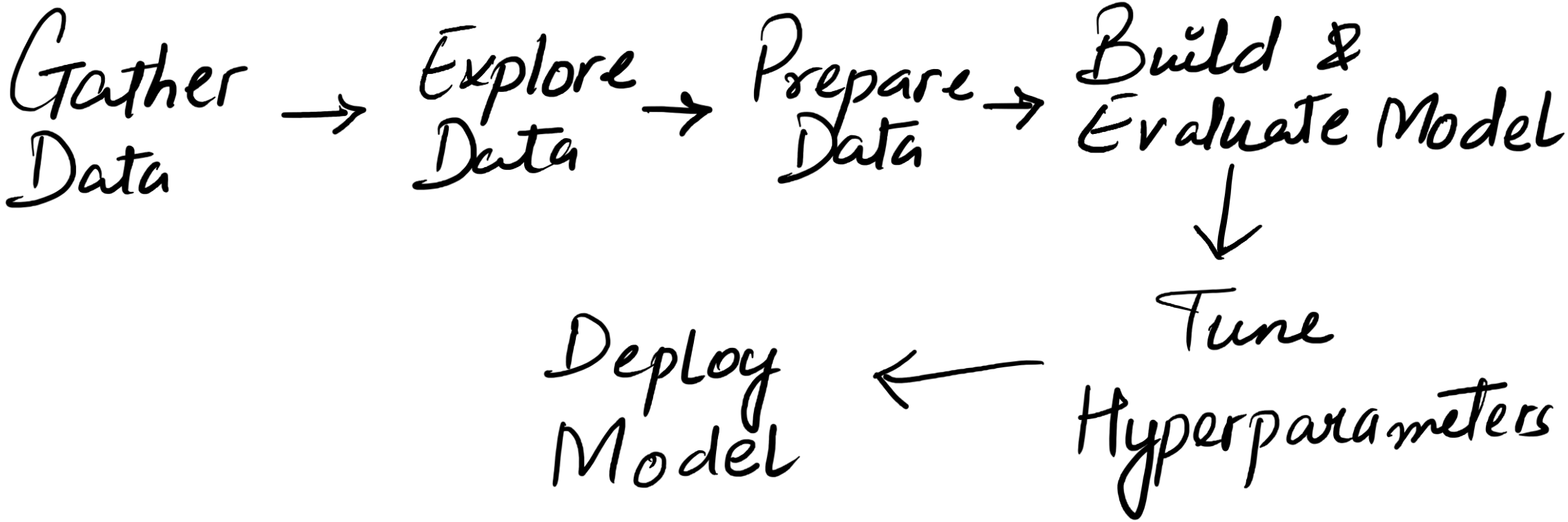
TensorFlow for Deep Learning

TANZEELA JAVID KALOO (32638)
ASSISTANT PROFESSOR
SYSTEMS AND ARCHITECTURE
LOVELY PROFESSIONAL UNIVERSITY

Steps of Solving an ML Problem



Steps of Solving an ML Problem

To perform these steps we need a tool

Python

Python

— Rich Ecosystem

libraries
Frameworks

Simply ML Tasks.

Very complex to implement
{- mathematical fn behind ML}

so

Engineers focus on problem Solving rather than implementation.

- Simplicity (english like code)
- Readability
- Supports Scientific Computation.
- Large and Active Community
- Open Source
- Platform Independent
  - Hardware & OS Architecture
    - Cloud
    - Local Machine

Libraries

Collection of

Code Block

functions classes } perform Specific tasks

- save time

Reusable

Frameworks

offers predifined architecture that dictates how to

- organize your code.
- provides libraries
- tools
- guidelines

to build a structured application

Libraries

Toolbox

- Tensorflow
- keras
- PyTorch
- Hugging Face Transformers
- Numpy
- Pandas
- Matplotlib
- Seaborn
- SciPy
- Scikit-learn
- Flask

- NLTK
- Plotly
- Statsmodels
- Opencv
- Pillow

Framework

Skeleton

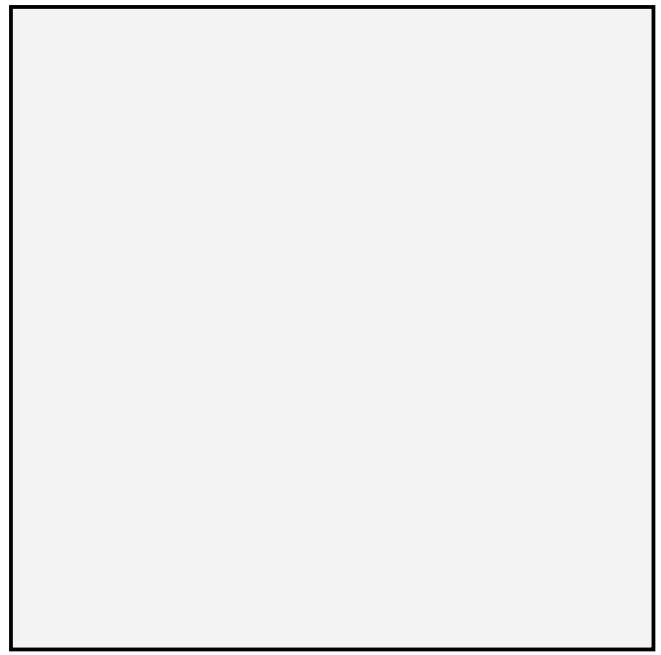
- **TensorFlow** — Google Brain (Python, C++, Java)
- **PyTorch** — Facebook's AI Research lab (FAIR) Python (C++ backend)
- **keras** — François Chollet (Now owned by TF) Python
- **MXNet** — Apache Software Foundation (Python, Scala, C++, Julia)
- **Caffe** — Berkeley Vision & Learning Center (C++ with Python)
- **JAX** — Google (Python)
- **DL4J** — Skymind (Java)
- **ONNX** (Open NN Exchange) — Facebook, Microsoft, others

