


# Keras Basics

Dong Kook Kim

# Reference

<https://keras.io/>

 Keras Documentation

Search docs

⊞ Home

Why use Keras

GETTING STARTED

Guide to the Sequential model

Guide to the Functional API

FAQ

MODELS

About Keras models

Sequential


Model (functional API)

LAYERS

About Keras layers

Core Layers

Docs » Home

 [Edit on GitHub](#)

## Keras: The Python Deep Learning library



# Keras

### You have just found Keras.

Keras is a high-level neural networks API, written in Python and capable of running on top of **TensorFlow**, **CNTK**, or **Theano**. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

# Book for Keras

- 프랑소와 솔레, "케라스 창시자에게 배우는 딥러닝 (Deep Learning with Python)", 2019, 길벗



- Best Keras book to read

# What is Keras ?

- A high-level neural networks API in Python
- Developed with a focus on enabling fast experimentation
- Runs on top of [TensorFlow](#), [CNTK](#), or [Theano](#)
- Modular – Building model is just stacking layers and connecting computational graphs

# Why use Keras ?

- Useful for **easy** and **fast** prototyping (through user friendliness, modularity, and extensibility)
- Supports both CNN and RNN, as well as combinations of the two
- Runs seamlessly on CPU and GPU
- Almost any architecture can be designed using this framework
- Open source code – Large community support

# TensorFlow and Keras

- **tf.keras** is TensorFlow's implementation of the Keras API specification

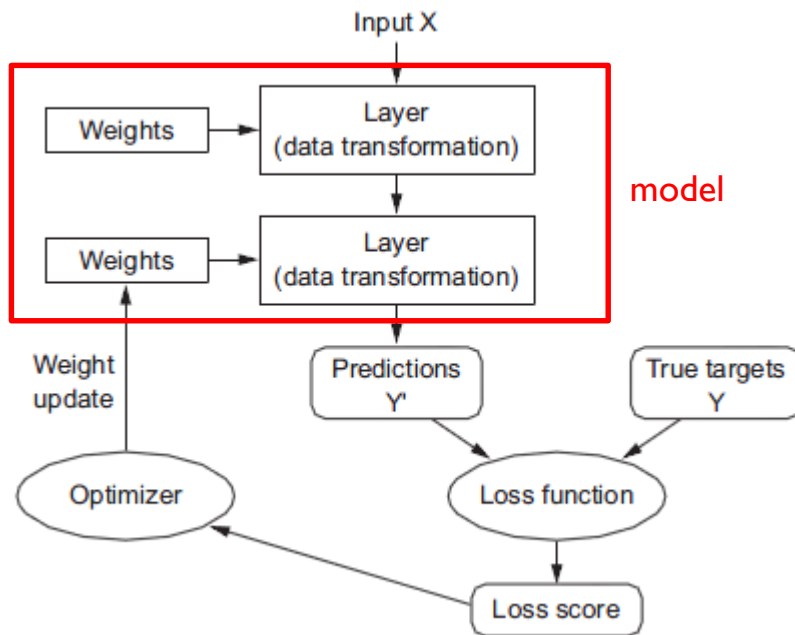
```
import tensorflow as tf  
  
print(tf.__version__)  
print(tf.keras.__version__)
```

# 5 Steps for Learning in Keras

- Step 1. **Data** – preparing the data (input and target) and preprocessing it
- Step 2. **Model** – defining the model architecture and build the computational graph
- Step 3. **Loss function and Optimizers** - specify the loss function and optimizer and configure the learning process
- Step 4. **Train and evaluate the model**
- Step 5. **Test the model**

# 5 Steps for Learning in Keras

- Relationship between the **model, layers, loss function, optimizer**





# 5 Steps for Learning in Keras

Step 1.

Prepare Data,  
Preprocessing

- Input/Target
- images
- texts
- videos
- audios

- load\_data()

Step 2.

Define the  
model

- Sequential  
Functional API
- Layers
- MLP
- CNN
- RNN

- Sequential( )  
- add()  
- Dense()  
...

Step 3.

Loss function  
Optimizer

- MSE
- Cross  
entropy
- SGD
- RMSprop
- Adam

- compile( )

Step 4.

