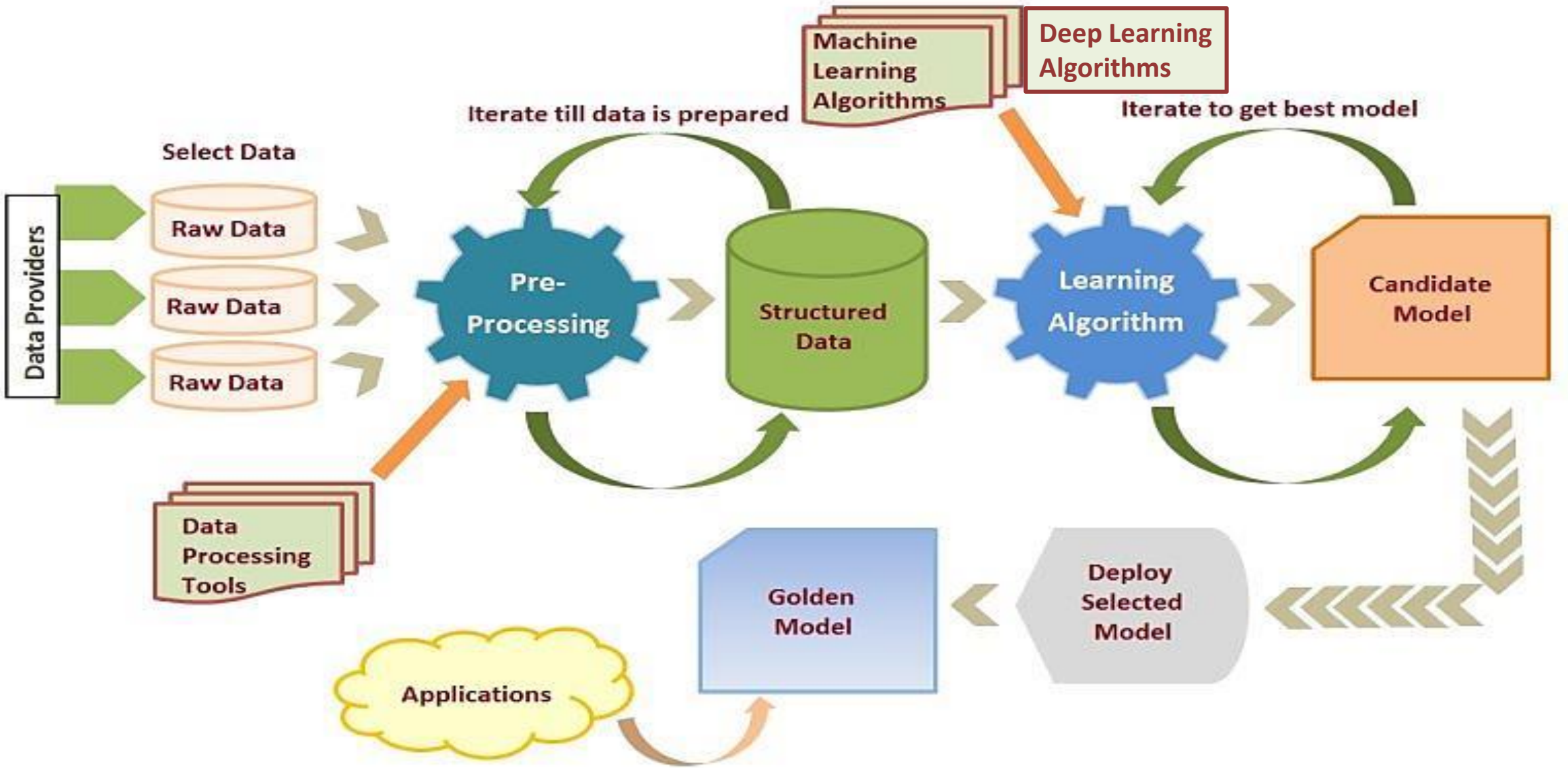


Implementasi *Deep Learning*: Matlab dan Python-Keras-Tensorflow

Achmad Benny Mutiara

2021

Proses Machine Learning / Deep Learning



Definisi Deep Learning

- **Deep learning** adalah jenis pembelajaran mesin dengan model belajar dengan melakukan tugas-tugas klasifikasi langsung dari gambar, teks, atau suara.
- **Deep Learning** biasanya diimplementasikan menggunakan arsitektur jaringan saraf.
- Istilah “**Deep**" mengacu pada jumlah lapisan dalam jaringan — semakin banyak lapisan, semakin dalam jaringan (deeper network).
- Jaringan saraf tradisional hanya mengandung 2 atau 3 lapisan, sedangkan **Deep Network** dapat memiliki ratusan.

Aplikasi Deep Learning

Beberapa Contoh Aplikasi DL

- Kendaraan yang bisa menyetir sendiri me-lambat saat mendekati penyeberangan pejalan kaki.
- ATM menolak uang kertas palsu.
- Aplikasi smartphone memberikan terjemahan instan tanda jalan berbahasa asing

