COMP7035

Python for Data Analytics and Artificial Intelligence

Keras

Renjie Wan, Jun Qi

12/03/2024

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

What we will learn?

Topic			**Hours**
I.	Python Fundamentals		12
	A.	Program control and logic	
	B.	Data types and structures	
	C.	Function	
	D.	File I/O	
II.	Numerical Computing and Data Visualization Tools and libraries such as		9
	A.	NumPy	
	B.	Matplotlib	
	C.	Seaborn	
III.	Exploratory Data Analysis (EDA) with Python Tools and libraries such as		9
	A.	Pandas	
	B.	Sweetviz	
IV.	Artificial Intelligence and Machine Learning with Python Tools and libraries such as		9
	A.	Scikit-learn	
	B.	**Keras**	

Key Components of Keras Pipeline

1. Load Data
2. Define Keras Model
3. Fit Keras Model
4. Evaluate Keras Model
5. Tie It All Together
6. Make Predictions

- Dataset?
- Learning Target?
- Model Structure?
- Loss function?
- Optimizer?
- Training Loop?

Define Keras Model

Keras Layers

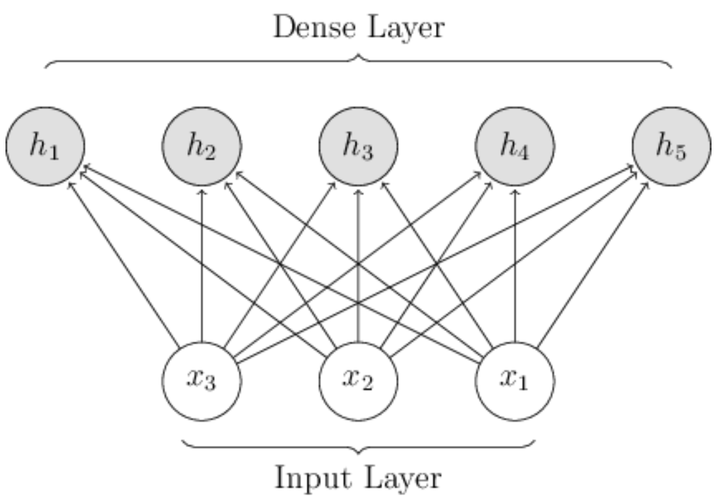
A DNN is built from many different types of layers, which mainly include

- Linear layers (Dense)
- RNN layers
- CNN layers

Define Keras Model

Keras Layers

- Linear layers (Dense)
- RNN layers
- CNN layers



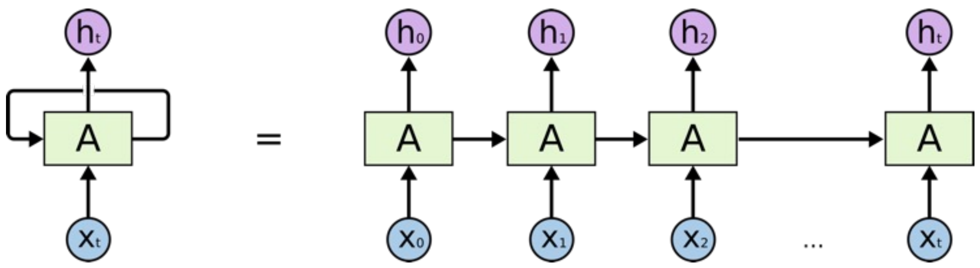

```
Dense( units, activation=None,
use_bias=True,
kernel_initializer="glorot_uniform",
bias_initializer="zeros",
kernel_regularizer=None,
bias_regularizer=None,
activity_regularizer=None,
kernel_constraint=None,
bias_constraint=None, **kwargs )
```

You can also specify the input dimension as introduced before

Define Keras Model

Keras Layers

- Linear layers (Dense)
- RNN layers
- CNN layers



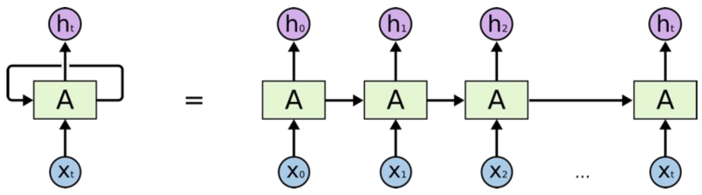

```
LSTM(units, activation="tanh",
recurrent_activation="sigmoid",
use_bias=True,
kernel_initializer="glorot_uniform",
recurrent_initializer="orthogonal",
bias_initializer="zeros",
unit_forget_bias=True,
kernel_regularizer=None,
recurrent_regularizer=None,
bias_regularizer=None,
activity_regularizer=None,
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,
time_major=False, unroll=False, **kwargs )
```

- RNN layers takes the sequence as input and output a new sequence.
- Mainly used to model sequential data (speech, stock price, etc)
- The most used RNN is LSTM (Long Short-Term Memory).

Define Keras Model

- Linear layers (Dense)
- RNN layers
- CNN layers

Keras Layers



Run the codes, check, and understand the dimensions of different tensors.
The first dimension of x is batch size, which will be studied later.

```
# Define the model using functional API:
from keras.layers import Dense,LSTM
import keras
import numpy as np

x = keras.backend.constant(np.random.randn(1,100,128))
lstm_layer1 = LSTM(512,return_sequences=True)
lstm_layer2 = LSTM(512,return_sequences=False)
y1 = lstm_layer1(x)
y2 = lstm_layer2(x)

for ii in [x,y1,y2]:
  print(ii.shape)
```

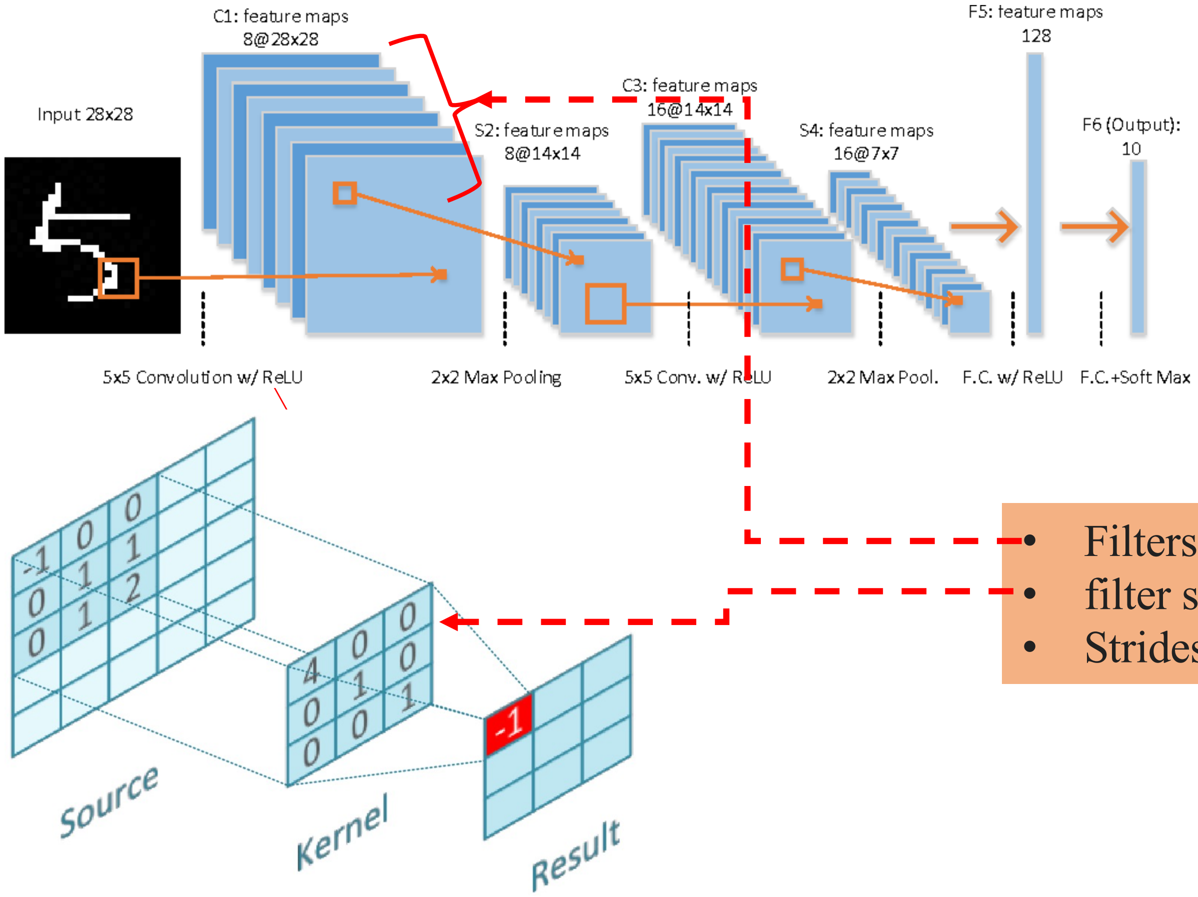
```
(1, 100, 128)
(1, 100, 512)
(1, 512)
```

