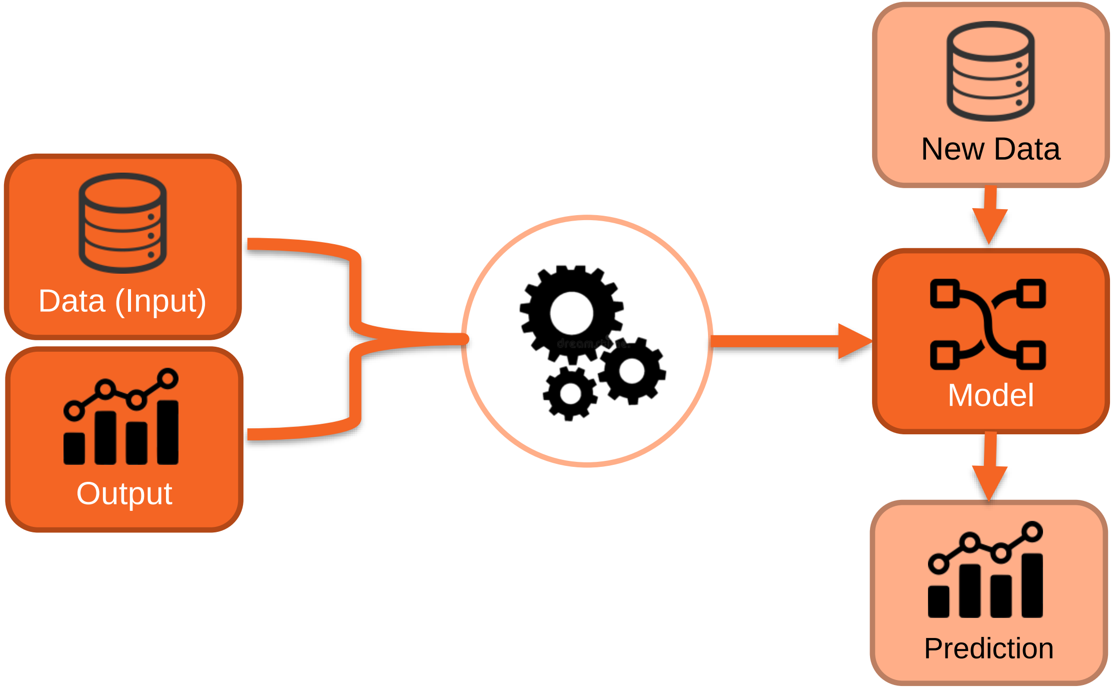
Menjadi Powerful Dengan Tensor(flow)

Part 2

Aditya Firman Ihsan • Tensorflow Developer Certified
KK Data Science Telkom University

Review dulu

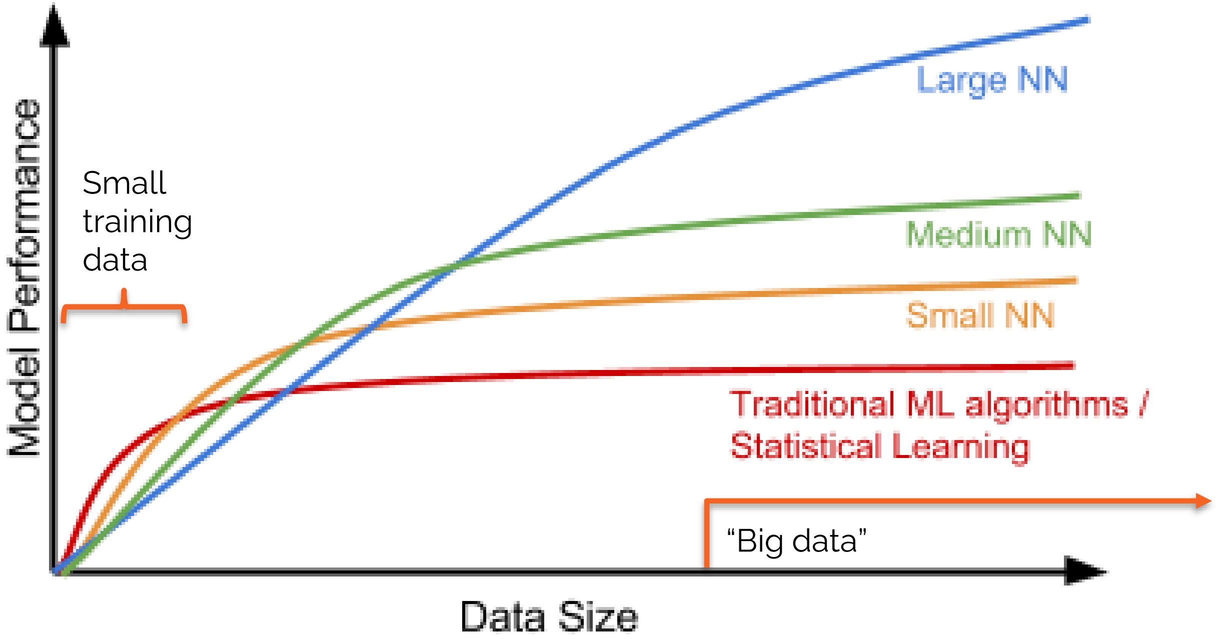
Bagaimana mesin belajar?



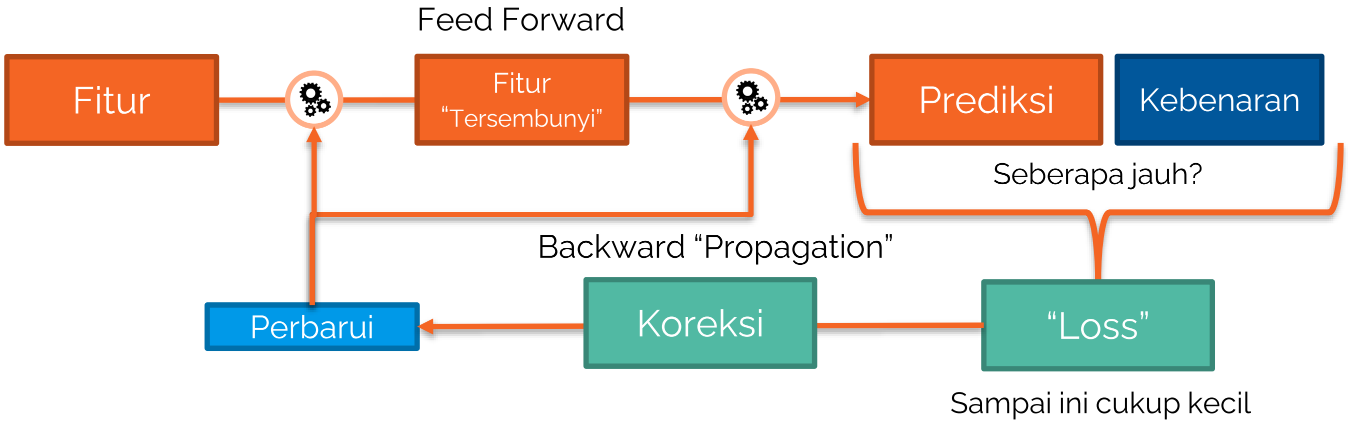
Bagaimana mesin belajar?

- Supervised
  - Classification
    - SVM
    - Naïve Bayes
    - Nearest Neighbor
    - Neural Networks
  - Regression
    - GLM
    - SVR
    - Decision Trees
    - Neural Networks
- Unsupervised
  - Clustering
    - K-Means
    - Gaussian Mixture
    - Hidden Markov
    - Neural Networks
- Reinforcement
  - Control
    - Monte Carlo
    - Q-Learning
    - SARSA
    - Neural Networks

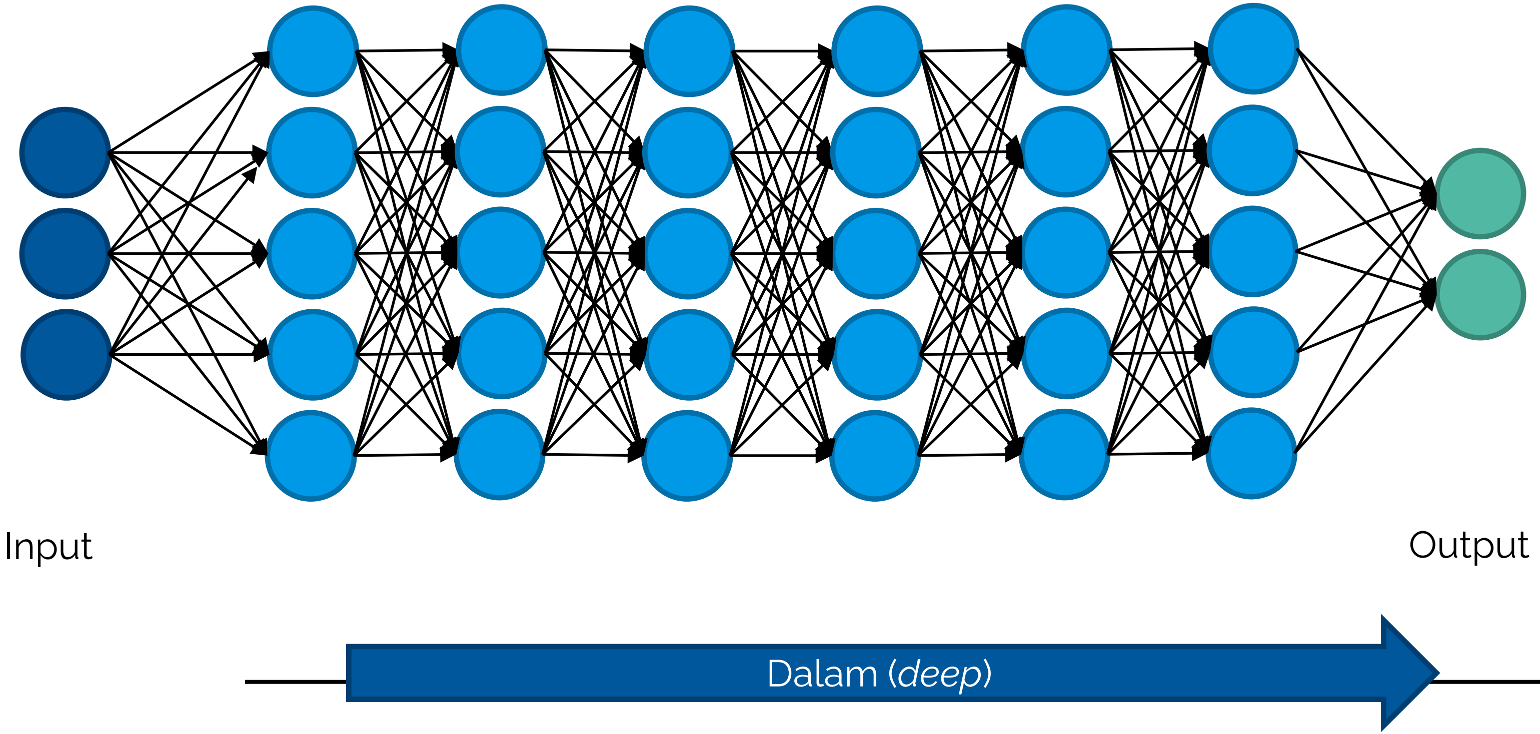
Kenapa Neural Network (NN)?



