CAPAIAN PEMBELAJARAN PRAKTIKUM

1. Menjelaskan konsep dari pembelajaran mesin kemudian diterpakan pada berbagai permasalahan – (C2)
2. Mengimplementasikan algoritma dan/atau metode di pembelajaran mesin sehingga mendapatkan pemecahan masalah dan model yang sesuai - (C3)
3. Menghasilkan suatu model terbaik dari permasalahan dan terus melakukan perbaikan jika terdapat perkembangan di bidang rekayasa cerdas – (C6)

MINGGU 8

Pembelajaran Mendalam

DESKRIPSI TEMA

Mahasiswa mempelajari algoritma pembelajaran terbimibng dengan pembelajaran mendalam dan algoritma Convolutional Neural Network serta menerapkannya dengan Bahasa pemrograman Python

CAPAIAN PEMBELAJARAN MINGGUAN (SUB- CAPAIAN PEMBELAJARAN)

Mahasiswa dapat menerapkan algoritma dengan pembelajaran mendalam sehingga bisa menghasilkan model terbaik dengan bahasa pemrograman Python (SCPMK-08)

PERALATAN YANG DIGUNAKAN

Anaconda Python 3

Laptop atau Personal Computer

LANGKAH-LANGKAH PRAKTIKUM

How to Instal Tensorflow and Keras in Python 3.8

Lets install Jupyter, which the editor we will use in this course :

```
conda install -y jupyter
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

We will actually lunch Jupyter later.

```
conda env create -f tensorflow.yml -n tensorflow
```

```
CondaValueError: prefix already exists: /Users/vastyoverbeek/opt/anaconda3/envs/tensorflow
```

Install the tensorflow.yml (inserted in this file). Run the following command at the same directory that containts the file (tensorflow.yml).

```
conda activate tensorflow
```

For now, lets add jupyter support to your new environment

```
conda install nb_conda
```

```
Collecting package metadata (current_repodata.
```

Testing your Environment

Run your Jupyter notebook and check the versions expected.

```python
import sys

import tensorflow.keras
import pandas as pd
import sklearn as sk
import tensorflow as tf
```

From the code, we get print the output with this following code :

```python
print(f"Tensor Flow Version: {tf.__version__}")
print(f"Keras Version: {tensorflow.keras.__version__}")
print()
print(f"Python {sys.version}")
print(f"Pandas {pd.__version__}")
print(f"Scikit-Learn {sk.__version__}")
```

The output is :

