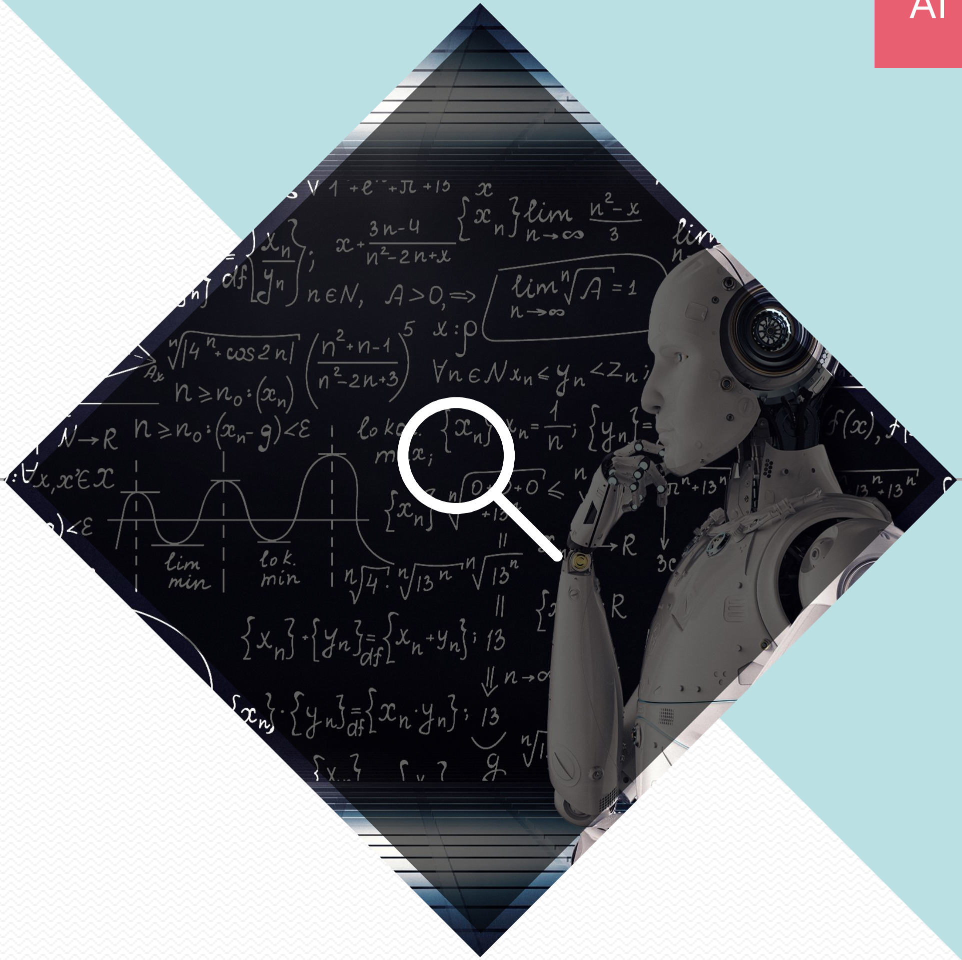
Before Start

WHAT YOU WILL LEARN FROM THIS SESSION

- Hands on practice of adopting CNN knowledge.
- Designing & building a model for image classification.
- Training & inferencing workflows in TensorFlow.
- Getting help on TensorFlow modules & functions.
- Tuning hyper-parameter.

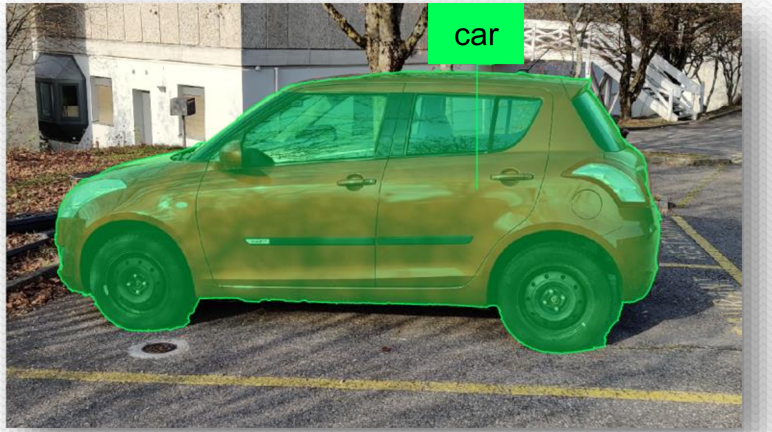
AI TRAINING SERIES



CNN hands on

AI Use Case

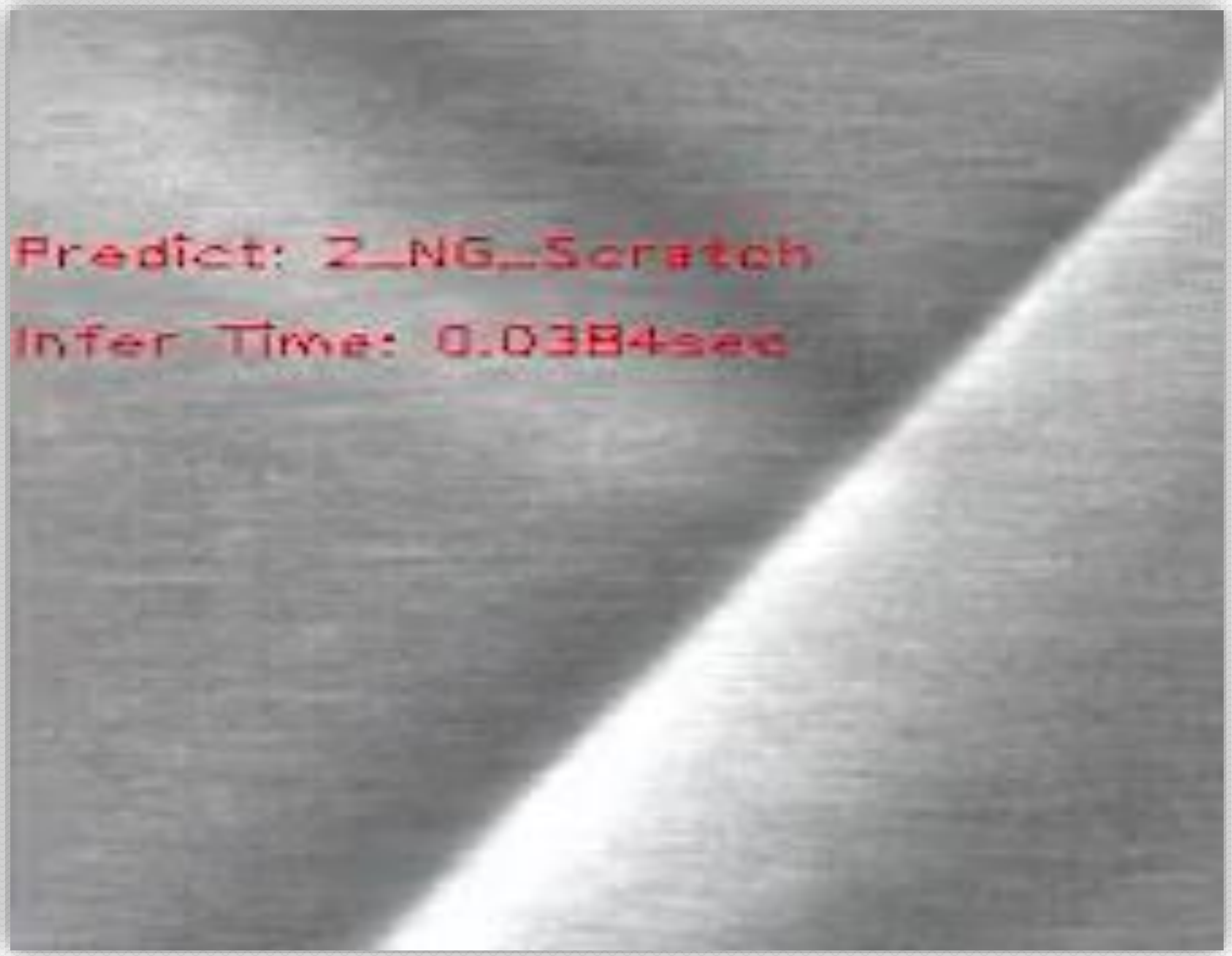
- Classification
  - Classify the whole that one or identify it in an image
- Detection
  - Track that object in a video or identify it in an image
- Segmentation
  - Highlight the isolated region and groups every pixel in that delineated shape for later processing