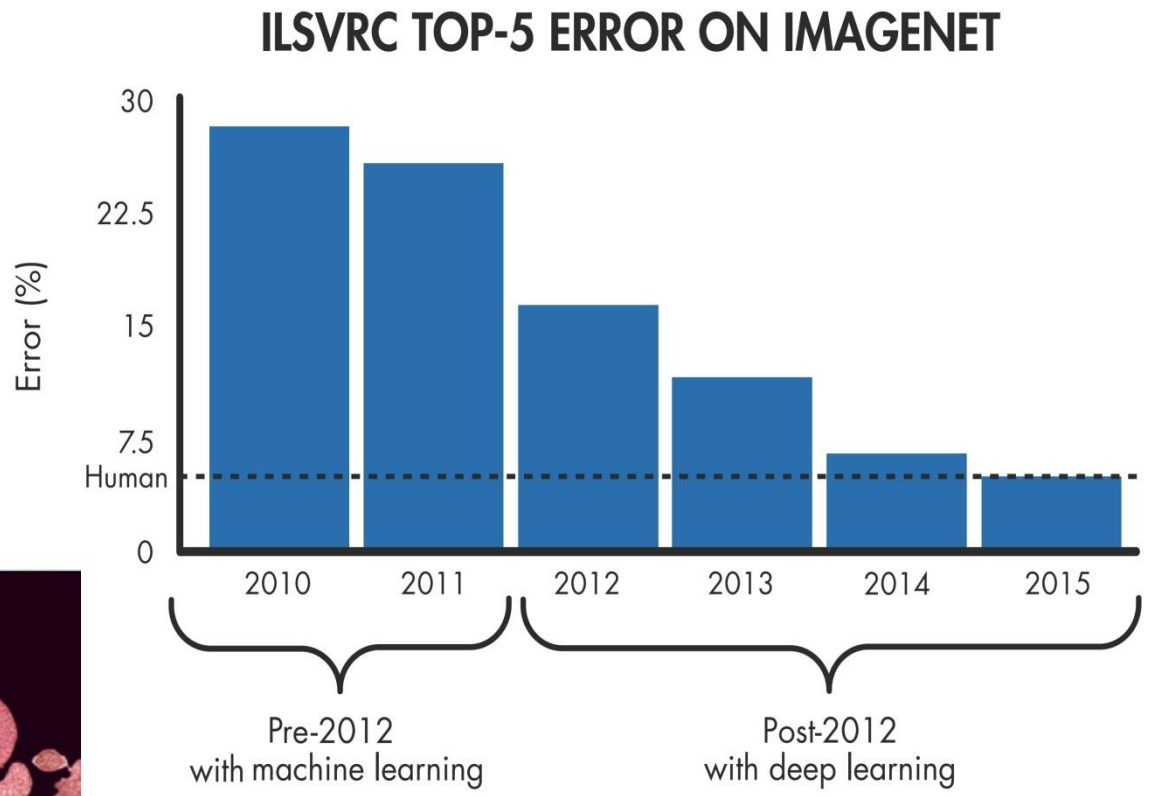


DL sangat cocok untuk aplikasi identifikasi seperti pengenalan wajah, terjemahan teks, pengenalan suara, dan sistem bantuan pengemudi tingkat lanjut, termasuk, klasifikasi jalur dan pengenalan rambu lalu lintas.

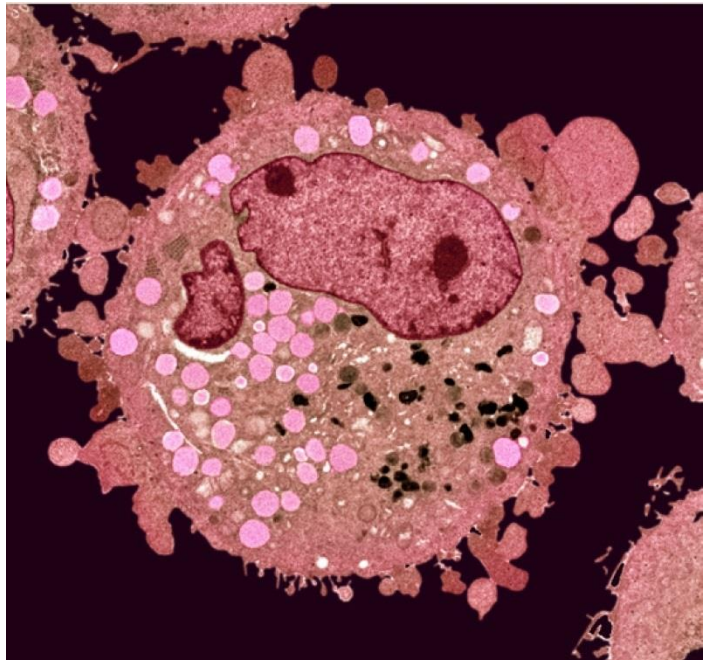
Apa yang Membuat DL Menjadi Terdepan?

Hanya kunci satu kata, akurasi

Tool dan teknik canggih telah secara dramatis meningkatkan algoritme **DL** — ke titik di mana mereka dapat mengungguli manusia dalam mengklasifikasikan gambar, menang melawan pemain GO terbaik dunia, atau memungkinkan asisten yang dikendalikan suara seperti **Amazon Echo®** dan **Google Home** untuk menemukan dan mengunduh yang lagu baru yang kita suka.



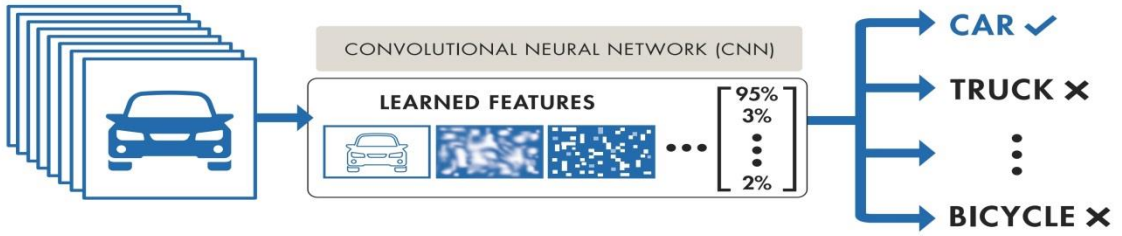
Peneliti UCLA membangun mikroskop canggih yang menghasilkan **dataset** dimensi tinggi yang digunakan untuk melatih jaringan **DL** untuk mengidentifikasi sel-sel kanker dalam sampel jaringan.



Apa yang Membuat DL Menjadi Terdepan?

