턴스의 경로

3. 반려동물 피부질환 데이터톤 소개

- 목표

- 전체 데이터는 반려동물 피부 증상 2종 + 무증상
 - 구진_플라크, 태선화_과다색소침착, 무증상
- 본 대회는 주어진 데이터와 Baseline Code를 활용하여 가장 높은 성능의 진단 AI 모델을 만드는 것이 목표
- 데이터 변경 및 추가는 불가능하고, 주어진 데이터를 이용하여 전처리, 증강 기법 적용 및 인공지능 모델이나 구성 변경 등이 가능함

Overview

- A1 구진_플라크 증상 상세 예시



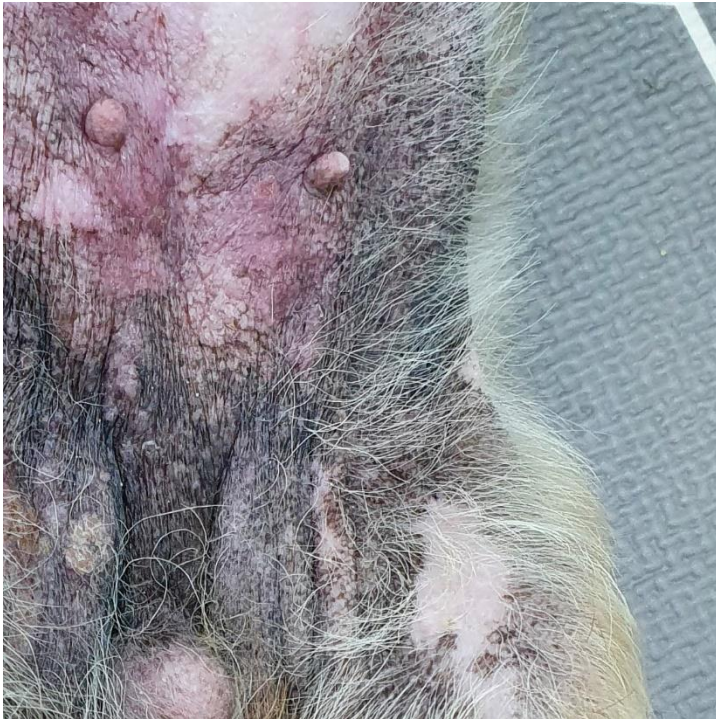
<구진>



<플라크>

Overview

- A3 구진_플라크 증상 상세 예시



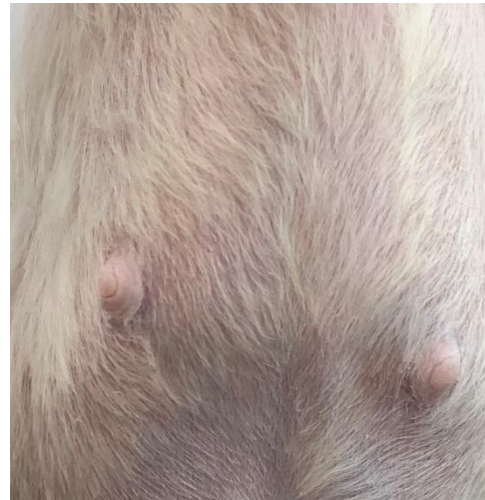
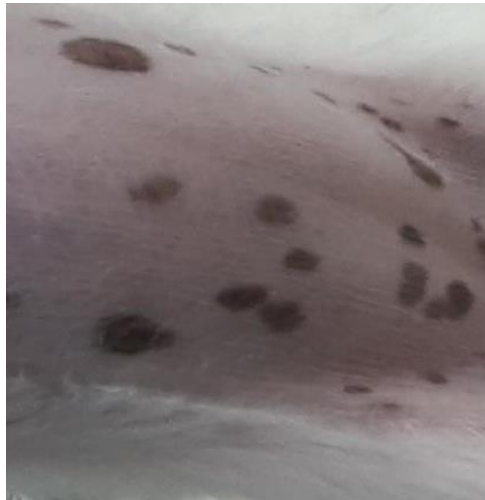
<구진>