Apa yang dilakukan NN?



Apa yang dilakukan NN?



Kenapa harus tensor?

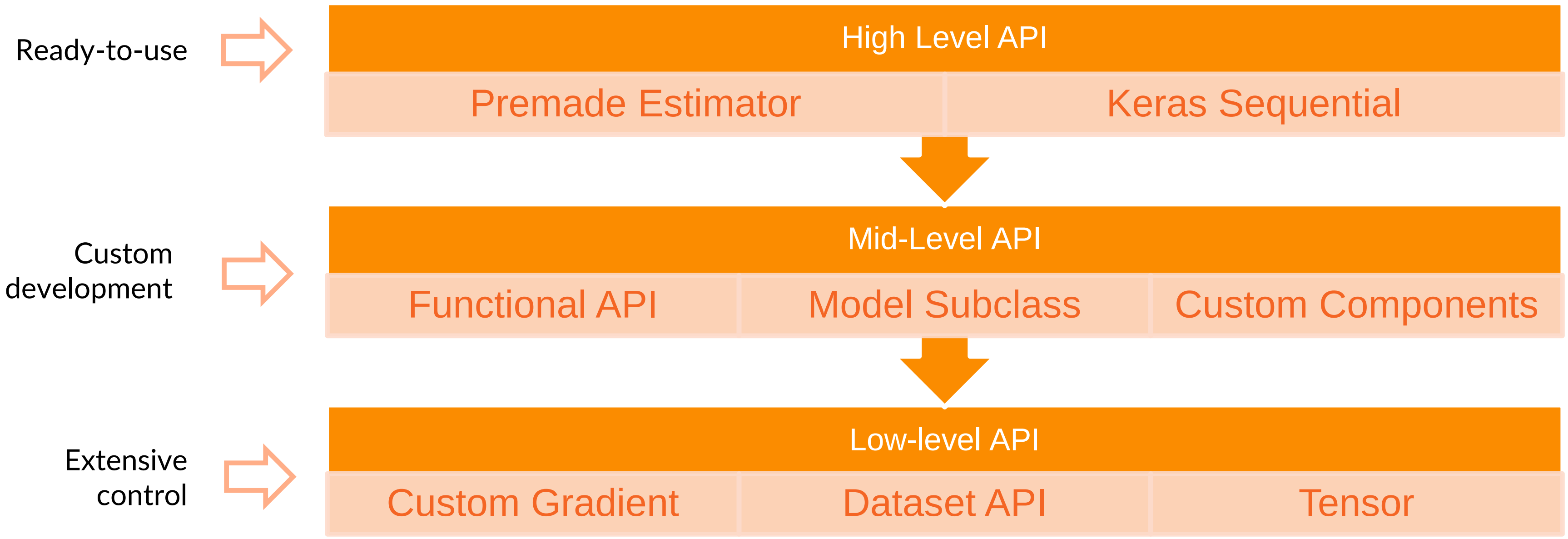
Tensor merupakan array yang menerapkan paradigma *differentiable programming*

Sederhananya, konsep program dimana perhitungan numerik dapat dihitung turunannya melalui graf komputasi yang dibangun

Tensorflow

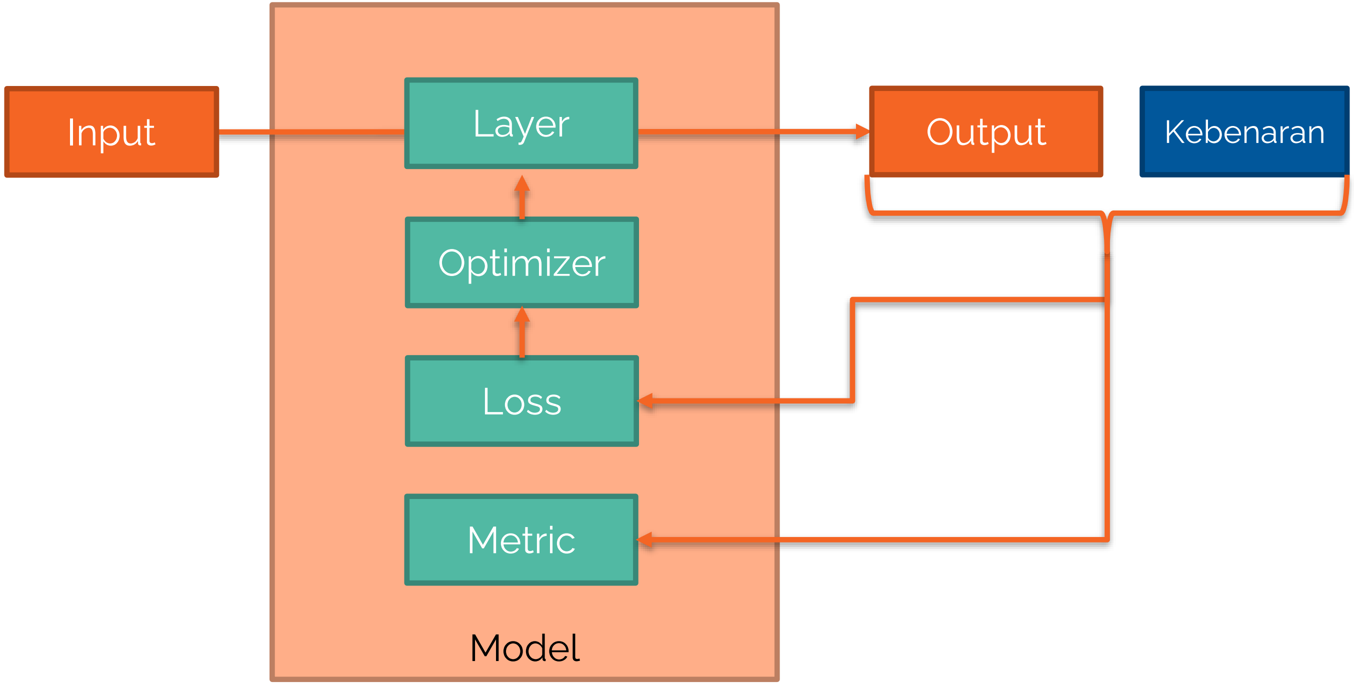
sebuah framework *Deep Learning*, yang memanfaatkan konsep tensor untuk efektivitas komputasi

Fleksibilitas TF

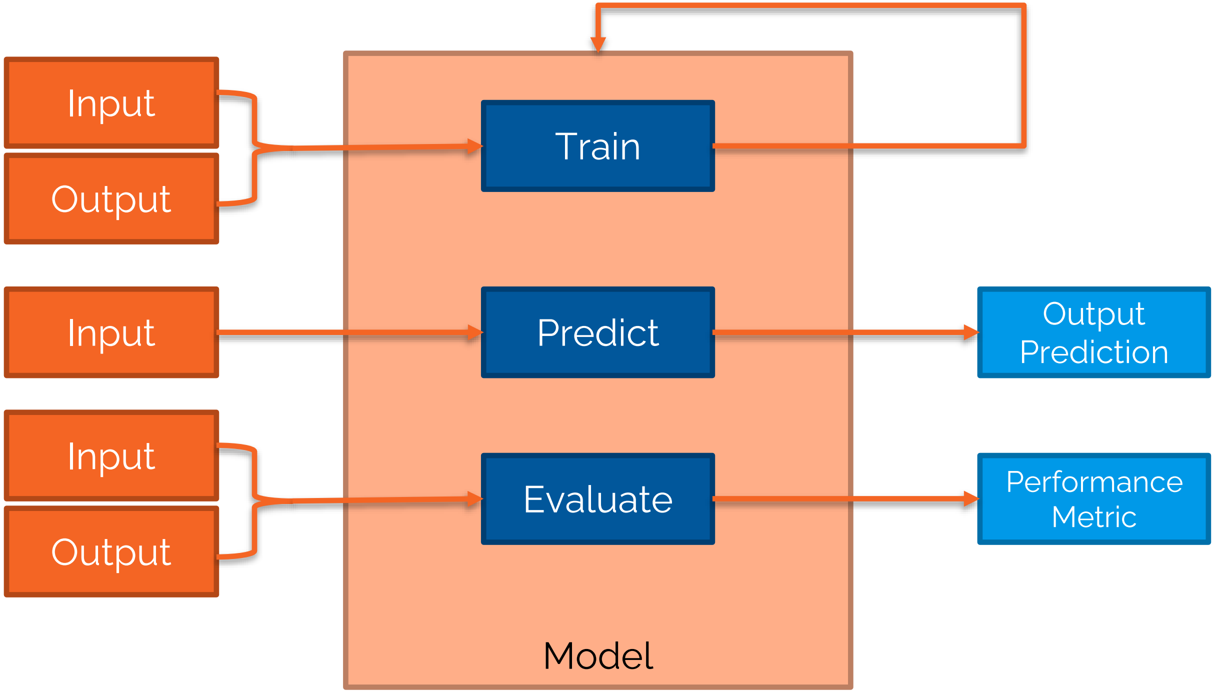


Bagaimana Memakainya?

Apa yang mendefinisikan suatu model NN?



Apa yang mendefinisikan suatu model NN?