TensorFlow

End to end platform for ML

- Multidimensional Array (similar to Numpy)
- GPU & distributed processing
- Model Construction, training & export

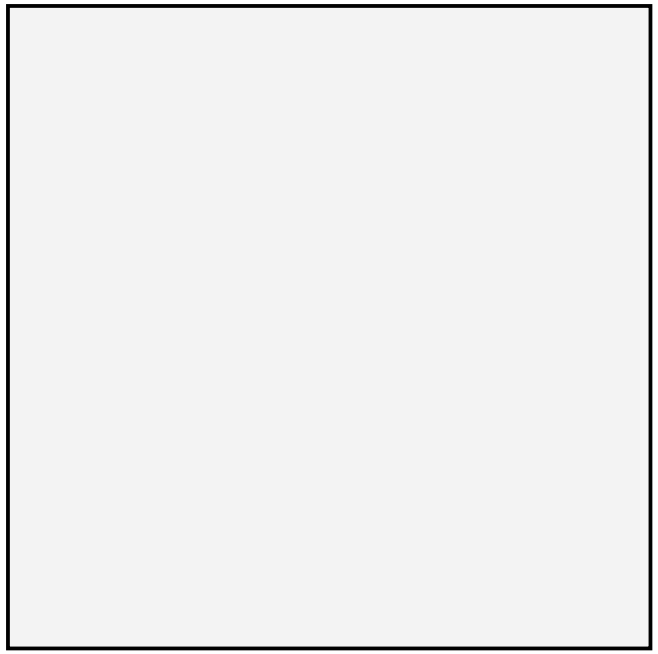
→ Tensors (kind of a special name to MD arrays)



Logical Structure
(organizes
code
for reusability)

