KERAS

Software Analysis

01076024 SOFTWARE ARCHITECTURE AND DESIGN

TABLE OF CONTENTS

ABOUT THE SOFTWARE

Keras คือ Open Source Deep learning API สำหรับการพัฒนา deep learning model ที่เขียนขึ้นในภาษา Python โดยทำงานอยู่บน libraries ที่มีชื่อว่า Tensorflow และ Theano

theano

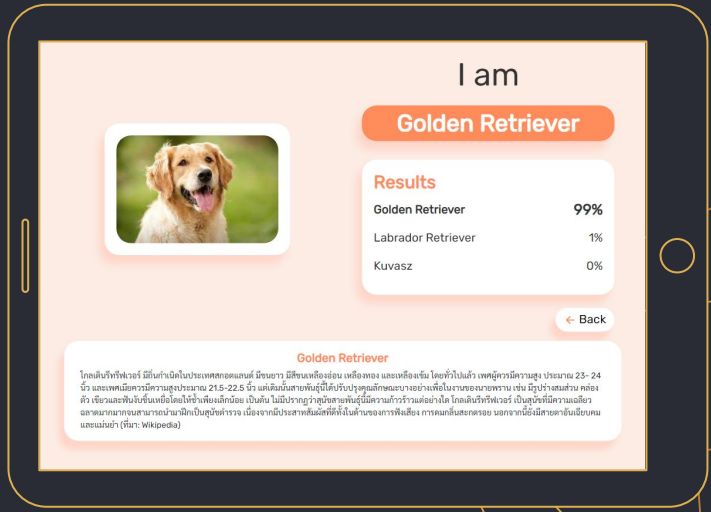
Keras

https://www.upgrad.com/blog/how-netflix-uses-machine-learning/

ABOUT THE SOFTWARE

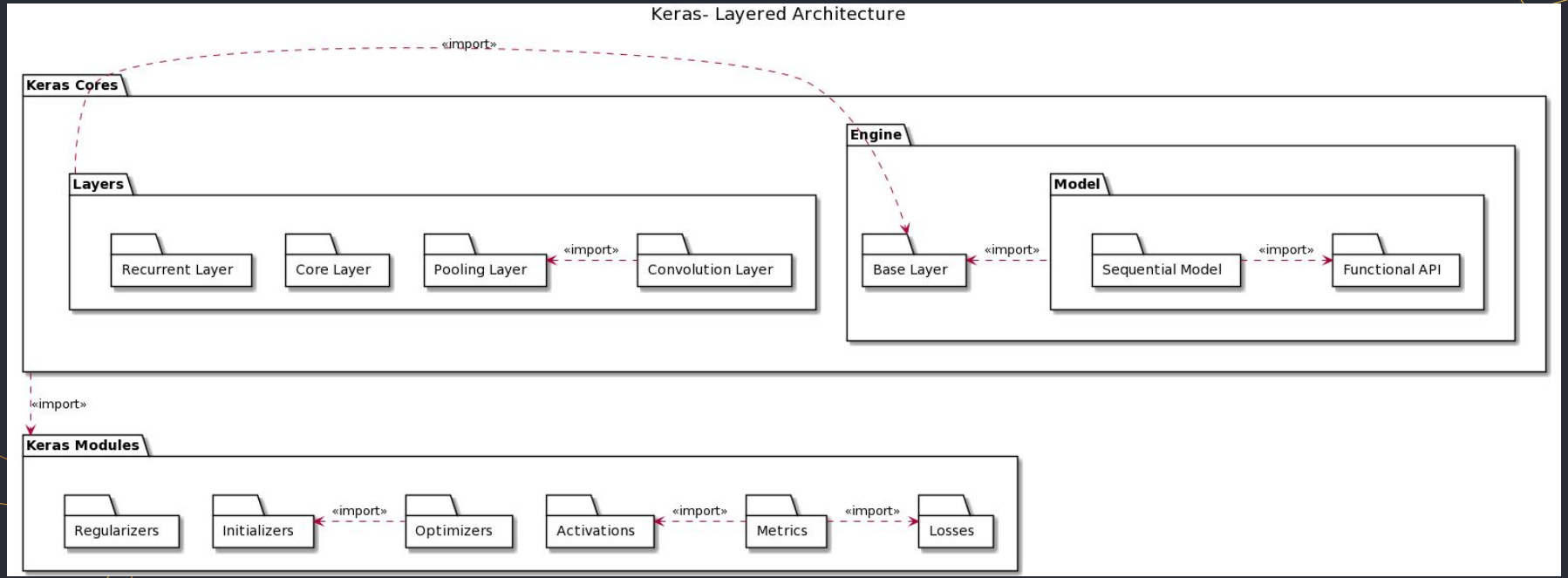
ในการนำ Keras ไปประยุกต์ใช้งาน สามารถทำได้ด้วยการนำผลลัพธ์ของการ train deep learning model ไปใช้ในการต่อยอดทำแอปพลิเคชั่นต่างๆได้ เช่น