Define Keras Model

Keras Layers

- Linear layers (Dense)
- RNN layers
- CNN layers

2D convolution



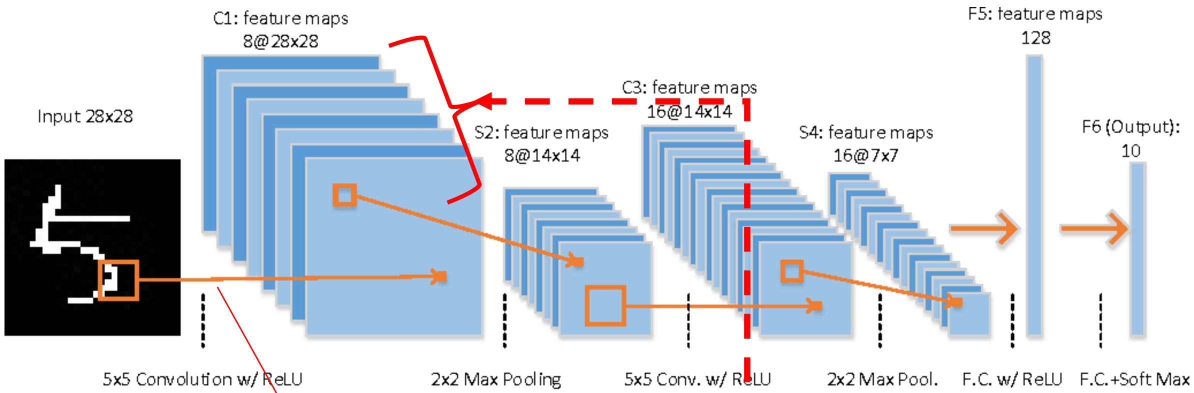
- Filters (how many do we generate)
- filter size (how large do we compute)
- Strides (how far do we move)

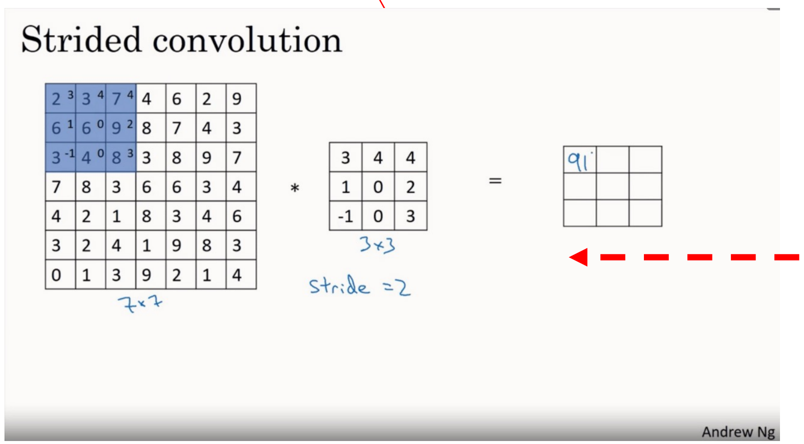
Define Keras Model

Keras Layers

- Linear layers (Dense)
- RNN layers
- CNN layers

2D convolution





- Filters (how many do we generate)
- filter size (how large do we compute)
- Strides (how far do we move)

Define Keras Model

Keras Layers

```
# Define the model using functional API:
from keras.layers import Dense,Conv2D
import keras
import numpy as np

x = keras.backend.constant(np.random.randn(16,28,
28,1))
n_filter = 16
filter_size = 2
cnn_layer1 = Conv2D(n_filter,filter_size,input_sh
ape=(28,28,1))
y1 = cnn_layer1(x)

n_filter = 32
filter_size = 4
cnn_layer2 = Conv2D(n_filter,filter_size,input_sh
ape=(28,28,1))
y2 = cnn_layer2(x)


for ii in [x,y1,y2]:
  print(ii.shape)
```

- Linear layers (Dense)
- RNN layers
- CNN layers

2D convolution

```
tf.keras.layers.Conv2D(
filters, kernel_size,
strides=(1, 1),
padding="valid",
data_format=None,
dilation_rate=(1, 1),
groups=1, activation=None,
use_bias=True,
kernel_initializer="glorot_uni
form",
bias_initializer="zeros",
kernel_regularizer=None,
bias_regularizer=None,
activity_regularizer=None,
kernel_constraint=None,
bias_constraint=None, **kwargs
```

Define Keras Model

Keras Layers

```
# Define the model using functional API:
from keras.layers import Dense,Conv2D
import keras
import numpy as np

x = keras.backend.constant(np.random.randn(16,28,
28,1))
n_filter = 16
filter_size = 2
cnn_layer1 = Conv2D(n_filter,filter_size,input_sh
ape=(28,28,1))
y1 = cnn_layer1(x)

n_filter = 32
filter_size = 4
cnn_layer2 = Conv2D(n_filter,filter_size,input_sh
ape=(28,28,1))
y2 = cnn_layer2(x)


for ii in [x,y1,y2]:
  print(ii.shape)
```

When using this layer as the first layer in a model, provide the keyword argument input_shape

- Linear layers (Dense)
- RNN layers
- CNN layers

2D convolution

```
tf.keras.layers.Conv2D(
filters, kernel_size,
strides=(1, 1),
padding="valid",
data_format=None,
dilation_rate=(1, 1),
groups=1, activation=None,
use_bias=True,
kernel_initializer="glorot_uni
form",
bias_initializer="zeros",
kernel_regularizer=None,
bias_regularizer=None,
activity_regularizer=None,
kernel_constraint=None,
bias_constraint=None, **kwargs
```