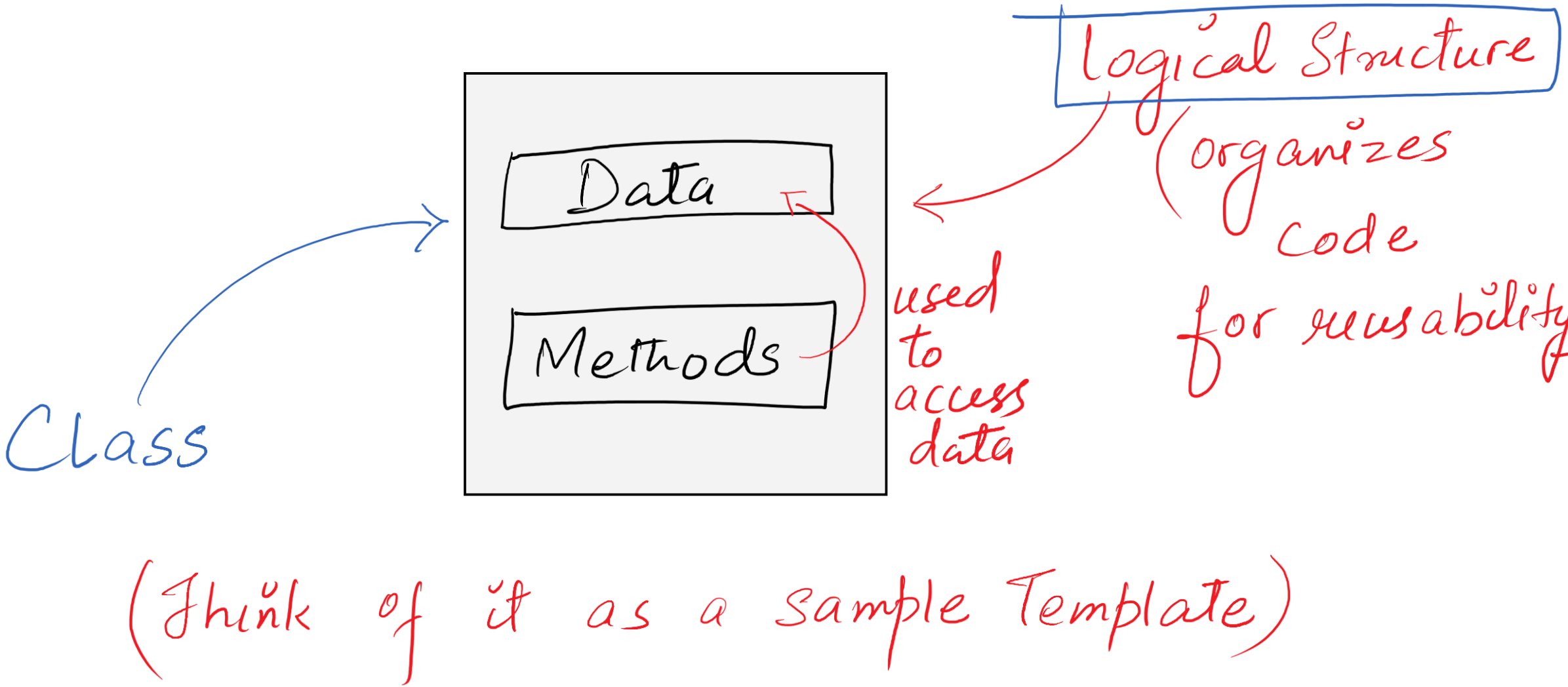
Logical Structure

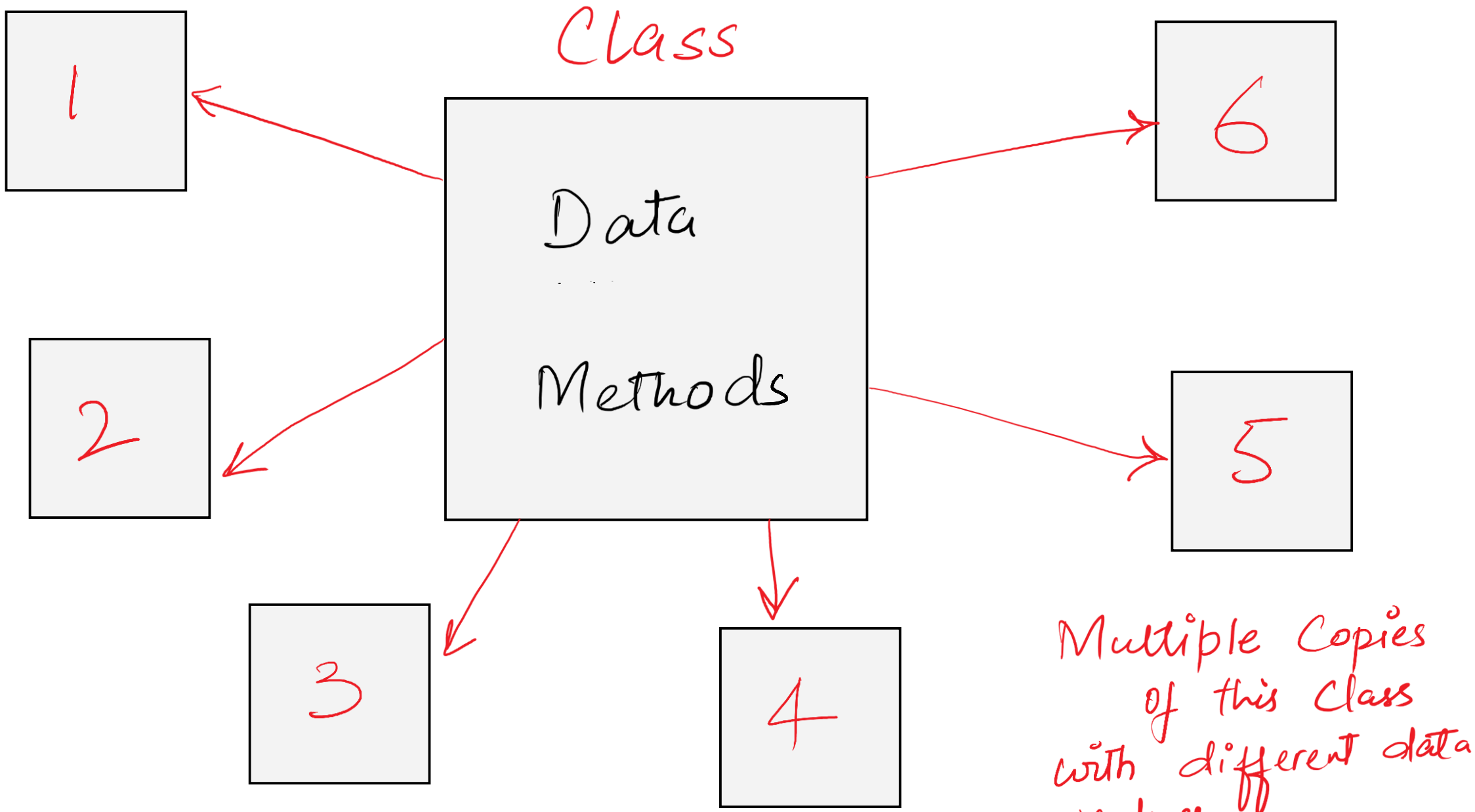
(organizes code for reusability)