Train the  
model

- fit( )

Step 5.

Test the  
model

- evaluate( )

Keras  
APIs

# Exercise 01-7.

`tf2-01-7-keras_mnist_example.py`

# 5 Steps using Keras : MNIST example

- **Step 0.** Import tensorflow `import tensorflow as tf`

- **Step 1. Data**

- load data : MNIST

```
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()
```

- preprocessing data

```
train_x = train_x.reshape(len(train_x), -1).astype('float32') / 255  
test_x = test_x.reshape(len(test_x), -1).astype('float32') / 255
```

```
train_y = tf.keras.utils.to_categorical(train_y)  
test_y = tf.keras.utils.to_categorical(test_y)
```

# 5 Steps using Keras : MNIST example

- Step 2. Model

```
model = tf.keras.models.Sequential()  
model.add(tf.keras.layers.Dense(512,input_shape=(784,),  
activation=tf.nn.relu))  
model.add(tf.keras.layers.Dense(10,activation=tf.nn.softmax))
```

- Step 3. Loss function and Optimizer

```
model.compile(optimizer= 'rmsprop',  
              loss='categorical_crossentropy', metrics = ['accuracy'])
```

# 5 Steps using Keras : MNIST example

- Step 4. Train and evaluate the model

```
model.fit(train_x, train_y, batch_size=100, epochs=20)  
print(model.summary())
```

- Step 5. Test the model

```
test_loss, test_acc = model.evaluate(test_x, test_y)  
print('test_loss = ', test_loss, 'test_acc = ', test_acc)
```

# Step I : Keras Datasets

DataSets	input	target	Train/test	Task
MNIST	gray images (28, 28, 1)	10 classes	60,000/ 10,000	handwritten digits classification
Fashion- MNIST	gray images (28, 28, 1)	10 classes	60,000/ 10,000	10 fashion categories classification
CIFAR10	color images (32, 32, 3)	10 classes	50,000/ 10,000	small image classification
CIFAR100	color images (32, 32, 3)	100 classes	50,000/ 10,000	small image classification
IMDB	Text data - variable seq	2 classes	25,000/ 25,000	movie reviews sentiment classification
Reuters	Test data - variable seq	46 topics	11,228/ 11,228	newswire topic classification
Boston housing price	Real data - 13-dim. vector	Median values of house	404/102	Boston housing price regression

# Step I : Keras Datasets

1. MNIST – handwritten digits classification, gray scale image
  - `tf.keras.dataset.mnist`
2. Fashion-MNIST – images of 10 fashion categories
  - `tf.keras.dataset.fashion_mnist`
3. CIFAR10 and CIFAR100 – small image classification
  - `tf.keras.dataset.cifar10`, - `tf.keras.dataset.cifar100`
4. IMDB – movie reviews sentiment classification
  - `tf.keras.dataset.imdb`
5. Reuters – newswire topic classification
  - `tf.keras.dataset.reuters`
6. Boston housing price regression dataset
  - `tf.keras.dataset.boston_housing`