Tiga enabler teknologi memungkinkan tingkat akurasi ini:

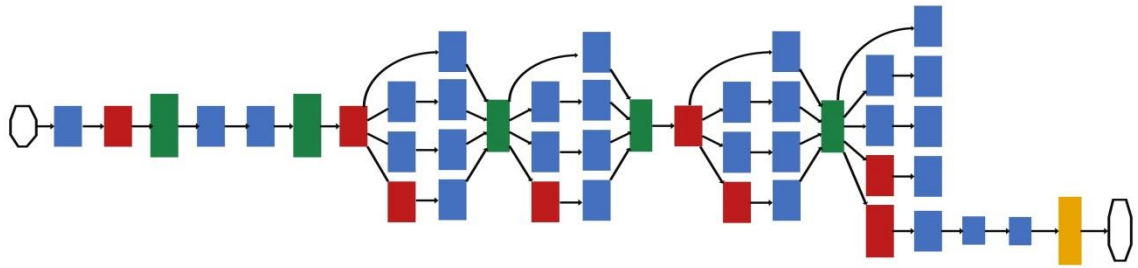
- **Akses mudah ke sejumlah besar data berlabel**
  - Datasets seperti **ImageNet** dan **PASCAL VoC** tersedia secara bebas, dan berguna untuk pelatihan berbagai jenis objek.



- **Peningkatan kekuatan komputasi**
  - GPU berkinerja tinggi mempercepat pelatihan sejumlah besar data yang diperlukan untuk **DL**, mengurangi waktu pelatihan dari minggu ke jam.



- **Model pra-pelatihan dibangun oleh para ahli**
  - Model seperti **AlexNet** dapat dilatih ulang untuk melakukan tugas-tugas pengenalan baru meng-gunakan teknik yang disebut **transfer learning**. Sementara **AlexNet** dilatih pada 1,3 juta gambar resolusi tinggi untuk mengenali 1000 objek yang berbeda, **pembelajaran transfer** yang akurat dapat dicapai dengan dataset yang jauh lebih kecil.

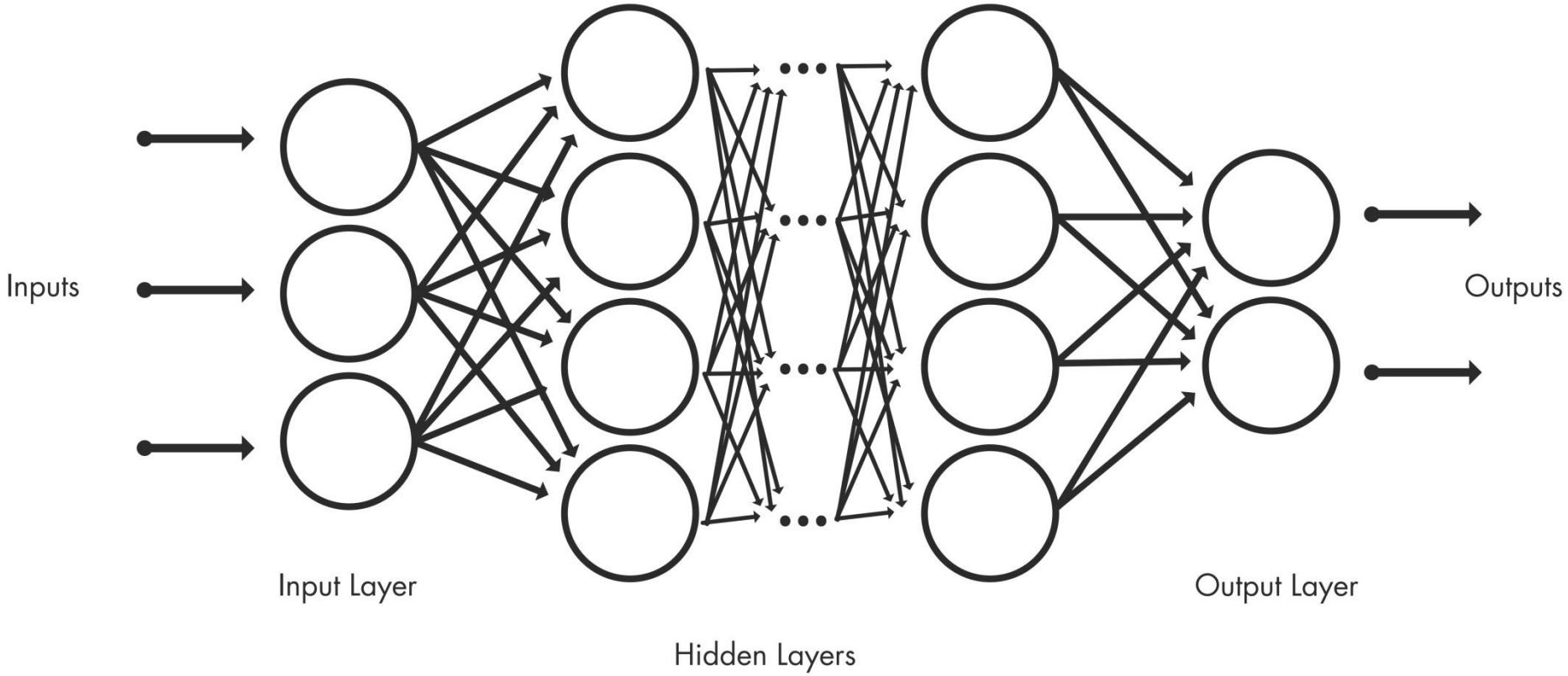


Bagan Umum Deep Neural Netwok (DNN)

DNN menggabungkan beberapa lapisan pemrosesan nonlinear, menggunakan elemen sederhana yang beroperasi secara paralel dan terinspirasi oleh sistem saraf biologis.