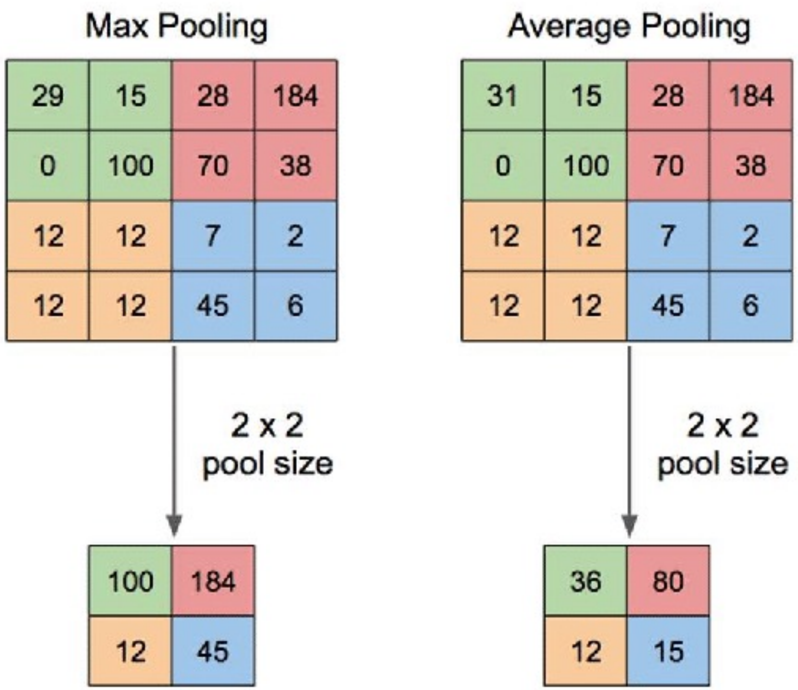
Define Keras Model

Keras Layers

```
MaxPooling2D(pool_size)
```

- Linear layers (Dense)
- RNN layers
- CNN layers

Pooling

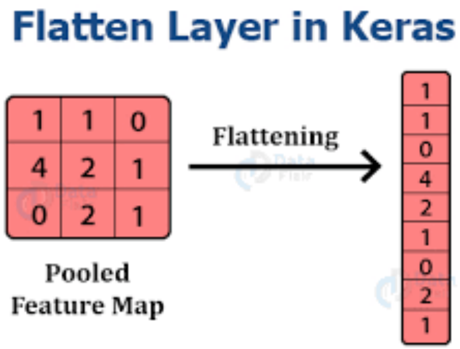



```python
# Define the model using functional API:
from keras.layers import Dense,Conv2D,MaxPool2D
import keras
import numpy as np

x = keras.backend.constant(np.random.randn(16,28,28,1))

n_filter = 32
filter_size = 4
cnn_layer2 = Conv2D(n_filter,filter_size,input_shape=(
28,28,1))
y1 = cnn_layer2(x)
pooling_layer = MaxPool2D(2)
y2 = pooling_layer(y1)

for ii in [x,y1,y2]:
  print(ii.shape)
```

```
(16, 28, 28, 1)
(16, 25, 25, 32)
(16, 12, 12, 32)
```

Define Keras Model

Other Common Keras Operations



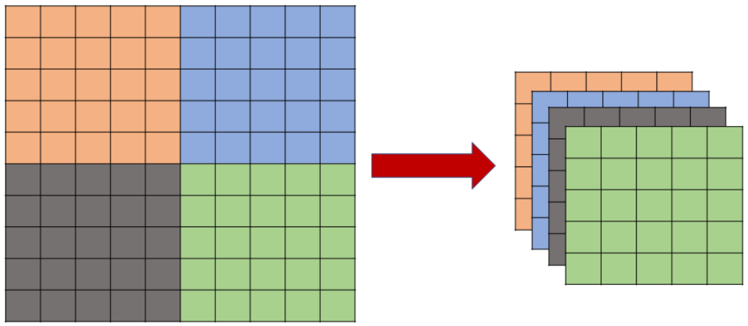

```
Flatten()
```

```
# Define the model using functional API:
from keras.layers import Flatten,Reshape
from keras.models import Sequential
import keras
import numpy as np

x = keras.backend.constant(np.random.randn(16,28,28,1))
y = Flatten()(x)
n_filter = 32
filter_size = 4
conv_layer = Conv2D(n_filter,filter_size,input_shape=(28,28,1))
z = conv_layer(x)
model = Sequential([conv_layer,Flatten()])
w = model(x)

for i in [x,y,z,w]:
  print(i.shape)
```

```
(16, 28, 28, 1)
(16, 784)
(16, 25, 25, 32)
(16, 20000)
```

Define Keras Model

Other Common Keras Operations



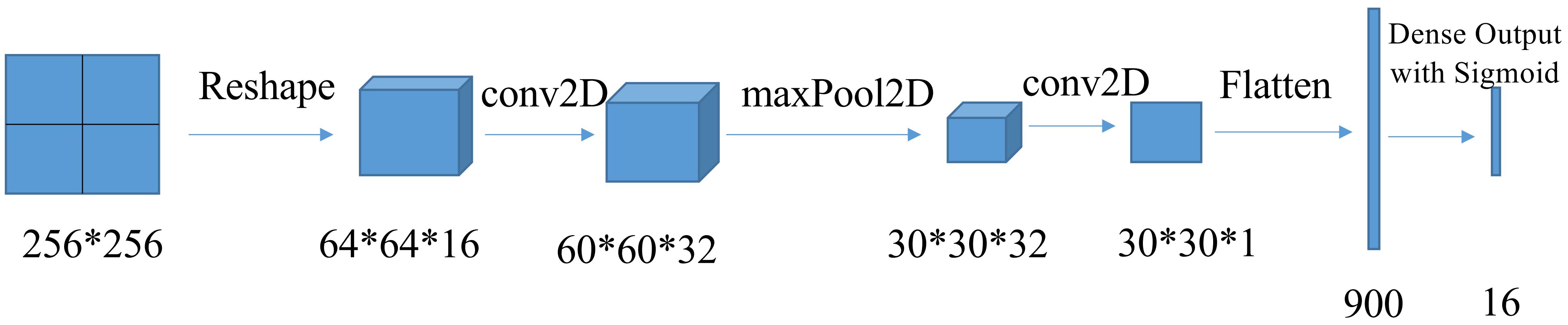
Reshape(target_shape)

```
# Define the model using functional API:
from keras.layers import Flatten,Reshape
from keras.models import Sequential
import keras
import numpy as np

x = keras.backend.constant(np.random.ran
dn(16,28,28,1))
y = Reshape((14,14,-1))(x)
print(y.shape)
y = Reshape((-1,1))(x)
print(y.shape)
```

```
(16, 14, 14, 4)
(16, 784, 1)
```

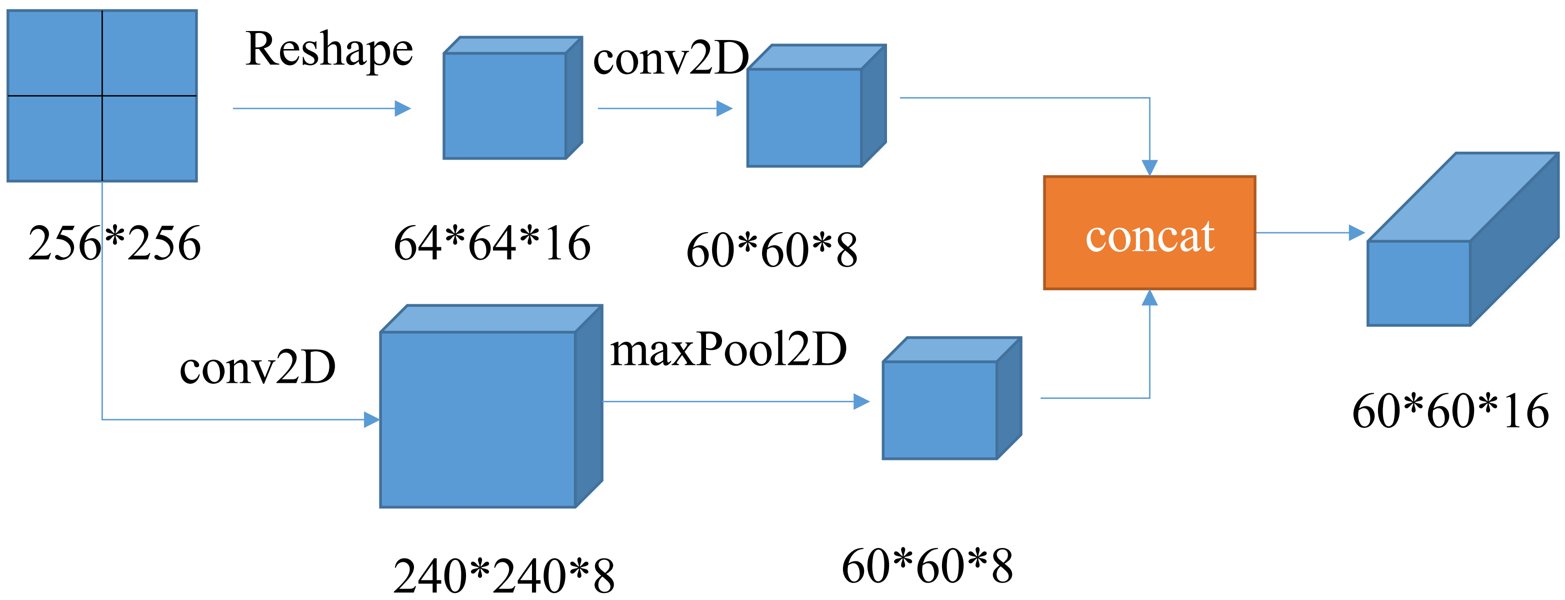
Exercise

1. Define the following model using Sequential API. Assuming padding is not used.



Exercise

1. Define the following model using Sequential API.Assuming padding is not used.
2. Define the following model using Functional API.