## Exercise 01-8.

tf2-01-8-keras\_mnist\_display.py

tf2-01-9-keras\_fashion\_mnist\_display.py

tf2-01-10-keras\_cifar10\_display.py

tf2-01-11-keras\_cifar100\_display.py

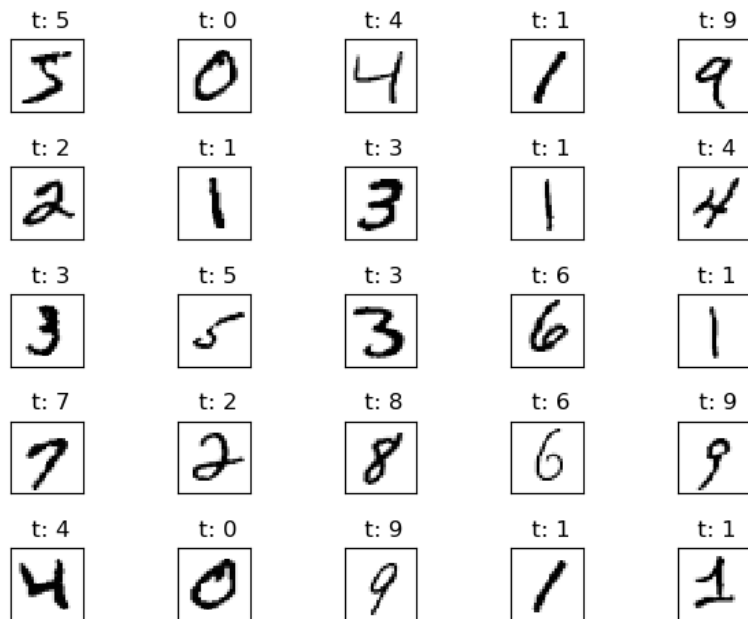
tf2-01-12-keras\_imdb\_display.py

tf2-01-13-keras\_boston\_display.py

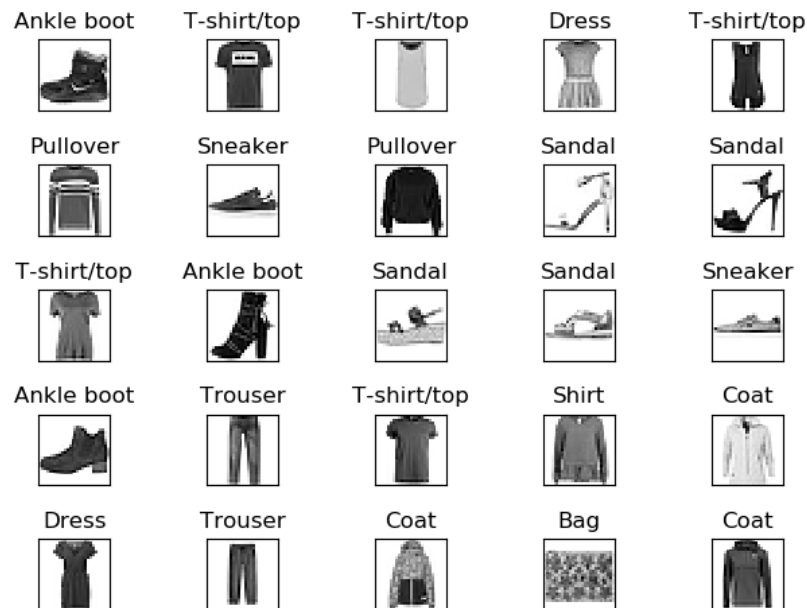


# MNIST vs Fashion MNIST

## ● MNIST



## ● Fashion-MNIST

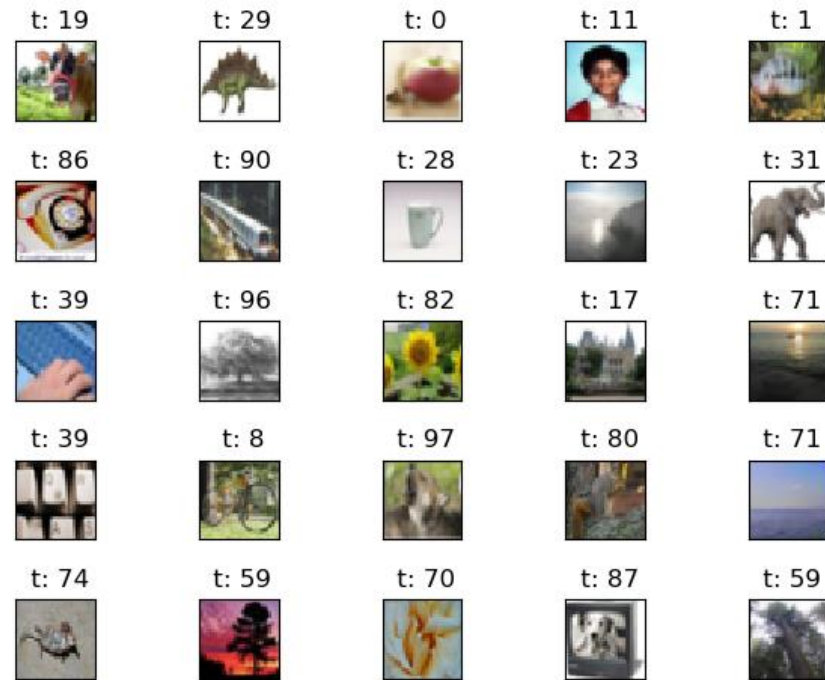


# CIFAR10 vs CIFAR100

## ● CIFAR10



## ● CIFAR100



# IMDB

- decoded training data 1 - : target - **positive sentiment**

this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert is an amazing actor and now the same being director father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for retail and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also congratulations to the two little boy's that played the of norman and paul they were just brilliant children are often left out of the praising list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all