Classification

- Classify the whole that one or identify it in an image

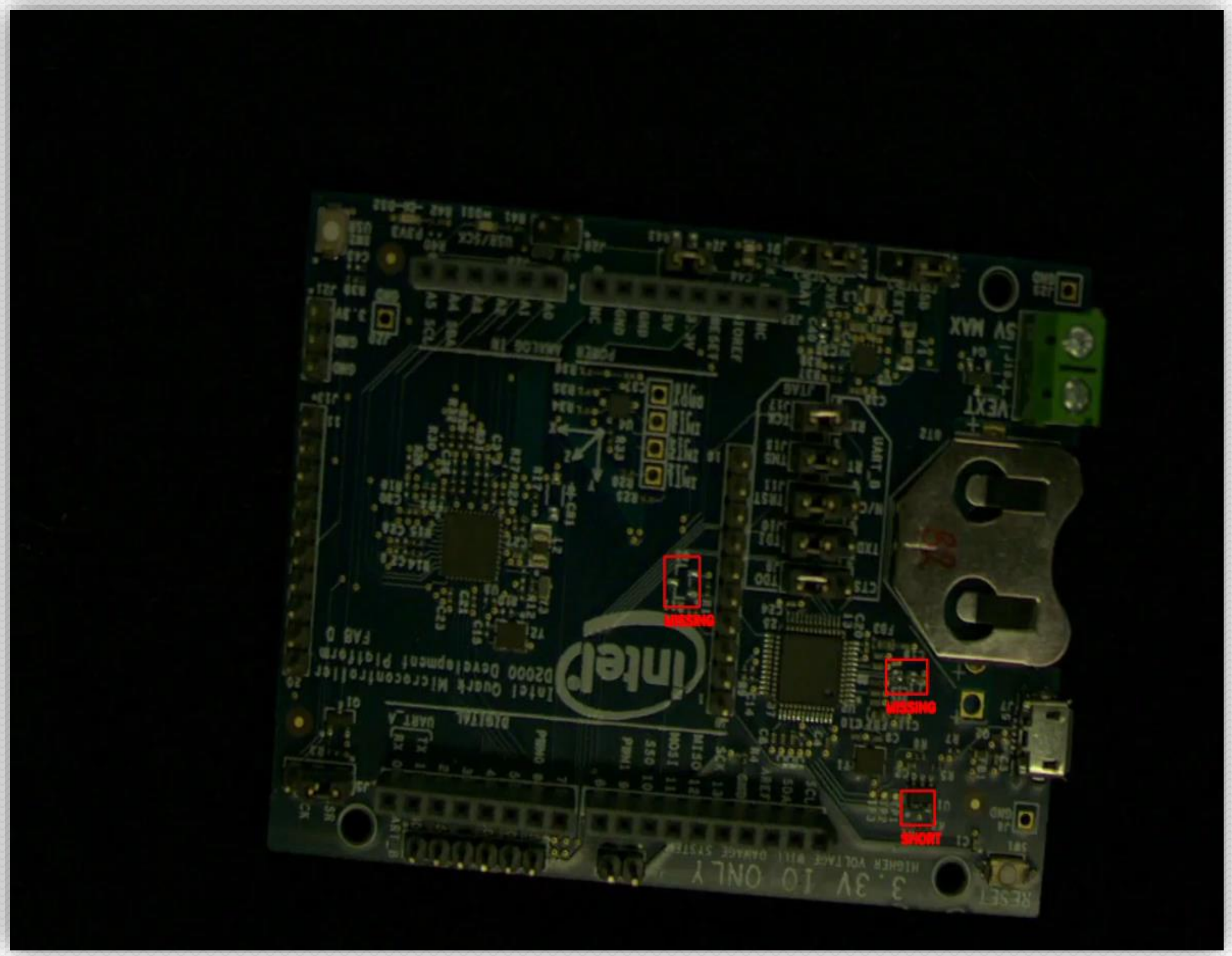
Example



Detection

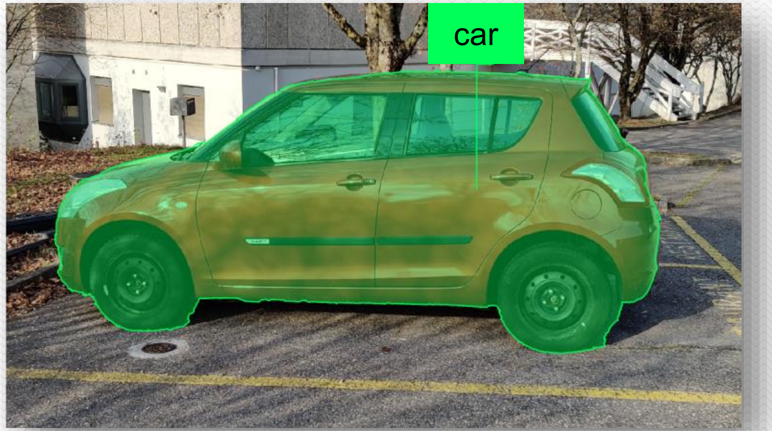
- Track that object in a video or identify it in an image