<플라크>

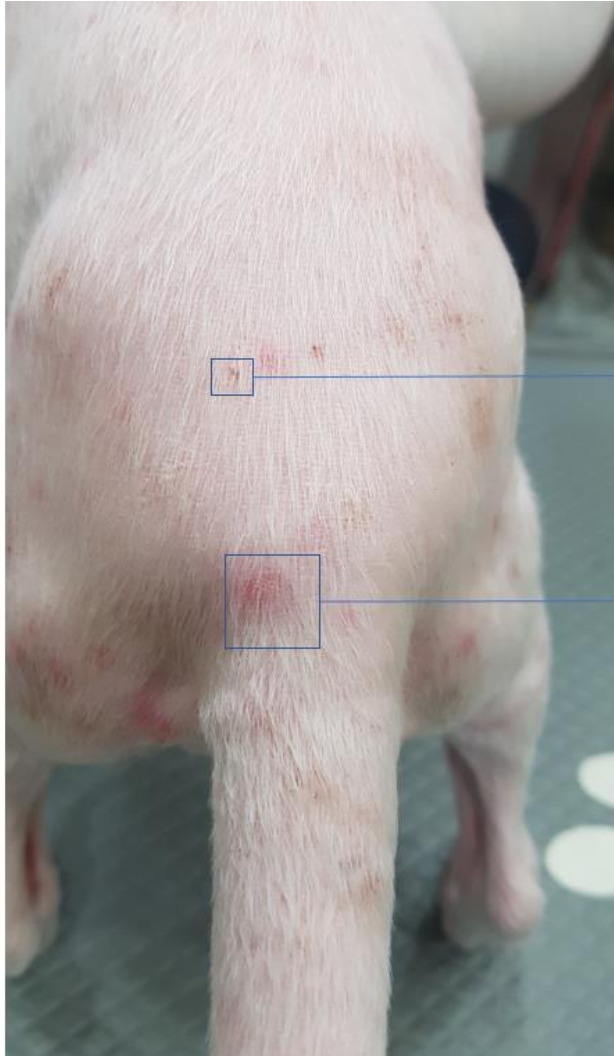
Overview

- 무증상 상세 예시



Overview

- 구진_플라크 예시 이미지

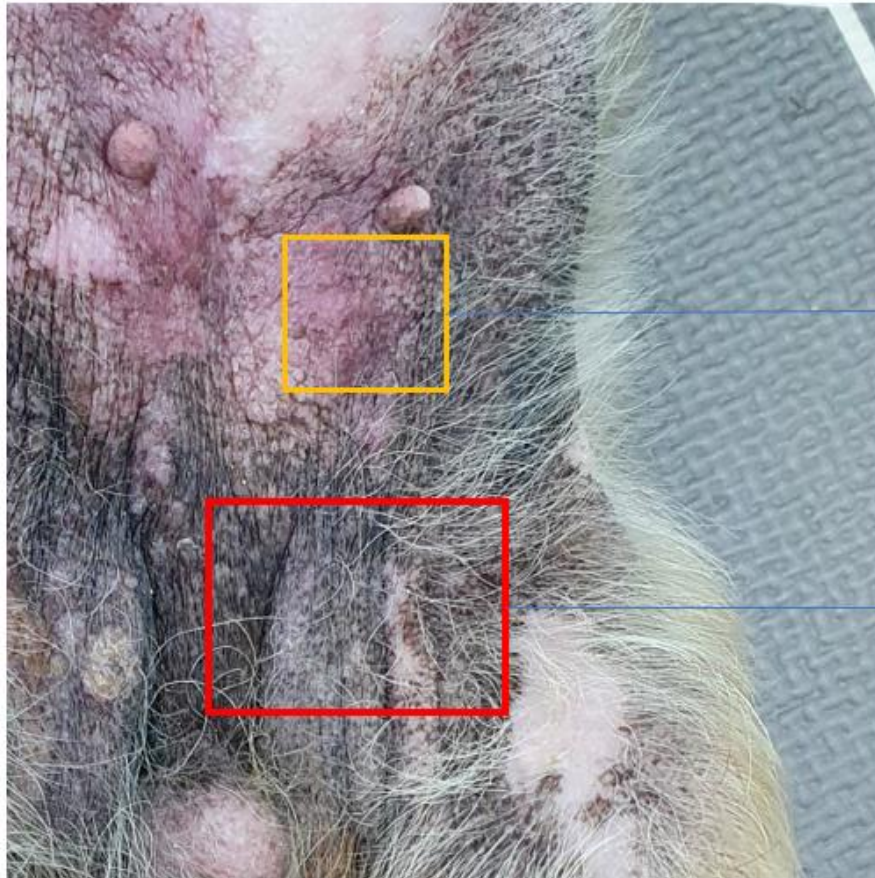


구진

플라크 (구진보다
1cm이상 큰 증상)

Overview



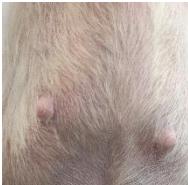
- 태선화_과다색소침착 예시 이미지



태선화 (두꺼워진
피부)

과다색소침착 (검게
변함)

• 데이터 정의(1) – 이미지 데이터

증상명	증증도	정의
 구진_플라크	유	<ul style="list-style-type: none"> - 구진: 작고 단단하게 융기된 피부변경. 병변 크기 (1CM)에 따라 구진과 플라크로 구분됨 - 플라크: 구진보다 1CM 더 큰 병변
 태선화_과다색소침착	유	<ul style="list-style-type: none"> - 태선화: 코끼리 다리 같이 두꺼워지고 단단해진 피부병 - 과다색소침착: 상피 또는 진피에 멜라닌 침착
 무증상	무	<ul style="list-style-type: none"> - 무증상: 어떤 증상도 없는 정상 피부

태선화/ 과다색소침착 추가 설명

- 태선화 : 코끼리 다리 피부처럼 두꺼워졌다는 한자 뜻. 피부에 주름 등 정상적인 피부조직 보다 두꺼워진 상태
- 과다 색소침착 : 염증들이 계속 반복되면 원래 피부색과 다르게 피부에 색소를 유발하는 세포들에 과해져, 피부색이 변질됨 (붉은색, 검은색, 갈색, 회색 등)