- สร้างเว็บแอปพลิเคชั่นสำหรับการจำแนกประเภทสิ่งต่างๆ เช่น คน สัตว์ สิ่งของ
- สร้าง feature ย่อยๆสำหรับแอปพลิเคชัน โดยแอปพลิเคชั่นชื่อดังที่มีการใช้งานสามารถยกตัวอย่างได้เช่น Netflix, Uber, Yelp, Instacart, Zocdoc, Square



ตัวอย่างเว็บแอป : เว็บแอปพลิเคชั่นทำนายสายพันธุ์สุนัข

ARCHITECTURAL STYLES: LAYERED



QUALITY ATTRIBUTES: PROS

PERFORMANCE

เพิ่มความเร็วในการ train model โดยใช้วิธีการ mixed precision training

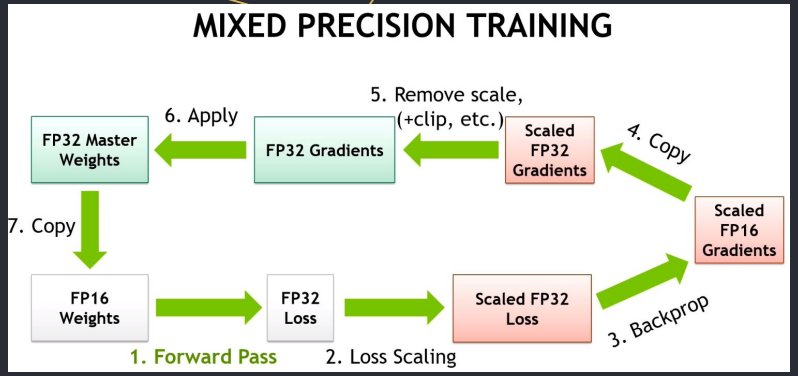
SCALABILITY

รองรับการเพิ่มหรือลดอุปกรณ์ประมวลผล เช่น CPU, GPU ในการ train model

PORTABILITY

สามารถ deploy ได้บนหลากหลาย platform

QUALITY ATTRIBUTES: PERFORMANCE



Why?

เพราะ Keras เป็น API ที่ใช้ในงาน deep learning ดังนั้นจึงมีความต้องการให้การ train model ใช้ระยะเวลาสั้นลงและสามารถทำงานได้อย่างมีประสิทธิภาพ

How?

Keras มีเทคนิคที่ช่วยเพิ่มความเร็วในการ train model โดยใช้วิธีการ mixed precision training ซึ่งเป็นการใช้ floating-point ชนิด 16 bits และ 32 bits ผสมกัน โดยในระหว่างการ train จะมีการเปลี่ยนชนิดข้อมูล จาก float32 เป็น float16 หรือ bfloat16 ส่งผลให้มีการใช้ Memory น้อยลงทำให้ประสิทธิภาพสูงขึ้น 3 เท่าบน GPUs รุ่นใหม่ และสูงขึ้น 60% บน TPUs

Src: https://keras.io/api/mixed_precision/

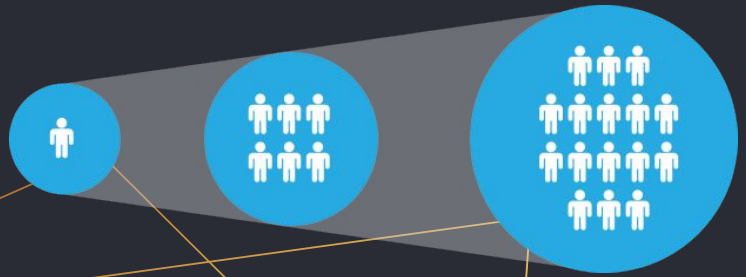
QUALITY ATTRIBUTES: SCALABILITY

Why?

เพราะ Keras เป็น API ที่มีการใช้งานในกลุ่มผู้ใช้งานที่หลากหลาย ตั้งแต่ระดับผู้ใช้งานทั่วไปจนถึงระดับอุตสาหกรรม หรืองานวิจัย ดังนั้นจึงต้องมีการออกแบบให้รองรับอุปกรณ์ประมวลผลตั้งแต่จำนวนน้อยๆไปจนถึงอุปกรณ์ประมวลผลจำนวนมากถึงหลักหนึ่งพันตัว

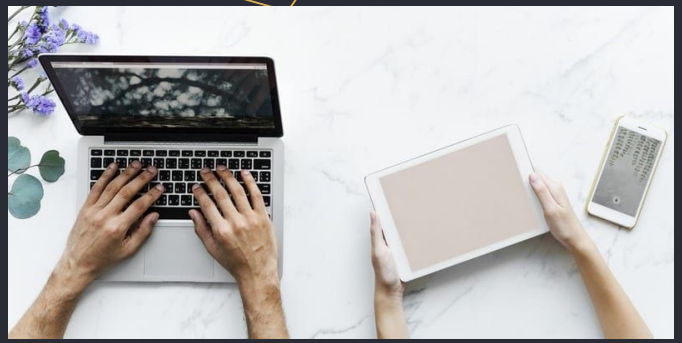
How?