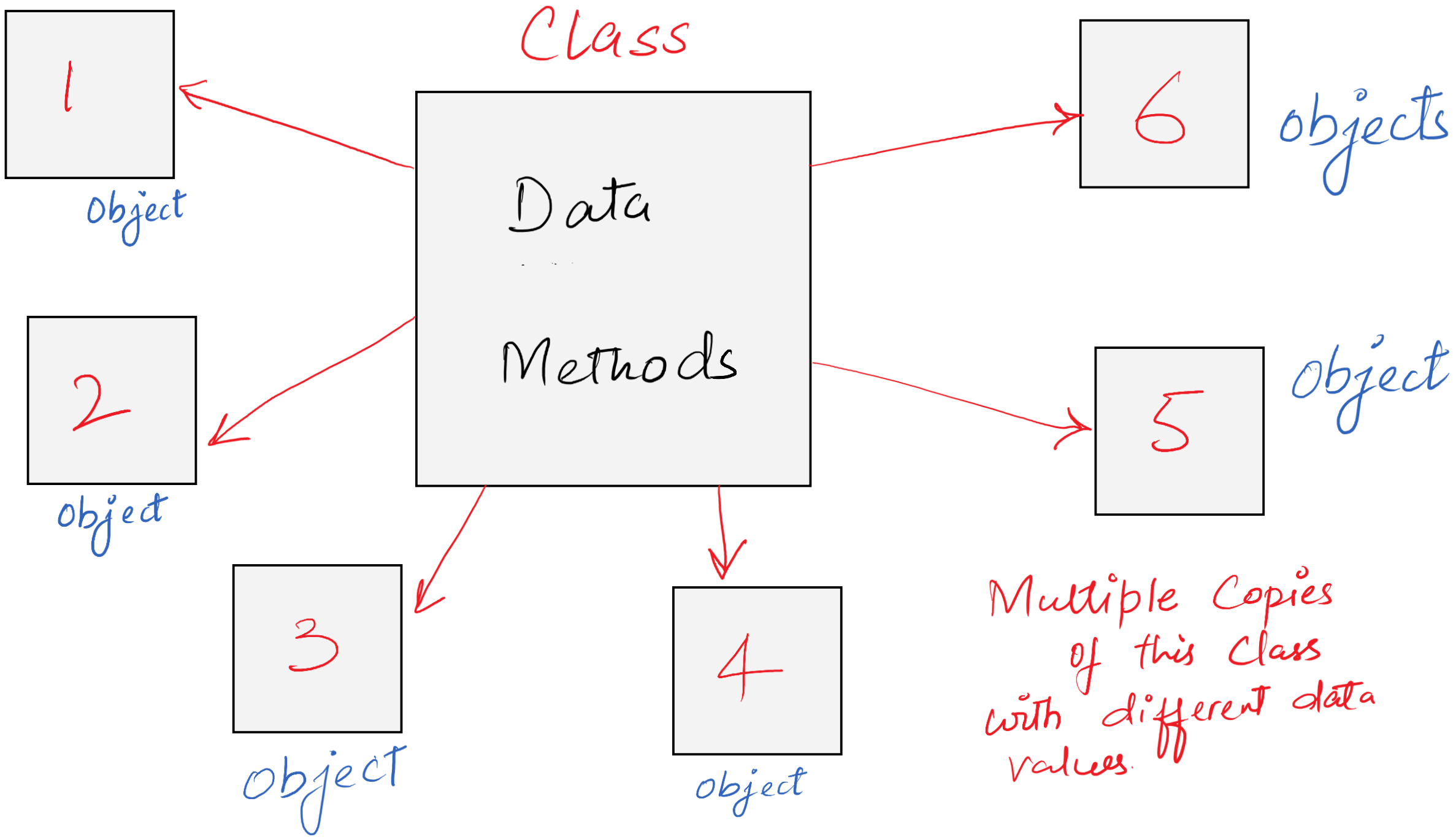


Class

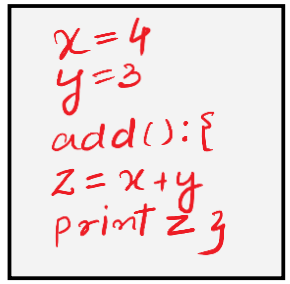
(Think of it as a sample Template)


Logical Structure
(organizes code for reusability)
Class
Data
Methods
used to access data
(Think of it as a sample Template)


Class
Data
Methods
1
2
3
4
5
6
Multiple Copies of this Class with different data values.


Class
Data
Methods
1
object
2
object
3
object
4
object
5
object
6
objects
Multiple Copies
of this Class
with different data
values.

Objects

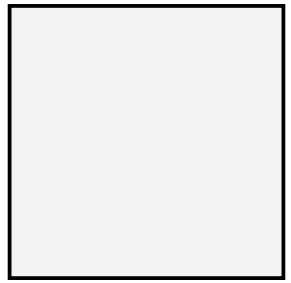


01

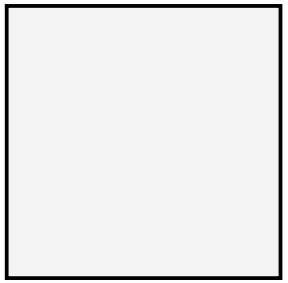
- unique ID
- State (represented by the value of its attributes)
- Behavior (Defined by its methods)