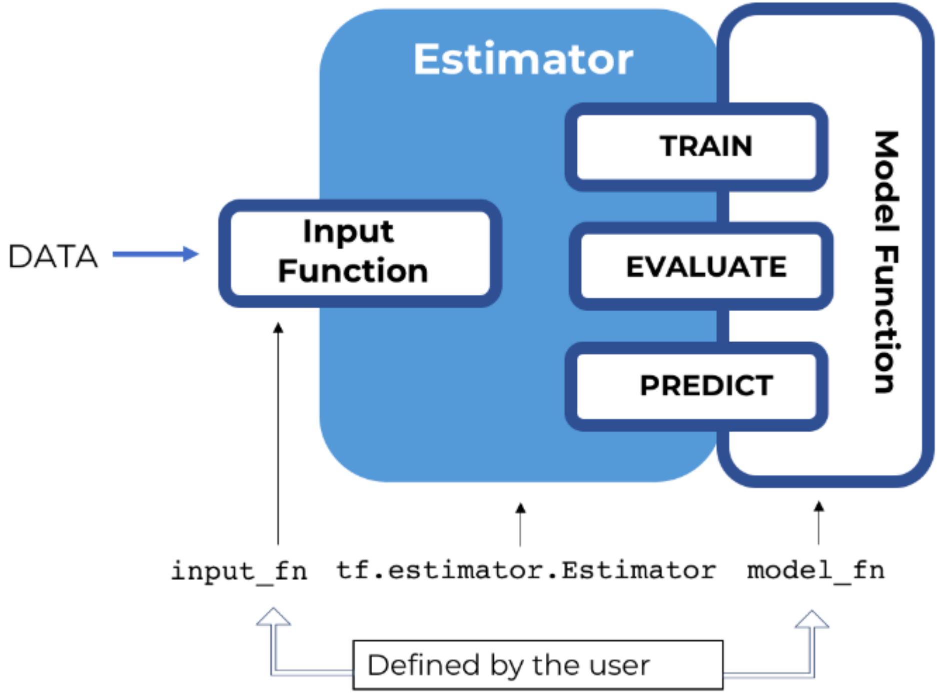
High Level API

Mid Level API

Low Level API

1. Premade Estimator

Estimator adalah kelas di TF yang membungkus suatu model siap pakai.



High Level API

Mid Level API

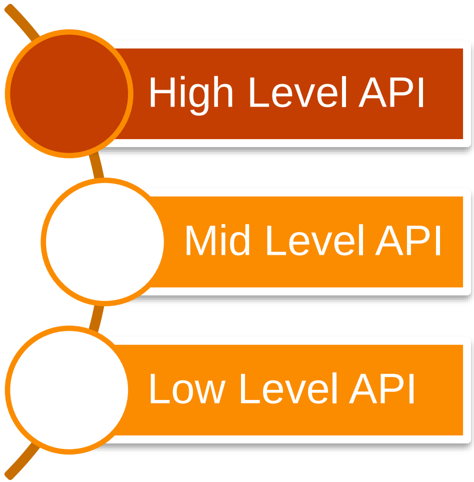
Low Level API

1. Premade Estimator

Beberapa Estimator yang disiapkan TF:

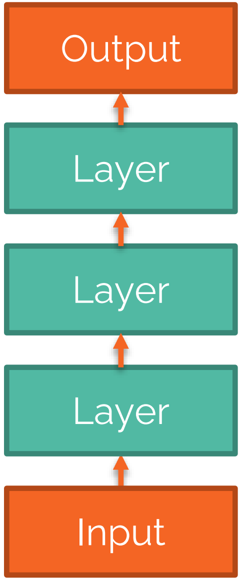
BaselineRegressor, BoostedTreesClassifier, BoostedTreesRegressor, DNNClassifier, DNNLinearCombinedClassifier, DNNLinearCombinedRegressor, DNNRegressor, LinearClassifier, LinearRegressor, etc

```
model = tf.estimator.LinearClassifier(feature_columns=feature_columns)
model.train(train_input_fn)
result = model.evaluate(eval_input_fn)
```



2. Keras Sequential API

Model sekuensial menyediakan konstruksi model yang sederhana dengan cukup menumpuk layer dalam list



High Level API

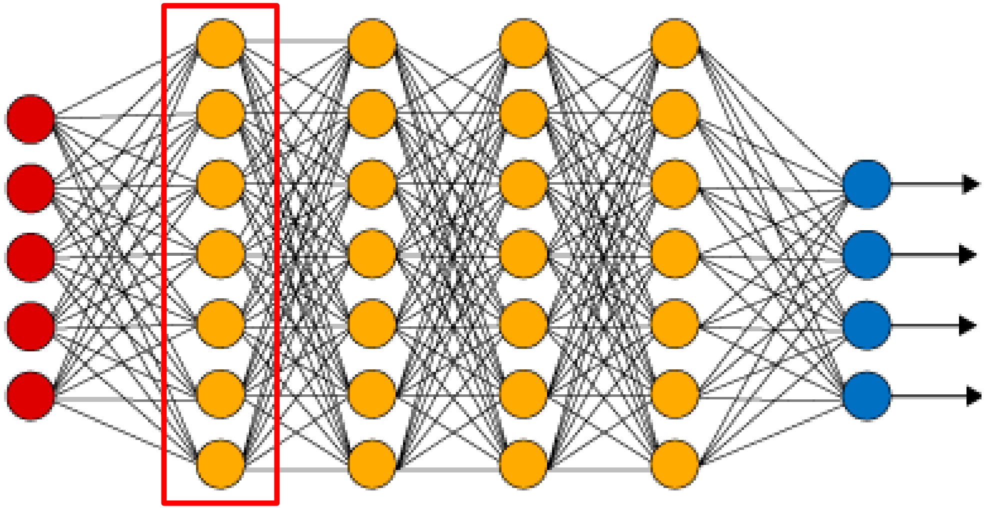
Mid Level API

Low Level API

2. Keras Sequential API

Membangun DNN yang sekuensial, layer demi layer.
Misal untuk *Fully Connected Network* (FCN) dgn 4 Hidden layer:

```
from tf.keras.layers import Dense as D
tf.keras.Sequential([
    D(7, activation='relu', input_shape=(5,)),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(4, activation='softmax')
])
```



High Level API

Mid Level API

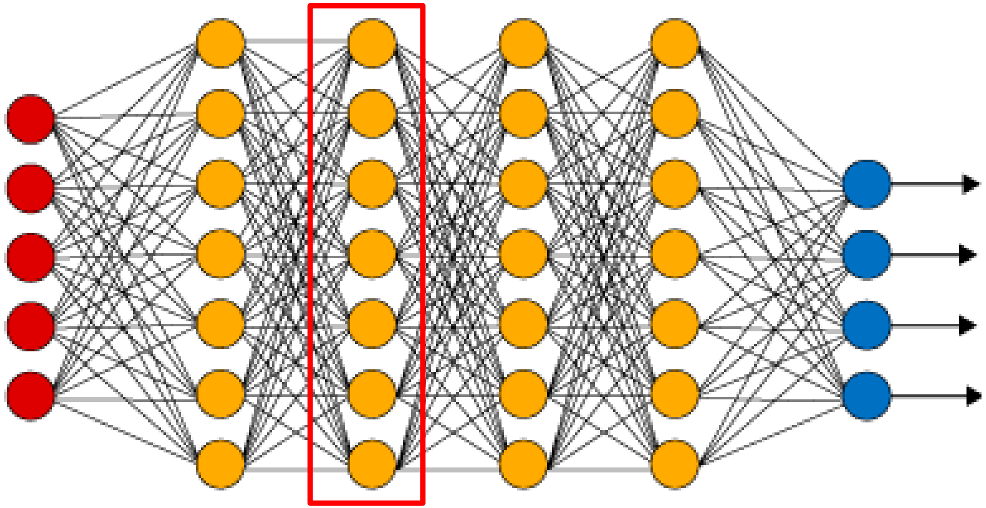
Low Level API

2. Keras Sequential API

Membangun DNN yang sekuensial, layer demi layer.

Misal untuk *Fully Connected Network* (FCN) dgn 4 Hidden layer:

```
from tf.keras.layers import Dense as D
tf.keras.Sequential([
    D(7, activation='relu', input_shape=(5,)),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(4, activation='softmax')
])
```

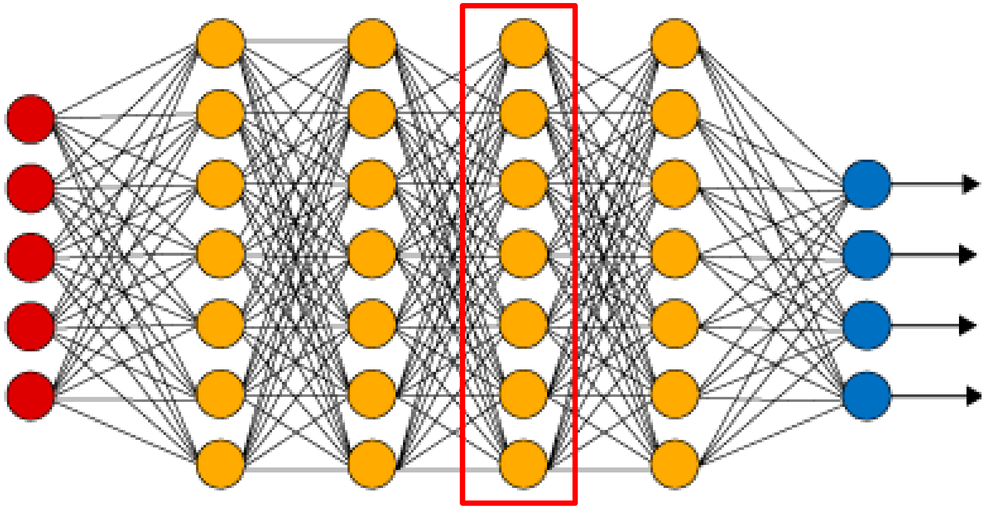


- High Level API
- Mid Level API
- Low Level API

2. Keras Sequential API

Membangun DNN yang sekuensial, layer demi layer.
Misal untuk *Fully Connected Network* (FCN) dgn 4 Hidden layer:

```python
from tf.keras.layers import Dense as D
tf.keras.Sequential([
    D(7, activation='relu', input_shape=(5,)),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(4, activation='softmax')
])
```



