Keras

(Python Framework)

dayche.com . گروه دایکه

دای‌کهِ
DAYCHE

Introduction

- An open-source framework for developing the deep neural networks in python
  - Developed by Froncois Challot, a software engineer in Google
- Frameworks

- Tensorflow
- PyTorch
- Theano
- Caffe
- MxNet

- Keras built on Tensorflow and Theano
- Theano – Fast computational capacity, written completely with Python
- Tensorflow – deep learning frameworks and importantly distributed processing support, written with C++ and Python
- Keras enjoy the mentioned features plus the user-friendly properties

Backend configuration

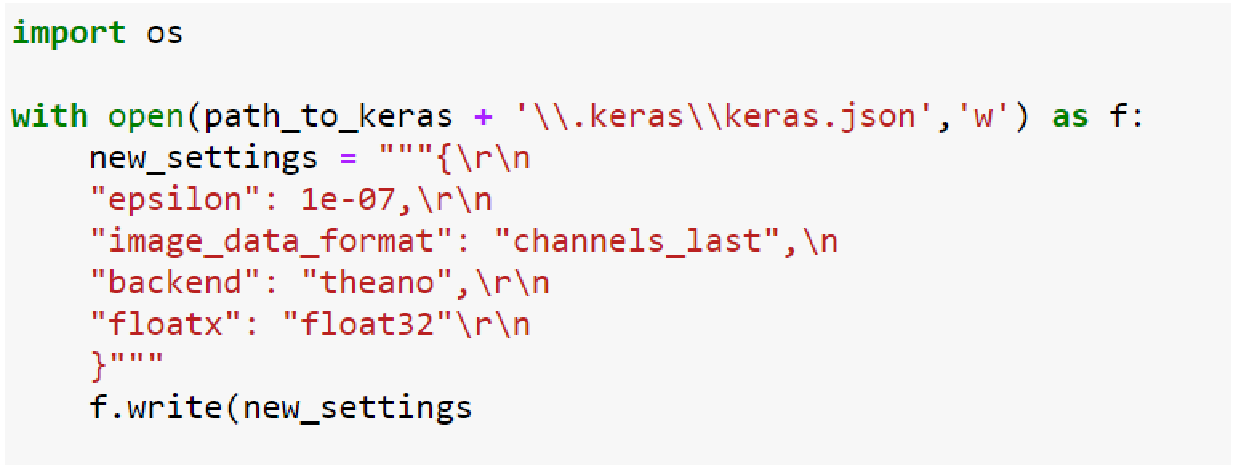
- Backend implementation, Tensorflow and Theano

Default backend = Tensorflow

```
{
    "image_data_format": "channels_last",
    "epsilon": 1e-07,
    "floatx": "float32",
    "backend": "tensorflow"
}
```