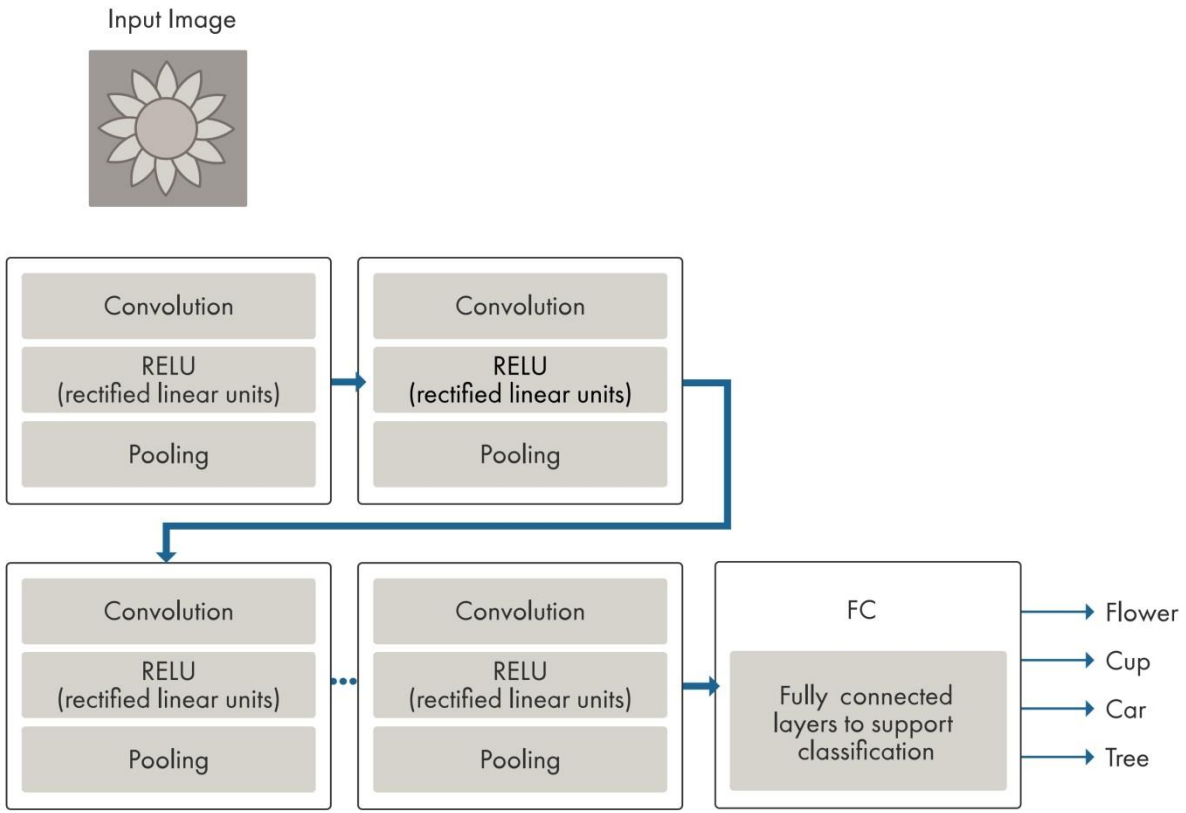
Terdiri dari lapisan input, beberapa lapisan tersembunyi, dan lapisan keluaran.

Lapisan saling berhubungan melalui node, atau neuron, dengan setiap lapisan tersembunyi menggunakan output dari lapisan sebelumnya sebagai inputnya.



Bagaimana DNN Belajar ?

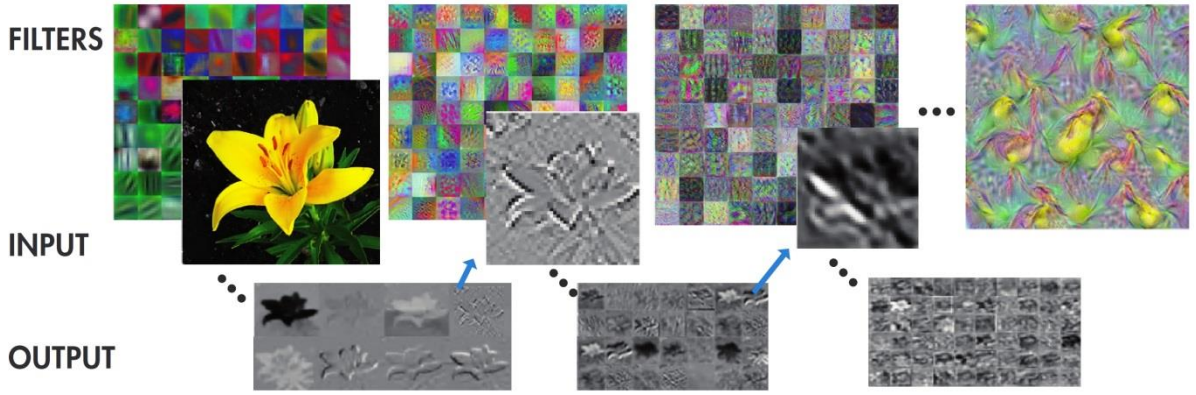
Katakanlah kita memiliki satu set gambar di mana setiap gambar berisi satu dari empat kategori objek yang berbeda, dan kita ingin DNN untuk secara otomatis mengenali objek mana yang ada di setiap gambar. Kita memberi label gambar agar kita memiliki data pelatihan untuk jaringan.



Dengan menggunakan data pelatihan ini, jaringan kemudian dapat mulai memahami fitur spesifik objek dan mengaitkannya dengan kategori yang sesuai.

Setiap lapisan dalam jaringan mengambil data dari lapisan sebelumnya, mentransformasikannya, dan meneruskannya. Jaringan meningkatkan kompleksitas dan detail dari apa yang dipelajari dari lapisan ke lapisan.

Perhatikan bahwa jaringan belajar langsung dari data — kita **tidak memiliki pengaruh atas fitur apa yang sedang dipelajari.**



Convolutional Neural Networks (CNN)

Jaringan saraf convolutional (**CNN**, atau **ConvNet**) adalah salah satu algoritma paling populer dari **DL** untuk gambar dan video.