Example

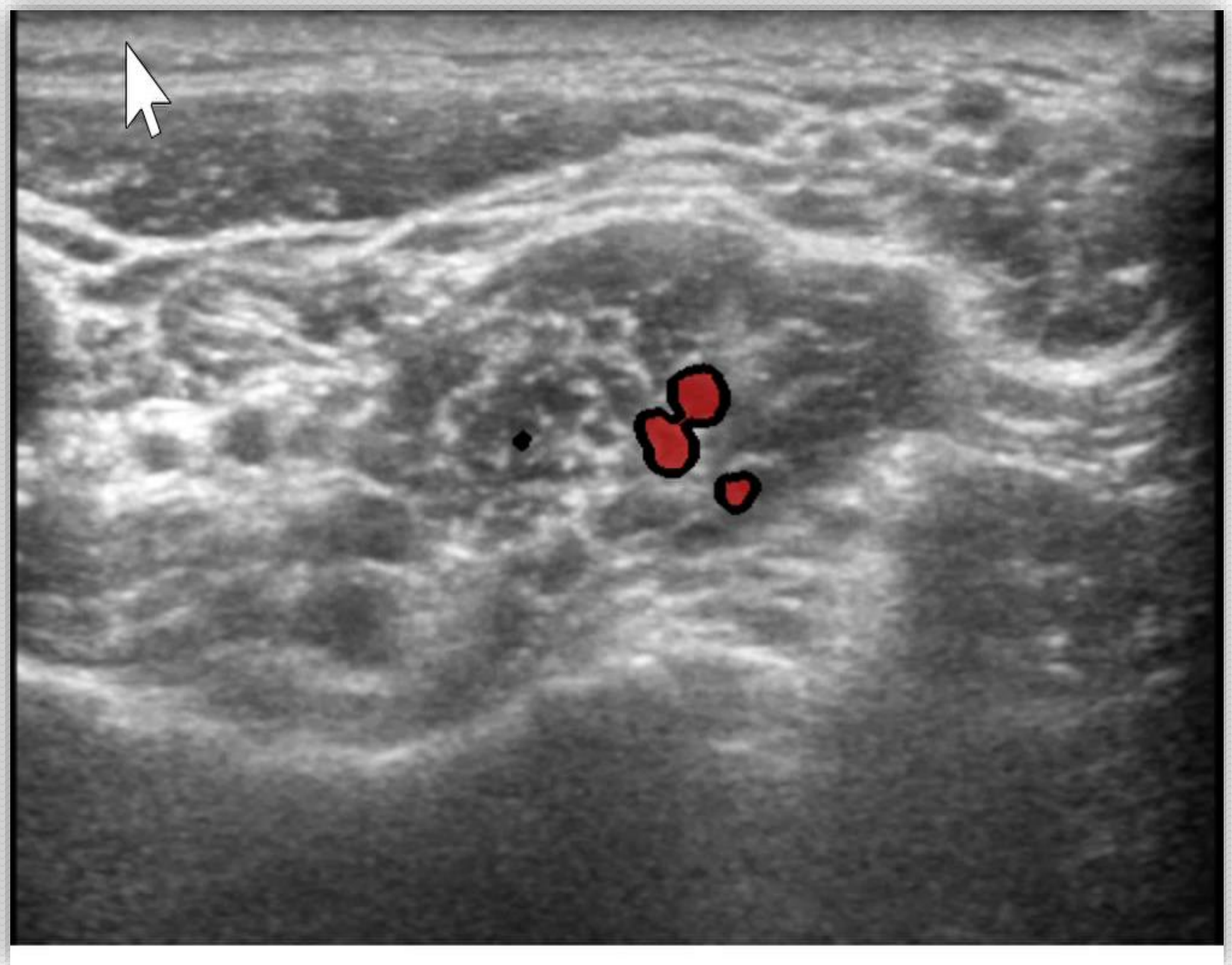


Segmentation

- Highlight the isolated region and groups every pixel in that delineated shape for later processing



Example



Classification Annotation

- Classification
  - Directory name is label name.