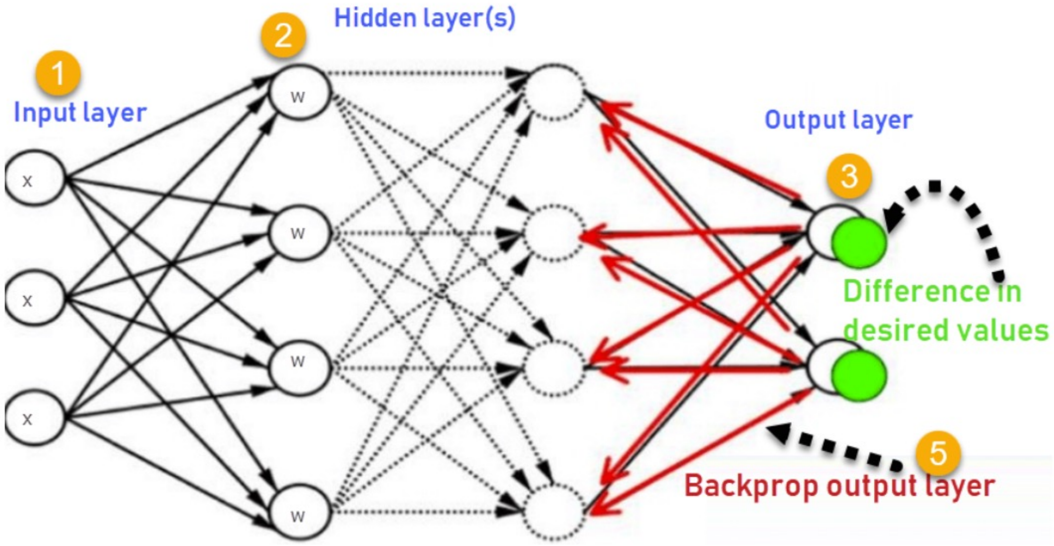
Key Components of Keras Pipeline

1. Load Data
2. Define Keras Model
3. Fit Keras Model
4. Evaluate Keras Model
5. Tie It All Together
6. Make Predictions

- Dataset?
- Learning Target?
- Model Structure?
- Loss function?
- Optimizer?
- Training Loop?

Loss Function

The DNN model is optimized through a feedback backpropogation (BP). The loss function is used to judge how well the model can approximate the desired output.



Types of loss functions

- Regression Loss Functions
- Classification Loss Functions
- Probabilistic Loss Functions
- Custom Loss Functions

Loss Function

Regression Loss Functions

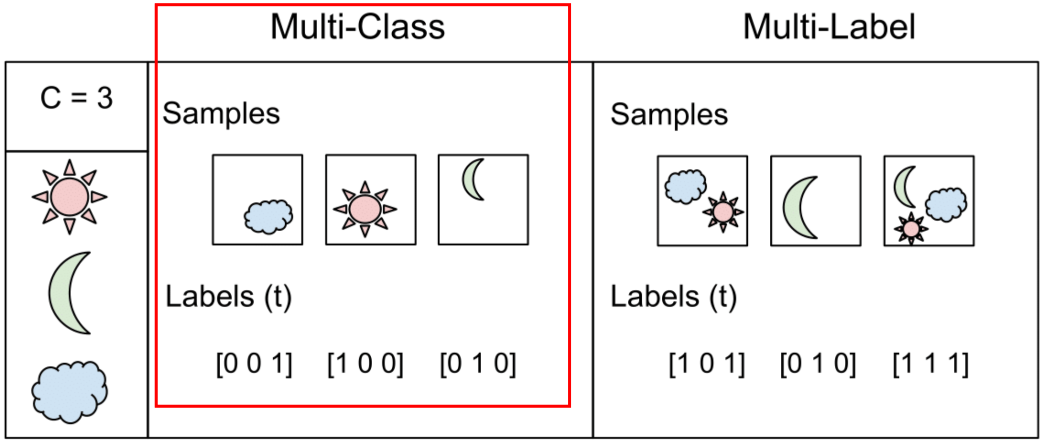
- **Mean Squared Error (MSE)**

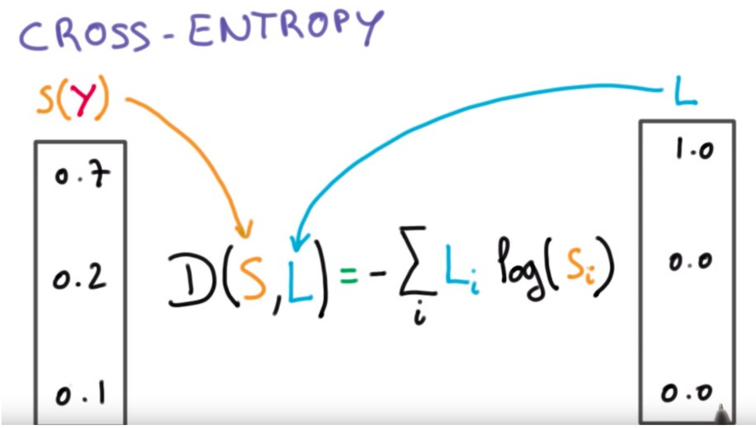
```
import keras
from keras.losses import MeanAbsoluteError,MeanSquaredError
mse = MeanSquaredError()
y_true = [[0., 0.3], [0., 1.]]
y_pred = [[1., 1.], [1., 0.]]
z = mse(y_true, y_pred)
print(z)
print(z.numpy())
```

Loss Function

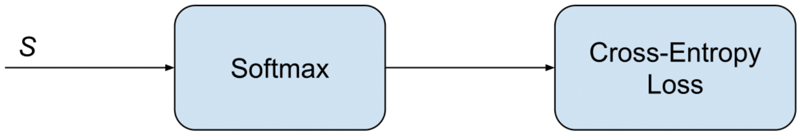
Classification Loss Functions

- **Categorical Cross-Entropy**






```
import keras
from keras.losses import CategoricalCrossentropy
loss = CategoricalCrossentropy()
y_true = [[0, 1, 0], [0, 0, 1]]
y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
z = loss(y_true, y_pred)
print(z.numpy())
```



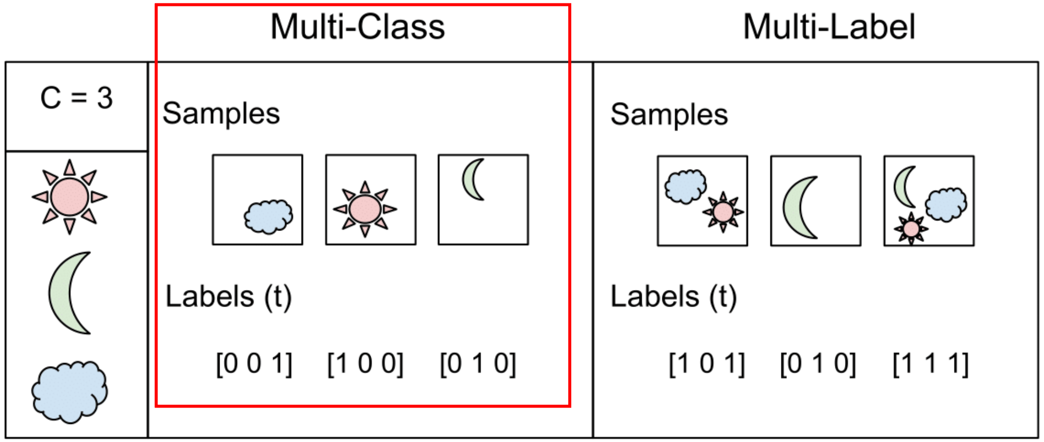
$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