Seperti jaringan saraf lainnya, **CNN** terdiri dari **lapisan input**, **lapisan output**, dan **banyak lapisan tersembunyi** di antaranya.

Lapisan Deteksi Fitur

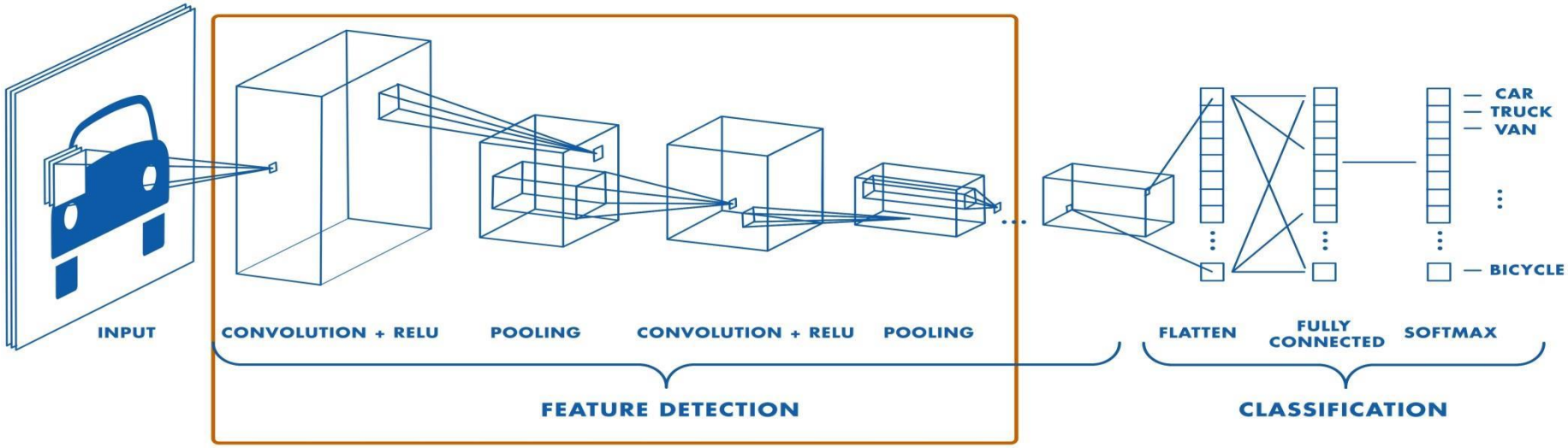
Lapisan-lapisan ini melakukan **salah satu** dari **tiga jenis** operasi pada data: **konvolusi**, **pooling**, atau **unit linear yang diperbaiki** (**ReLU**).

Konvolusi menempatkan gambar input melalui serangkaian filter konvolusional, yang masing-masing mengaktifkan fitur tertentu dari gambar.

Pooling menyederhanakan output dengan melakukan non-linear downsampling, mengurangi jumlah parameter yang perlu dipelajari jaringan.

Rectified linear unit (ReLU) memungkinkan pelatihan yang lebih cepat dan lebih efektif dengan memetakan nilai negatif menjadi nol dan mempertahankan nilai positif.

Ketiga operasi ini diulang lebih dari puluhan atau ratusan lapisan, dengan setiap lapisan belajar untuk mendeteksi fitur yang berbeda.



Convolutional Neural Networks (CNN)

Lapisan Klasifikasi

Setelah deteksi fitur, arsitektur **CNN** bergeser ke klasifikasi.

Lapisan berikutnya ke yang terakhir adalah **lapisan yang sepenuhnya terhubung** (**FC**) yang menghasilkan vektor dimensi **K** di mana **K** adalah **jumlah kelas** yang akan dapat diprediksi oleh jaringan.

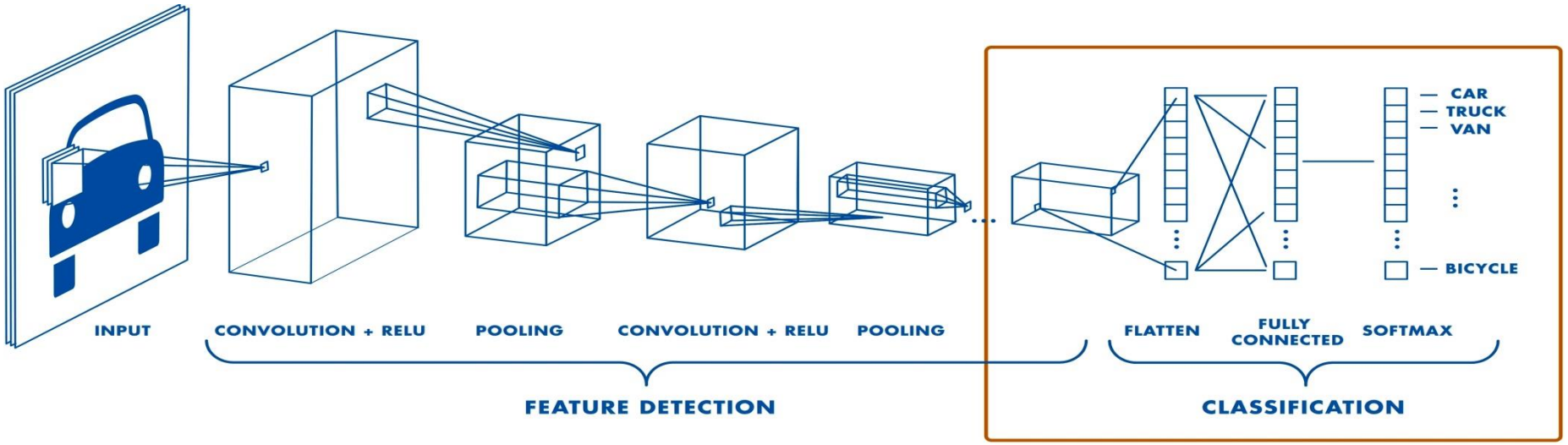
Vektor ini berisi probabilitas untuk setiap kelas dari gambar apa pun yang diklasifikasikan.