Car

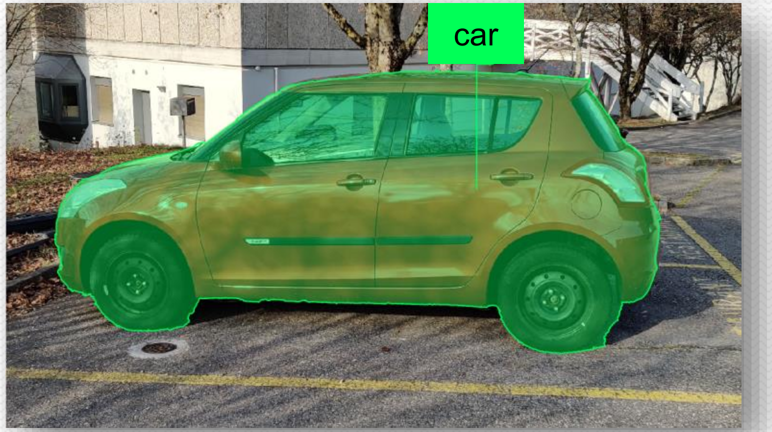
Detection / Segmentation Annotation

- Object Detection / Segmentation
  - Dedicated meta-file to contain label info

+

meta-file

XML or TXT or JSON or etc



+

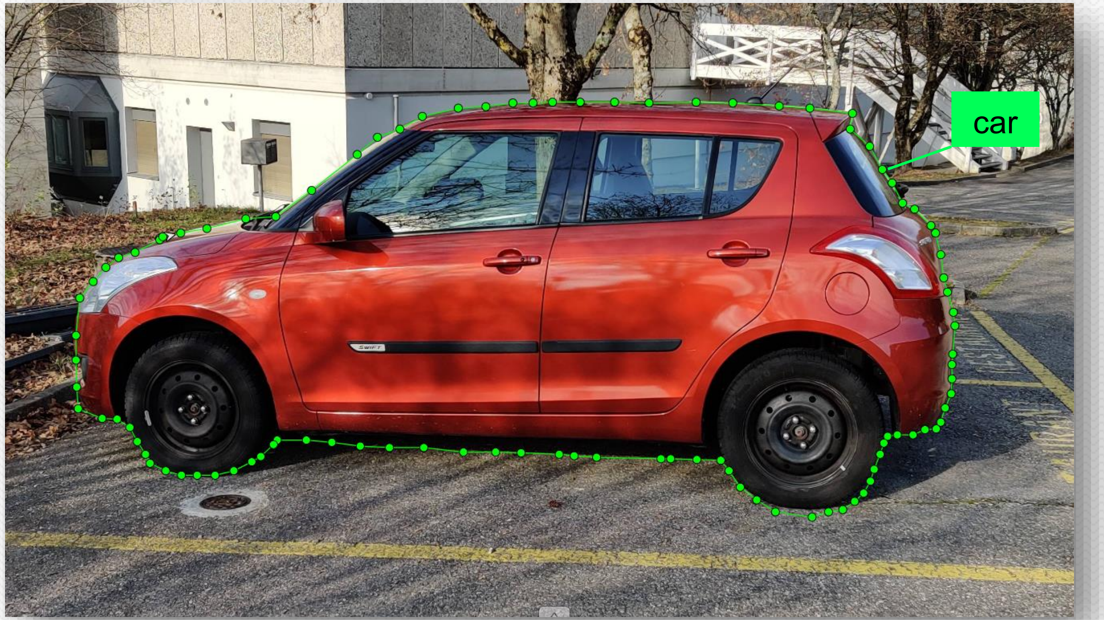
meta-file

XML or TXT or JSON or etc

Detection / Segmentation Annotation

- Example
  - Segmentation
  - COCO format

IMG_20201123_132631.png



instances.json

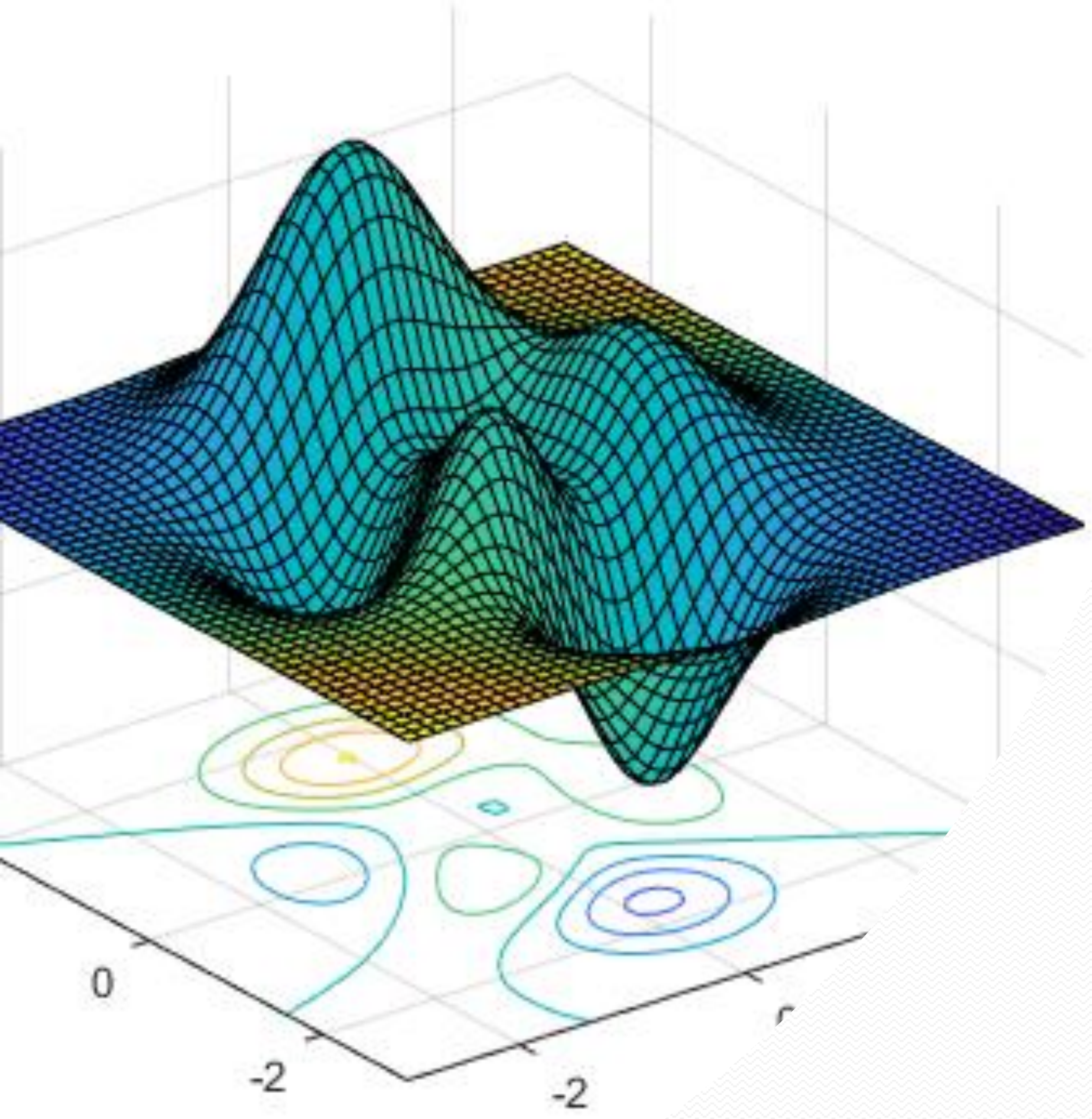

```
{
...(Snipped)...
"categories": [
  {
    "id": 1,
    "name": "human",
    "supercategory": ""
  },
  {
    "id": 2,
    "name": "car",
    "supercategory": ""
  },
],
"images": [
  {
    "id": 1,
    "width": 2666,
    "height": 1540,
    "file_name": "IMG_20201123_132631.png",
    "license": 0,
    "flickr_url": "",
    "coco_url": "",
    "date_captured": 0
  }
],
"annotations": [{
  "segmentation": [[395.29, 591.17, 447.18, 578.20, 512.04, 567.08, 608.40, 542.99, 682.53, 537.43, 771.48,
  472.57, 884.53, 383.61, 934.57, 340.99, 990.16, 320.60, 1045.76, 289.10, 1134.71, 268.71, 1203.28,
  265.01, 1269.99, 255.74, 1320.03, 255.74, 1368.21, 253.89, 1436.78, 253.89, 1522.03, 253.89, 1607.27,
  255.74, 1724.02, 253.89, 1814.83, 255.74, 1929.73, 263.16, 2005.71, 268.71, 2111.34, 281.69, 2105.78,
  320.60, 2153.96, 363.23, 2185.47, 420.68, 2209.56, 454.03, 2235.50, 504.07, 2265.16, 518.90, 2305.93,
  559.67, 2317.04, 578.20, 2330.02, 631.94, 2337.43, 689.39, 2346.70, 724.60, 2361.52, 774.64, 2365.23,
  807.99, 2359.67, 878.41, 2357.81, 904.36, 2357.81, 939.57, 2355.96, 976.63, 2341.14, 1011.84, 2330.02,
  1047.05, 2318.90, 1063.73, 2291.10, 1065.59, 2261.45, 1078.56, 2222.53, 1078.56, 2198.44,
  ...(Snipped)..., 1091.53, 686.24, 1091.53, 676.97, 1102.65, 645.47, 1132.30, 623.23, 1145.27, 578.75,
  1167.51, 532.42, 1178.63, 487.95, 1178.63, 449.03, 1178.63, 410.11, 1167.51, 371.20, 1148.98, 360.08,
  1126.74, 339.69, 1095.24, 321.16, 1060.03, 289.66, 1039.64, 252.59, 1037.79, 193.29, 1013.70, 189.58,
  959.95, 187.73, 893.24, 187.73, 832.08, 197.00, 739.42, 230.35, 698.65, 261.86, 663.44, 293.36, 639.35,
  334.13, 626.38, 395.29, 594.88, 402.70, 587.46]],
  "area": 600.4,
  "iscrowd": 1,
  "Image_id:" 2,
  "bbox": [473.05, 395.45, 38.65, 28.92],
  "category_id": 15,
  "id": 934}]
}
```


Label name

Image information

All of points info to close contour for object

* COCO (Common Objects in Context) : https://cocodataset.org/



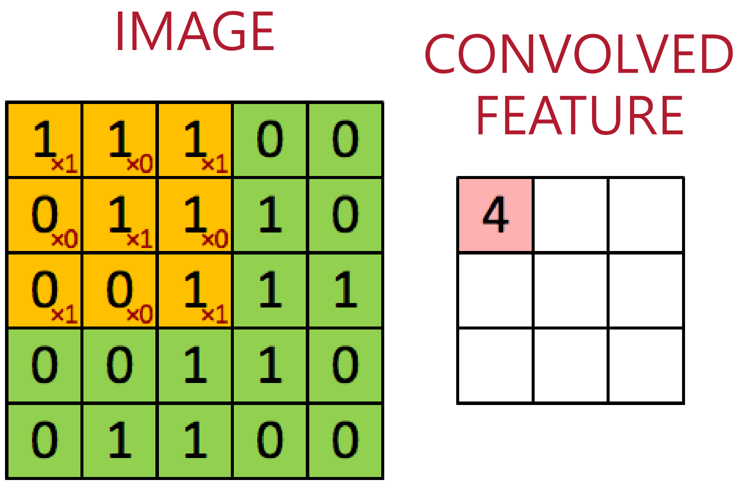
CONVOLUTION

CONVOLUTION

MATHEMATICAL OPERATION

Slides one function over another

e.g., Photo filters

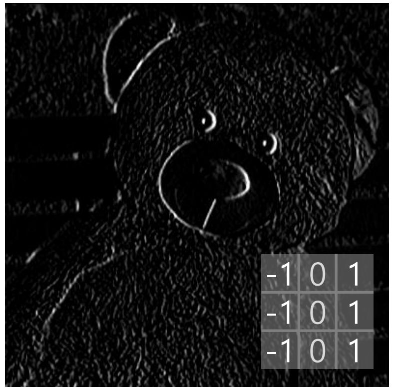


ORIGINAL

SHARPEN

EMBOSS

EXTRACT HORIZONTAL LINE

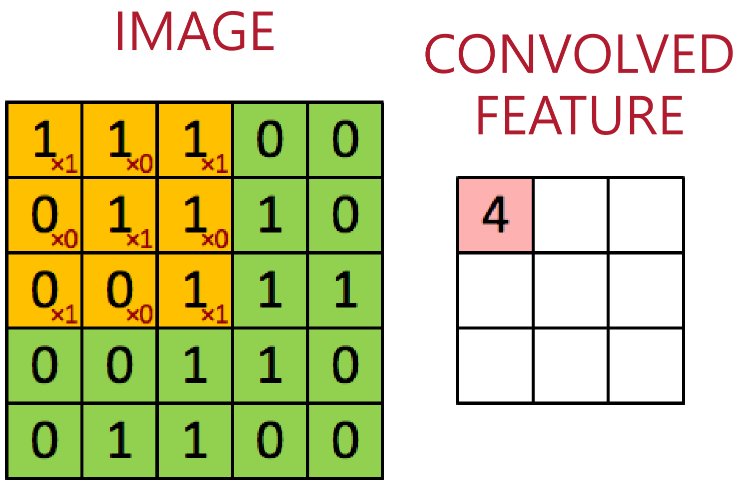


EXTRACT VERTICAL LINE

CONVOLUTION

MATHEMATICAL OPERATION

1차 미분 연산자이용한 Edge detect



Original

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Edge Detection

CONVOLUTION

```python
import cv2
import numpy as np

img = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
kernel = np.array([[1, 1, 1],[1, -8, 1], [1, 1, 1]])
print(kernel)
output = cv2.filter2D(img, -1, kernel)
cv2.imshow('edge', output)
cv2.waitKey(0)
```

Homework #1

- 아래 나열된 필터를 적용해 보고, 해당 코드를 github에 upload
- https://en.wikipedia.org/wiki/Kernel_(image_processing)

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

VIRTUAL ENVIRONMENT

- Python3 virtual environment.
  - `$mkdir intro && cd intro`
  - `$sudo apt install python3-venv`
  - `$python3 -m venv .venv`
  - `$source .venv\bin\activate`

- Download & install python modules.
  - `(.venv)$ pip install tensorflow`
  - `(.venv)$ pip install numpy`
  - `(.venv)$ pip install matplotlib`
  - `(.venv)$ pip install pyQt5==5.14`