Keras มีการรองรับการเพิ่มหรือลดอุปกรณ์ประมวลผล เช่น GPU หรือ TPU ในการ train model ได้ด้วยวิธีการทำ Multi-GPU and distributed training โดยจะมี 2 รูปแบบคือ Data parallelism และ Model parallelism



Src: https://keras.io/why_keras/#keras-has-strong-multigpu-amp-distributed-training-support

QUALITY ATTRIBUTES: PORTABILITY



Why?

เพราะในปัจจุบันมีผู้ใช้งานบริการที่เกี่ยวกับ AI บนหลากหลาย platform เช่น Web Browser, Application บน Smartphone และ Embedded system

How?

Keras ถูกออกแบบให้ model ที่เป็นผลลัพธ์ของการ train สามารถนำไปใช้งานได้บนหลากหลาย platform เช่น Android, iOS, WebServer โดยสามารถใช้วิธี save model ที่ train แล้วจากนั้นก็ให้ service ของแต่ละ platform นั้นเรียกใช้ API ได้

Src: https://keras.io/why_keras/#keras-makes-it-easy-to-turn-models-into-products

QUALITY ATTRIBUTES: CONS

TESTABILITY

Debug และ Test ยากเมื่อเกิดข้อผิดพลาดขึ้น

Problem

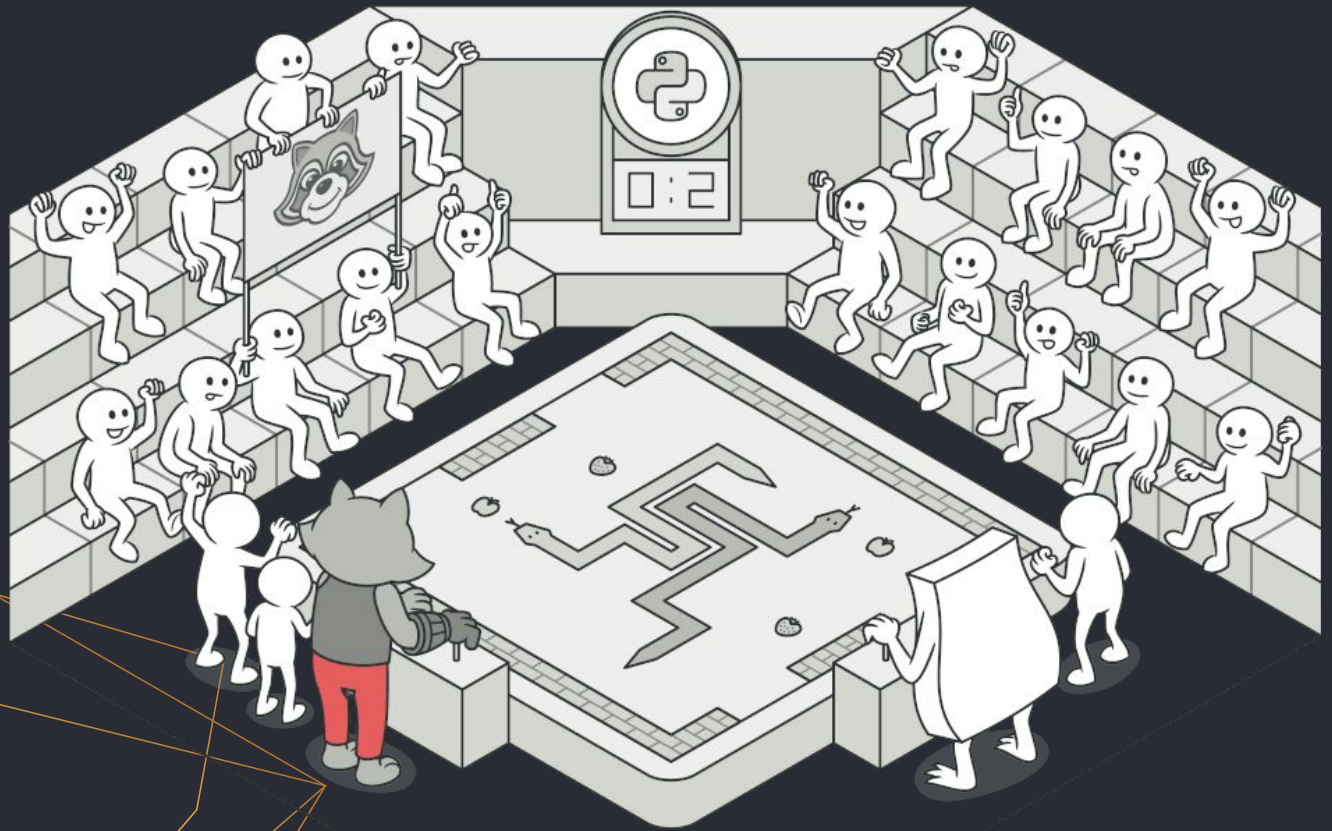
ในมุมมองของผู้ใช้งานนั้น การทำ Debug และการ Test นั้นเป็นไปได้ด้วยความยากลำบากเมื่อเกิดข้อผิดพลาดขึ้น เนื่องจาก keras มีการทำงานในลักษณะของ High-Level API ซึ่งอาจทำให้เราหาสาเหตุและต้นตอของปัญหาได้ยาก