Lapisan terakhir dari arsitektur **CNN** menggunakan fungsi **softmax** untuk memberikan hasil klasifikasi.

Catatan:

***Tidak ada formula** yang tepat untuk memilih lapisan.*

***Pendekatan terbaik** adalah dengan mencoba beberapa dan melihat seberapa baik mereka bekerja - atau menggunakan jaringan yang sudah ada sebelumnya.*



Apa Perbedaan antara DL dan ML?

DL adalah subtipe **ML**.

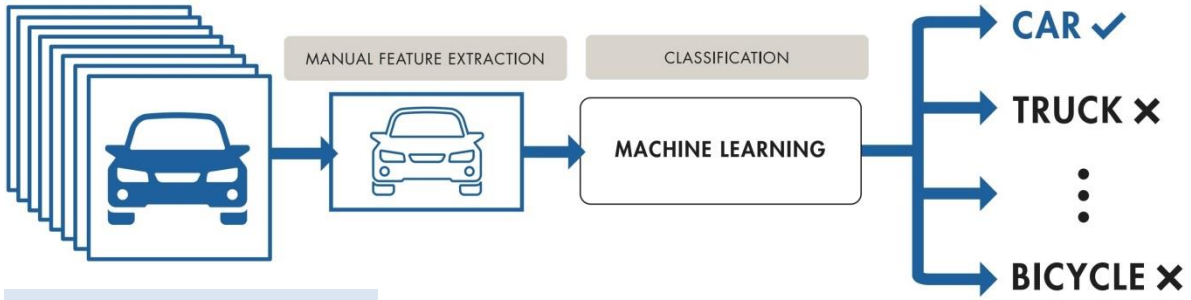
Dengan **ML**, kita mengekstraksi fitur gambar yang relevan secara manual.

Dengan **DL** , kita memasukkan gambar mentah langsung ke **DNN** yang mempelajari fitur secara otomatis.

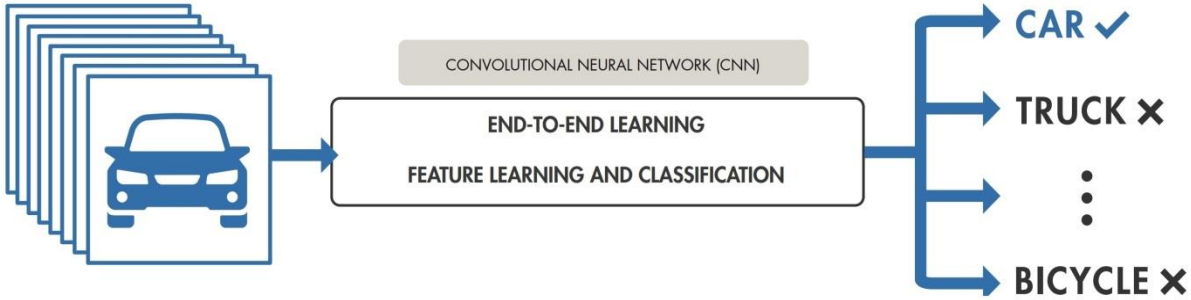
DL sering membutuhkan ratusan ribu atau jutaan gambar untuk hasil terbaik.

DL juga **intensif secara komputasi** dan membutuhkan GPU berkinerja tinggi.

Machine Learning Tradisional



Deep Learning



ML	DL
+ Hasil bagus dengan dataset kecil	– Membutuhkan dataset yang sangat besar
+ Cepat untuk melatih model	– Intensif secara komputasi
– Perlu mencoba berbagai fitur dan pengklasifikasi untuk mencapai hasil terbaik	Mempelajari fitur dan pengklasifikasi secara otomatis
– Akurasi datar (stabil tanpa kemajuan)	Akurasi tidak terbatas