CONTENTS

IMAGE CLASSIFICATION TRAINING WORKFLOW

- 데이터셋 준비
  - 학습용 / 검증용
- 학습 모델 형성
  - Convolution layer
  - Pooling layer
  - Dense(FC) layer
- 컴파일
- 학습 및 테스트

train.py

```
# Prepare datasets.
# Construct a model.
# Compile the model.
# Train & evaluate
# Save the model.
```

VIRTUAL ENVIRONMENT

- Python3 virtual environment.
  - `$mkdir intro && cd intro`
  - `$sudo apt install python3-venv`
  - `$python3 -m venv .venv`
  - `$source .venv\bin\activate`

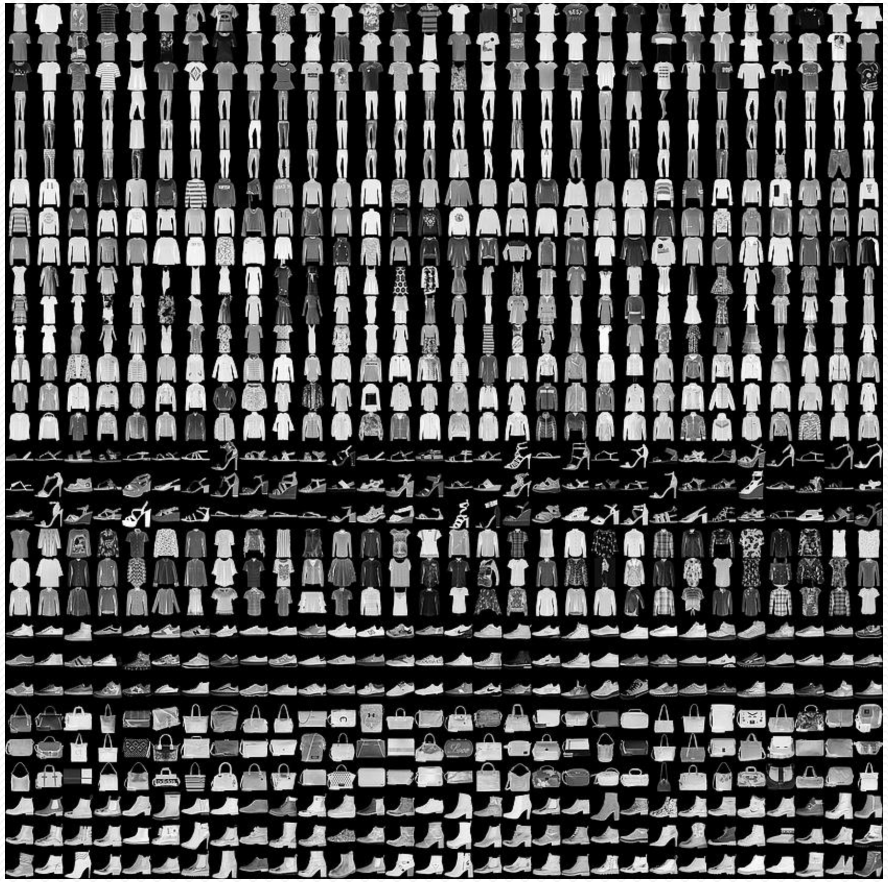
- Download & install python modules.
  - `(.venv)$ pip install tensorflow`
  - `(.venv)$ pip install numpy`
  - `(.venv)$ pip install matplotlib`
  - `(.venv)$ pip install pyQt5==5.14`

INITIAL SETTING & IMPORT MODULES

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

DATASET PREPARATION (Hand and Fashion)

```
# Model load: MNIST / Fashion MNIST Dataset
mnist = tf.keras.datasets.mnist
fashion_mnist = tf.keras.datasets.fashion_mnist
```



DATASET PREPARATION

mnist 혹은 fashion_mnist 데이터셋을 준비하고, training할 이미지셋과 test할 이미지 셋을 구분해준다.
그 후 이미지의 색상을 정규화를 시켜주기 위해 255로 색상 값을 나눈다.

```
# Model load: MNIST / Fashion MNIST Dataset
fashion_mnist = tf.keras.datasets.fashion_mnist
(f_image_train, f_label_train), (f_image_test, f_label_test) = fashion_mnist.load_data()
# normalized iamges
f_image_train, f_image_test = f_image_train / 255.0, f_image_test / 255.0
```

DATASET PREPARATION (cont'd)

fashion_mnist 의 레이블은 숫자로 저장이 되어 있기 때문에 레이블과 클래스 이름을 매핑을 해줘야 한다.

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

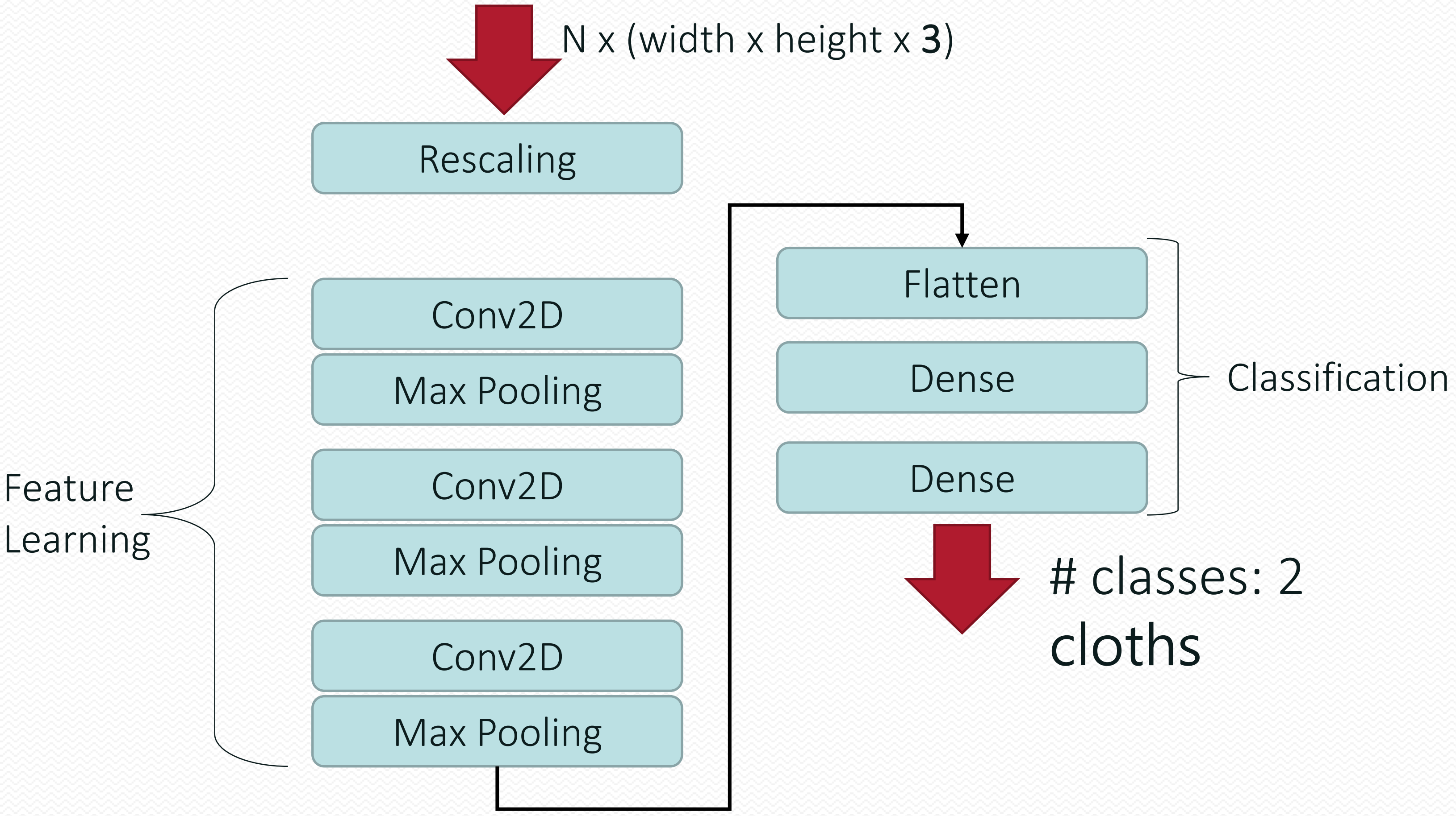
레이블	클래스
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