Suggestion

ควรมีการเพิ่ม assertions หรือ raising exceptions ใน source code ให้มากขึ้นและมีความละเอียดยิ่งขึ้นเพื่อใช้ในการแสดง error ให้แก่ผู้ใช้งานเพื่อทำให้รู้ถึงต้นตอของปัญหาได้ง่ายยิ่งขึ้น

Src: https://analyticsindiamag.com/pytorch-vs-keras-who-suits-you-the-best/

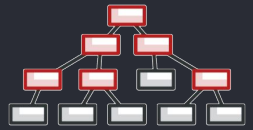
DESIGN PATTERNS



DESIGN PATTERNS

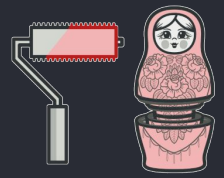
ADAPTER

- DataAdapter
- DatasetCreator



COMPOSITE

- Sequential
- Model
- Layer



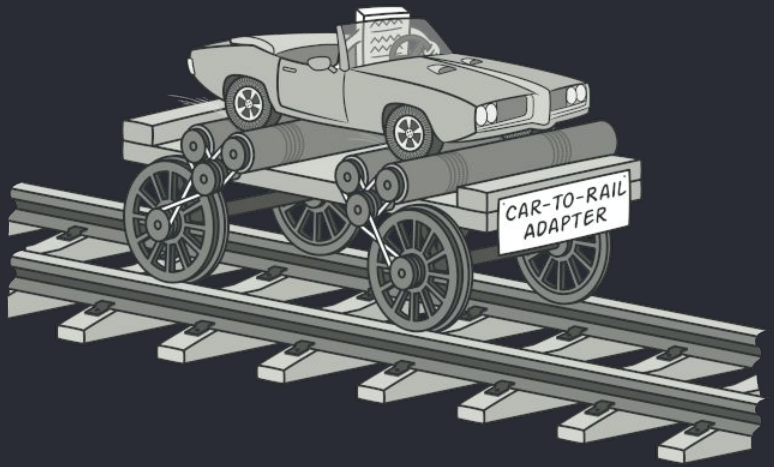
DECORATOR

- Layer
- Wrapper
- Bidirectional
- TimeDistributed

ITERATOR

- Iterator
- DirectoryIterator