- decoded training data 2 - : target - **negative sentiment**

big hair big boobs bad music and a giant safety pin these are the words to best describe this terrible movie i love cheesy horror movies and i've seen hundreds but this had got to be on of the worst ever made the plot is paper thin and ridiculous the acting is an abomination the script is completely laughable the best is the end showdown with the cop and how he worked out who the killer is it's just so damn terribly written the clothes are sickening and funny in equal measures the hair is big lots of boobs bounce men wear those cut tee shirts that show off their sickening that men actually wore them and the music is just trash that plays over and over again in almost every scene there is trashy music boobs and taking away bodies and the gym still doesn't close for all joking aside this is a truly bad film whose only charm is to look back on the disaster that was the 80's and have a good old laugh at how bad everything was back then

# Boston Housing Price

- Input : 13-dimensional vector, Target : scalar

```
[ 1.23247 0.      8.14   0.     0.538  6.142  91.7  3.9769  4.    307.    21.    396.9  18.72 ] 15.2  
[ 3.69311 0.      18.1   0.     0.713  6.376  88.4  2.5671  24.    666.    20.2   391.43  14.65 ] 17.7
```

- crim : 1.23247 zn : 0.0 indus : 8.14 chas : 0.0 nox : 0.538 rm : 6.142 age : 91.7 dis : 3.9769 rad : 4.0 tax : 307.0 ptratio : 21.0 black : 396.9  
lstat : 18.72 medv : 15.2

# crim : 자치시(town) 별 1인당 범죄율  
# zn : 25,000 평방피트를 초과하는 거주지역의 비율  
# indus : 비소매상업지역이 점유하고 있는 토지의 비율  
# chas : 찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)  
# nox : 10ppm 당 농축 일산화질소  
# rm : 주택 1가구당 평균 방의 개수  
# age : 1940년 이전에 건축된 소유주택의 비율  
# dis : 5개의 보스턴 직업센터까지의 접근성 지수  
# rad : 방사형 도로까지의 접근성 지수  
# tax : 10,000 달러 당 재산세율  
# ptratio : 자치시(town)별 학생/교사 비율  
# black :  $1000(Bk - 0.63)^2$ , 여기서 Bk는 자치시별 흑인의 비율을 말함.  
# lstat : 모집단의 하위계층의 비율(%)  
# medv : 본인 소유의 주택가격(중앙값) (단위: \$1,000) / target으로 사용