DATASET PREPARATION (show)

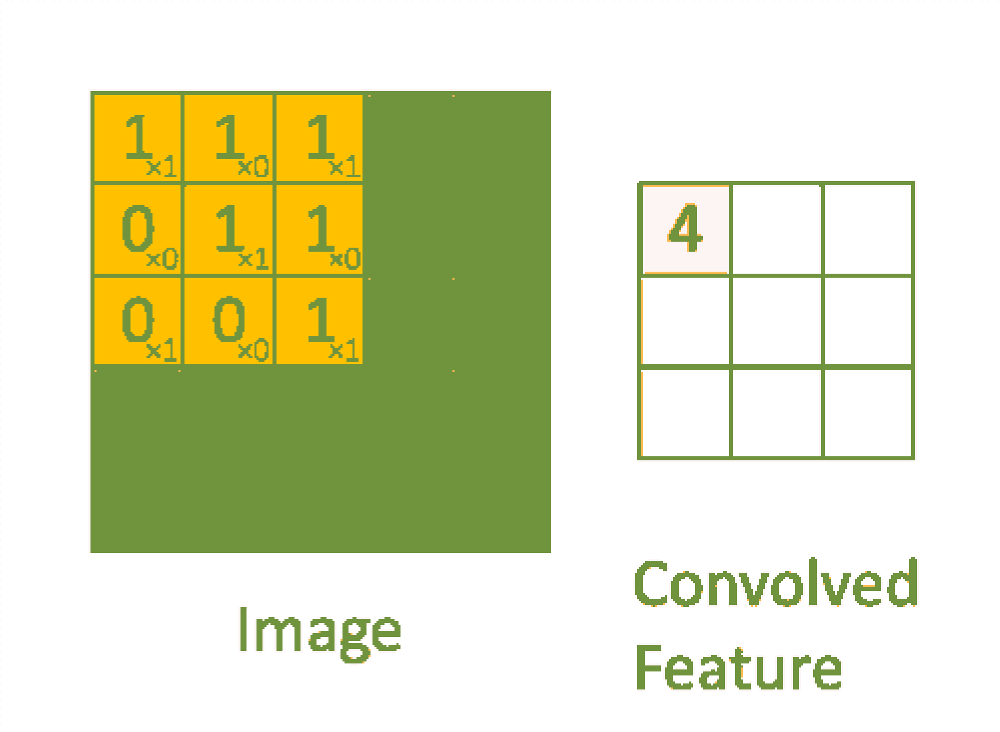
```
plt.figure(figsize=(10,10))
for i in range(10):
    plt.subplot(3,4,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(f_image_train[i])
    plt.xlabel(class_names[f_label_train[i]])
plt.show()
```

MODEL CONSTRUCTION

- [tf.keras.Sequential()]
- [tf.keras.layers]
  - Rescaling
  - Convolution
  - Pooling
  - Flatten
  - Dense(ANN/FC)



CONVOLUTION LAYER

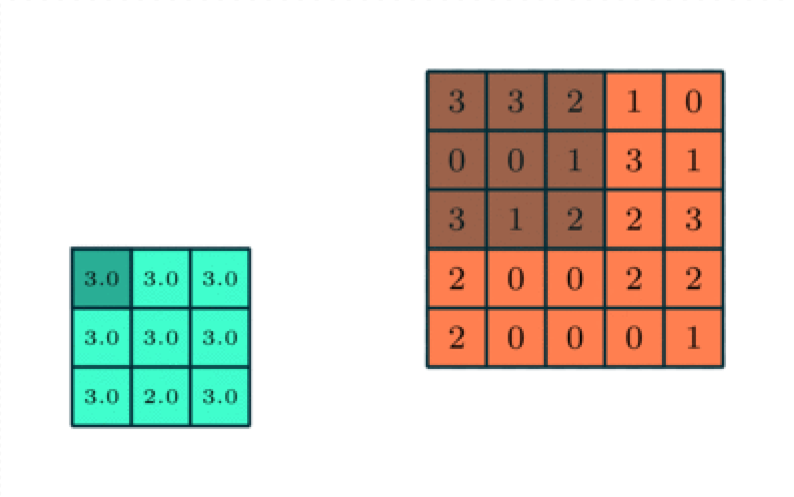


- Image: 5x5, 1 channel
- Kernel: 3x3
- Stride: 1
- Padding: 0

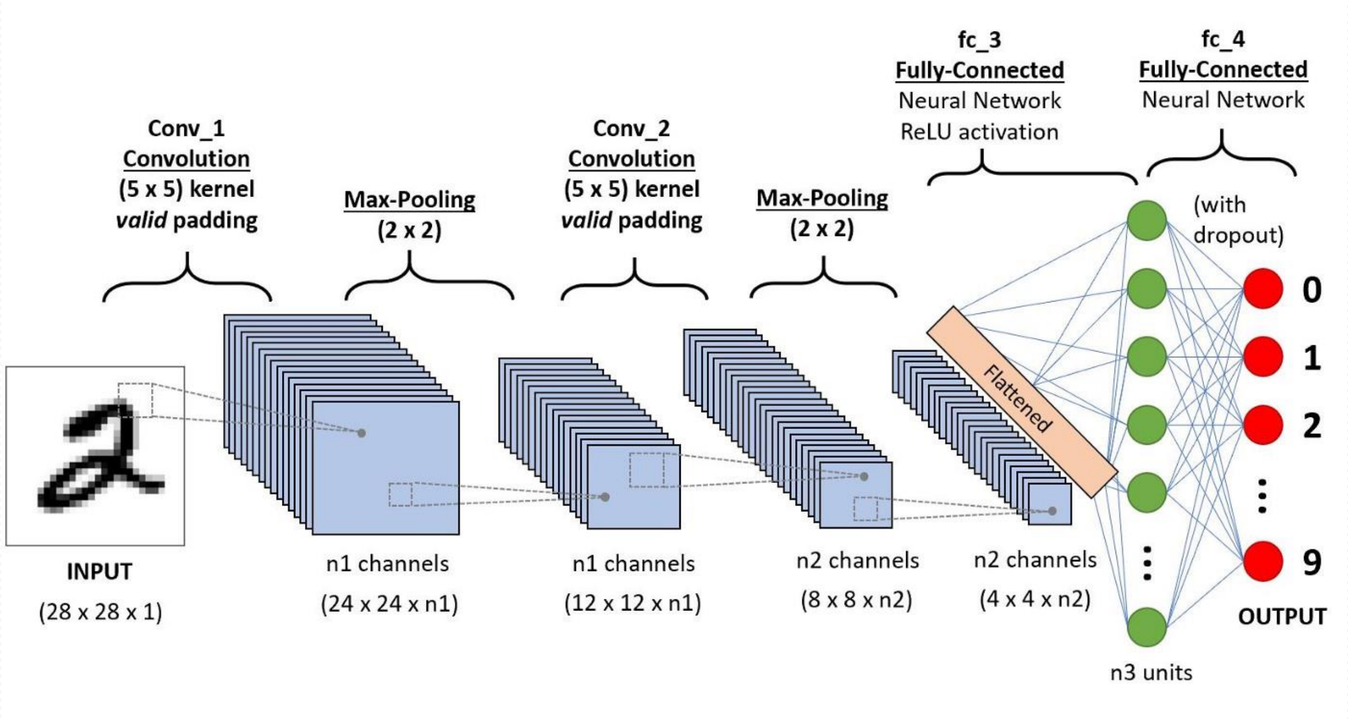
*Animations from [A Comprehensive Guide to Convolutional Neural Networks]