```
Tensor Flow Version: 2.4.0
Keras Version: 2.4.0

Python 3.8.12 (default, Oct 12 2021, 06:23:56)
[Clang 10.0.0 ]
Pandas 1.3.3
Scikit-Learn 0.24.2
```

Convolutional Neural Networks (CNN)

CNN is a neural network technology that has profoundly impacted the area of computer vision (CV). This advantage in CNN is due to years of research on biological eyes. In other words, CNN utilize overlapping fields of input to simulate features of biological eyes.

Until this breakthrough, AI had been unable to reproduce the capabilities of biological vision.

The Dataset in This Course

There are many datasets for computer vision. Two of the most popular are the MNIST digits dataset and the CIFAR image datasets.

MNIST Digits dataset

The MNIST digits data set is very popular in the neural network research community. A sample of it can be seen in Figure 1.

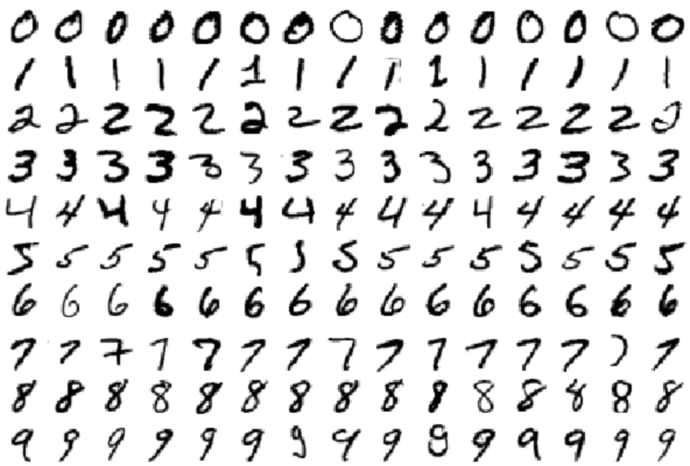


Figure 1 MNIST Dataset (LeCun et al, 1998)

This dataset was generated from scanned from, such as seen in Figure 2.

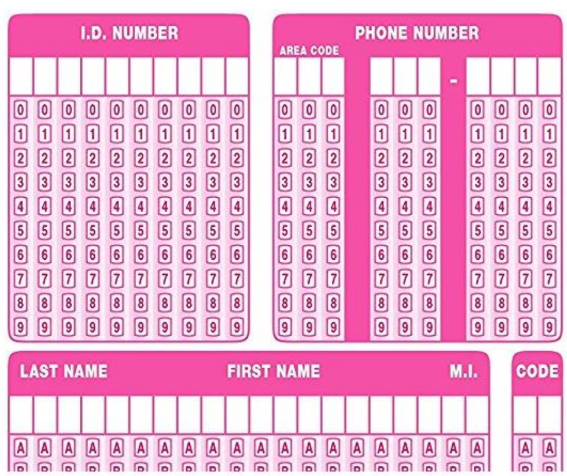


Figure 2 Exam Form

MNIST Fashion Dataset

Fashion MNIST is a dataset from Zalando's article images- containing of a training set of 70 000 example and a test set of 10 000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is intended to serve as a direct drop in replacement for the original MNIST dataset for benchmarking machine learning algorithm. It shares the same image size and structure of training and testing splits. The dataset can be seen in Figure 3.

Figure 3 Fashion MNIST Dataset (Zalando, 2017)

TensorFlow with CNN

Access to Dataset –DIGITS

Keras provides build in access classes for MNIST. It is important to note that MNIS data arrives already separated into two sets :

- Train : neural network will be trained with thist
- Test : used for validation

1. Import the libraries

```
import tensorflow.keras
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
print("Shape of x_train: {}".format(x_train.shape))
print("Shape of y_train: {}".format(y_train.shape))
print()
print("Shape of x_test: {}".format(x_test.shape))
print("Shape of y_test: {}".format(y_test.shape))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 4s 0us/step
Shape of x_train: (60000, 28, 28)
Shape of y_train: (60000,)

Shape of x_test: (10000, 28, 28)
Shape of y_test: (10000,)
```

2. Display the Digits

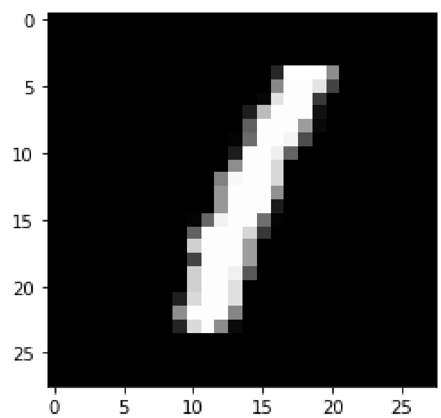
```
from IPython.display import display
import pandas as pd

# Display as text
pd.set_option('display.max_columns', 15)
pd.set_option('display.max_rows', 5)

print("Shape for dataset: {}".format(x_train.shape))
print("Labels: {}".format(y_train))

# Single MNIST digit
single = x_train[0]
print("Shape for single: {}".format(single.shape))

pd.DataFrame(single.reshape(28,28))
```

```
Shape for dataset: (60000, 28, 28)
Labels: [5 0 4 ... 5 6 8]
Shape for single: (28, 28)
```

3. Lets display as image