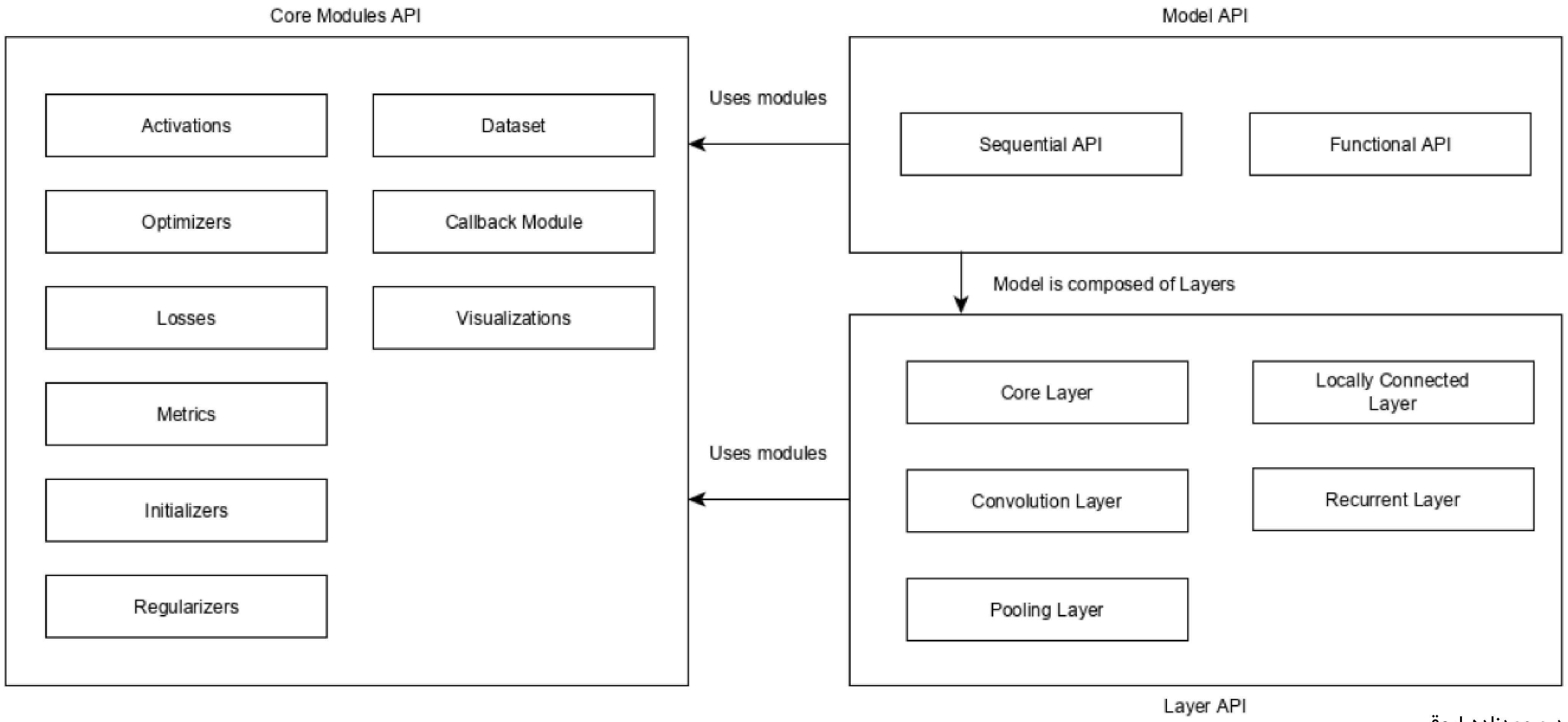
How to change the backend configuration?

- There is no need to do that!



```python
import os

with open(path_to_keras + '\\.keras\\keras.json','w') as f:
    new_settings = """{\r\n
    "epsilon": 1e-07,\r\n
    "image_data_format": "channels_last",\n
    "backend": "theano",\r\n
    "floatx": "float32"\r\n
    }"""
    f.write(new_settings
```

Architecture of Keras



Keras models

- Sequential class
  - A linear stack of layers into keras.Model

Type of activation function

?

```
model = keras.models.Sequential(name = 'Our First Model')
model.add(keras.layers.Dense(8, input_shape = (16, )))
model.summary()
```

```
Model: "Our First Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 8)                 136
=================================================================
Total params: 136
Trainable params: 136
Non-trainable params: 0
```

Sequential model

- Different way of adding layers

```
model = keras.models.Sequential(name = 'Our First Model')
model.add(keras.layers.Dense(8, input_shape = (16, ), name = 'FC1'))

model = keras.models.Sequential(name = 'Our First Model')
model.add(keras.Input(shape = (16, )))
model.add(keras.layers.Dense(8, name = 'FC1'))

model = keras.models.Sequential(
    [keras.layers.Dense(8, input_shape = (16, ), name = 'FC1')], name = 'Our First Model')
```