example

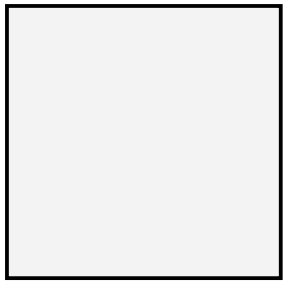
What is a class capable of doing



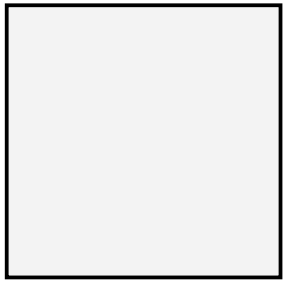
02



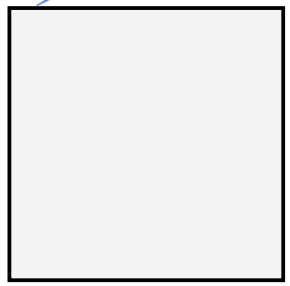
03



04

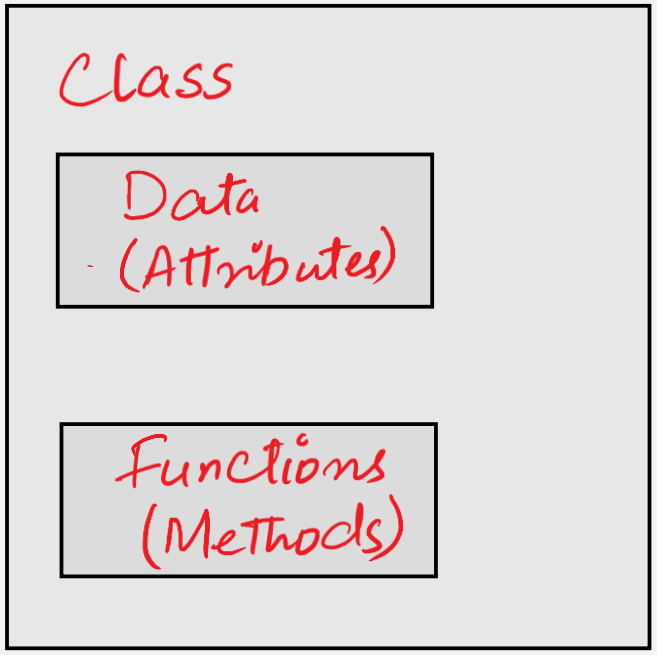


06



05

Modules



- keras (High end API)
- nn
- operations
- train
- losses
- math
- image
- save_model
- estimator
- random
- metrices
- audio
- data

Used to handle end-to-end ML tasks

Modules

Class

Data
(Attributes)

Functions
(Methods)

- keras (High end API)
- nn
- operations
- train
- losses
- math
- image
- save_model
- estimator
- random
- metrices
- audio
- data

tensorflow.keras

or

tf.keras

if tensorflow is imported as tf

"import tensorflow as tf"

Used to handle end-to-end ML tasks

Modules

Class

Data
(Attributes)

Functions
(Methods)

- model
- layers
- optimizers
- Losses
- metrics
- preprocessing
- regularizers
- activations

These are module in Keras API again a module

tf.keras.optimizers

- keras (High end API)
- nn
- operations
- train
- losses
- math
- image
- save_model
- estimator
- random
- metrices
- audio
- data

tensorflow.keras or tf.keras

if tensorflow is imported as tf

"import tensorflow as tf"

Used to handle end-to-end ML tasks

Modules

Class

Data
(Attributes)

Functions
(Methods)

deprecated b'coz of keras
- ~~Optimizer~~
- Checkpoint
- CheckpointManager
- ~~GradientDescent Optimizer~~

Deprecated

tf.train.Checkpoint

- keras (High end API)
- nn
- operations
- train
- losses
- math
- image
- save_model
- estimator
- random
- metrices
- audio
- data

tensorflow.train
or
tf.train

if tensorflow is imported as tf
"import tensorflow as tf"

used to handle end-to-end ML tasks

Modules

Class

Data
(Attributes)

Functions
(Methods)

- ImageProcessor
- ResizeMethod
- rgb_to_grayscale
- grayscale_to_rgb
- random_brightness
- random_contrast
- random_hue
- random_saturation
- random_*
- extract_patches
- central_crop