```python
# Display as image
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
digit = 105 # Change to choose new digit
a = x_train[digit]
plt.imshow(a, cmap='gray', interpolation='nearest')
print("Image (#{}): Which is digit '{}'".format(digit,y_train[digit]))
```

```
Image (#105): Which is digit '1'
```



4. We can show the dataset with random. Using random library to show the figure

```python
import random

ROWS = 6
random_indices = random.sample(range(x_train.shape[0]), ROWS*ROWS)

sample_images = x_train[random_indices, :]

plt.clf()

fig, axes = plt.subplots(ROWS,ROWS,
                         figsize=(ROWS,ROWS),
                         sharex=True, sharey=True)

for i in range(ROWS*ROWS):
    subplot_row = i//ROWS
    subplot_col = i%ROWS
    ax = axes[subplot_row, subplot_col]

    plottable_image = np.reshape(sample_images[i,:], (28,28))
    ax.imshow(plottable_image, cmap='gray_r')

    ax.set_xbound([0,28])

plt.tight_layout()
plt.show()
```

5. Split the data and input the parameter of CNN algorithm before make a model (train the data)

   The parameters are :

   - Batch size : 128
   - Epoch : 12 epoch
   - Number of classes = 10 class
   - Picture size : 28x28
   - Data type of training and testing : floating number
   - Activated function : ReLu
   - Model compiling optimizer : ADAM optimizer

```
import tensorflow.keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras import backend as K
batch_size = 128
num_classes = 10
epochs = 12
# input image dimensions
img_rows, img_cols = 28, 28
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# Normalize the data
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print("Training samples: {}".format(x_train.shape[0]))
print("Test samples: {}".format(x_test.shape[0]))
```

```
# convert class vectors to binary class matrices
y_train = tensorflow.keras.utils.to_categorical(y_train, num_classes)
y_test = tensorflow.keras.utils.to_categorical(y_test, num_classes)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
```

```
x_train shape: (60000, 28, 28, 1)
Training samples: 60000
Test samples: 10000
```

6.Training the CNN – DIGITS datasets. This can take awhile

```
import tensorflow as tf
import time

start_time = time.time()

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=2,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss: {}'.format(score[0]))
print('Test accuracy: {}'.format(score[1]))

elapsed_time = time.time() - start_time
print("Elapsed time: {}".format(hms_string(elapsed_time)))
```

The output from 12 epoch DIGITS dataset :

```
Epoch 1/12
469/469 - 45s - loss: 0.2474 - accuracy: 0.9243 - val_loss: 0.0574 - val_accuracy: 0.9813
Epoch 2/12
469/469 - 47s - loss: 0.0886 - accuracy: 0.9735 - val_loss: 0.0363 - val_accuracy: 0.9882
Epoch 3/12
469/469 - 45s - loss: 0.0660 - accuracy: 0.9800 - val_loss: 0.0356 - val_accuracy: 0.9878
Epoch 4/12
469/469 - 46s - loss: 0.0547 - accuracy: 0.9834 - val_loss: 0.0329 - val_accuracy: 0.9894
Epoch 5/12
469/469 - 50s - loss: 0.0455 - accuracy: 0.9859 - val_loss: 0.0365 - val_accuracy: 0.9890
Epoch 6/12
469/469 - 48s - loss: 0.0401 - accuracy: 0.9874 - val_loss: 0.0309 - val_accuracy: 0.9903
Epoch 7/12
469/469 - 47s - loss: 0.0346 - accuracy: 0.9890 - val_loss: 0.0286 - val_accuracy: 0.9915
Epoch 8/12
469/469 - 49s - loss: 0.0309 - accuracy: 0.9898 - val_loss: 0.0255 - val_accuracy: 0.9931
Epoch 9/12
469/469 - 47s - loss: 0.0276 - accuracy: 0.9910 - val_loss: 0.0299 - val_accuracy: 0.9914
Epoch 10/12
469/469 - 48s - loss: 0.0255 - accuracy: 0.9916 - val_loss: 0.0285 - val_accuracy: 0.9912
Epoch 11/12
469/469 - 50s - loss: 0.0234 - accuracy: 0.9923 - val_loss: 0.0278 - val_accuracy: 0.9922
Epoch 12/12
469/469 - 51s - loss: 0.0229 - accuracy: 0.9926 - val_loss: 0.0278 - val_accuracy: 0.9926
Test loss: 0.027780190110206604
Test accuracy: 0.9926000237464905
```

7. Evaluate accuracy from DIGITS dataset