- NumpyArrayIterator
- DataFrameIterator

ADAPTER



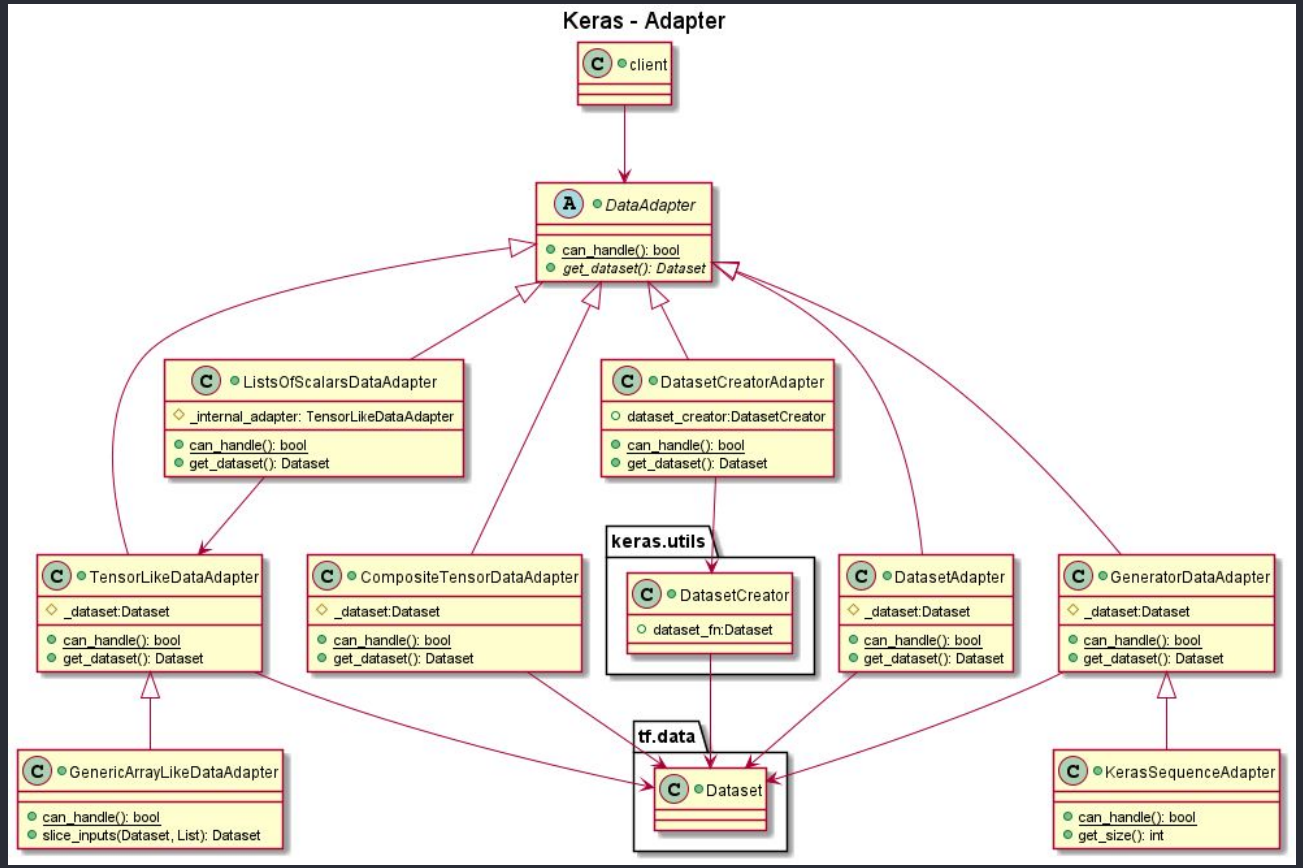
Adapter คือ Design Pattern ที่ทำหน้าที่แปลง object ให้สามารถทำงานร่วมกับตัวอื่นได้

ซึ่งผู้พัฒนามีความต้องการให้ชนิดข้อมูลที่หลากหลาย เช่น python list, numpy array, generator และอื่นๆ สามารถแปลงเป็น tf.data.Dataset ที่สามารถใช้ทำงานกับ Keras ได้อย่างถูกต้อง

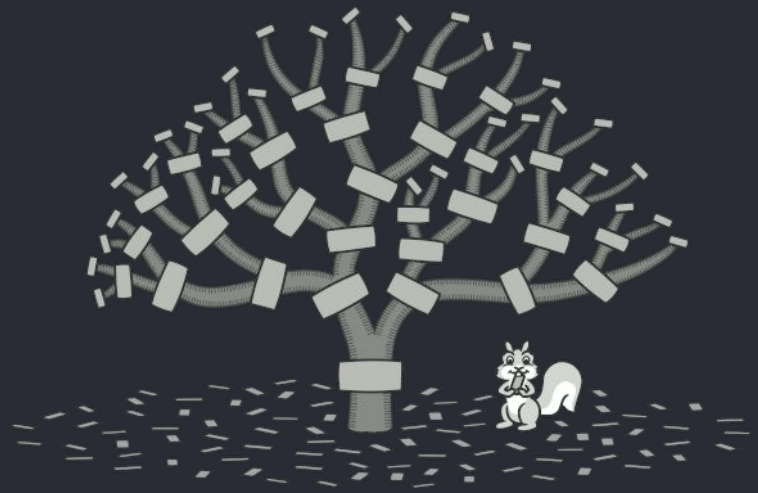
เป็นไปตามหลักการ SOLID Principle

- [x] Single-responsibility principle
- [x] Open–closed principle
- [ ] Liskov substitution principle
- [ ] Interface segregation principle
- [ ] Dependency inversion principle