tf.image.rgb_to_grayscale

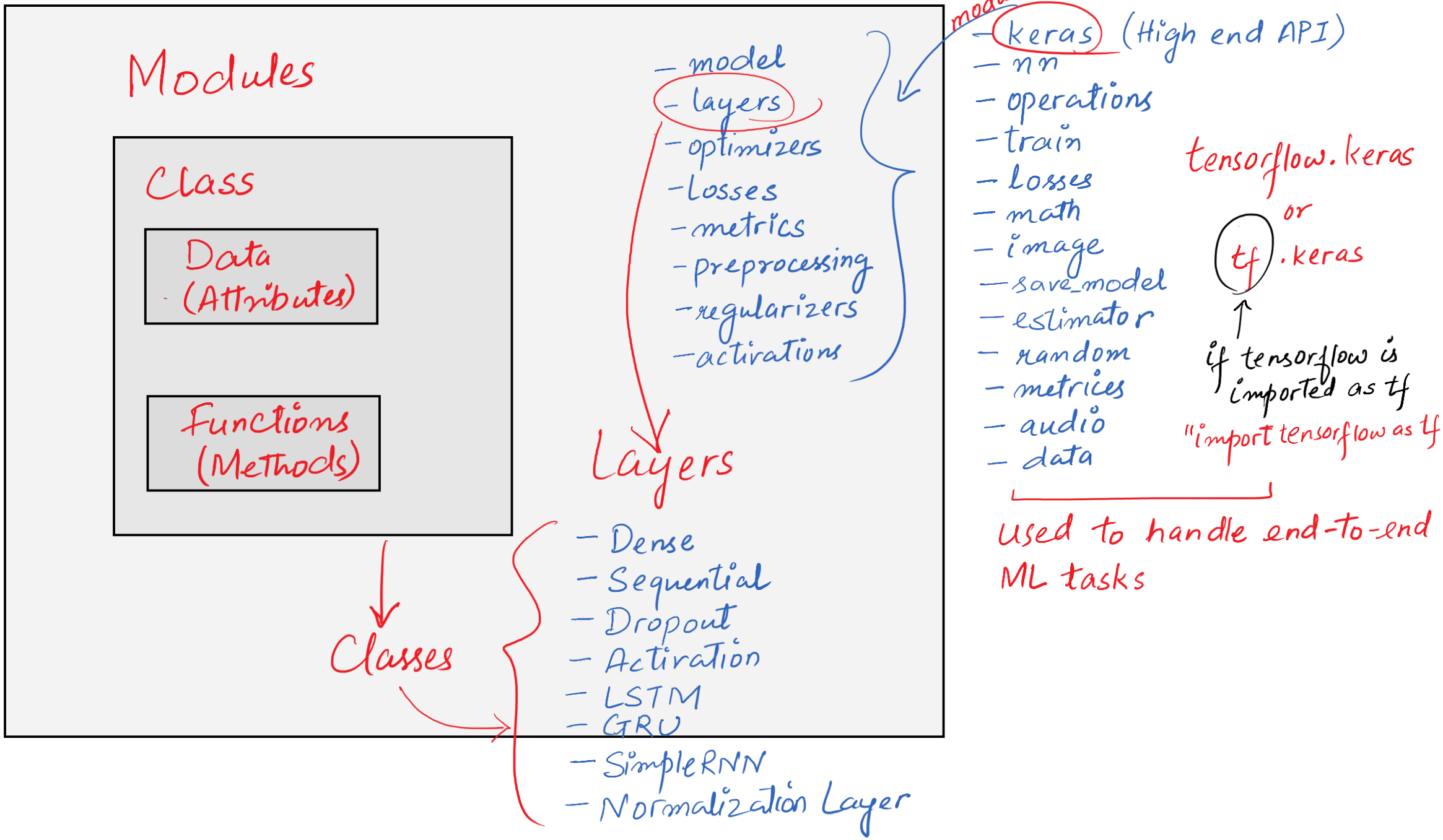
- keras (High end API)
- nn
- operations
- train
- losses
- math
- image
- save_model
- estimator
- random
- metrices
- audio
- data