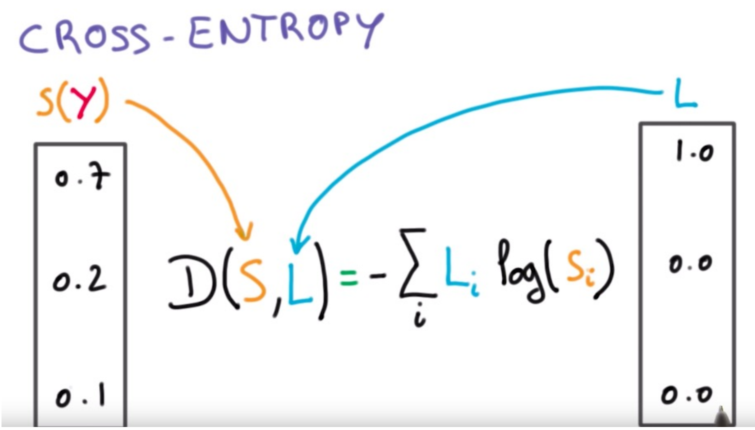
- The prediction contains only one class
- The label is one-hot

Loss Function

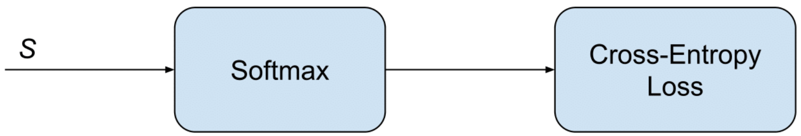
Classification Loss Functions

- **Categorical Cross-Entropy**






```
import keras
from keras.losses import SparseCategoricalCrossentropy
loss = SparseCategoricalCrossentropy()
y_true = [1, 2]
y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
z = loss(y_true, y_pred)
print(z.numpy())
```



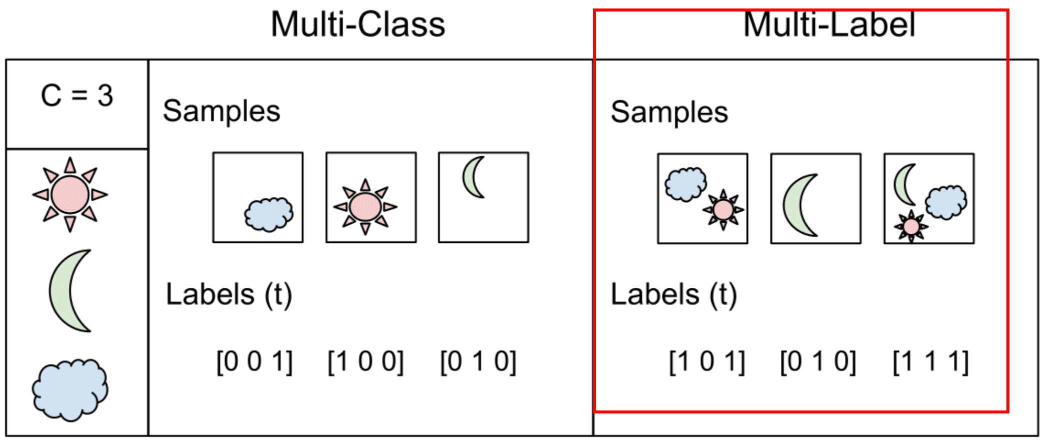
$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

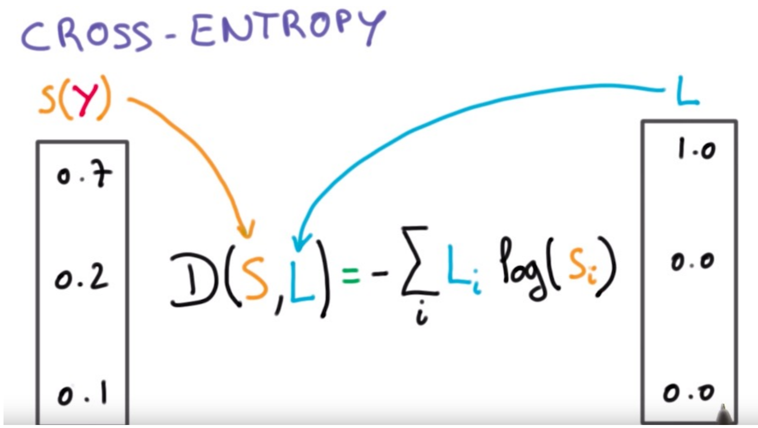
- The prediction contains only one class
- The label is an integer.

Loss Function

Probabilistic Loss Functions

- **Binary Cross-Entropy**






```python
import keras
from keras.losses import BinaryCrossentropy
loss = BinaryCrossentropy()
y_true = [0, 1, 0, 0]
y_pred = [-18.6, 0.51, 2.94, -12.8]
z = loss(y_true, y_pred)
print(z.numpy())
```

$$CE = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

$$f(s_i) = \frac{1}{1 + e^{-s_i}}$$

- The prediction can contain multiple classes
- The label is multi-hot.

Loss Function

Custom Loss Functions

We can also define the loss as a function of prediction and target.

```
def custom_loss(y_true, y_pred):
    # calculate loss, using y_pred
    return loss
```

Supposing we want to define a loss function as MAE+2*MSE

```
import keras
from keras.losses import MeanAbsoluteError,MeanSquaredError
def my_loss(y_true, y_pred):
    loss = MeanAbsoluteError(y_true, y_pred)+2*MeanSquaredError(y_true, y_pred)
    return loss
```

Loss Function

For the MNIST task, the loss function would be

```
loss = CategoricalCrossentropy()
```

for one-hot labels;

Or

```
loss = SparseCategoricalCrossentropy()
```

for integer-valued labels.

Key Components of Keras Pipeline

1. Load Data
2. Define Keras Model
3. Fit Keras Model
4. Evaluate Keras Model
5. Tie It All Together
6. Make Predictions

- Dataset?
- Learning Target?
- Model Structure?
- Loss function?
- Optimizer?
- Training Loop?

Optimizer

The optimizer specifies how to update the DNN weights according to the loss function values.