Sequential model

- First layer is required to have input dimension

Delayed-build pattern model

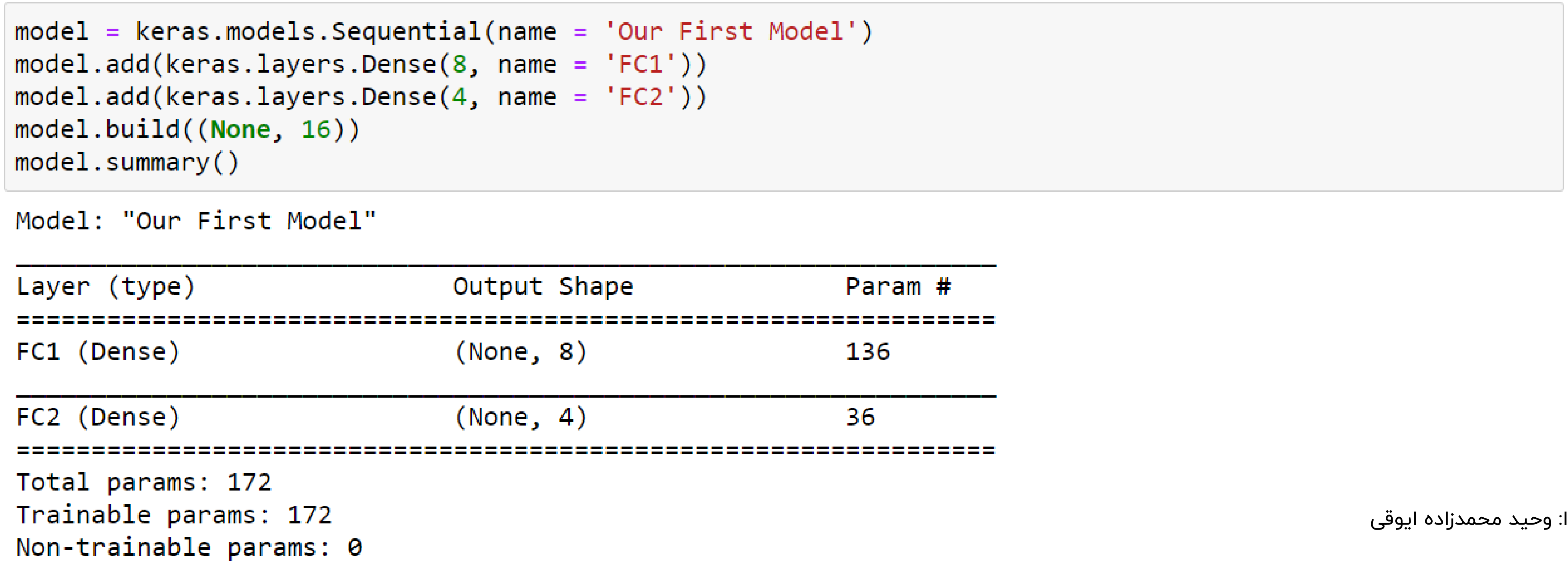
```
model = keras.models.Sequential(name = 'Our First Model')
model.add(keras.layers.Dense(8, name = 'FC1'))
```

The mode has not yet been created!

```
ValueError: This model has not yet been built. Build the model first by calling `build()` or calling
`fit()` with some data, or specify an `input_shape` argument in the first layer(s) for automatic buil
d.
```

Sequential model

- Delayed-build pattern
  - Build method, Fit method – infer automatically



```python
model = keras.models.Sequential(name = 'Our First Model')
model.add(keras.layers.Dense(8, name = 'FC1'))
model.add(keras.layers.Dense(4, name = 'FC2'))
model.build((None, 16))
model.summary()
```

```
Model: "Our First Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
FC1 (Dense)                  (None, 8)                 136       
_________________________________________________________________
FC2 (Dense)                  (None, 4)                 36        
=================================================================
Total params: 172
Trainable params: 172
Non-trainable params: 0
```

Keras models

- Model class
  - Group layers into an object – appropriate for complex projects

```
model = keras.models.Model(inp, out, name = 'Our first model' )
```

  - There are two way for creating such a model: Functional API, sub-classing

- Functional API – we start from input to output

```
inputs = keras.Input(shape=(16,))
x = keras.layers.Dense(8, name = 'FC1')(inputs)
outputs = keras.layers.Dense(4, name = 'FC2')(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.summary()
```

Keras models

```
class MyModel(keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.dense1 = keras.layers.Dense(8, name = 'FC1')
    self.dense2 = keras.layers.Dense(4, name = 'FC2')
  def call(self, inputs):
    x = self.dense1(inputs)
    return self.dense2(x)

model = MyModel()
model.build((None, 16))
```

Define layers in __init__

Define forward pass in call

```
model.summary()
```

```
Model: "my_model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
FC1 (Dense)                  multiple                  136
_________________________________________________________________
FC2 (Dense)                  multiple                  36
=================================================================
Total params: 172
Trainable params: 172
Non-trainable params: 0
```

Training APIs

- Upon building a model, an object of deep model, we need to train the model, how?
- There are some step regard to tutorial known on training a model . Creating an object is equivalent to fixing the structure, then we need to determine loss function, evaluation metrics, and optimization method.
  - Compile method