- 정상적인 피부세포가 변성을 일으켜서 색이나 두께가 변해서 각각 발병된 경우도 있지만 태선화가 있는 피부는 대부분 과다색소침착이 존재함. 태선화가 두꺼워지는 과정에서 색이 변함.

Overview

• 데이터 정의(2) – Json

: 1개의 이미지 데이터당 1개의 Json 데이터

※ JSON 구성 요소

M E T A			
항목명	설명	비고	
Raw data ID	파일명		
Copyright	저작권자		
Resolution	해상도		
Date	촬영일자		
Breed	품종		
Age	나이	1~30	
Gender	성별	M=수컷/F=암컷	
Region	촬영위치	L=다리 H=머리 B=몸통 A=연접부	
Camera type	촬영 장비	IMG / CYT	
Species	반려종	D=개 C=고양이	
Lesions	증상	A1 = 구진_플라크 A3=태선화_색소과다침	
Diagnosis	질병	공백	
Path	이미지폴더명		

L A B E L			
항목명	설명	비고	
Polygon	폴리곤		
Color	라벨링 색상	#56bcec, ,,,	
Location	라벨링 좌표	X1=210,y1=258,,,	
Label	증상	A1 = 구진_플라크 A3=태선화_색소과다침	
Type	저작도구 종류	Polygon	
Box	바운딩박스		
Bounding Box	라벨링 좌표	Xmin, ymin, Xmax, ymax	
Label	라벨	바운딩박스	
Type	저작도구 종류	Box	
항목명	설명	비고	
inspRejectYn	반려처리 여부 (교차검증)	Y/N	