High Level API

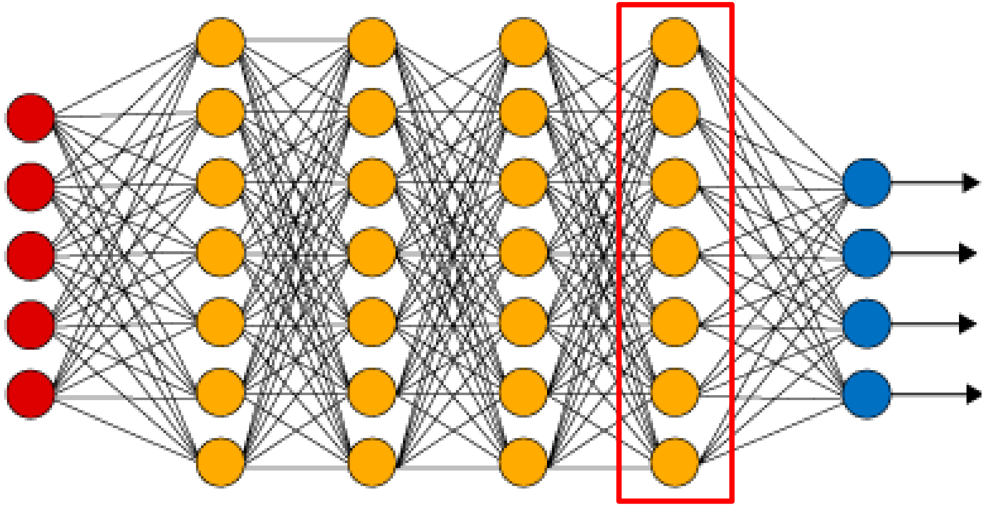
Mid Level API

Low Level API

2. Keras Sequential API

Membangun DNN yang sekuensial, layer demi layer.
Misal untuk *Fully Connected Network* (FCN) dgn 4 Hidden layer:

```
from tf.keras.layers import Dense as D
tf.keras.Sequential([
    D(7, activation='relu', input_shape=(5,)),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(4, activation='softmax')
])
```



High Level API

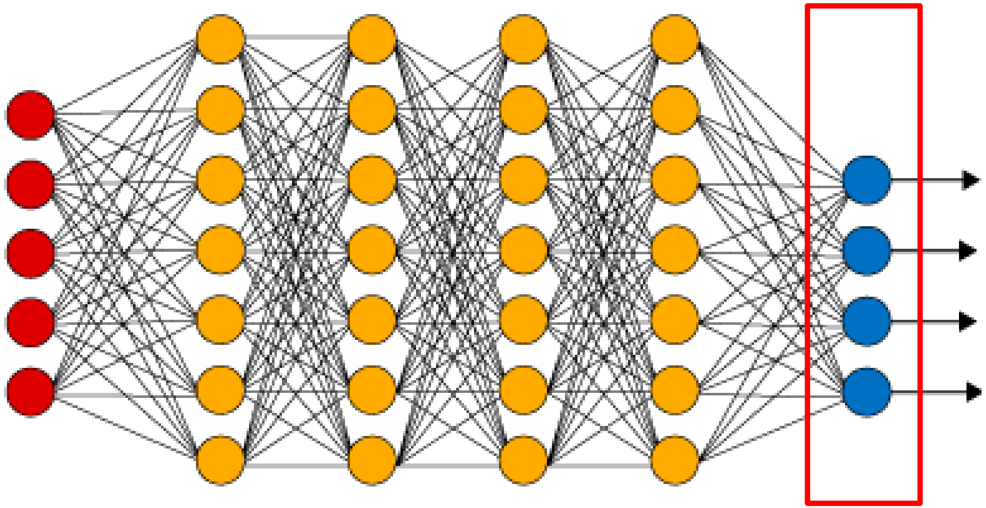
Mid Level API

Low Level API

2. Keras Sequential API

Membangun DNN yang sekuensial, layer demi layer.
Misal untuk *Fully Connected Network* (FCN) dgn 4 Hidden layer:

```
from tf.keras.layers import Dense as D
tf.keras.Sequential([
    D(7, activation='relu', input_shape=(5,)),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(7, activation='relu'),
    D(4, activation='softmax')
])
```



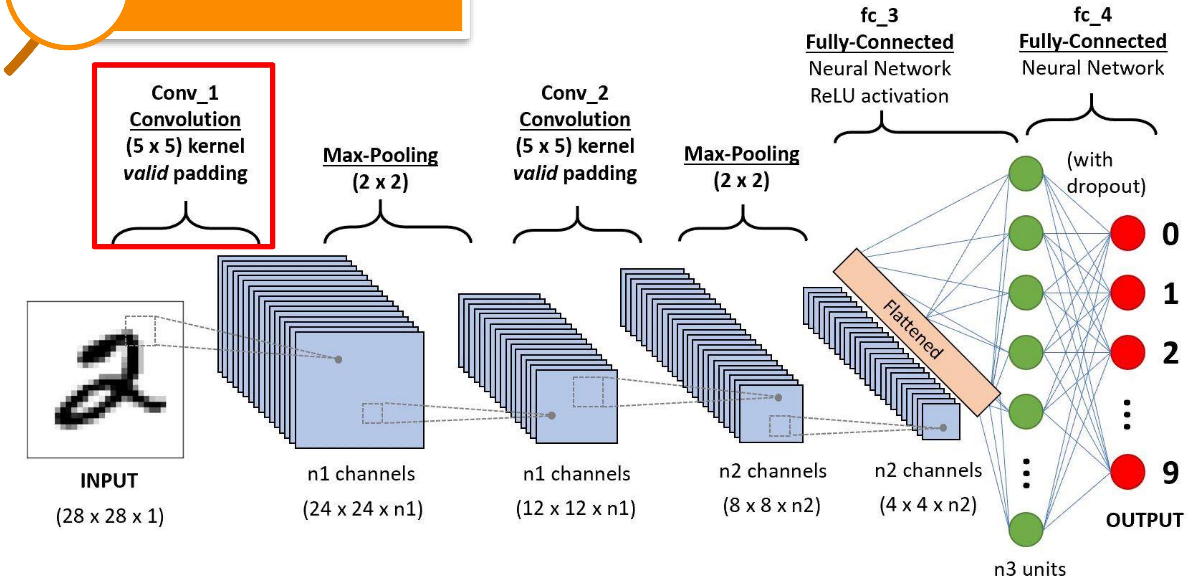
High Level API

Mid Level API

Low Level API

2. Keras Sequential API

Ataupun *Convolutional Neural Network* (CNN) sederhana seperti berikut




```
from tf.keras import layers as L
tf.keras.Sequential([
  L.Conv2D(filters=32,
           kernel_size=(5,5),
           input_shape=(28, 28, 1)),
  L.MaxPool2D((2,2)),
  L.Conv2D(filters=32,
           kernel_size=(5,5)) ,
  L.MaxPool2D((2,2)),
  L.Flatten(),
  L.Dense(256, activation='relu'),
  L.Dense(10, activation='softmax')
])
```

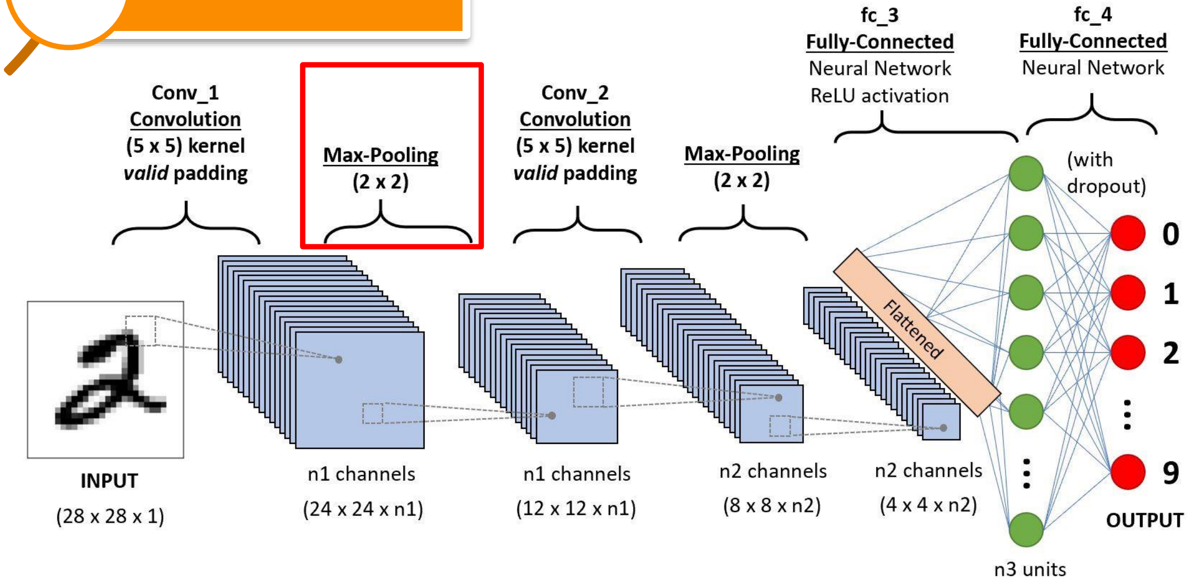
High Level API

Mid Level API

Low Level API

2. Keras Sequential API

Ataupun *Convolutional Neural Network* (CNN) sederhana seperti berikut




```
from tf.keras import layers as L
tf.keras.Sequential([
  L.Conv2D(filters=32,
           kernel_size=(5,5),
           input_shape=(28, 28, 1)),
  L.MaxPool2D((2,2)),
  L.Conv2D(filters=32,
           kernel_size=(5,5)) ,
  L.MaxPool2D((2,2)),
  L.Flatten(),
  L.Dense(256, activation='relu'),
  L.Dense(10, activation='softmax')
])
```

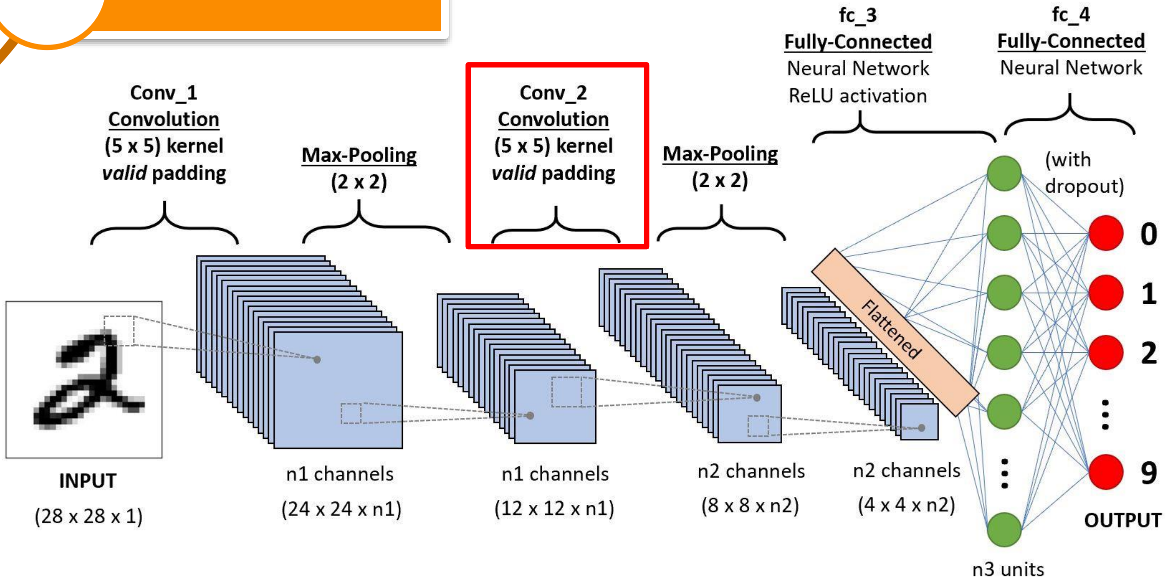
High Level API

Mid Level API

Low Level API

2. Keras Sequential API

Ataupun *Convolutional Neural Network* (CNN) sederhana seperti berikut




```
from tf.keras import layers as L
tf.keras.Sequential([
  L.Conv2D(filters=32,
           kernel_size=(5,5),
           input_shape=(28, 28, 1)),
  L.MaxPool2D((2,2)),
  L.Conv2D(filters=32,
           kernel_size=(5,5)) ,
  L.MaxPool2D((2,2)),
  L.Flatten(),
  L.Dense(256, activation='relu'),
  L.Dense(10, activation='softmax')
])
```