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

Optimizer

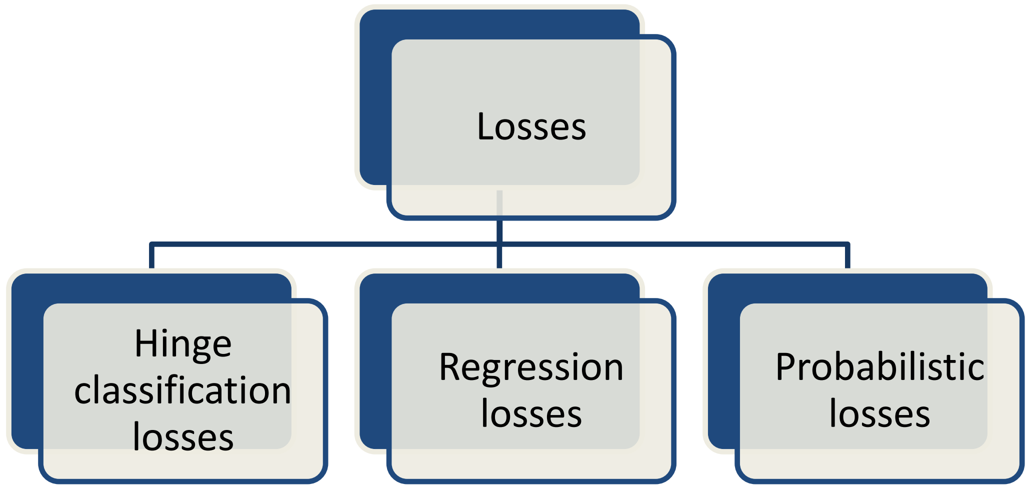
- There are two ways for setting optimization method
  - String - default value is RMSProp
  - Optimizer instance