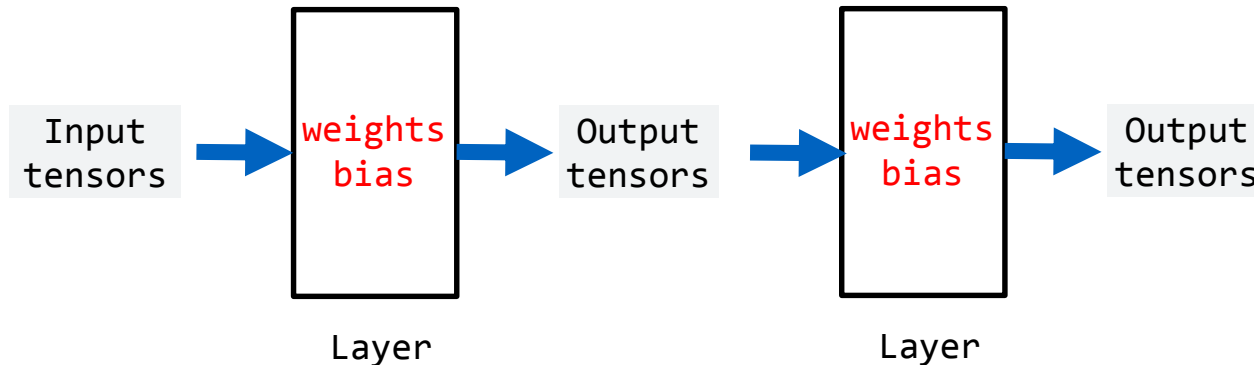
# Step I : Keras Data Preprocessing

- Image Preprocessing : see <https://keras.io/preprocessing/image/>
  - ImageDataGenerator class
  - ImageDataGenerator methods
- Text Preprocessing : see <https://keras.io/preprocessing/text/>
  - text\_to\_word\_sequence()
  - etc. ...
- Sequence Preprocessing : <https://keras.io/preprocessing/sequence/>
  - TimeseriesGenerator()
  - etc. ...

# Step 2 : Keras **Layers**

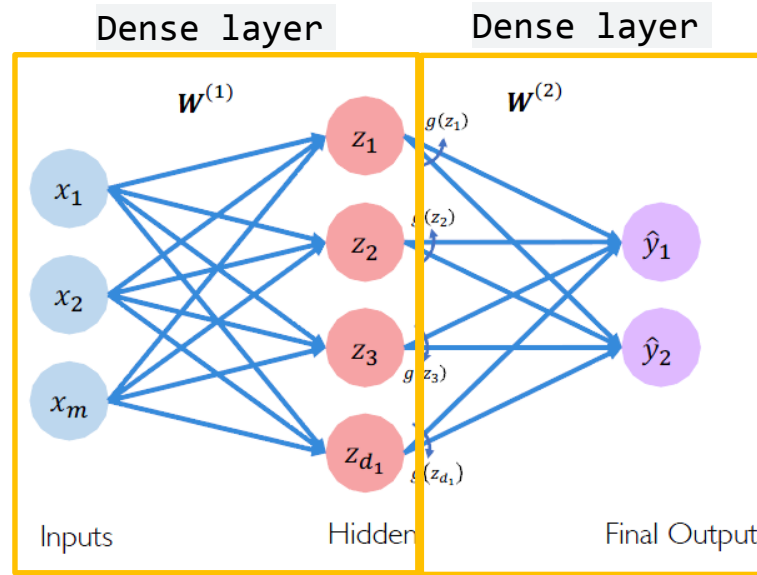
- **Layers**

- Building blocks, the **fundamental data structure** of Keras
- Data processing module that takes as input one or more tensors and that outputs one or more tensors
- Building DL models is done by clipping together compatible layers to form useful data-transformation pipelines



# Step 2 : Keras Layers

- Two Dense Layers
- Two-layers NN



```
model = Sequential()  
model.add(Dense(100, input_shape=(784,), activation='relu'))  
model.add(Dense(10, activation='softmax'))
```

# Keras Core Layers

- **Dense** : densely-connected NN layer
  - **Activation** : applies an activation function to an output
  - **Dropout** : applies dropout to the input
  - **Flatten** : flatten the input
  - **Input** : instantiate a Keras tensor
  - **Reshape** : reshapes an output to a certain shape
  - etc. ...
- 
- See <https://keras.io/layers/core/>



# Keras Layers

- **Convolutional Layers** : Conv1D, Conv2D, Conv3D ...
- **Pooling Layers** : MaxPooling1D, MaxPooling2D ...
- **Recurrent Layers** : RNN, LSTM, GRU ...
- **Embedding Layers** : Embedding
- **Merge Layers** : Add, Maximum, Concatenate ...
- **Advanced Activation Layers** : ReLU, Softmax, LeakyReLU ...
- **Normalization Layers** : BatchNormalization
- etc. ...
  
- See <https://keras.io/layers/about-keras-layers/>

# Keras Initializers

- Initializations define the way to set **the initial random weights** of Keras layers
1. `tf.keras.initializers.Zeros()`
  2. `tf.keras.initializers.Ones()`
  3. `tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.1, seed=None)`
  4. `tf.keras.initializers.RandomUniform(minval=-0.1, maxval=0.1, seed=None)`
  5. `tf.keras.initializers.glorot_normal()`
  6. `tf.keras.initializers.he_normal()`
  7. etc. ...
- see <https://keras.io/initializers/>