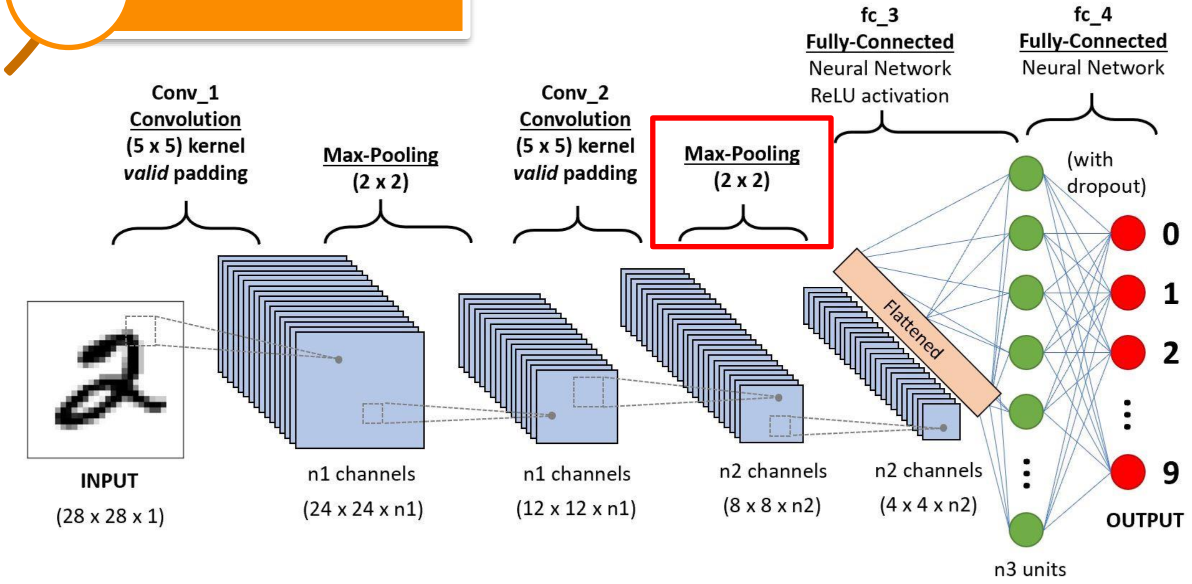
High Level API

Mid Level API

Low Level API

2. Keras Sequential API

Ataupun *Convolutional Neural Network* (CNN) sederhana seperti berikut




```python
from tf.keras import layers as L
tf.keras.Sequential([
  L.Conv2D(filters=32,
           kernel_size=(5,5),
           input_shape=(28, 28, 1)),
  L.MaxPool2D((2,2)),
  L.Conv2D(filters=32,
           kernel_size=(5,5)) ,
  L.MaxPool2D((2,2)),
  L.Flatten(),
  L.Dense(256, activation='relu'),
  L.Dense(10, activation='softmax')
])
```

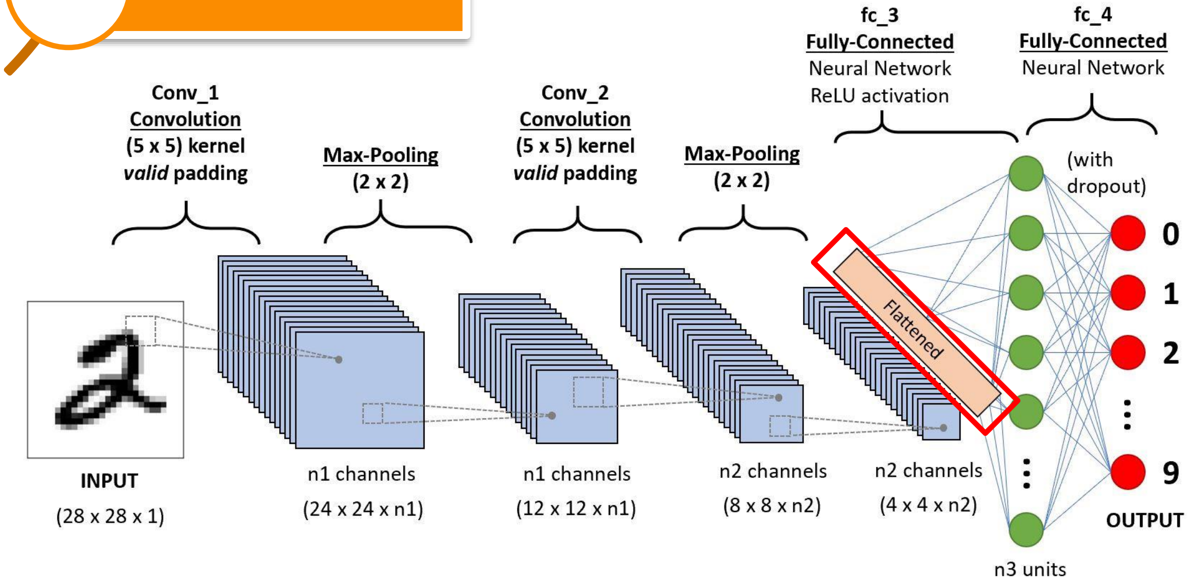
High Level API

Mid Level API

Low Level API

2. Keras Sequential API

Ataupun *Convolutional Neural Network* (CNN) sederhana seperti berikut




```
from tf.keras import layers as L
tf.keras.Sequential([
  L.Conv2D(filters=32,
           kernel_size=(5,5),
           input_shape=(28, 28, 1)),
  L.MaxPool2D((2,2)),
  L.Conv2D(filters=32,
           kernel_size=(5,5)) ,
  L.MaxPool2D((2,2)),
  L.Flatten(),
  L.Dense(256, activation='relu'),
  L.Dense(10, activation='softmax')
])
```

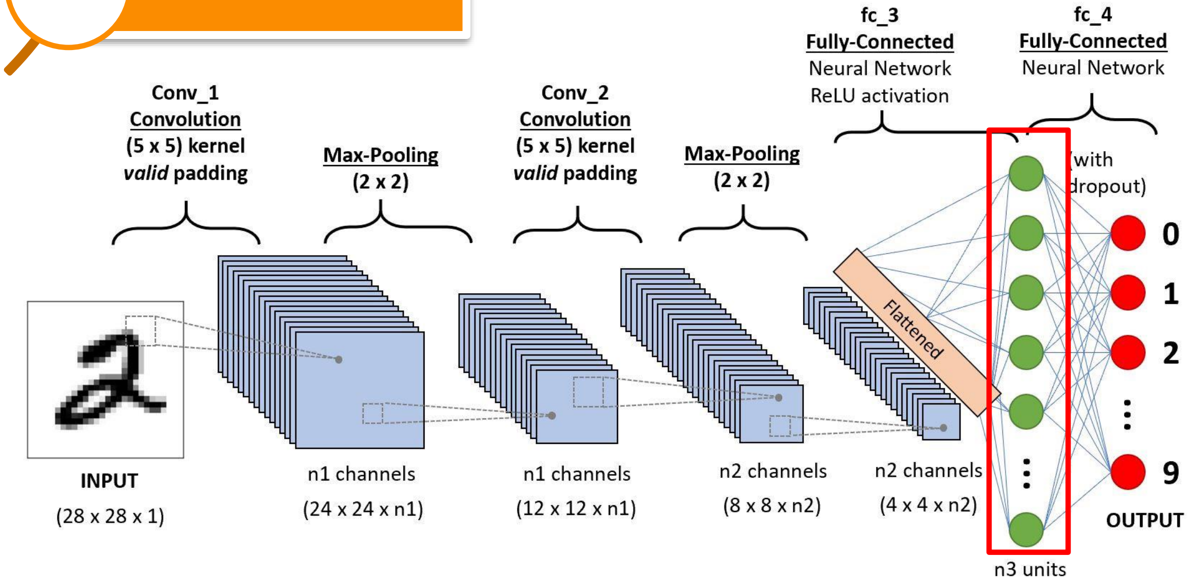
High Level API

Mid Level API

Low Level API

2. Keras Sequential API

Ataupun *Convolutional Neural Network* (CNN) sederhana seperti berikut




```
from tf.keras import layers as L
tf.keras.Sequential([
  L.Conv2D(filters=32,
           kernel_size=(5,5),
           input_shape=(28, 28, 1)),
  L.MaxPool2D((2,2)),
  L.Conv2D(filters=32,
           kernel_size=(5,5)) ,
  L.MaxPool2D((2,2)),
  L.Flatten(),
  L.Dense(256, activation='relu'),
  L.Dense(10, activation='softmax')
])
```