Overview

- 데이터 규모 및 형식(1)

	증상명	중증도	총 데이터 수
	A1 구진_플라크	유	약 1,000 개
	A3 태선화_과다색소침착	유	약 1,000 개
	A7 무증상	무	약 1,000 개

→ 반려동물 피부 유증상 2종, 무증상에 대해 각각 약 1,000개, 총 3,000 개의 이미지 데이터 및 라벨 정보가 담긴 Json 데이터 제공

- 데이터 규모 및 형식(2)
 - 데이터 형식
 - Jpg, Png 형식의 이미지 데이터와 Json 데이터
 - 데이터 규모



Train Data
80%



Test Data
10%



Validation Data
10%

평가시 사용


```
# from ops import *  
train_path = 'D:/train/' # 경로 마지막에 반드시 '/'를 기입해야합니다.  
val_path = 'D:/val/'  
model_name = 'inception_resnet_v2'  
epoch = 100
```

```
① if __name__ == '__main__':  
    fine_tunning = Fine_tunning(train_path=train_path,  
                                model_name=model_name,  
                                epoch=epoch)  
    ② history = fine_tunning.training()  
    ③ fine_tunning.save_accuracy(history)
```

Entry point

- ① 모델 학습 관리 객체 생성
- ② 모델 학습
- ③ 모델 및 학습 결과 저장

Baseline Code – Import data

```
class Import_data:
    def __init__(self, train_path):
        self.train_path = train_path
        self.test_path = val_path

    def train(self):
        # generator 생성
        train_datagen = ImageDataGenerator(rescale=1. / 255,
                                           featurewise_std_normalization=True,
                                           zoom_range=0.2,
                                           channel_shift_range=0.1,
                                           rotation_range=20,
                                           width_shift_range=0.2,
                                           height_shift_range=0.2,
                                           horizontal_flip=True
                                           )

        train_generator = train_datagen.flow_from_directory(
            self.train_path,
            target_size=(224, 224),
            batch_size=8
        )
        val_generator = train_datagen.flow_from_directory(
            self.test_path,
            target_size=(224, 224),
            batch_size=8
        )

        return train_generator, val_generator
```

1

① ImageDataGenerator

: 입력 영상데이터의 전처리 객체

- rescale = 화소값의 정규화
- Featurewise_std_normalization : 특징의 정규화
- zoom_range : 입력 영상 확대 및 축소 범위
- Channel_shift_range : 채널 이동 범위
- Rotation_range : 영상 회전 범위
- width_shift_range : 영상 좌우로 이동 범위
- height_shift_range : 영상 상하로 이동 범위
- Horizontal_flip : 영상 좌우 반전

Baseline Code – Load model

```
class Load_model:
    def __init__(self, train_path):
        self.num_class = len(os.listdir(train_path)) # 클래스 수

    # 모델 정의
    def build_network(self):
        # Instantiates the Inception-ResNet v2 architecture
        network = InceptionResNetV2(include_top=False,
                                     weights='imagenet',
                                     input_tensor=None,
                                     input_shape=(224, 224, 3),
                                     pooling='avg')

        model = Sequential()
        model.add(network)
        model.add(Dense(2048, activation='relu'))
        model.add(Dense(self.num_class, activation='softmax'))
        model.summary()

        return model
```