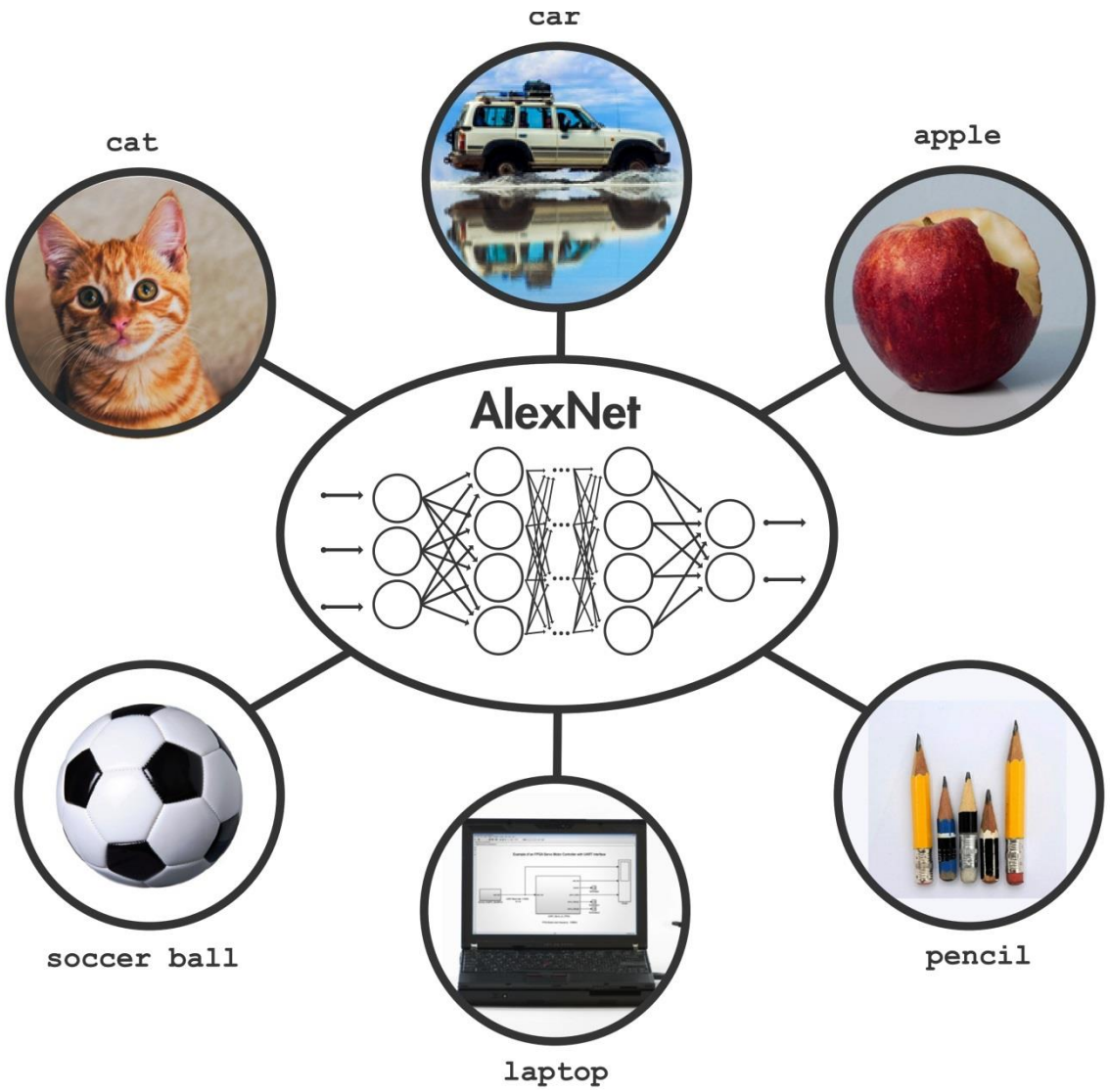
Memulai dengan Deep Learning

Belajar **DL**: **cara cepat dan mudah** untuk memulai adalah dengan menggunakan jaringan yang ada, seperti **AlexNet**, **CNN** yang dilatih pada lebih dari satu juta gambar.

AlexNet paling umum digunakan untuk klasifikasi gambar.

AlexNet dapat mengklasifikasikan gambar ke dalam 1000 kategori yang berbeda, termasuk keyboard, mouse komputer, pensil, dan peralatan kantor lainnya, serta berbagai jenis anjing, kucing, kuda, dan hewan lainnya.

AlexNet pertama kali diterbitkan pada tahun 2012, dan telah menjadi model yang terkenal di komunitas penelitian.