ADAPTER


Keras - Adapter
client
DataAdapter
can_handle(): bool
get_dataset(): Dataset
ListsOfScalarsDataAdapter
_internal_adapter: TensorLikeDataAdapter
can_handle(): bool
get_dataset(): Dataset
DatasetCreatorAdapter
dataset_creator:DatasetCreator
can_handle(): bool
get_dataset(): Dataset
keras.utils
DatasetCreator
dataset_fn:Dataset
TensorLikeDataAdapter
_dataset:Dataset
can_handle(): bool
get_dataset(): Dataset
CompositeTensorDataAdapter
_dataset:Dataset
can_handle(): bool
get_dataset(): Dataset
DatasetAdapter
_dataset:Dataset
can_handle(): bool
get_dataset(): Dataset
GeneratorDataAdapter
_dataset:Dataset
can_handle(): bool
get_dataset(): Dataset
tf.data
Dataset
GenericArrayLikeDataAdapter
can_handle(): bool
slice_inputs(Dataset, List): Dataset
KerasSequenceAdapter
can_handle(): bool
get_size(): int

COMPOSITE



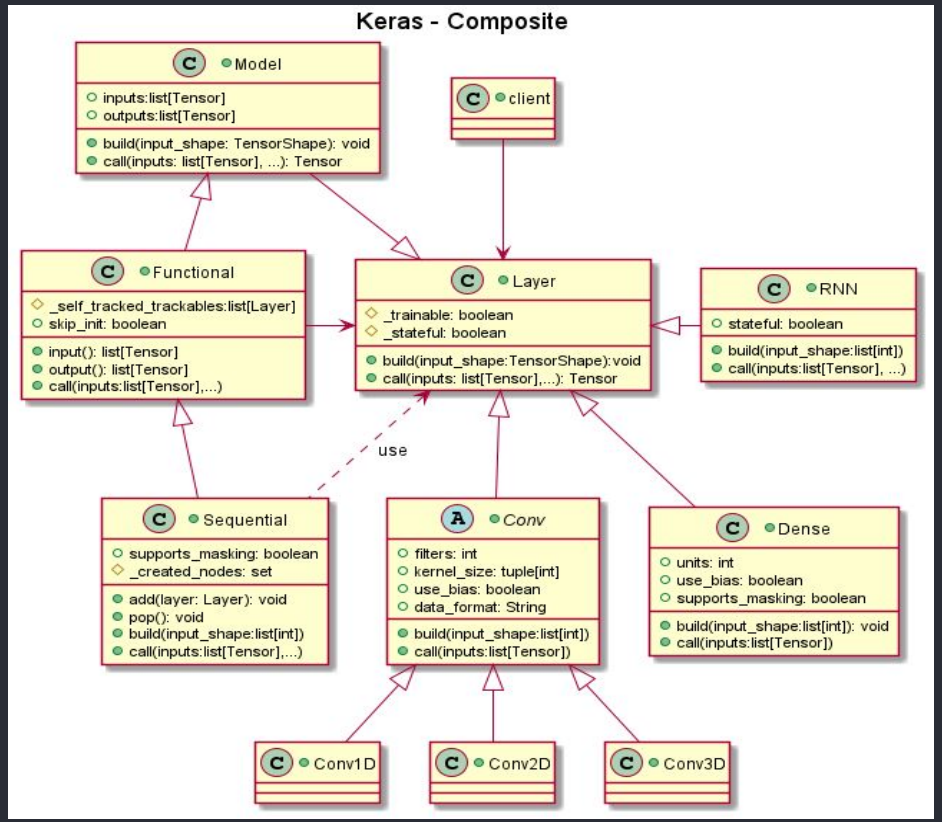
Composite คือ Design Pattern ที่เก็บ object เป็นโครงสร้างแบบ tree และใช้งานเป็นในลักษณะ recursive เป็นทอด ๆ เข้าไปใน object

ซึ่งผู้พัฒนามีความต้องการให้ instance ของ Sequential สามารถเก็บ instance ของ Sequential ด้วยกันเอง หรือ เก็บ instance ของ Layer ได้ ทำให้ง่ายต่อ client code

เป็นไปตามหลักการ SOLID Principle

- Single-responsibility principle
- ☑ Open–closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle

COMPOSITE


Keras - Composite
Model
inputs:list[Tensor]
outputs:list[Tensor]
build(input_shape: TensorShape): void
call(inputs: list[Tensor], ...): Tensor
client
Functional
_self_tracked_trackables:list[Layer]
skip_init: boolean
input(): list[Tensor]
output(): list[Tensor]
call(inputs:list[Tensor],...)
Layer
_trainable: boolean
_stateful: boolean
build(input_shape:TensorShape):void
call(inputs: list[Tensor],...): Tensor
RNN
stateful: boolean
build(input_shape:list[int])
call(inputs:list[Tensor], ...)
use
Sequential
supports_masking: boolean
_created_nodes: set
add(layer: Layer): void
pop(): void
build(input_shape:list[int])
call(inputs:list[Tensor],...)
Conv
filters: int
kernel_size: tuple[int]
use_bias: boolean
data_format: String
build(input_shape:list[int])
call(inputs:list[Tensor])
Dense
units: int
use_bias: boolean
supports_masking: boolean
build(input_shape:list[int]): void
call(inputs:list[Tensor])
Conv1D
Conv2D
Conv3D