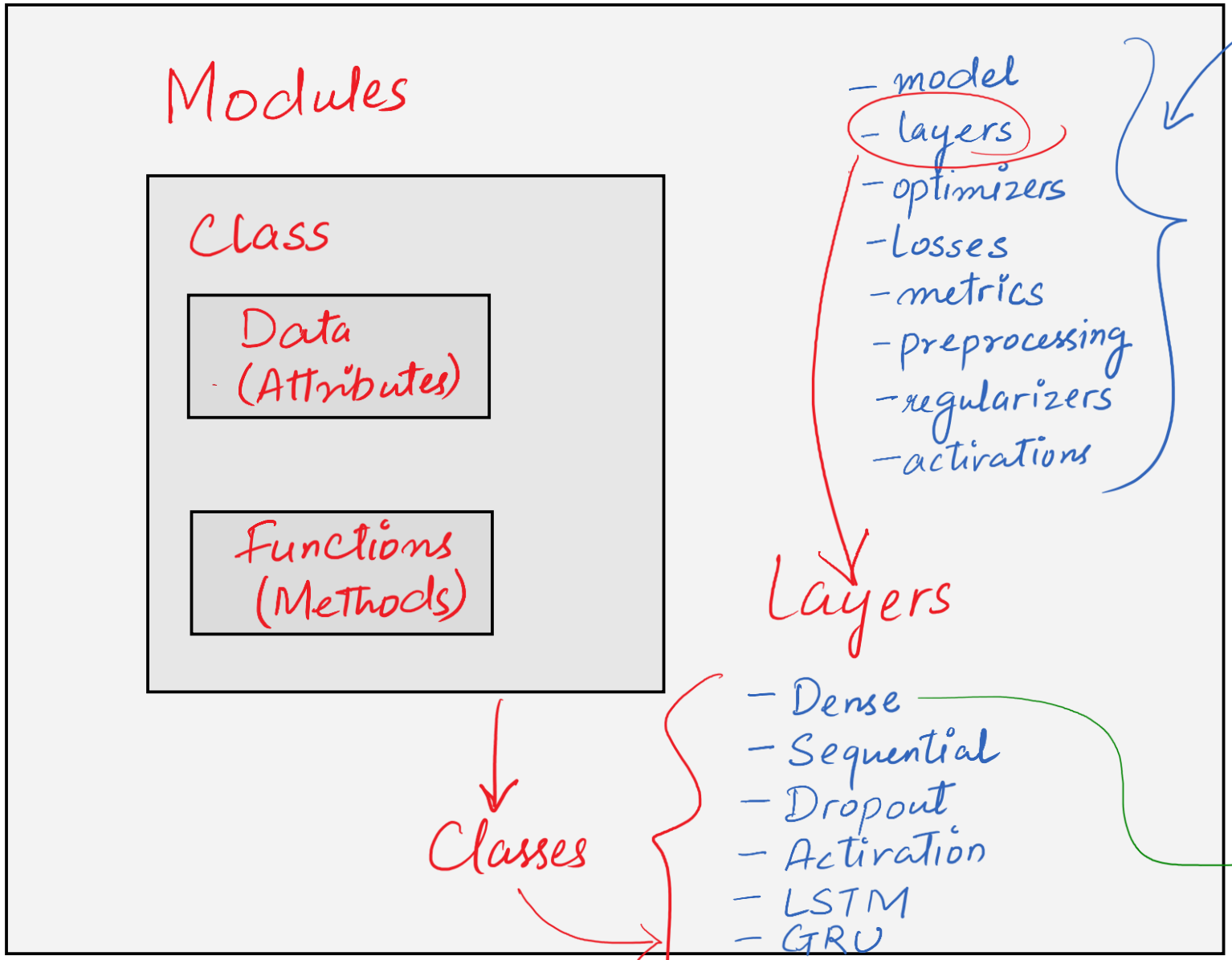
tensorflow.image
or
tf.image

if tensorflow is imported as tf

"import tensorflow as tf"

Used to handle end-to-end ML tasks


Modules
Class
Data
(Attributes)
Functions
(Methods)
- model
- layers
- optimizers
- Losses
- metrics
- preprocessing
- regularizers
- activations
Layers
Classes
- Dense
- Sequential
- Dropout
- Activation
- LSTM
- GRU
- SimpleRNN
- Normalization Layer
module
keras (High end API)
- nn
- operations
- train
- losses
- math
- image
- save_model
- estimator
- random
- metrics
- audio
- data
tensorflow.keras
or
tf.keras
if tensorflow is imported as tf
"import tensorflow as tf"
Used to handle end-to-end ML tasks


Modules
Class
Data
(Attributes)
Functions
(Methods)
Classes
- model
- layers
- optimizers
- Losses
- metrics
- preprocessing
- regularizers
- activations
Layers
- Dense
- Sequential
- Dropout
- Activation
- LSTM
- GRU
- SimpleRNN
- Normalization Layer
module
- keras (High end API)
- nn
- operations
- train
- losses
- math
- image
- save_model
- estimator
- random
- metrices
- audio
- data
tensorflow.keras
or
tf.keras
if tensorflow is imported as tf
"import tensorflow as tf"
used to handle end-to-end ML tasks
Fully Connected NN layer
neurons
Activation fxn
input_shape

```
keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    lora_rank=None,
    **kwargs
)
```

class

attributes

The Model class

`keras.Model()`

```
inputs = keras.Input(shape=(37,))
x = keras.layers.Dense(32, activation="relu")(inputs)
outputs = keras.layers.Dense(5, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

summary method

```
Model.summary(
    line_length=None,
    positions=None,
    print_fn=None,
    expand_nested=False,
    show_trainable=False,
    layer_range=None,
)
```

get_layer method

```
Model.get_layer(name=None, index=None)
```

```
model.add(layers.Dense(64, activation=activations.relu))
```

```
from keras import layers
from keras import activations

model.add(layers.Dense(64))
model.add(layers.Activation(activations.relu))
```

```
keras.activations.relu(x, negative_slope=0.0, max_value=None, threshold=0.0)
```

```
keras.activations.sigmoid(x)
```

Sigmoid activation function.

It is defined as: `sigmoid(x) = 1 / (1 + exp(-x))`.

```
keras.activations.softsign(x)
```

Softsign activation function.

Softsign is defined as: `softsign(x) = x / (abs(x) + 1)`.

Recurrent layers

- LSTM layer
- LSTM cell layer
- GRU layer
- GRU Cell layer
- SimpleRNN layer
- TimeDistributed layer
- Bidirectional layer
- ConvLSTM1D layer
- ConvLSTM2D layer
- ConvLSTM3D layer
- Base RNN layer
- Simple RNN cell layer
- Stacked RNN cell layer

```
keras.layers.SimpleRNN(
    units,
    activation="tanh",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    unroll=False,
    seed=None,
    **kwargs
)
```

Convolution layers

- Conv1D layer
- Conv2D layer
- Conv3D layer
- SeparableConv1D layer
- SeparableConv2D layer
- DepthwiseConv1D layer
- DepthwiseConv2D layer
- Conv1DTranspose layer
- Conv2DTranspose layer
- Conv3DTranspose layer

```
keras.layers.Conv1D(
    filters,
    kernel_size,
    strides=1,
    padding="valid",
    data_format=None,
    dilation_rate=1,
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Available optimizers

- SGD
- RMSprop
- Adam
- AdamW
- Adadelta
- Adagrad
- Adamax
- Adafactor
- Nadam
- Ftrl
- Lion
- Lamb
- Loss Scale Optimizer

```
keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=None,
    loss_scale_factor=None,
    gradient_accumulation_steps=None,
    name="adam",
    **kwargs
)
```

How to create a Neural Network Model

```
inputs = keras.Input(shape=(37,))
x = keras.layers.Dense(32, activation="relu")(inputs)
outputs = keras.layers.Dense(5, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

```
keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    lora_rank=None,
    **kwargs
)
```

The Sequential class

```
keras.Sequential(layers=None, trainable=True, name=None)
```

add method

```
Sequential.add(layer, rebuild=True)
```

pop method

```
Sequential.pop(rebuild=True)
```

`Sequential` groups a linear stack of layers into a `Model`.