# Keras Activation Functions

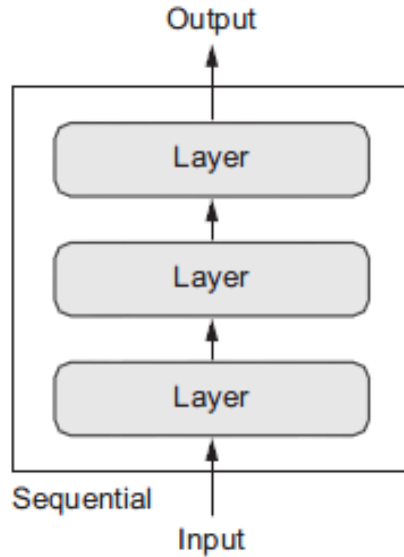
- **Activation function** : function that defines the **output of a layer** given an input
  1. `tf.keras.activations.linear()`
  2. `tf.keras.activations.sigmoid()`
  3. `tf.keras.activations.tanh()`
  4. `tf.keras.activations.softmax()`
  5. `tf.keras.activations.relu()`
  6. etc. ...
- see <https://keras.io/activations/> and <https://keras.io/layers/advanced-activations/>

# Step 2 : Keras Models

- **Models : a graph of layers**
  - 1) a simple stack of layers, 2) an arbitrary topology of layers
- **The Sequential Model**
  - a linear stack of layers
  - useful for building simple models such as FCNN, CNN, RNN, AE
- **The Functional API**
  - multi-input and multi-output models
  - complex model with 2 or more branches
  - models with shared (weights) layers, or with non-sequential data flow
- **Model Subclassing**
  - useful for creating your own fully-customizable model

# Sequential Models

- The network has one input and one output
- A linear stack of layers



# Exercise 01-14.

tf2-01-14-keras\_sequential.py

# Sequential Models

```
from tf.keras.models import Sequential
from tf.keras.layers import Dense, Activation
```

```
model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels) # starts training
```

or,

```
model = Sequential()
model.add(Dense(32, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))
```

# Functional API

- Import class called 'Model'
- Each layer explicitly returns a tensor
- Pass the returned tensor to the next layer as input
- Explicitly mention model inputs and outputs
- Multi-input, multi-output, arbitrary static graph topologies



# Exercise 01-15.

tf2-01-15-keras\_functionalAPI.py

# Functional API

```
from tf.keras.layers import Input, Dense
from tf.keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

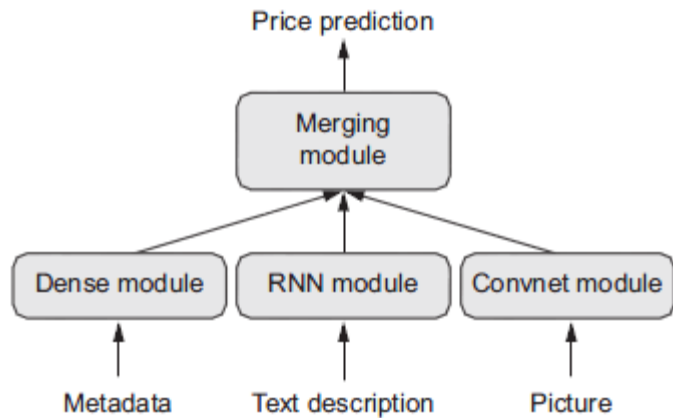
# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)

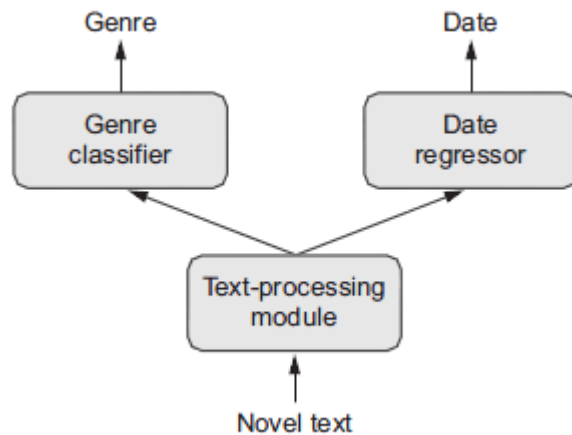
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels) # starts training
```