DECORATOR

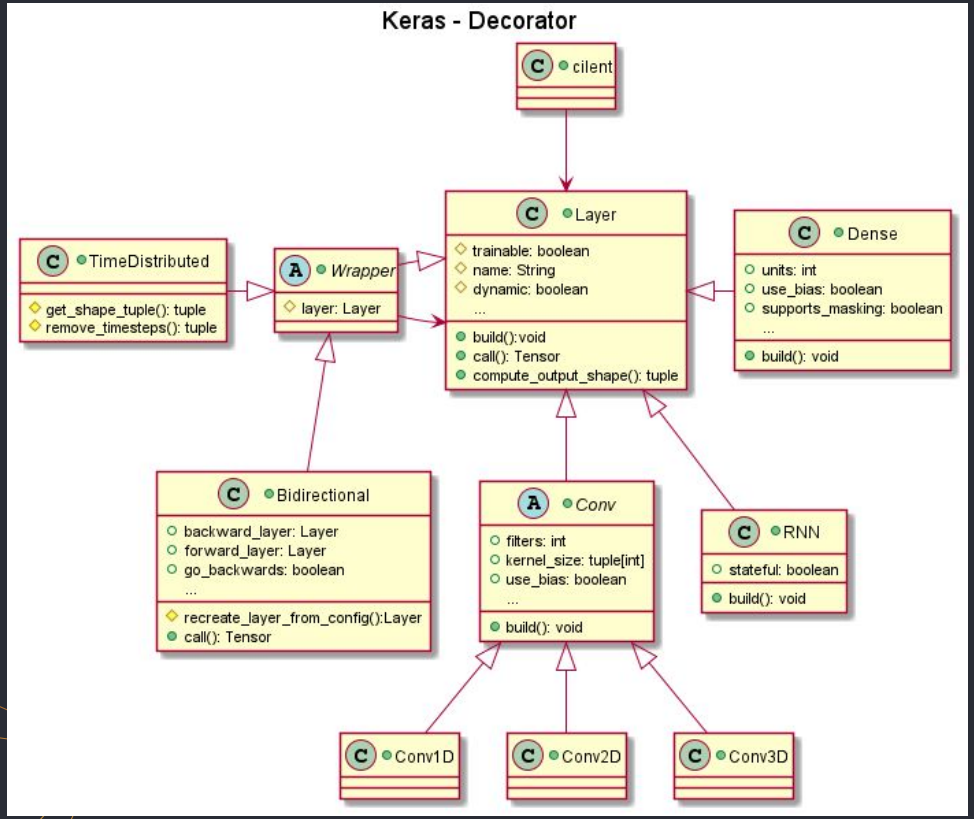
Decorator คือ Design Pattern ที่สามารถเพิ่มและลดความสามารถของ object ขณะ runtime ได้

ซึ่งผู้พัฒนามีความต้องการให้ Layer สามารถเพิ่มความสามารถได้ไปในทิศทางใดทิศทางหนึ่ง เช่น ความสามารถในการทำให้ input ที่รับมาสามารถแบ่งเป็นช่วงของเวลาได้ หรือ ความสามารถในการทำให้ Layer สามารถที่จะส่งค่าได้ทั้งสองทิศทาง

เป็นไปตามหลักการ SOLID Principle

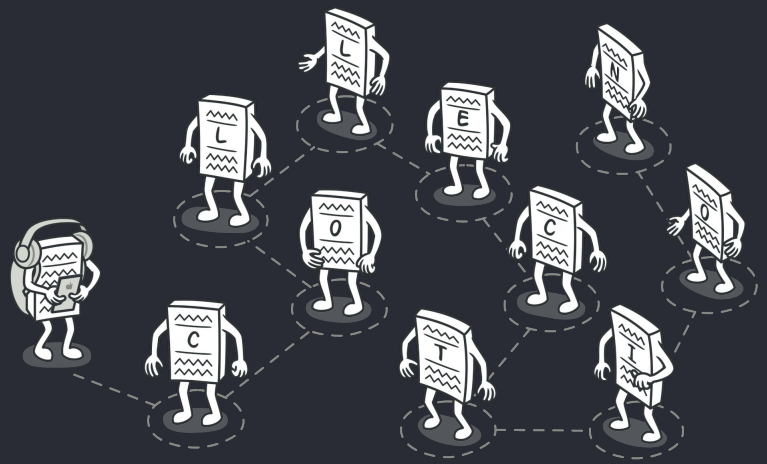
- [x] Single-responsibility principle
- [ ] Open–closed principle
- [ ] Liskov substitution principle
- [ ] Interface segregation principle
- [ ] Dependency inversion principle