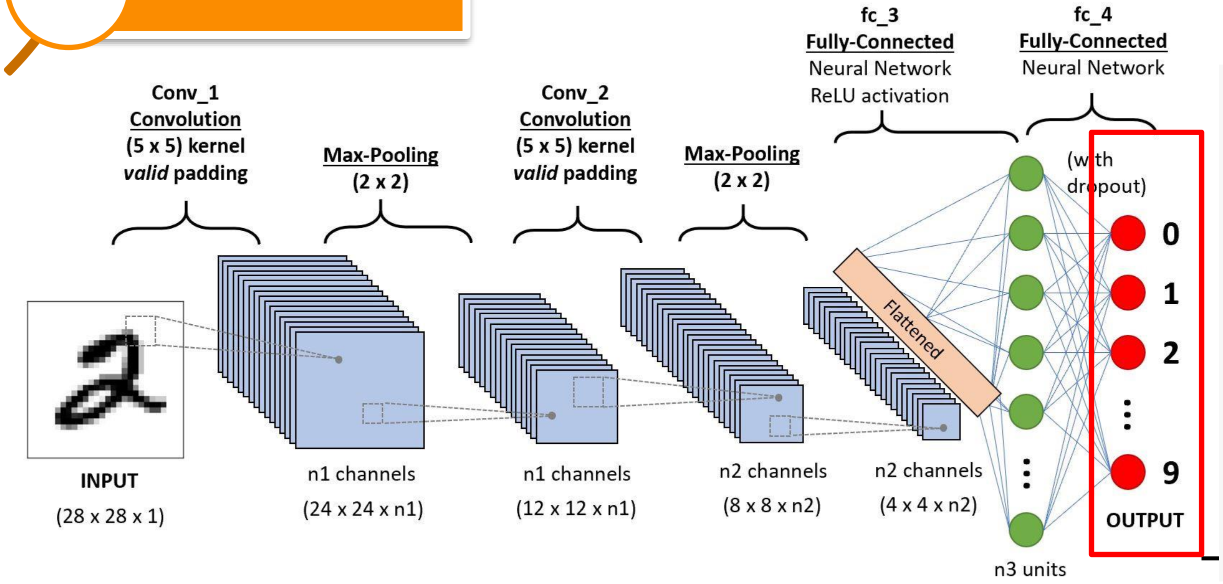
High Level API

Mid Level API

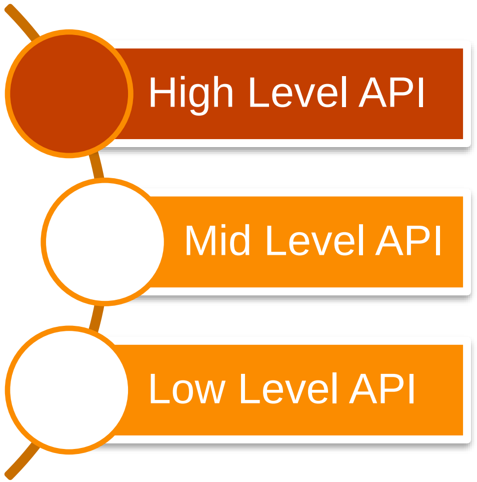
Low Level API

2. Keras Sequential API

Ataupun *Convolutional Neural Network* (CNN) sederhana seperti berikut




```
from tf.keras import layers as L
tf.keras.Sequential([
  L.Conv2D(filters=32,
           kernel_size=(5,5),
           input_shape=(28, 28, 1)),
  L.MaxPool2D((2,2)),
  L.Conv2D(filters=32,
           kernel_size=(5,5)) ,
  L.MaxPool2D((2,2)),
  L.Flatten(),
  L.Dense(256, activation='relu'),
  L.Dense(10, activation='softmax')
])
```



2. Keras Sequential API

Beberapa layer yang tersedia:

- Dense
- Batch Normalization
- Dropout

(Khusus CNN)

- Convolutional (1D/2D/3D, Transpose/Separable)
- Pooling (Max/Avg, Global, 1D/2D/3D)
- Cropping (1D/2D/3D)

(Khusus RNN)

- Embedding
- SimpleRNN
- GRU
- LSTM
- Bidirectional

dll

High Level API

Mid Level API

Low Level API

2. Keras Sequential API

Proses kompilasi dan melatih model dapat dilakukan dengan sederhana

```
model.compile(optimizer='adam', loss='mae')
model.fit(x, y, batch_size=32, epochs=50)
```

High Level API

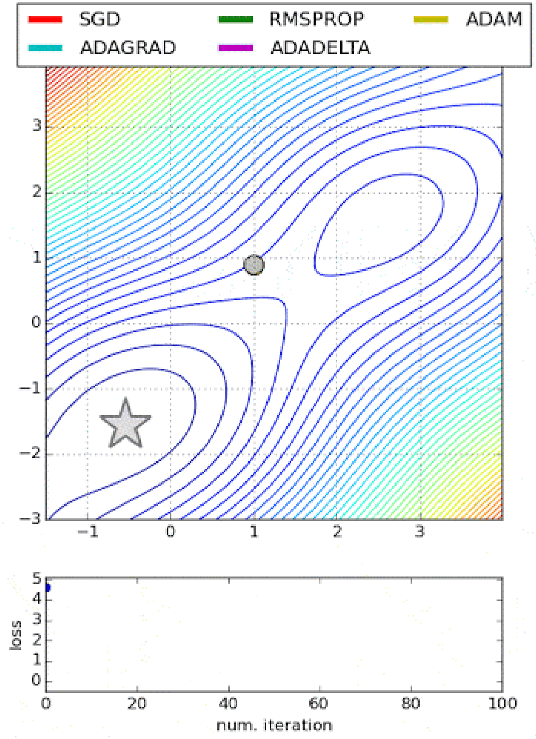
Mid Level API

Low Level API

2. Keras Sequential API

Beberapa optimizer yang tersedia:

- SGD (Stochastic Gradient Descent)
- Adam (Adaptive Moment Estimation)
- RMSProp (Root Mean Squared Propagation)
- AdaGrad (Adaptive Gradient Descent)
- AdaDelta (AdaGrad with Delta)



High Level API

Mid Level API

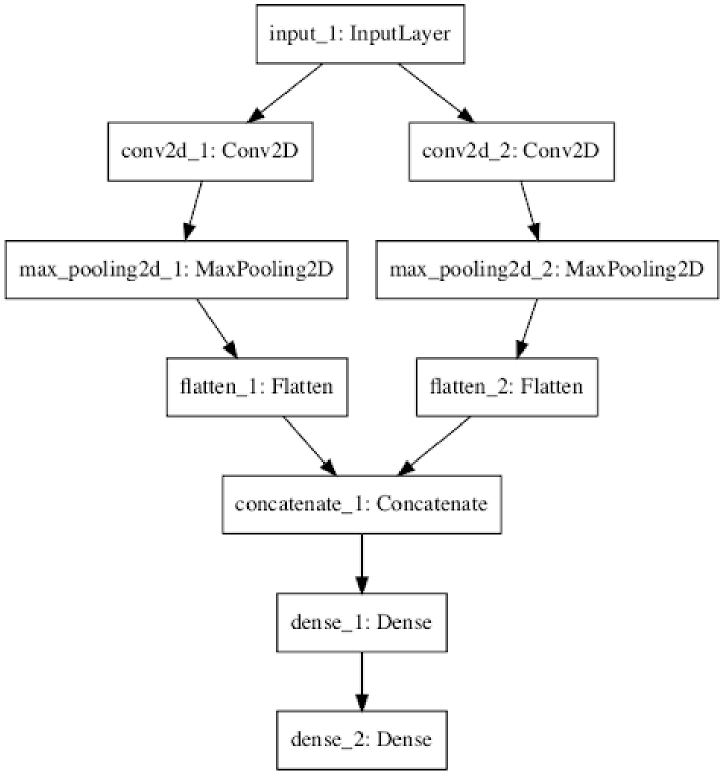
Low Level API

2. Keras Sequential API

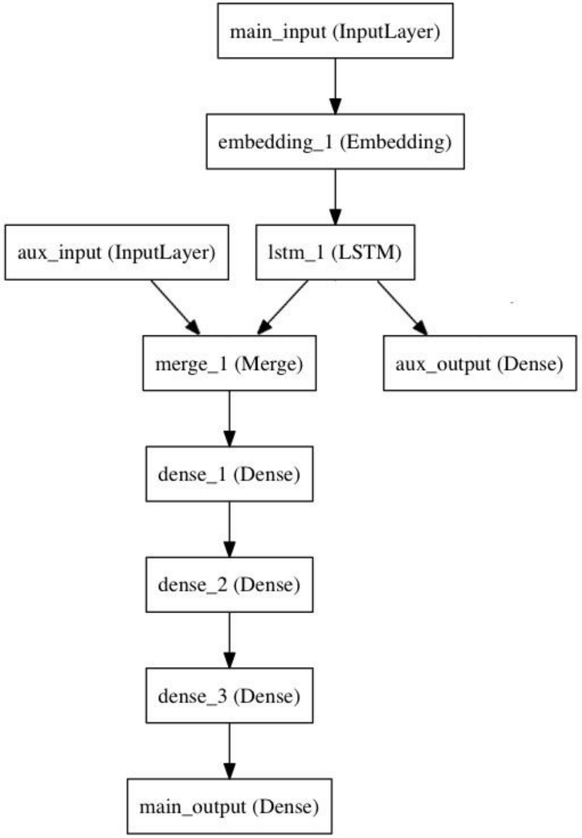
Beberapa fungsi loss yang tersedia:

- Binary & Categorical Crossentropy
- MSE
- RMSE
- MAE
- Huber
- Cosine Similarity

Bagaimana kalau kita punya model seperti ini



Atau ini?



Model DNN “lurus” begitu saja tidak akan banyak bermanfaat.

Selain model sekuensial, TF memiliki dua cara lain untuk membangun model: *functional API* dan *model subclassing*

High Level API

Mid Level API

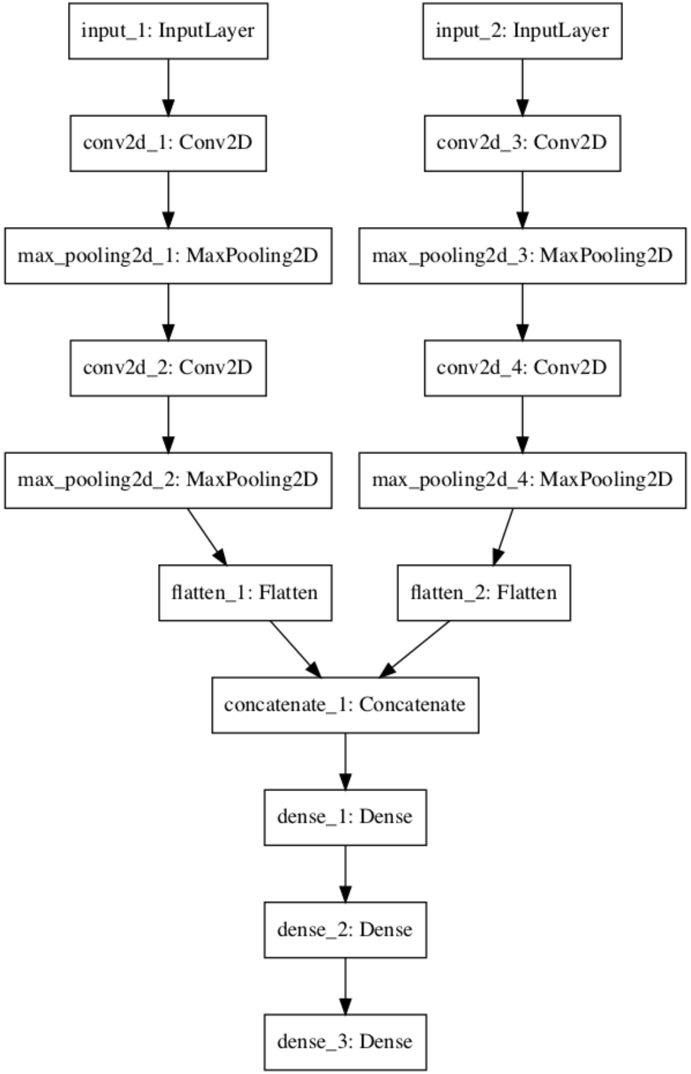
Low Level API

1. Keras Functional API

Membangun model dengan **memerlakukan setiap layer seperti fungsi**, yang secara fleksibel bisa ditentukan inputnya dari layer mana ke layer mana.