① 모델: Inception-Resnet-v2

Baseline Code – Fine tuning: Training

```
def training(self):
    data_name = self.train_path.split('/')
    data_name = data_name[len(data_name)-3]

    # 옵티마이저 정의
    optimizer = tf.keras.optimizers.SGD(learning_rate=0.001, decay=1e-5, momentum=0.999, nesterov=True)

    # 모델 생성
    model = self.load_model.build_network()

    # 학습모델 저장할 경로 생성
    save_folder = './model_saved/' + data_name + '/' + self.model_name + '_' + str(self.epoch) + '/'
    if not os.path.exists(save_folder):
        os.makedirs(save_folder)

    # 훈련 중 주기적으로 모델 저장
    check_point = ModelCheckpoint(save_folder + 'model-{epoch:03d}-{acc:03f}-{val_acc:03f}.h5', verbose=1,
                                  monitor='val_acc', save_best_only=True, mode='auto')

    # 모델 컴파일
    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['acc'])

    # 모델 학습
    history = model.fit_generator(
        self.train_data,
        steps_per_epoch=self.train_data.samples / self.train_data.batch_size,
        epochs=self.epoch,
        validation_data=self.val_data,
        validation_steps=self.val_data.samples / self.val_data.batch_size,
        callbacks=[check_point],
        verbose=1)

    return history
```

Epoch 9/10
405/404 [=====] - ETA: 0s - loss: 0.8283 - acc: 0.6449
Epoch 00009: val_acc improved from 0.68486 to 0.71464, saving model to ./model_saved/
/data_skin/inception_resnet_v2_10/model-009-0.644932-0.714640.h5

- ① 옵티마이저 정의
- ② 정의된 모델 생성
- ③ 훈련 중 주기적으로 학습
모델 저장
- ④ 모델 컴파일
- ⑤ 모델 학습

Baseline Code – Fine tuning: Save accuracy

```
def save_accuracy(self, history):
    # 학습모델 저장 경로
    data_name = self.train_path.split('/')
    data_name = data_name[len(data_name)-3]
    save_folder = './model_saved/' + data_name + '/' + self.model_name + '_' + str(self.epoch) + '/'

    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(len(acc))
    epoch_list = list(epochs)

    # csv 저장
    df = pd.DataFrame({'epoch': epoch_list, 'train_accuracy': acc, 'validation_accuracy': val_acc},
                      columns=['epoch', 'train_accuracy', 'validation_accuracy'])
    df_save_path = save_folder + 'accuracy.csv'
    df.to_csv(df_save_path, index=False, encoding='euc-kr')

    # Accuracy 그래프 이미지 저장
    plt.plot(epochs, acc, 'b', label='Training acc')
    plt.plot(epochs, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    save_path = save_folder + 'accuracy.png'
    plt.savefig(save_path)
    plt.cla()

    # Loss 그래프 이미지 저장
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
    save_path = save_folder + 'loss.png'
    plt.savefig(save_path)
    plt.cla()

    # 마지막 모델을 제외하고 삭제
    name_list = os.listdir(save_folder)
    h5_list = []
    for name in name_list:
        if '.h5' in name:
            h5_list.append(name)
    h5_list.sort()
    h5_list = [save_folder + name for name in h5_list]
    for path in h5_list[:len(h5_list) - 1]:
        os.remove(path)
    K.clear_session()
```

- ① 학습을 통해 기록된 Accuracy, Loss 값 csv 저장
- ② Accuracy 그래프 이미지 저장
- ③ Loss 그래프 이미지 저장
- ④ 학습 모델 저장 경로에 마지막 학습 모델 제외하고 삭제

- 평가기준

- Accuracy 점수 : 제공하지 않은 별도 검증 데이터를 이용한 모델의 Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- 제출 파일

- keras_test.py : 모델 등 변경사항에 따른 있더라도 Base line code를 참조하여 수정된 파일, Test Data를 활용하여 검증 필요
- 기타 소스코드
- accuracy.csv : Accuracy 출력 값
- accuracy.png : Accuracy 출력 그래프
- loss.png : Loss 출력 그래프
- model-xxx-xxx-xxx.h5 : 출력 모델
- 또는, 상기 파일 및 기타 소스코드에 대한 인스턴스의 경로

Thank You

Q&A