```
model = keras.Sequential()
model.add(keras.Input(shape=(16,)))
model.add(keras.layers.Dense(8))
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(8))
model.add(keras.layers.Dense(4))
```

```
model = keras.Sequential()
model.add(keras.Input(shape=(16,)))
model.add(keras.layers.Dense(8))
len(model.weights)  # Returns "2"
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(8))
model.add(keras.layers.Dense(1))
model.compile(optimizer='sgd', loss='mse')
# This builds the model for the first time:
model.fit(x, y, batch_size=32, epochs=10)
```

`compile` method

```
Model.compile(
    optimizer="rmsprop",
    loss=None,
    loss_weights=None,
    metrics=None,
    weighted_metrics=None,
    run_eagerly=False,
    steps_per_execution=1,
    jit_compile="auto",
    auto_scale_loss=True,
)
```

```
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss=keras.losses.BinaryCrossentropy(),
    metrics=[
        keras.metrics.BinaryAccuracy(),
        keras.metrics.FalseNegatives(),
    ],
)
```

fit method

```
Model.fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose="auto",
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
)
```

evaluate method

```
Model.evaluate(
    x=None,
    y=None,
    batch_size=None,
    verbose="auto",
    sample_weight=None,
    steps=None,
    callbacks=None,
    return_dict=False,
    **kwargs
)
```

predict method

```
Model.predict(x, batch_size=None, verbose="auto", steps=None, callbacks=None)
```

save method

```
Model.save(filepath, overwrite=True, zipped=None, **kwargs)
```

```
model = keras.Sequential(
    [
        keras.layers.Dense(5, input_shape=(3,)),
        keras.layers.Softmax(),
    ],
)
model.save("model.keras")
loaded_model = keras.saving.load_model("model.keras")
```

Accuracy metrics

`Accuracy` class

```
keras.metrics.Accuracy(name="accuracy", dtype=None)
```

Usage with `compile()` API:

```
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=[keras.metrics.Accuracy()])
```

Image segmentation metrics

`IoU` class

```
keras.metrics.IoU(
    num_classes,
    target_class_ids,
    name=None,
    dtype=None,
    ignore_class=None,
    sparse_y_true=True,
    sparse_y_pred=True,
    axis=-1,
)
```

```
iou = true_positives / (true_positives + false_positives + false_negatives)
```

Usage with `compile()` API:

```
model.compile(
    optimizer='sgd',
    loss='mse',
    metrics=[keras.metrics.IoU(num_classes=2, target_class_ids=[0])])
```

Available datasets

MNIST digits classification dataset

- load_data function

CIFAR10 small images classification dataset

- load_data function

CIFAR100 small images classification dataset

- load_data function

IMDB movie review sentiment classification dataset

- load_data function
- get_word_index function

Reuters newswire classification dataset

- load_data function
- get_word_index function

Fashion MNIST dataset, an alternative to MNIST

- load_data function

California Housing price regression dataset

- load_data function

Fashion MNIST dataset, an alternative to MNIST

`load_data` function [source]

```
keras.datasets.fashion_mnist.load_data()
```

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

KerasTuner

hyperparameter optimization framework

```
pip install keras-tuner
```

```
import keras_tuner
import keras
```

```
def build_model(hp):
  model = keras.Sequential()
  model.add(keras.layers.Dense(
      hp.Choice('units', [8, 16, 32]),
      activation='relu'))
  model.add(keras.layers.Dense(1, activation='relu'))
  model.compile(loss='mse')
  return model
```

```
tuner = keras_tuner.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=5)
```

Start the search and get the best model:

```
tuner.search(x_train, y_train, epochs=5, validation_data=(x_val, y_val))
best_model = tuner.get_best_models()[0]
```

```python
def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Flatten())
    model.add(
        layers.Dense(
            # Tune number of units.
            units=hp.Int("units", min_value=32, max_value=512, step=32),
            # Tune the activation function to use.
            activation=hp.Choice("activation", ["relu", "tanh"]),
        )
    )
    # Tune whether to use dropout.
    if hp.Boolean("dropout"):
        model.add(layers.Dropout(rate=0.25))
    model.add(layers.Dense(10, activation="softmax"))
    # Define the optimizer learning rate as a hyperparameter.
    learning_rate = hp.Float("lr", min_value=1e-4, max_value=1e-2, sampling="log")
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
        loss="categorical_crossentropy",
        metrics=["accuracy"],
    )
    return model


build_model(keras_tuner.HyperParameters())
```

```
hp = keras_tuner.HyperParameters()
print(hp.Int("units", min_value=32, max_value=512, step=32))
```

```
tuner = keras_tuner.RandomSearch(
    hypermodel=build_model,
    objective="val_accuracy",
    max_trials=3,
    executions_per_trial=2,
    overwrite=True,
    directory="my_dir",
    project_name="helloworld",
)
```