```python
# Set the desired TensorFlow output level
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss: {}'.format(score[0]))
print('Test accuracy: {}'.format(score[1]))
```

```
Test loss: 0.027780190110206604
Test accuracy: 0.9926000237464905
```

```python
from sklearn import metrics

small_x = x_test[1:100]
small_y = y_test[1:100]
small_y2 = np.argmax(small_y,axis=1)
pred = model.predict(small_x)
pred = np.argmax(pred,axis=1)
score = metrics.accuracy_score(small_y2, pred)
print('Accuracy: {}'.format(score))
```

```
Accuracy: 1.0
```

Fashion MNIST Dataset

8. Import the libraries

```
import tensorflow.keras
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
print("Shape of x_train: {}".format(x_train.shape))
print("Shape of y_train: {}".format(y_train.shape))
print()
print("Shape of x_test: {}".format(x_test.shape))
print("Shape of y_test: {}".format(y_test.shape))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 8s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 1s 0us/step
Shape of x_train: (60000, 28, 28)
Shape of y_train: (60000,)

Shape of x_test: (10000, 28, 28)
Shape of y_test: (10000,)
```

9. Display the Fashion MNIST

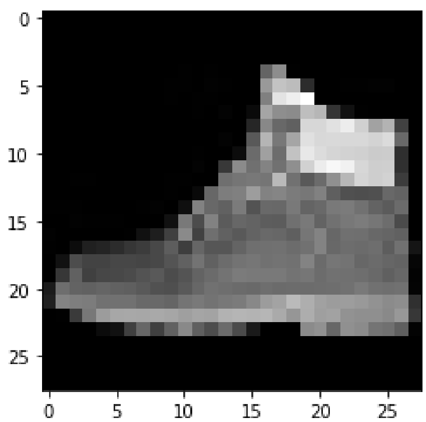
10. Let’s display as an image and text

```
# Display as text
from IPython.display import display
import pandas as pd

print("Shape for dataset: {}".format(x_train.shape))
print("Labels: {}".format(y_train))

# Single MNIST digit
single = x_train[0]
print("Shape for single: {}".format(single.shape))

pd.set_option('display.max_columns', 7)
pd.set_option('display.max_rows', 10)
pd.DataFrame(single.reshape(28,28))
```

```
Shape for dataset: (60000, 28, 28)
Labels: [9 0 0 ... 3 0 5]
Shape for single: (28, 28)
```

```
# Display as image
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
digit = 90 # Change to choose new article
a = x_train[digit]
plt.imshow(a, cmap='gray', interpolation='nearest')
print("Image (#{}): Which is digit '{}'".format(digit,y_train[digit]))
```

```
Image (#90): Which is digit '9'
```



11. We can show the dataset with random. Using random library to show the figure

```
import random

ROWS = 6
random_indices = random.sample(range(x_train.shape[0]), ROWS*ROWS)

sample_images = x_train[random_indices, :]

plt.clf()

fig, axes = plt.subplots(ROWS,ROWS,
                         figsize=(ROWS,ROWS),
                         sharex=True, sharey=True)

for i in range(ROWS*ROWS):
    subplot_row = i//ROWS
    subplot_col = i%ROWS
    ax = axes[subplot_row, subplot_col]

    plottable_image = np.reshape(sample_images[i,:], (28,28))
    ax.imshow(plottable_image, cmap='gray_r')

    ax.set_xbound([0,28])

plt.tight_layout()
plt.show()
```

12. Split the data and input the parameter of CNN algorithm before make a model (train the data)

The parameters are :

- Batch size : 128
- Epoch : 12 epoch
- Number of classes = 10 class
- Picture size : 28x28
- Data type of training and testing : floating number
- Activated function : ReLu
- Model compiling optimizer : ADAM optimizer