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad

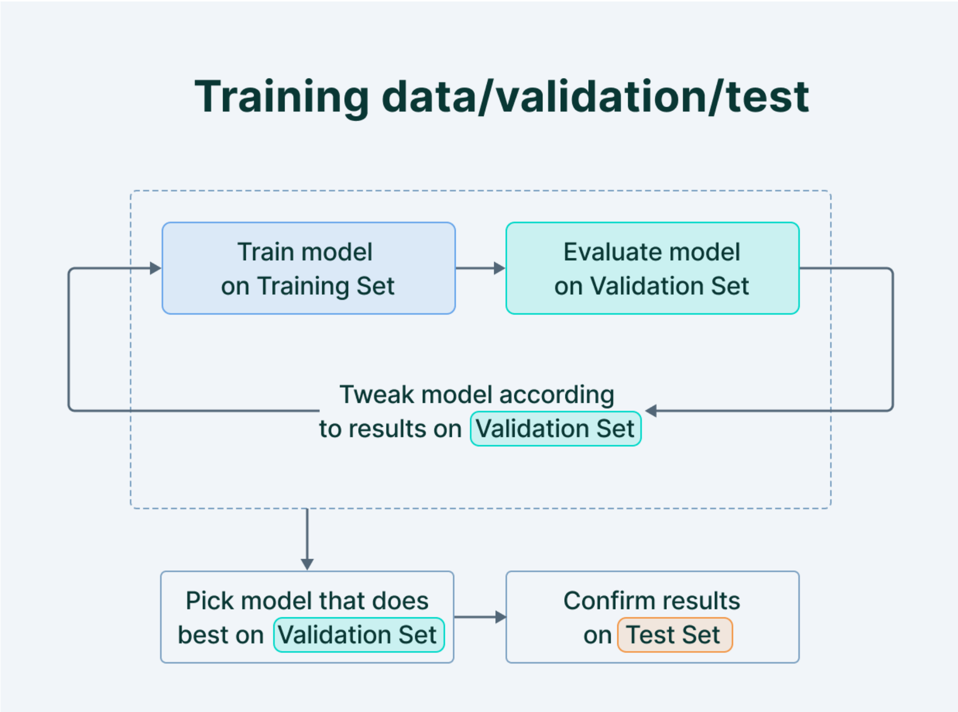
- e.g.,

```
optimizer = keras.optimizers.Adam()
optimizer = keras.optimizers.Adam(learning_rate=1e-3)
```

Compile Model

- **There is one more thing before actually training the model:**
  - How to evaluate the generalization performance, in order to save the best model?

Metrics



During training, we compute the metrics on the validation dataset and choose the model with the best evaluation metric.

Commonly used metric:

- Accuracy
- MSE
- …

The loss function itself can also be taken as the metric.

Compile Model

After we have figured out the loss function and optimizer, we can compile a defined model for training.

```
model.compile(optimizer=keras.optimizers.Adam(),
              loss=keras.losses.SparseCategoricalCrossentropy(),
              metrics=[tferas.metrics.Accuracy())
```

or

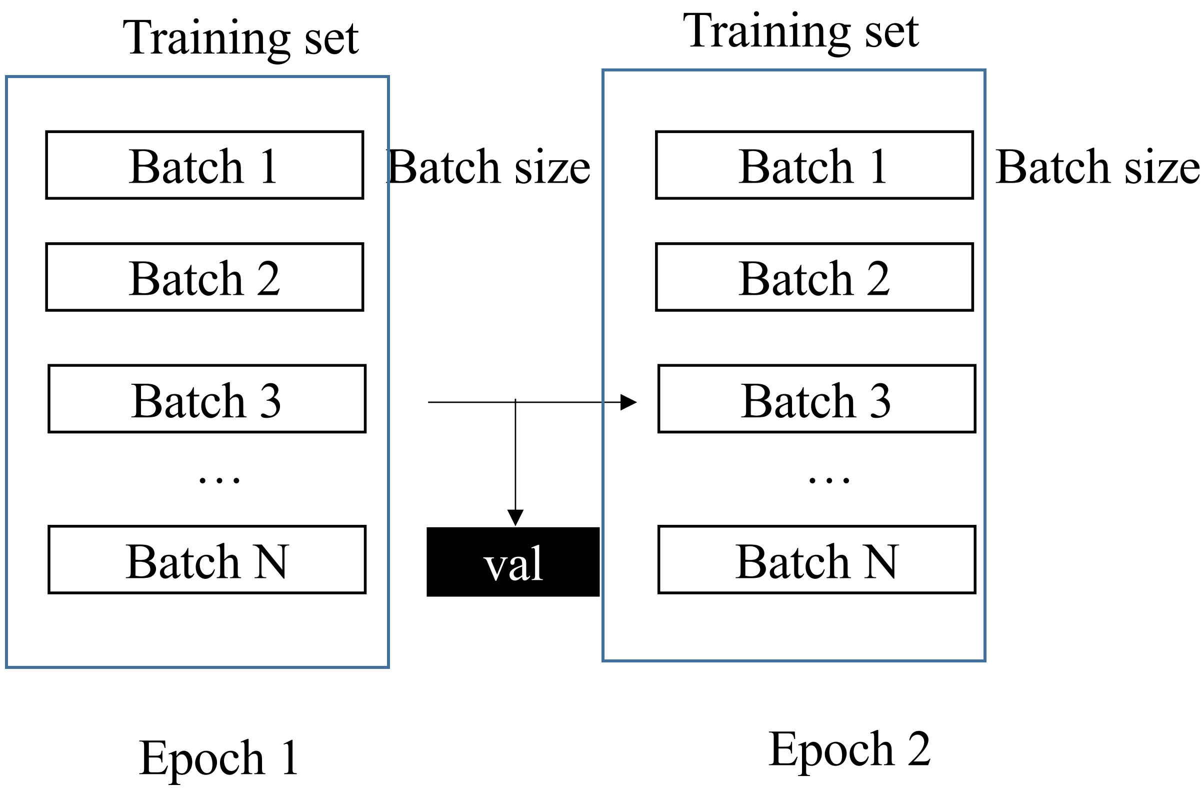
```
model.compile(optimizer='adam',
              loss= 'sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Fit Model

The model can be optimized using the built-in API

model.fit

```
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,
max_queue_size=10, workers=1,
use_multiprocessing=False, )
```



Fit Model

The model can be optimized using the built-in API

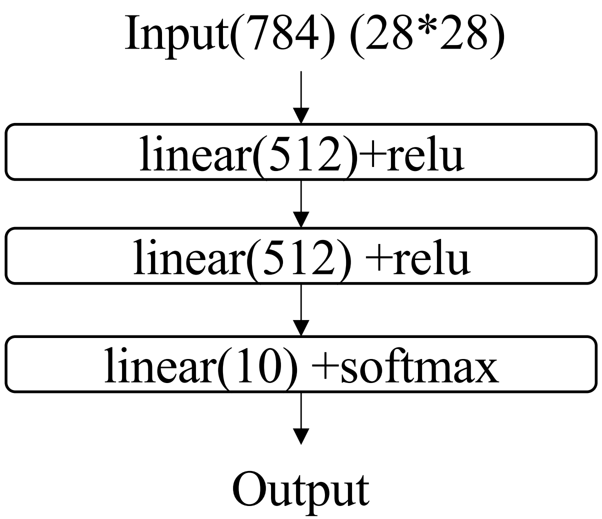
model.fit

```
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,
max_queue_size=10, workers=1,
use_multiprocessing=False, )
```

```
batch_size = 128
epochs = 15
model.fit(x_train, y_train, batch_size=batch_size
, epochs=epochs, validation_split=0.1)
```

Exercise

1. Define the following model using Sequential API. Assuming padding is not used.
2. Define the following model using Functional API.
3. Use the structure as below to train a model for the MNIST task. Try using the entire dataset and data generator to train the model.



Key Components of Keras Pipeline

1. Load Data
2. Define Keras Model
3. Fit Keras Model
4. Evaluate Keras Model
5. Tie It All Together
6. Make Predictions

Dataset?

Learning Target?

Model Structure?

Loss function?

Optimizer?

Training Loop?

Evaluate Model

using model.evaluate to evaluate the model, it will give us the loss and the metric values.