Opt=Keras.optimizers. + SGD
RMSProp
Adam
Adadelta
Adagrad
Adamax

Each method has its own parameters need to be tuned

Loss function

- There are two ways for setting loss function
  - String
  - Loss instance

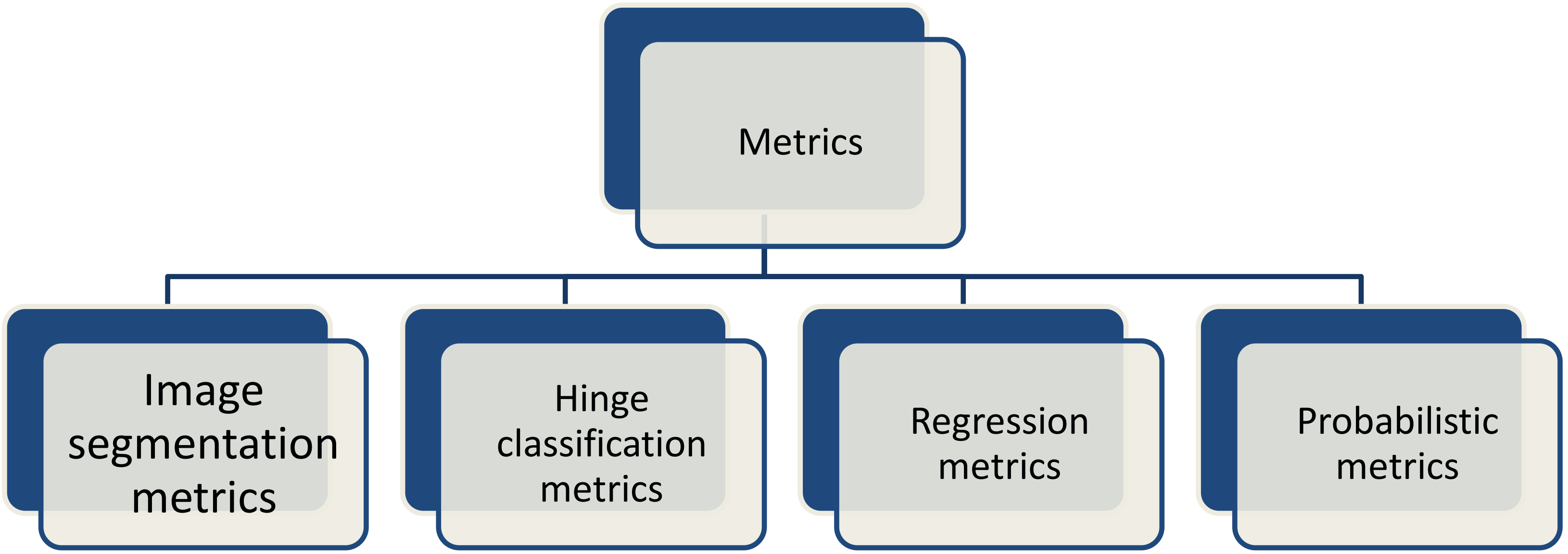


Opt=Keras.losses. + BinaryCrossEntopy CategoricalCrossEntopy

In case of multi-output problem we can pass a dictionary of losses

Metrics

- There are two ways for setting loss function
  - String
  - Loss instance



A simple model

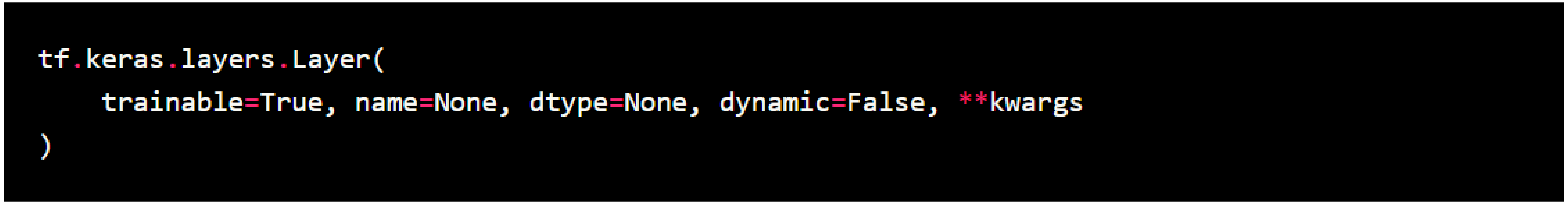
```python
import keras
import tensorflow as tf
model = keras.models.Sequential(name = 'First model')
model.add(keras.layers.Dense(8, activation = 'relu'))
model.add(keras.layers.Dense(4, activation = 'softmax'))
model.build((None, 16))
model.summary()
```

```
Model: "First model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 36
=================================================================
Total params: 172
Trainable params: 172
Non-trainable params: 0
_________________________________________________________________
```

```python
opt = tf.keras.optimizers.SGD()
loss = keras.losses.BinaryCrossentropy()
model.compile(optimizer=opt, loss = loss, metrics = 'binary_crossentropy')
```