```
import tensorflow.keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras import backend as K
batch_size = 128
num_classes = 10
epochs = 12
# input image dimensions
img_rows, img_cols = 28, 28
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
#Normalize dataset
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print("Training samples: {}".format(x_train.shape[0]))
print("Test samples: {}".format(x_test.shape[0]))
```

```
# convert class vectors to binary class matrices
y_train = tensorflow.keras.utils.to_categorical(y_train, num_classes)
y_test = tensorflow.keras.utils.to_categorical(y_test, num_classes)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
```

```
x_train shape: (60000, 28, 28, 1)
Training samples: 60000
Test samples: 10000
```

13. Training the CNN – DIGITS datasets. This can take awhile

```
import tensorflow as tf
import time

start_time = time.time()

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=2,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss: {}'.format(score[0]))
print('Test accuracy: {}'.format(score[1]))
```

The output from 12 epoch DIGITS dataset :

```
Epoch 1/12
469/469 - 44s - loss: 0.5295 - accuracy: 0.8130 - val_loss: 0.3404 - val_accuracy: 0.8784
Epoch 2/12
469/469 - 44s - loss: 0.3475 - accuracy: 0.8773 - val_loss: 0.2919 - val_accuracy: 0.8949
Epoch 3/12
469/469 - 44s - loss: 0.2976 - accuracy: 0.8931 - val_loss: 0.2604 - val_accuracy: 0.9054
Epoch 4/12
469/469 - 44s - loss: 0.2636 - accuracy: 0.9041 - val_loss: 0.2420 - val_accuracy: 0.9124
Epoch 5/12
469/469 - 46s - loss: 0.2412 - accuracy: 0.9128 - val_loss: 0.2525 - val_accuracy: 0.9071
Epoch 6/12
469/469 - 46s - loss: 0.2193 - accuracy: 0.9202 - val_loss: 0.2217 - val_accuracy: 0.9215
Epoch 7/12
469/469 - 46s - loss: 0.2061 - accuracy: 0.9236 - val_loss: 0.2237 - val_accuracy: 0.9198
Epoch 8/12
469/469 - 47s - loss: 0.1886 - accuracy: 0.9296 - val_loss: 0.2202 - val_accuracy: 0.9199
Epoch 9/12
469/469 - 47s - loss: 0.1768 - accuracy: 0.9339 - val_loss: 0.2119 - val_accuracy: 0.9249
Epoch 10/12
469/469 - 47s - loss: 0.1644 - accuracy: 0.9383 - val_loss: 0.2087 - val_accuracy: 0.9260
Epoch 11/12
469/469 - 48s - loss: 0.1547 - accuracy: 0.9420 - val_loss: 0.2077 - val_accuracy: 0.9277
Epoch 12/12
469/469 - 48s - loss: 0.1483 - accuracy: 0.9434 - val_loss: 0.2141 - val_accuracy: 0.9301
Test loss: 0.2141042947769165
Test accuracy: 0.9301000237464905
```

REFERENSI

1. Geron A. 2017. Hands on Machine Learning wirh Scikit Learn and TensorFlow. O Reilly Media Inc
2. Van derPlas J. 2016. Pyton Data Science Handbook. Oreilly Media Inc
3. LeCun Y, Cortes C, Burges CJC. 1998. The MNIST Database of Handwritten Digits. Internet : http://yann.lecun.com/exdb/mnist/ .
4. Krizhevsky A, Nair V, Hinton G. 2009. The CIFAR Dataset. Internet : https://www.cs.toronto.edu/~kriz/cifar.html