POOLING LAYER

- Feature map: 5x5, 1 channel
- Kernel: 3x3
- Stride: 1
- Padding: 0

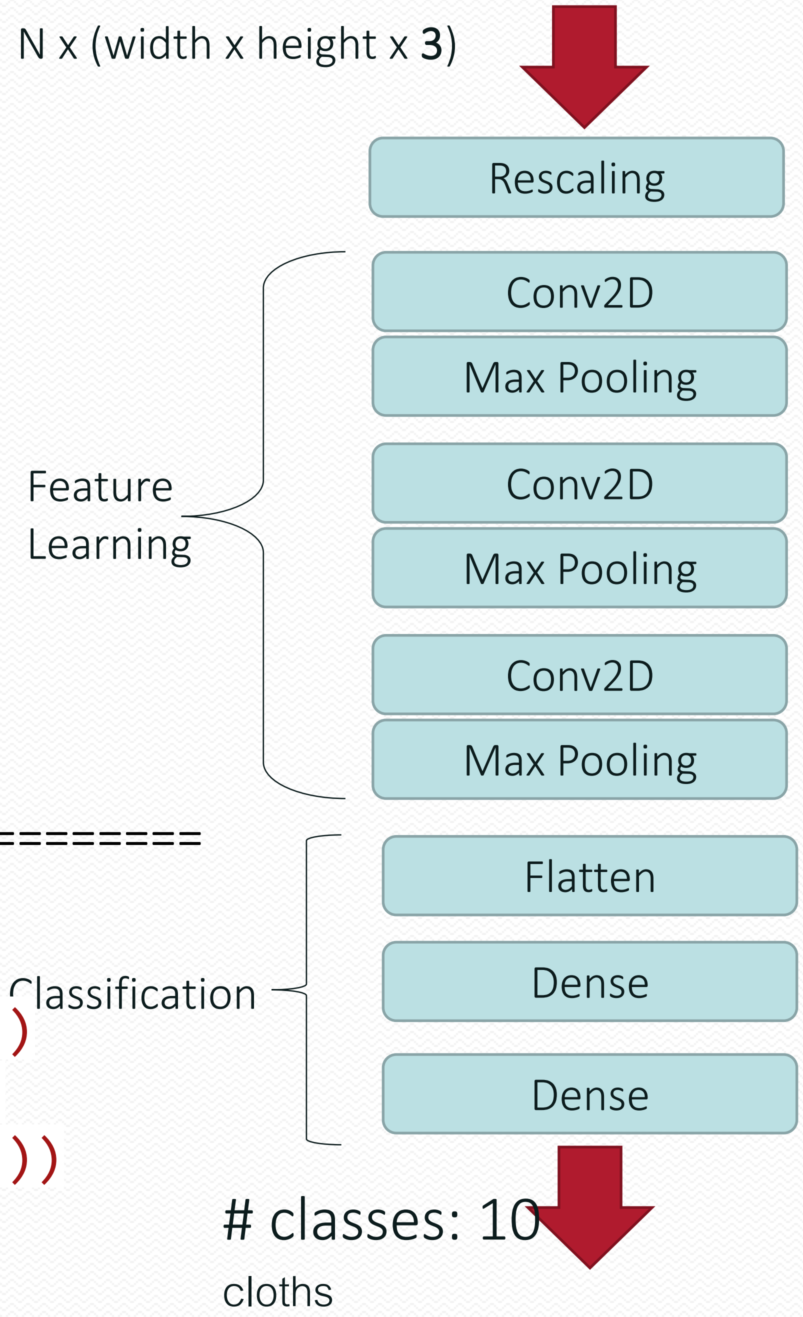


- Max pooling
- Average pooling



MODEL CONSTRUCTION

```
# CNN
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(64, (3, 3), \
  activation='relu', input_shape=(28, 28, 1)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3),
activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3),
activation='relu'))
==============================================
# ANN
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu’))
model.add(tf.keras.layers.Dense(64, activation='relu’))
model.add(tf.keras.layers.Dense(10,activation='softmax'))
```



MODEL COMPILATION

- [tf.keras.Sequential.compile()]()
  - optimizer
  - loss
  - Metrics
- [tf.model.fit()]()
  - validation_data
  - epochs
  - Hyper parameters
    - Epochs
    - Dataset size
    - Batch size
    - Optimizer / loss function
- Save model
  - [tf.keras.Model.save()]()

```
model.compile(
  optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'],
)
model.fit(image_train, label_train, epochs=10, batch_size=10)
model.summary()
model.save(fashion_mnist.h5’)
```

MODEL TRAINING

```
Epoch 1/10
250/250 [==============================] - 210s 840ms/step - loss: 0.6527 -
accuracy: 0.6428 - val_loss: 0.5764 - val_accuracy: 0.6965
Epoch 2/10
250/250 [==============================] - 222s 888ms/step - loss: 0.5223 -
accuracy: 0.7293 - val_loss: 0.5381 - val_accuracy: 0.7275
Epoch 3/10
250/250 [==============================] - 225s 898ms/step - loss: 0.4456 -
accuracy: 0.7894 - val_loss: 0.5187 - val_accuracy: 0.7500
Epoch 4/10
250/250 [==============================] - 224s 895ms/step - loss: 0.3441 -
accuracy: 0.8429 - val_loss: 0.5366 - val_accuracy: 0.7545
Epoch 5/10
250/250 [==============================] - 225s 899ms/step - loss: 0.2362 -
accuracy: 0.9046 - val_loss: 0.6528 - val_accuracy: 0.7575
Epoch 6/10
250/250 [==============================] - 238s 953ms/step - loss: 0.1323 -
accuracy: 0.9465 - val_loss: 0.8491 - val_accuracy: 0.7460
Epoch 7/10
250/250 [==============================] - 232s 927ms/step - loss: 0.0696 -
accuracy: 0.9734 - val_loss: 1.0140 - val_accuracy: 0.7570
Epoch 8/10
250/250 [==============================] - 238s 952ms/step - loss: 0.0591 -
accuracy: 0.9801 - val_loss: 1.0195 - val_accuracy: 0.7415
Epoch 9/10
250/250 [==============================] - 242s 967ms/step - loss: 0.0339 -
accuracy: 0.9893 - val_loss: 1.2181 - val_accuracy: 0.7460
Epoch 10/10
250/250 [==============================] - 248s 991ms/step - loss: 0.0341 -
accuracy: 0.9875 - val_loss: 1.3117 - val_accuracy: 0.7545
```

INFERENCE TRAINED MODELS

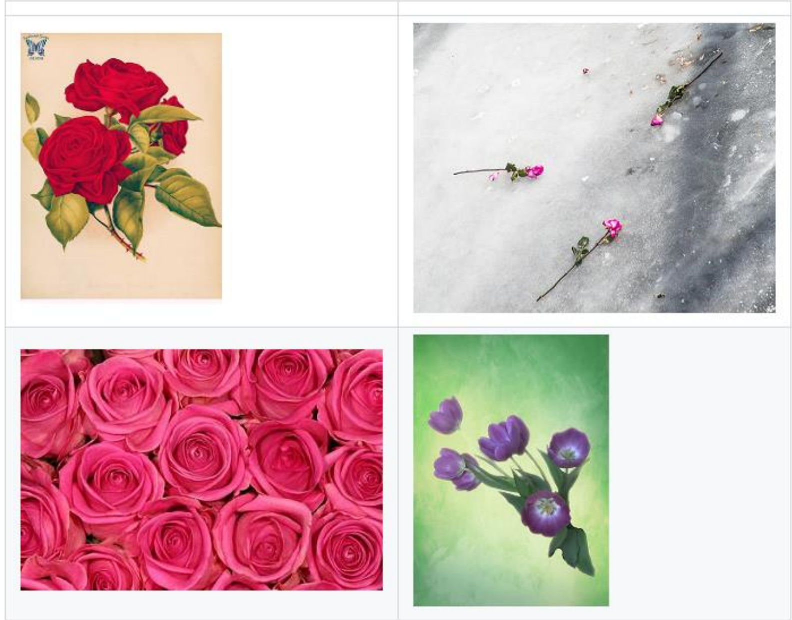
- Load model
  - tf.keras.models.load_model()