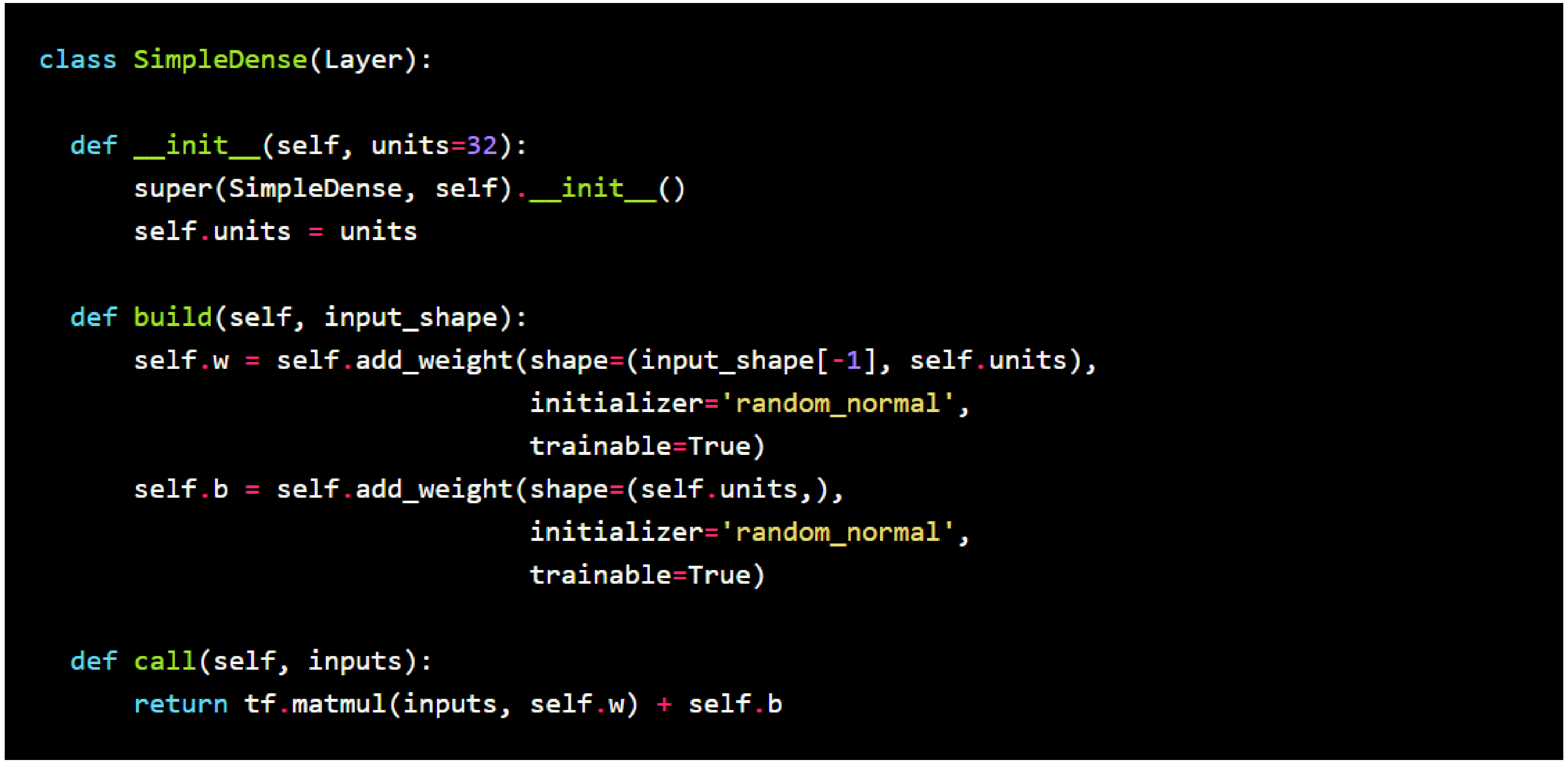
Layer APIs

- Base layer
  - All layers in keras inherit the base layer, so in case we need to develop a new layer we should do the same as other layers, RBF layers, Rough layers, flexible layers, and so in.



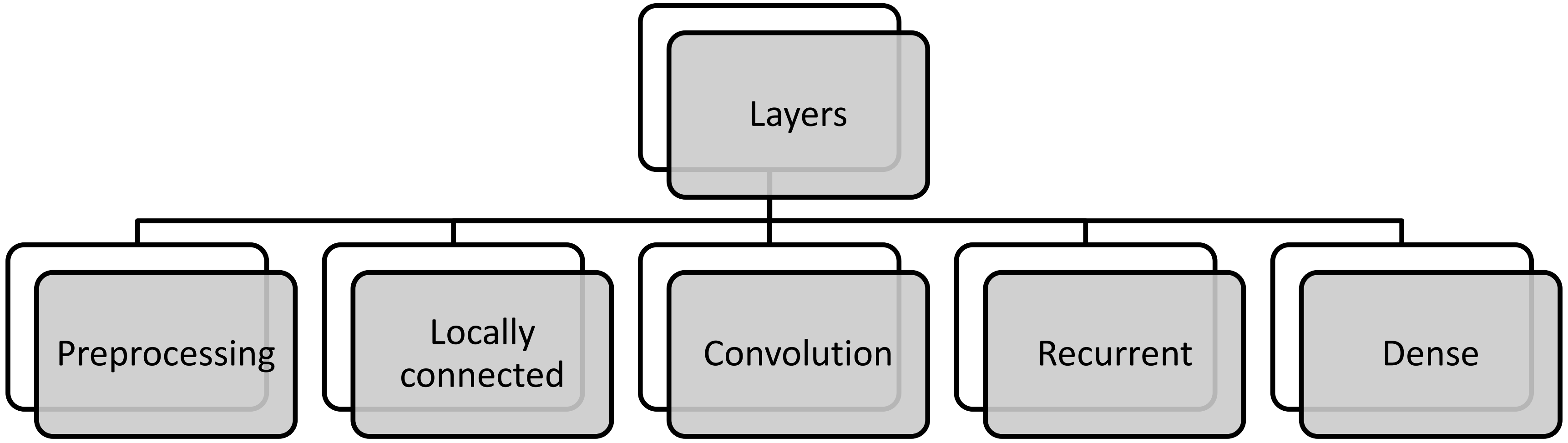
```
tf.keras.layers.Layer(
    trainable=True, name=None, dtype=None, dynamic=False, **kwargs
)
```