DECORATOR




```
inputs = tf.keras.Input(shape=(10, 128, 128, 3))
conv_2d_layer = tf.keras.layers.Conv2D(64, (3, 3))
outputs = tf.keras.layers.TimeDistributed(conv_2d_layer)(inputs)
outputs.shape
TensorShape([None, 10, 126, 126, 64])
```

ITERATOR



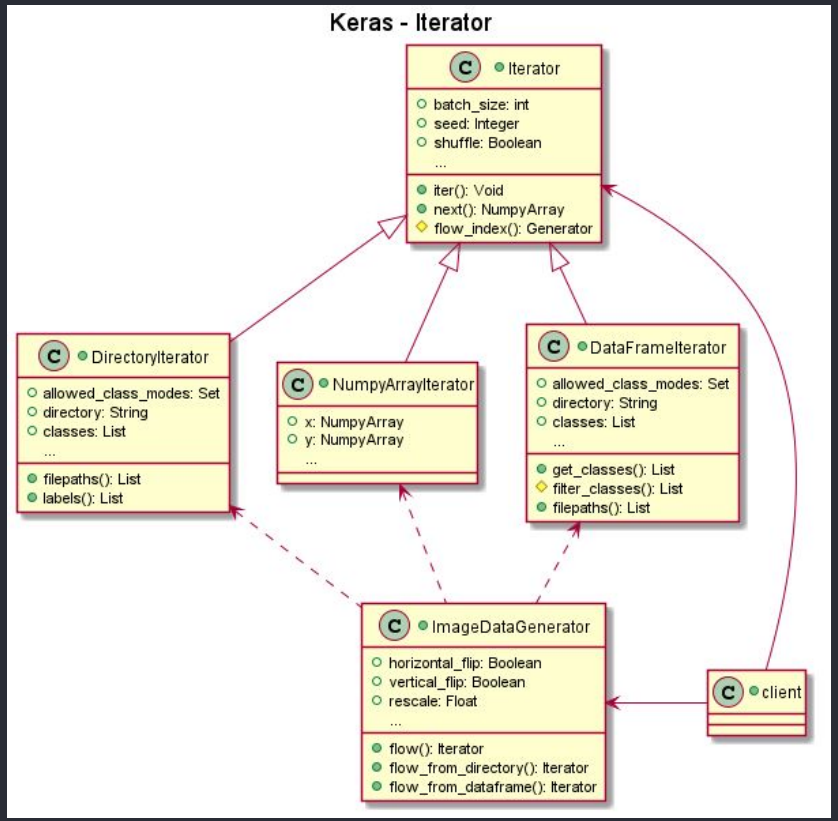
Iterator คือ Design Pattern ที่ทำงานกับ collections ต่างๆโดยซ่อนความซับซ้อนภายใน collection นั้น

ซึ่งผู้พัฒนามีความต้องการให้ client code ภายใน keras สามารถเข้าถึงข้อมูลที่ถูกเก็บไว้ภายใน collections ต่างๆได้โดยที่ต้องการซ่อนความซับซ้อนต่างๆภายใน collection นั้นและรวมถึง implementation ด้วย

เป็นไปตามหลักการ SOLID Principle

- [x] Single-responsibility principle
- [x] Open–closed principle
- [ ] Liskov substitution principle
- [ ] Interface segregation principle
- [ ] Dependency inversion principle

ITERATOR


Keras - Iterator
Iterator
batch_size: int
seed: Integer
shuffle: Boolean
...
iter(): Void
next(): NumpyArray
flow_index(): Generator
DirectoryIterator
allowed_class_modes: Set
directory: String
classes: List
...
filepaths(): List
labels(): List
NumpyArrayIterator
x: NumpyArray
y: NumpyArray
...
DataFrameIterator
allowed_class_modes: Set
directory: String
classes: List
...
get_classes(): List
filter_classes(): List
filepaths(): List
ImageDataGenerator
horizontal_flip: Boolean
vertical_flip: Boolean
rescale: Float
...
flow(): Iterator
flow_from_directory(): Iterator
flow_from_dataframe(): Iterator
client

THANK YOU

62010525 PAPHANKORN TANAROJ

62010602 PONLAPAT SANGUANSIRIKUL

62010604 PHONLAPAT JONGWATANASIRI

62010615 PATPUM HAKAEW

62010619 PATTAPON JANCHOO

62010785 LIKHITBHUM LHIKHITNGAM