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2

model = tf.keras.models.load_model(‘./fashion_mnist.h5')
fashion_mnist = tf.keras.datasets.fashion_mnist
(f_image_train, f_label_train), (f_image_test, f_label_test) =
fashion_mnist.load_data()

f_image_train, f_image_test = f_image_train / 255.0, f_image_test / 255.0

num = 10
predict = model.predict(f_image_train[:num])
print(f_label_train[:num])
print(" * Prediction, ", np.argmax(predict, axis = 1))
```

cnn overview (openvino notebooks ex)flower classification)

[openvino_notebooks/notebooks/301-tensorflow-training-openvino at 2024.0 · openvinotoolkit/openvino_notebooks (github.com)](openvino_notebooks/notebooks/301-tensorflow-training-openvino)

Training to Deployment with TensorFlow and OpenVINO™



In this directory, you will find two Jupyter notebooks. The first is an end-to-end deep learning training tutorial which borrows the open source code from the TensorFlow image classification tutorial, demonstrating how to train the model and then convert to OpenVINO Intermediate Representation (OpenVINO IR). It leverages the tf_flowers dataset which includes about 3,700 photos of flowers.

The second notebook demonstrates how to quantize the OpenVINO IR model that was created in the first notebook. Post-training quantization speeds up inference on the trained model. The quantization is performed with the Post-training Quantization with NNCF from OpenVINO Toolkit. A custom dataloader and a metric will be defined, and accuracy and performance will be computed for the original OpenVINO IR model and the quantized model on CPU and iGPU (if available).

Transfer learning & Fine tuning

https://keras.io/guides/transfer_learning/

https://www.tensorflow.org/tutorials/images/transfer_learning?hl=ko

- **Transfer learning** consists of taking features learned on one problem, and leveraging them on a new, similar problem. For instance, features from a model that has learned to identify racoons may be useful to kick-start a model meant to identify tanukis.

- The most common incarnation of transfer learning in the context of deep learning is the following workflow:

  1. Take layers from a previously trained model.
  2. Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
  3. Add some new, trainable layers on top of the frozen layers.
     They will learn to turn the old features into predictions on a new dataset.
  4. Train the new layers on your dataset.

Transfer learning & Fine tuning

https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNetV3Small

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
import tensorflow.keras.layers as layers
from tensorflow.keras.layers import 
Dense,Flatten,BatchNormalization,Conv2D
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import numpy as np
import pickle
```

Transfer learning & Fine tuning: datasets preparing (1)

데이터 준비 튜토일얼: https://www.tensorflow.org/tutorials/load_data/images?hl=ko

Tensorflow 데이터 카탈로그: https://www.tensorflow.org/datasets/catalog/overview?hl=ko

AUTOTUNE: https://www.tensorflow.org/guide/data_performance?hl=ko

```
img_height  = 255
img_width = 255
batch_size = 32
AUTOTUNE = tf.data.AUTOTUNE # 병렬연산을 할 것인지에 대한 인자를 알아서 처리하도록.

# Dataset 준비 (https://www.tensorflow.org/tutorials/load_data/images?hl=ko)
(train_ds, val_ds, test_ds), metadata = tfds.load(
    'tf_flowers',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)
num_classes = metadata.features['label'].num_classes
label_name = metadata.features['label'].names
print(label_name, ", classnum : ", num_classes)
```

Transfer learning & Fine tuning: datasets preparing (2)

```python
def prepare(ds, shuffle=False, augment=False):
    preprocess_input = tf.keras.applications.mobilenet_v3.preprocess_input
    # Resize and rescale all datasets.
    ds = ds.map(lambda x, y: (tf.image.resize(x, [img_height, img_width]), y),
                num_parallel_calls=AUTOTUNE)
    # 전처리 적용
    ds = ds.map(lambda x, y: (preprocess_input(x), y),
                num_parallel_calls=AUTOTUNE)
    # Batch all datasets
    ds = ds.batch(batch_size)
    # Use data augmentation only on the training set.
    if augment:
        data_augmentation = tf.keras.Sequential([
            layers.RandomFlip("horizontal_and_vertical"),
            layers.RandomRotation(0.2),
        ])
        ds = ds.map(lambda x, y: (data_augmentation(x, training=True), y),
            num_parallel_calls=AUTOTUNE)
    # 데이터 로딩과 모델 학습이 병렬로 처리되기 위해
    # prefetch()를 사용해서 현재 배치가 처리되는 동안 다음 배치의 데이터를 미리 로드 하도록 함.
    return ds.prefetch(buffer_size=AUTOTUNE)
```

Transfer learning & Fine tuning: datasets preparing (3)

```
train_ds = prepare(train_ds, shuffle=True, augment=True)
val_ds = prepare(val_ds)
test_ds = prepare(test_ds)
```

Transfer learning & Fine tuning: model build

https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNetV3Small

https://www.tensorflow.org/tutorials/images/transfer_learning?hl=ko

```
# include_top -> ANN 부분 직접 수정
base_model = tf.keras.applications.MobileNetV3Small(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape = (img_height, img_width, 3),
    include_top = False)
# 기본 모델의 가중치 동결
base_model.trainable = False

inputs = tf.keras.Input(shape=(img_height, img_width, 3))
# 추론, 학습에서 다르게 동작하는 layer들을 추론/학습 중 하나로만
동작하게 함.
x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = tf.keras.layers.Dense(num_classes,
activation='softmax')(x)
model = tf.keras.Model(inputs, outputs)
```

Transfer learning & Fine tuning: model compile & training

```
model.summary()
model.compile(optimizer = 'adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(train_ds, epochs=15, validation_data=val_ds)
model.save('transfer_learning_flower.keras')
with open('history_flower', 'wb') as file_pi:
    pickle.dump(history.history, file_pi)
```

Transfer learning & Fine tuning: Inference (1)

```
import tensorflow as tf
# Helper libraires
import numpy as np
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds

img_height  = 255
img_width = 255
batch_size = 32
AUTOTUNE = tf.data.AUTOTUNE # 병렬연산을 할 것인지에 대한 인자를
알아서 처리하도록.

# Dataset 준비
(https://www.tensorflow.org/tutorials/load_data/images?hl=ko)
(train_ds, val_ds, test_ds), metadata = tfds.load(
    'tf_flowers',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)
```

Transfer learning & Fine tuning: Inference (2)

```
num = 20
def prepare(ds, batch = 1,  shuffle=False, augment=False):
    preprocess_input = tf.keras.applications.mobilenet_v3.preprocess_input
    # Resize and rescale all datasets.
    # x: image, y: label

    # 이미지 크기 조정
    ds = ds.map(lambda x, y: (tf.image.resize(x, [img_height, img_width]), y),
                num_parallel_calls=AUTOTUNE)

    # Batch all datasets
    ds = ds.batch(batch_size)

    # 데이터 로딩과 모델 학습이 병렬로 처리되기 위해
    # prefetch()를 사용해서 현재 배치가 처리되는 동안 다음 배치의 데이터를 미리 로드
하도록 함.
    return ds.prefetch(buffer_size=AUTOTUNE)
```

Transfer learning & Fine tuning: Inference (3)

```
num_classes = metadata.features['label'].num_classes
label_name = metadata.features['label'].names
print(label_name, ", classnum : ", num_classes, ", type: ", type(label_name))

test_ds = prepare(test_ds, num)
image_test, label_test = next(iter(test_ds))
image_test = np.array(image_test)
label_test = np.array(label_test, dtype='int')

# 모델 불러오기
model = tf.keras.models.load_model('transfer_learning_flower.keras')
model.summary()

predict = model.predict(image_test)
predicted_classes = np.argmax(predict, axis=1)
```

Transfer learning & Fine tuning: Inference (4)

```
print("실제 레이블 | 예측 레이블");
print("------------------------")
for ll in range((label_test.size)):
    print(label_name[label_test[ll]], "|",
label_name[predicted_classes[ll]])
print("------------------------")
# print("실제 레이블:", [label_name[idx] for idx in label_test])
# print("예측 레이블:", [label_name[idx] for idx in predicted_classes])

accuracy = np.mean(predicted_classes == label_test)
print(f"정확도: {accuracy:.2%}")
```

THANK YOU