# Functional API

- Multi-input mode

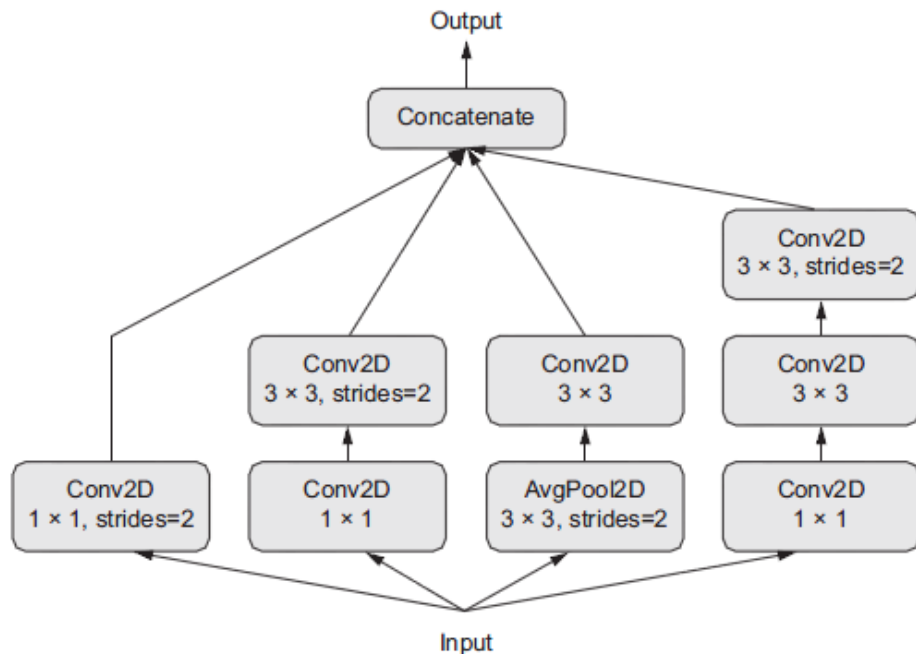


- Multi-output mode

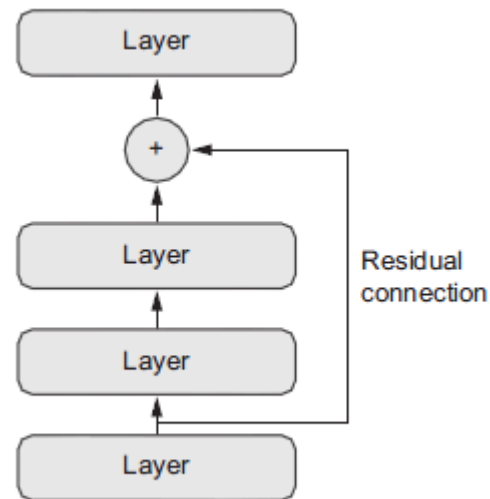


# Functional API

- An **Inception** module



- A **residual** connection



# Sequential vs Functional API

```
model = Sequential()  
model.add(Dense(32, input_shape=(784,)))  
model.add(Activation('relu'))  
model.add(Dense(10))  
model.add(Activation('softmax'))
```

```
inputs = Input(shape=(784,))  
x = Dense(64, activation='relu')(inputs)  
x = Dense(64, activation='relu')(x)  
predictions = Dense(10, activation='softmax')(x)  
model = Model(inputs=inputs,  
              outputs=predictions)
```

# Model subclassing

- Creating your own models by subclassing the 'Model' class

# Step 3 : Losses, Optimizers and Metrics

- Configures the model for training

```
compile(optimizer, loss=None, metrics=None, loss_weights=None, sample_weight_mode=None, weighted_metrics=None, target_tensors=None)
```

- **Loss function** : the quantity that will be minimized during training
- **Optimizer** : determines how the network will be updated based on the loss function
- **Metric** : function that is used to judge the performance of the model

# Loss Functions

- the quantity that will be minimized during training
- `tf.keras.losses.mean_squared_error` (MSE)
- `tf.keras.losses.mean_absolute_error` (MAE)
- `tf.keras.losses.hinge`
- `tf.keras.losses.categorical_crossentropy` (CE)
- `tf.keras.losses.binary_crossentropy` (CE)
- `tf.keras.losses.kullback_leibler_divergence`
- etc. ...
- see <https://keras.io/losses/>