```
Model.evaluate( x=None, y=None,
batch_size=None, verbose="auto",
sample_weight=None, steps=None,
callbacks=None,
max_queue_size=10, workers=1,
use_multiprocessing=False,
return_dict=False, **kwargs )
```

```
score = model.evaluate(x_test, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Key Components of Keras Pipeline

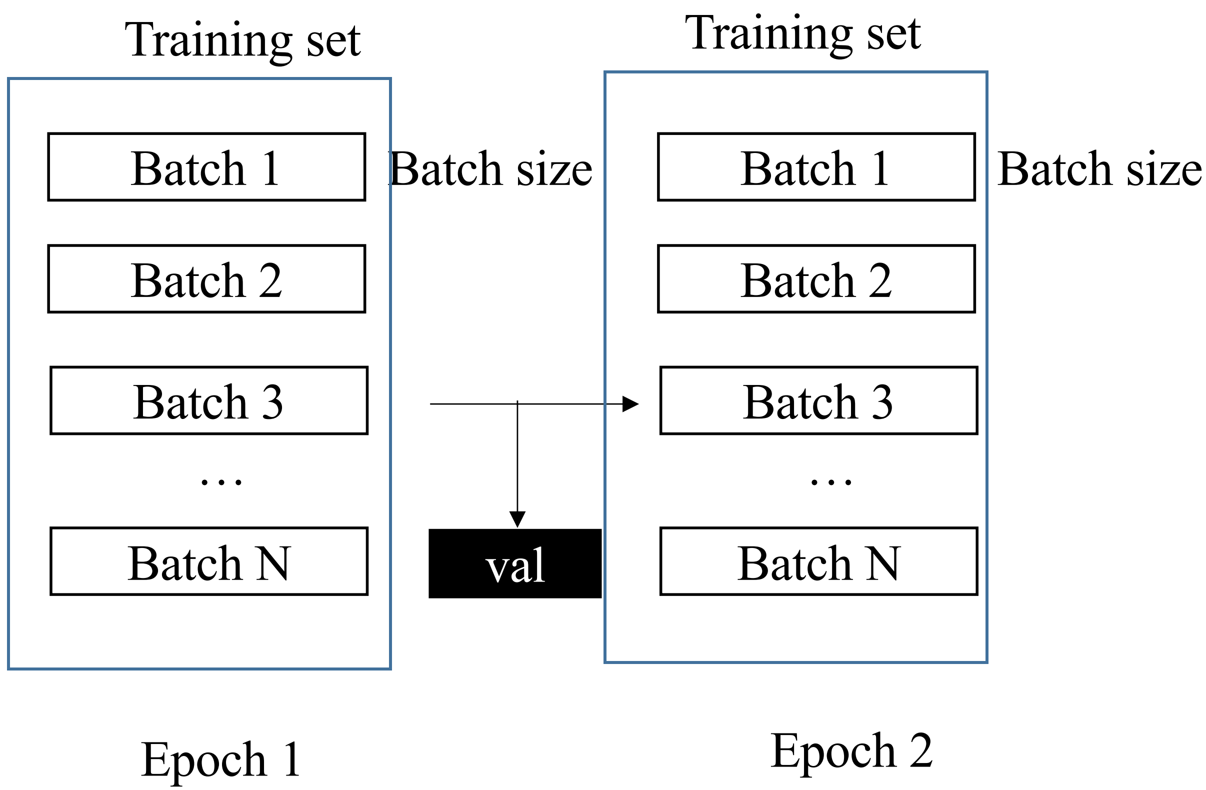
1. Load Data
2. Define Keras Model
3. Fit Keras Model
4. Evaluate Keras Model
5. Tie It All Together
6. Make Predictions

- Dataset?
- Learning Target?
- Model Structure?
- Loss function?
- Optimizer?
- Training Loop?

Customized Training Loop

Rather than using model.fit, the training loop can also be customized, in order to

- Save the model
- Adjust the learning rate
- etc



Customized Training Loop

The training loop design:

```
for epoch in range(epochs):
  for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):
        with tf.GradientTape() as tape:
            logits = model(x_batch_train, training=True)
            loss_value = loss_fn(y_batch_train, logits)
        grads = tape.gradient(loss_value, model.trainable_weights)
        optimizer.apply_gradients(zip(grads, model.trainable_weights))
```

- Open a for loop that iterates over epochs
- For each epoch, iterate over the dataset in batches
- For each batch, open a GradientTape() scope, forward pass and compute the loss
- Outside the scope, retrieve the gradients
- Use the optimizer to update the model

Customized Training Loop

The training loop design:

```
val_acc_metric = keras.metrics.SparseCategoricalAccuracy()
best_metric = 10000
for epoch in range(epochs):
  for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):
        with tf.GradientTape() as tape:
            logits = model(x_batch_train, training=True)
            loss_value = loss_fn(y_batch_train, logits)
        grads = tape.gradient(loss_value, model.trainable_weights)
        optimizer.apply_gradients(zip(grads, model.trainable_weights))
   for x_batch_val, y_batch_val in val_dataset:
        val_logits = model(x_batch_val, training=False)
        val_acc_metric.update_state(y_batch_val, val_logits)
    val_acc = val_acc_metric.result()
    if float(val_acc) < best_metric:
      best_metric = float(val_acc)
      model.save('path_to_location.h5')
    val_acc_metric.reset_states()
```

- Evaluate after each epoch, save the best model

Customized Training Loop

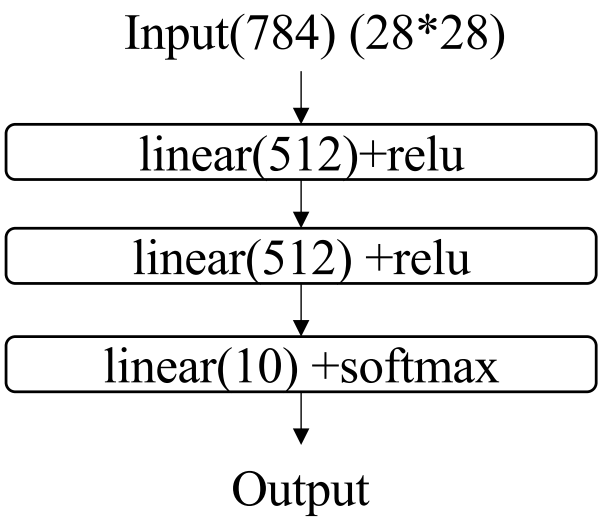
The training loop design:

```
import keras
from keras import backend as K
val_acc_metric = keras.metrics.SparseCategoricalAccuracy()
best_metric = 10000
for epoch in range(epochs):
  for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):
        with tf.GradientTape() as tape:
            logits = model(x_batch_train, training=True)
            loss_value = loss_fn(y_batch_train, logits)
        grads = tape.gradient(loss_value, model.trainable_weights)
        optimizer.apply_gradients(zip(grads, model.trainable_weights))
  K.set_value(model.optimizer.learning_rate, 0.001)
```

- Change the learning rate

Exercise

1. Define the following model using Sequential API. Assuming padding is not used.
2. Define the following model using Functional API.
3. Use the structure as below to train a model for the MNIST task. Try using the entire dataset and data generator to train the model.
4. Write customized training loop for the MNIST task.



Key Components of Keras Pipeline

1. Load Data
2. Define Keras Model
3. Fit Keras Model
4. Evaluate Keras Model
5. Tie It All Together
6. Make Predictions

Dataset?

Learning Target?

Model Structure?

Loss function?

Optimizer?

Training Loop?

Make Predictions

```
Model.predict( x,
batch_size=None,
verbose="auto",
steps=None,
callbacks=None,
max_queue_size=10,
workers=1,
use_multiprocessing=False,
)
```

```
prediction = model.predict(x_test)
```

Exercise

1. Define the following model using Sequential API. Assuming padding is not used.

```
# Define the model using functional API:
from keras.models import Sequential
from keras.layers import Dense,Conv2D,MaxPool2D,Input,Reshape,Flatten
import keras
import numpy as np

layers = [Input(shape=(256,256,1)),
      Reshape((64,64,-1)),
      Conv2D(filters=32,kernel_size=5),
      MaxPool2D(2),
      Conv2D(filters=1,kernel_size=1),
      Flatten(),
      Dense(16,activation='sigmoid')]
model = Sequential(layers)
model.summary()
```

Exercise

1. Define the following model using Sequential API.Assuming padding is not used.
2. Define the following model using Functional API.

```
# Define the model using functional API:
from keras.models import Model
from keras.layers import Dense,Conv2D,MaxPool2D,Input,Reshape,Flatten,concatenate
import keras
import numpy as np

def my_model():
  # define layers
  input_tensor = Input(shape=(256,256,1))
  reshape_layer = Reshape((64,64,-1))
  conv2d_1 = Conv2D(filters=8,kernel_size=5)
  conv2d_2 = Conv2D(filters=8,kernel_size=17)
  max_pool_layer = MaxPool2D(4)

  input_tensor_reshape = reshape_layer(input_tensor)
  conv1_out = conv2d_1(input_tensor_reshape)
  conv2_out = conv2d_2(input_tensor)
  max_pool_out = max_pool_layer(conv2_out)
  cat_out = concatenate([conv1_out,max_pool_out],axis=-1)
  model = Model(inputs=input_tensor, outputs=cat_out)
  return model

my_model = my_model()
my_model.summary()
```

Exercise

Use the structure as below to train a model for the MNIST task. Try using **the entire dataset** and the data generator to train the model.