Model cukup didefinisikan dengan menetapkan input dan outputnya

```
input = tf.keras.Input((10,))
dense1 = tf.keras.layers.Dense(32, activation='relu')(input)
dense2 = tf.keras.layers.Dense(32, activation='relu')(dense1)
output = tf.keras.layers.Dense(1)(dense2)

model = tf.keras.Model(inputs=input, output=output)
```



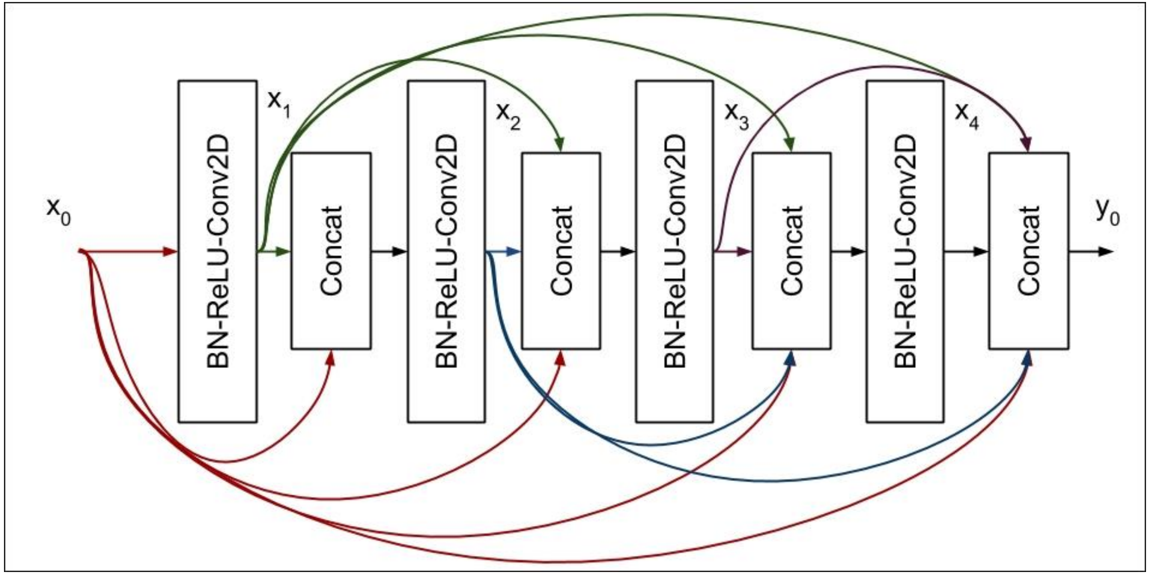

```python
from tf.keras import layers as L

input1 = Input((64,64,1))
conv11 = L.Conv2D(32, kernel_size=4, activation='relu')(input1)
pool11 = L.MaxPooling2D(pool_size=(2, 2))(conv11)
conv12 = L.Conv2D(16, kernel_size=4, activation='relu')(pool11)
pool12 = L.MaxPooling2D(pool_size=(2, 2))(conv12)
flat1 = Flatten()(pool12)

input2 = Input((32,32,3))
conv21 = L.Conv2D(32, kernel_size=4, activation='relu')(input2)
pool21 = L.MaxPooling2D(pool_size=(2, 2))(conv21)
conv22 = L.Conv2D(16, kernel_size=4, activation='relu')(pool21)
pool22 = L.MaxPooling2D(pool_size=(2, 2))(conv22)
flat2 = L.Flatten()(pool22)

merge = L.Concatenate()([flat1, flat2])

hidden1 = Dense(10, activation='relu')(merge)
hidden2 = Dense(10, activation='relu')(hidden1)
output = Dense(1, activation='sigmoid')(hidden2)
model = Model(inputs=[input1, input2], outputs=output)
```

High Level API

Mid Level API

Low Level API

Bagaimana kalau modelnya seperti ini?



Tentu dimungkinkan dengan Functional API, tapi kodenya akan sangat kompleks dan panjang

High Level API

Mid Level API

Low Level API

2. Model Subclassing

Membangun model dengan membuat kelas turunan Keras Model secara manual.

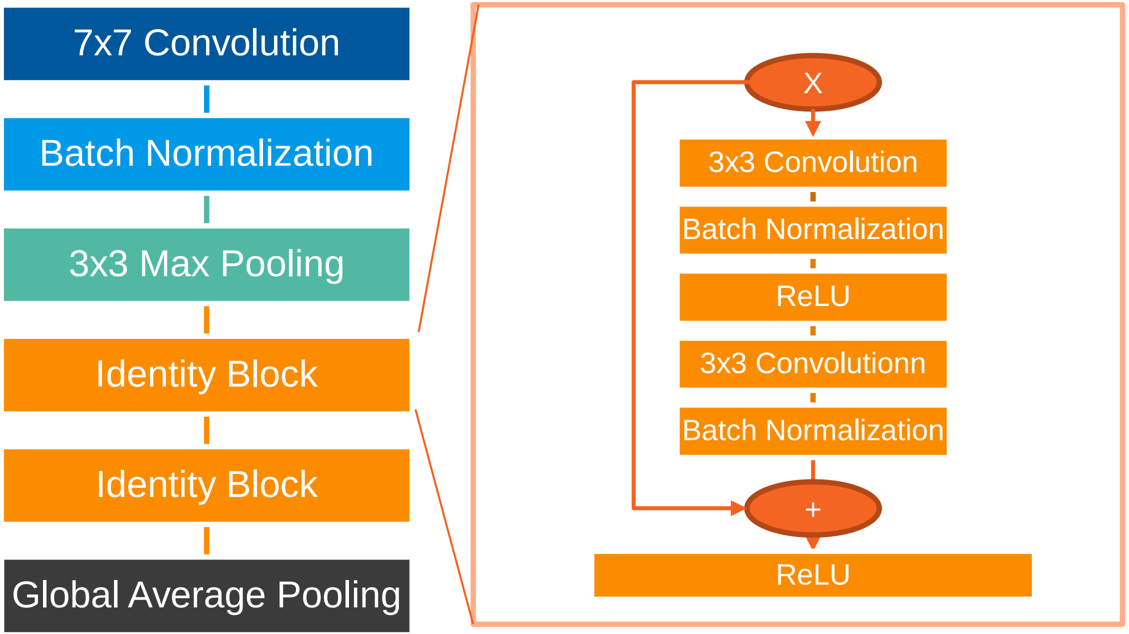
Bermanfaat terutama apabila terdapat banyak “blok” di dalam modelnya yang bisa dianggap sebagai submodel

```
class CustomModel(tf.keras.Model):
    def __init__(self, *args, **kwargs):
        super(CustomModel, self).__init__(name='')
        # di sini kita inisialisasi semua layer yang dibutuhkan

    def call(self, input):
        # di sini ditentukan apa yang akan dilakukan
        # bila model dipanggil dengan suatu input tensor
        output = None
        return output
```

High Level API
Mid Level API
Low Level API

2. Model Subclassing

Misal kita ingin bangun model seperti ini



High Level API

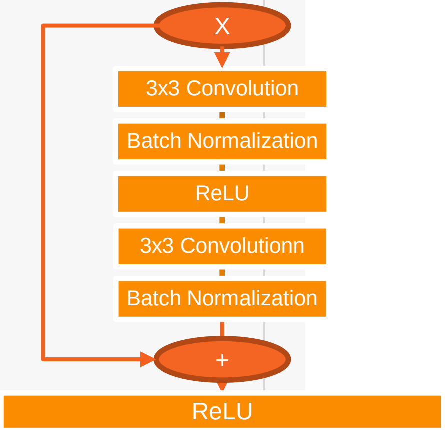
Mid Level API

Low Level API

2. Model Subclassing

Mula-mula, definisikan Identity Block sebagai sebuah model sendiri

```python
class IdentityBlock(tf.keras.Model):
    def __init__(self, filters, kernel_size):
        super(IdentityBlock, self).__init__(name='')
        self.conv1 = tf.keras.layers.Conv2D(filters, kernel_size, padding='same')
        self.bn1 = tf.keras.layers.BatchNormalization()
        self.conv2 = tf.keras.layers.Conv2D(filters, kernel_size, padding='same')
        self.bn2 = tf.keras.layers.BatchNormalization()
        self.act = tf.keras.layers.Activation('relu')
        self.add = tf.keras.layers.Add()

    def call(self, input_tensor):
        x = self.conv1(input_tensor)
        x = self.bn1(x)
        x = self.act(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.add([x, input_tensor])
        x = self.act(x)
        return x
```