How to create a non-existing layer

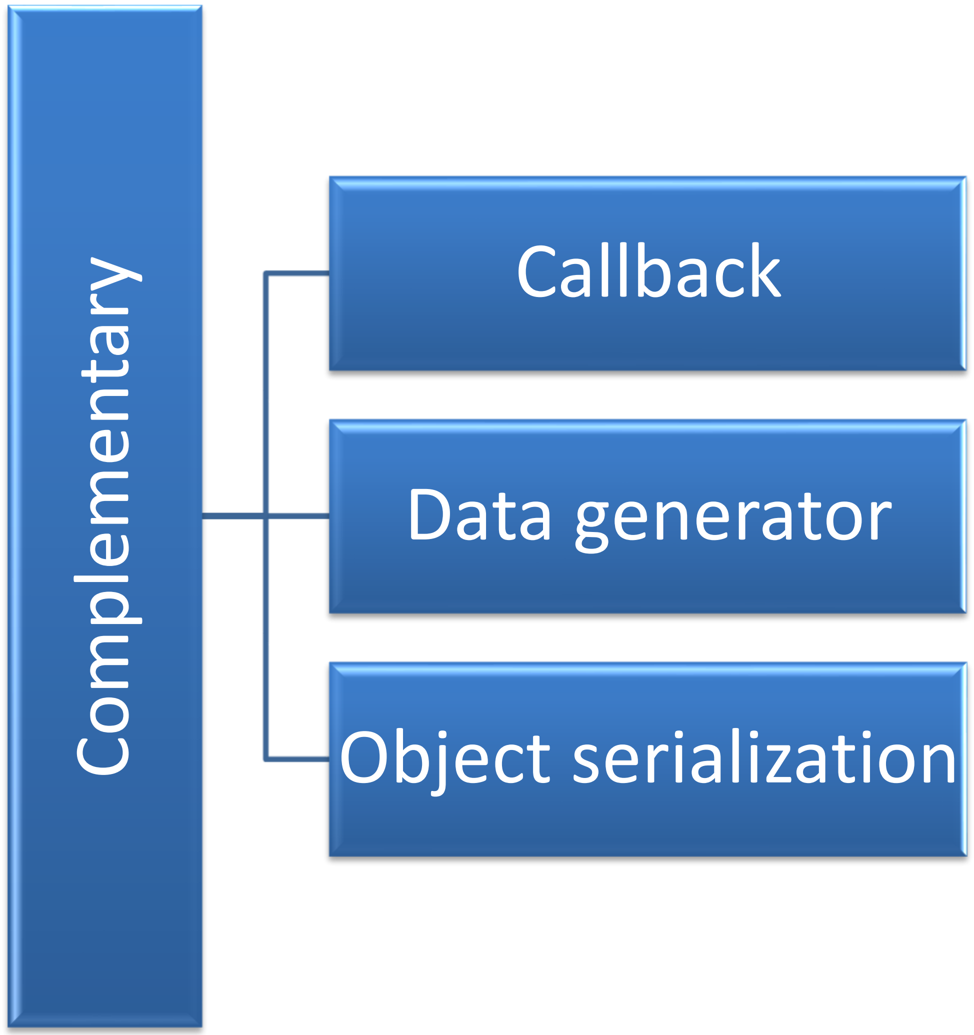


```
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):
      self.w = self.add_weight(shape=(input_shape[-1], self.units),
                               initializer='random_normal',
                               trainable=True)
      self.b = self.add_weight(shape=(self.units,),
                               initializer='random_normal',
                               trainable=True)

  def call(self, inputs):
      return tf.matmul(inputs, self.w) + self.b
```

Supporting layers



Complementary


Complementary
Callback
Data generator
Object serialization