Input(784) (28*28)

↓

linear(512)+relu

↓

linear(512) +relu

↓

linear(10) +softmax

↓

Output

See next page

Exercise

```
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense,Input
from keras.losses import SparseCategoricalCrossentropy

df_orig_train = pd.read_csv('/content/drive/MyDrive/keras/mnist_train.csv',header=None)
df_orig_test = pd.read_csv('/content/drive/MyDrive/keras/mnist_test.csv',header=None)
df_train_values = df_orig_train.values
df_test_values = df_orig_test.values
train_feat_ori,train_label_ori = df_train_values[:,1:]/255.0,df_train_values[:,0]
test_feat,test_label = df_test_values[:,1:]/255.0,df_test_values[:,0]
train_feat,val_feat = train_feat_ori[6000:],train_feat_ori[:6000]
train_label,val_label = train_label_ori[6000:],train_label_ori[:6000]

model = Sequential()
model.add(Input(shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam',loss= 'sparse_categorical_crossentropy',metrics=['accuracy'])

batch_size = 128
epochs = 8
model.fit(train_feat, train_label, batch_size=batch_size, epochs=epochs, validation_data = (va
l_feat,val_label))
```

Exercise

Use the structure as below to train a model for the MNIST task. Try using the entire dataset and the **data generator** to train the model.

Input(784) (28*28)

↓

linear(512)+relu

↓

linear(512) +relu

↓

linear(10) +softmax

↓

Output

See next page

Exercise

```python
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense,Input
from keras.losses import SparseCategoricalCrossentropy

import numpy as np
import linecache
import random
```

Exercise

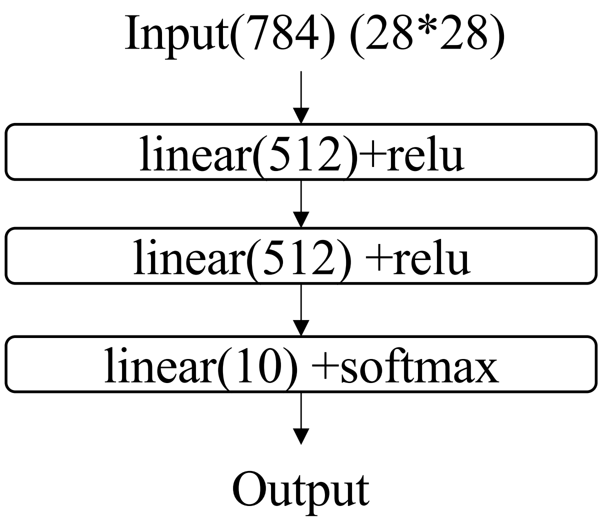
```
class DataGenerator(keras.utils.Sequence):
  'Generates data for Keras'
  def __init__(self, csv_path, indexes, bs):
    # initilizes some variables
    self.csv_path = csv_path
    self.norm_facor = 255.0
    self.indexes = indexes
    random.shuffle(self.indexes)
    self.bs = bs
  def __len__(self):
    # return the total number of samples in the dataset
    return len(self.indexes)//self.bs
  def __getitem__(self,index):
    # get one sample according to the index
    feat_all = []
    label_all = []
    for this_index in range(index*self.bs,(index+1)*self.bs):
      line_index = self.indexes[index]
      line_str = linecache.getline(self.csv_path,line_index)
      line_val = [int(i) for i in line_str.split(',')]
      label = line_val[0]
      feat = np.array(line_val[1:])/self.norm_facor
      feat_all.append(feat)
      label_all.append(label)

    return np.array(feat_all),np.array(label_all)
```

Exercise

```
indexes = [i for i in range(60000)]
train_index = indexes[6000:]
val_index = indexes[:6000]
batch_size = 128
train_set = DataGenerator('/content/drive/MyDrive/keras/mnist_train.csv',train_index,batch_siz
e)
val_set = DataGenerator('/content/drive/MyDrive/keras/mnist_train.csv',val_index,batch_size)

model = Sequential()
model.add(Input(shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam',loss= 'sparse_categorical_crossentropy',metrics=['accuracy'])

epochs = 8
model.fit(train_set, steps_per_epoch = len(train_set), epochs=epochs, validation_data = val_se
t,validation_steps=len(val_set))
```

Exercise

1. Define the following model using Sequential API. Assuming padding is not used.
2. Define the following model using Functional API.
3. Use the structure as below to train a model for the MNIST task. Try using the entire dataset and data generator to train the model.
4. Write customized training loop for the MNIST task.



Exercise

```
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense,Input

import numpy as np
import linecache
import random
import tensorflow as tf
```

Exercise

```
class DataGenerator(keras.utils.Sequence):
  'Generates data for Keras'
  def __init__(self, csv_path, indexes, bs):
    # initilizes some variables
    self.csv_path = csv_path
    self.norm_facor = 255.0
    self.indexes = indexes
    self.bs = bs
    random.shuffle(self.indexes)
  def __len__(self):
    # return the total number of samples in the dataset
    return (len(self.indexes))//self.bs-1
  def __getitem__(self,index):
    # get one sample according to the index
    feat_all = []
    label_all = []
    for this_index in range(index*self.bs,(index+1)*self.bs):
      line_index = self.indexes[index]
      line_str = linecache.getline(self.csv_path,line_index)
      line_val = [int(i) for i in line_str.split(',') if len(i)]
      label = line_val[0]
      feat = np.array(line_val[1:])/self.norm_facor
      feat_all.append(feat)
      label_all.append(label)

    return np.array(feat_all),np.array(label_all)
  def shuffle(self):
    random.shuffle(self.indexes)
```

Exercise

```
indexes = [i for i in range(60000)]
train_index = indexes[6000:]
val_index = indexes[:6000]
batch_size = 128
train_set = DataGenerator('/content/drive/MyDrive/keras/mnist_train.csv',train_index,
batch_size)
val_set = DataGenerator('/content/drive/MyDrive/keras/mnist_train.csv',val_index,batc
h_size)

model = Sequential()
model.add(Input(shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam',loss= 'sparse_categorical_crossentropy',metrics=['accu
racy'])

loss_fn = keras.losses.SparseCategoricalCrossentropy()
optimizer = keras.optimizers.Adam()
val_acc_metric = keras.metrics.SparseCategoricalAccuracy()
epochs = 15
best_metric = 100
```

Exercise

```
for epoch in range(epochs):
  train_set.shuffle()
  for step, (x_batch_train, y_batch_train) in enumerate(train_set):
        with tf.GradientTape() as tape:
            logits = model(x_batch_train, training=True)
            loss_value = loss_fn(y_batch_train, logits)
        grads = tape.gradient(loss_value, model.trainable_weights)
        optimizer.apply_gradients(zip(grads, model.trainable_weights))
  for x_batch_val, y_batch_val in val_set:
    val_logits = model(x_batch_val, training=False)
    val_acc_metric.update_state(y_batch_val, val_logits)
  val_acc = val_acc_metric.result()
  print("[{}/{}],val_acc:{}".format(epoch,epochs,float(val_acc)))
  if float(val_acc) < best_metric:
    best_metric = float(val_acc)
    model.save('path_to_location.h5')
  val_acc_metric.reset_states()
```