High Level API

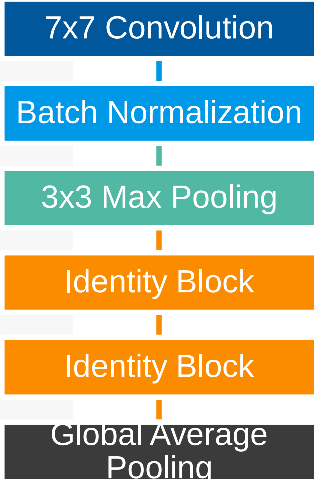
Mid Level API

Low Level API

2. Model Subclassing

Kemudian gunakan blok tersebut di dalam model yang sesungguhnya

```python
class MiniResNet(tf.keras.Model):
    def __init__(self, num_classes):
        super(MiniResNet, self).__init__()
        self.conv = tf.keras.layers.Conv2D(64, 7, padding='same')
        self.bn = tf.keras.layers.BatchNormalization()
        self.act = tf.keras.layers.Activation('relu')
        self.max_pool = tf.keras.layers.MaxPool2D((3, 3))
        self.id1a = IdentityBlock(64, 3)
        self.id1b = IdentityBlock(64, 3)
        self.global_pool = tf.keras.layers.GlobalAveragePooling2D()
        self.classifier = tf.keras.layers.Dense(num_classes, activation='softmax')

    def call(self, inputs):
        x = self.conv(inputs)
        x = self.bn(x)
        x = self.act(x)
        x = self.max_pool(x)
        x = self.id1a(x)
        x = self.id1b(x)
        x = self.global_pool(x)
        return self.classifier(x)
```

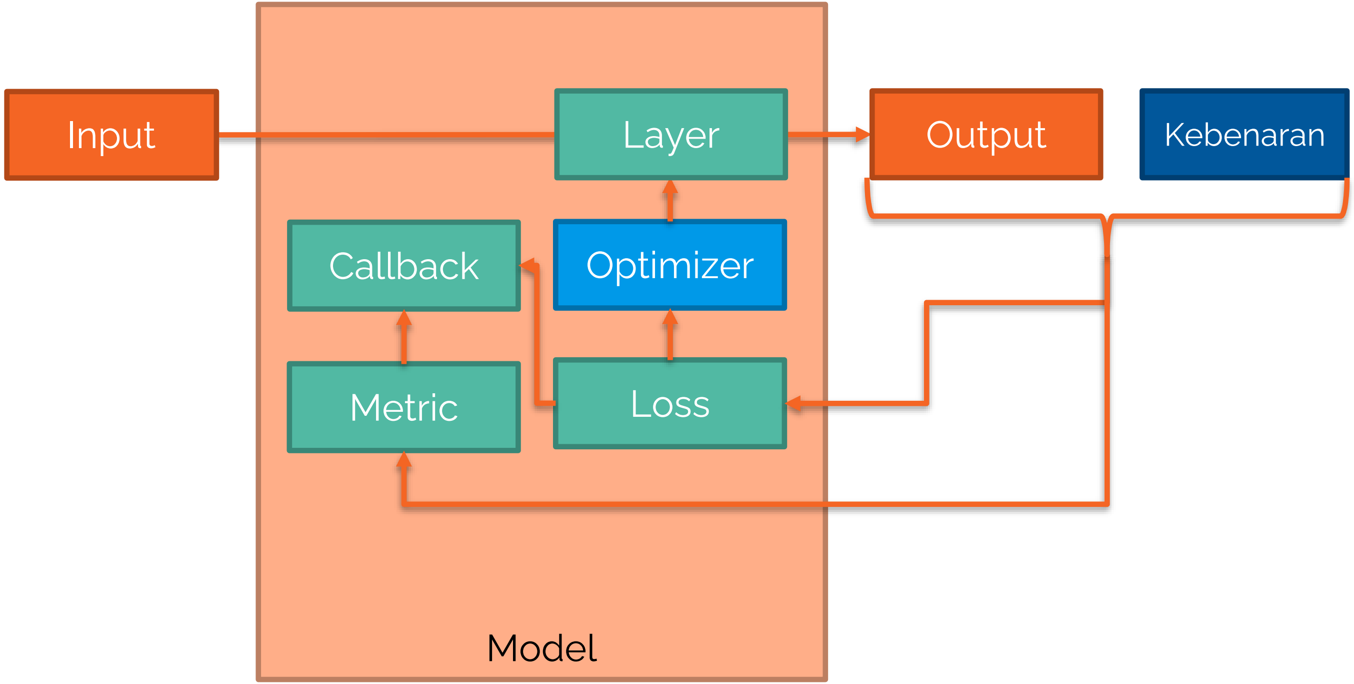


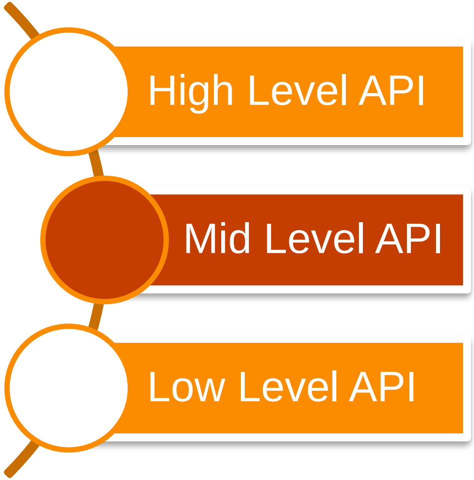
High Level API

Mid Level API

Low Level API

3. Custom components





3. Custom components

Semua komponen yang dibutuhkan ketika melatih model DNN:

Layer, *Loss*, *Metric*, dan juga *Callback*

bisa dibuat manual sesuai kebutuhan.
Khusus optimizer, override-nya harus dilakukan di low level API

High Level API

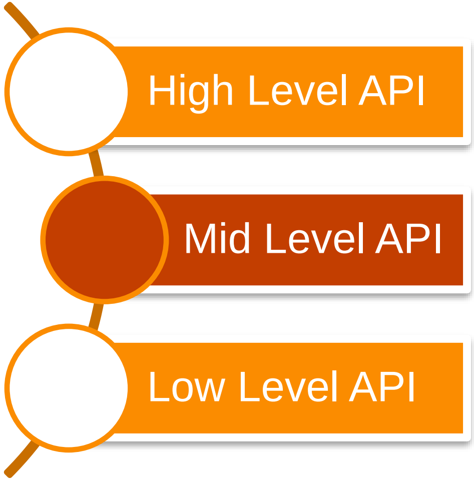
Mid Level API

Low Level API

3. Custom components

Format dasar custom layer:

```
class CustomLayer(tf.keras.layers.Layer):
    def __init__(self, *args, **kwargs):
        # inisiasi atribut
    def build(self, input_shape):
        # membangun parameter yang dipakai
    def call(self, inputs):
        # mengolah input jadi output dari layer ini
        return outputs
```



3. Custom components

Format dasar custom loss:

```
class CustomLoss(tf.keras.losses.Loss):
    def __init__(self, *args, **kwargs):
        # inisiasi atribut
    def call(self, y_true, y_pred):
        # menghitung loss dari label prediksi
        # dan label yang benar
        return loss
```

High Level API

Mid Level API

Low Level API

3. Custom components

Format dasar custom metric:

```
class CustomMetric(tf.keras.metrics.Metric):
    def __init__(self, *args, **kwargs):
        # inisiasi atribut
    def update_state(self, *args, **kwargs):
        # menghitung dan menambahkan nilai metrik
        # sesuai input yang diberikan
    def result(self):
        # mengembalikan akumulasi dari metrik yang
        # sudah dihitung
```

High Level API

Mid Level API

Low Level API

3. Custom components

Format dasar custom callback:

```
class CustomCallback(tf.keras.callbacks.Callback):
    def __init__(self, *args, **kwargs):
        # inisiasi atribut

    def on_epoch_begin(self, epoch, logs):
        # callback ketika epoch dimulai

    def on_epoch_end(self, epoch, logs):
        # callback ketika epoch berakhir

    def on_batch_begin(self, batch, logs):
        # callback ketika batch dimulai

    def on_batch_end(self, batch, logs):
        # callback ketika batch berakhir
```

High Level API

Mid Level API

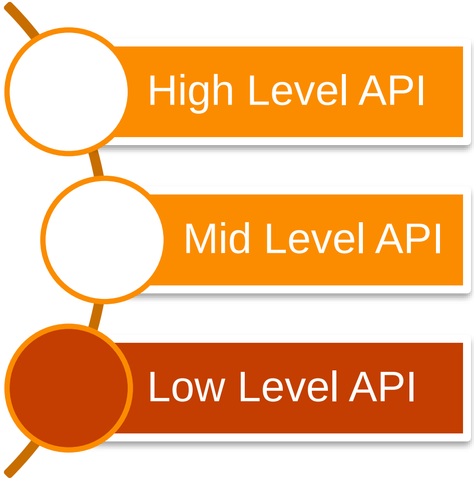
Low Level API

1. Gradient Tape

Gradient Tape merekam semua operasi yang dilakukan dan secara otomatis menghitung turunannya. Sangat memudahkan proses *backpropagation*

```python
x = tf.ones((2,2))
with tf.GradientTape() as t:
    t.watch(x)
    y = tf.reduce_sum(x)
    z = tf.square(y)
dz_dx = t.gradient(z, x)
print('dz_dx =', dz_dx.numpy())
```

```
dz_dx = [[8. 8.]
 [8. 8.]]
```



2. Dataset API

Tensorflow punya API khusus untuk mengelola data pipeline.
Misal, kita punya data time series, yang mau diolah menjadi input suatu model

```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer).map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

High Level API

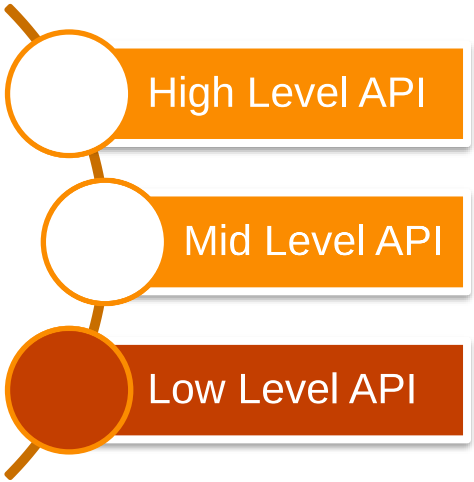
Mid Level API

Low Level API

2. Dataset API

```
array = np.arange(0, 20)
print(array)
dataset = windowed_dataset(array, 4, 3, 1000)
print("processed:")
for x, y in dataset.take(1):
  for i in range(len(x)):
    print(x.numpy()[i], "->", y.numpy()[i])
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
processed:
[1 2 3 4] -> 5
[3 4 5 6] -> 7
[15 16 17 18] -> 19
```



Dan masih banyak lagi

Low level API berarti semuanya serba *custom* dan *self-build*, memanfaatkan seluruh fitur inti dari Tensorflow.

Mau pakai yang mana?
Apapun modelnya, frameworknya Tensorflow

Yuk, coba dulu

bit.ly/hands-on-tf

Any Question?

TensorFlow™