# Optimizers

- A Optimizer determines how the network will be updated based on the loss function
- `tf.keras.optimizers.SGD`
- `tf.keras.optimizers.RMSprop`
- `tf.keras.optimizers.Adagrad`
- `tf.keras.optimizers.Adadelta`
- `tf.keras.optimizers.Adam`
- `tf.keras.optimizers.Adamax`
- `tf.keras.optimizers.Nadam`
- see <https://keras.io/optimizers/>

# Metrics

- A metric is a function that is to judge the performance of the model
- `tf.keras.metrics.binary_accuracy`
- `tf.keras.metrics.categorical_accuracy`
- `tf.keras.metrics.sparse_categorical_accuracy`
- `tf.keras.metrics.top_k_categorical_accuracy`
- `tf.keras.metrics.sparse_top_k_categorical_accuracy`
- see <https://keras.io/metrics/>

```
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['mae', 'acc'])
```

```
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=[metrics.mae,  
metrics.categorical_accuracy])
```

# Step 4. Training the model

- Trains the model for a given number of epochs

```
fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, 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_freq=1)
```

- x, y : input, target
- batch\_size :
- epochs :
- verbose : 0 = silent, 1 = progress bar, 2 = one line per epoch
- callbacks :
- validation\_split : (0~1)

# Keras Callbacks

- A **callback** is a set of functions to be applied at given stages of the training procedure
- You can use callbacks to get a view on internal states and statistics of the model during training
- Callbacks **Examples**
  - **Model checkpointing** : Saving the current weights of the model at different points during training
  - **Early stopping** : Interrupting training when the validation loss is no longer improving
  - Dynamically adjusting the value of certain parameters during training
    - : the learning rate of the optimizer
  - Logging training and validation metrics during training, or visualizing the representations learned by the model as they're updated

# Keras Callbacks

- `tf.keras.callbacks.ModelCheckpoint()`
- `tf.keras.callbacks.EarlyStopping()`
- `tf.keras.callbacks.LearningRateScheduler()`
- `tf.keras.callbacks.TensorBoard()`
- `tf.keras.callbacks.ReduceLROnPlateau()`
- `tf.keras.callbacks.CSVLogger()`
  
- etc. ...
  
- see <https://keras.io/callbacks/>

# Callbacks Example

- **ModelCheckpoint and EarlyStopping**
  - Interrupts training when improvement stops and saves the current weights after every epoch
- **TensorBoard** : the TensorFlow viusalization framework

```
callbacks_list = [  
    keras.callbacks.EarlyStopping(monitor='acc', patience=1,),  
    keras.callbacks.ModelCheckpoint(filepath='my_model.h5', monitor='val_loss', save_best_only=True)  
]
```

```
callbacks_list = [  
    keras.callbacks.TensorBoard(log_dir='my_log_dir', histogram_freq=1, embeddings_freq=1)  
]
```

```
History = model.fit(x, y, epochs=10, batch_size=32, callbacks=callbacks_list, validation_data  
=(x_val, y_val))
```

## Exercise 01-16.

tf2-01-16-keras\_no\_callbacks.py

tf2-01-17-keras\_callbacks.py

ModelCheckpoint and EarlyStopping

# Exercise 01-18.

tf2-01-18-keras\_tensorboard.py

TensorBoard

```
>> tensorboard --logdir=logs/
```



# Step 5. Test the model

- Returns the loss value & metrics values for the model in test mode.

```
evaluate(x=None, y=None, batch_size=None, verbose=1, sample_weight=None, steps=None, callbacks=None)
```

- Generates output predictions for the input samples.

```
predict(x, batch_size=None, verbose=0, steps=None, callbacks=None)  
predict_classes(x, batch_size=None, verbose=0, steps=None, callbacks=None)
```

# Save and Load Keras Models

- Save and Load model weights
  - `model.save_weights()`
  - `model.load_weights()`
- Save and Load models
  - `model.save()`
  - `tf.keras.models.load_model()`

# Exercise 01-19.

tf2-01-19-keras\_save\_load.py