Tool-Tool Implementasi

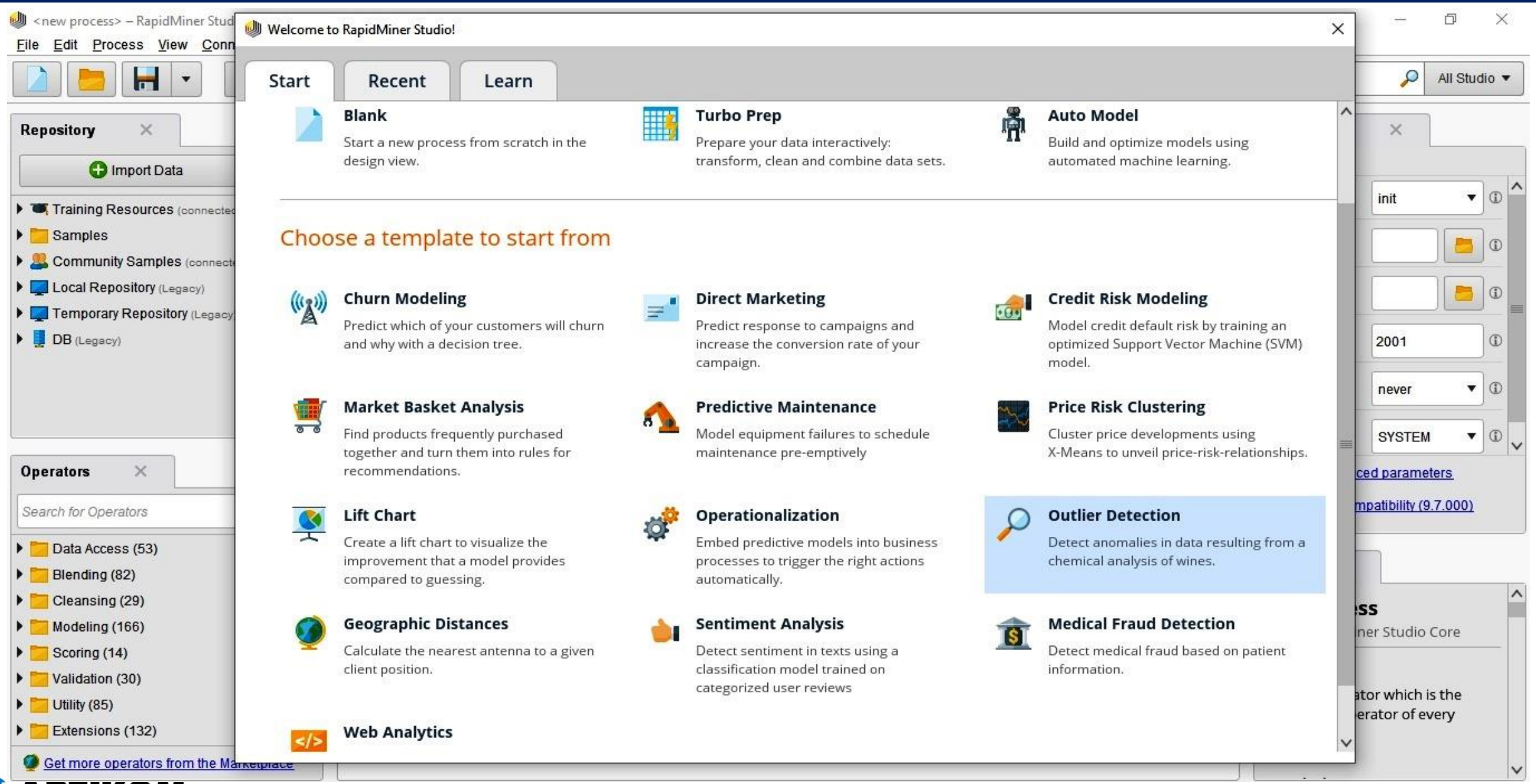
Tool Implementasi: RapidMiner

- RapidMiner https://rapidminer.com/

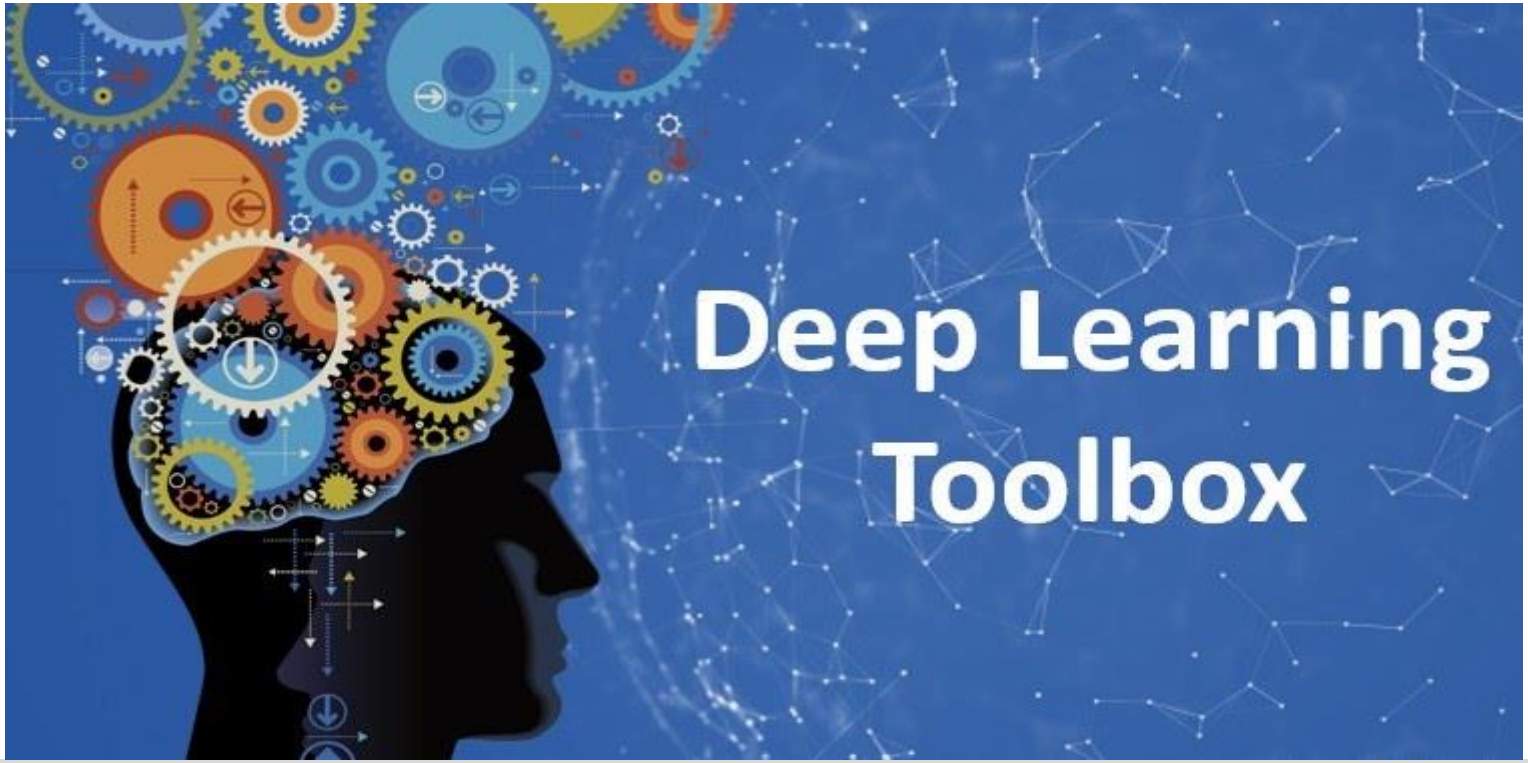


  - platform perangkat lunak data science
  - yang dikembangkan oleh perusahaan bernama sama dengan yang menyediakan lingkungan terintegrasi untuk **data preparation, machine learning, deep learning, text mining, and predictive analytics.**
  - Digunakan untuk bisnis dan komersial, juga untuk penelitian, pendidikan, pelatihan, *rapid prototyping*, dan pengembangan aplikasi serta mendukung semua langkah dalam proses **machine learning termasuk data preparation, results visualization, model validation and optimization**.
  - RapidMiner dikembangkan pada **open core model**. Dengan RapidMiner Studio **Free Edition**, yang terbatas untuk 1 prosesor logika dan 10.000 baris data, tersedia di bawah lisensi AGPL. **RapidMiner Studio 9.7 (https://my.rapidminer.com/nexus/account/index.html#downloads)** Harga komersial dimulai dari $2.500 dan tersedia dari pengembang.
- Link buku RapidMiner:

Tool Implementasi: RapidMiner



APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER



Deep Learning dengan MATLAB

Tool Implementasi: Matlab

- **Matlab** https://www.mathworks.com/products/matlab.html
  - Komersial versi terakhir R2020a
  - Tersedia Toolbox: **AI, Data Science, and Statistics**
    - Statistics and Machine Learning Toolbox
    - Deep Learning Toolbox
    - Reinforcement Learning Toolbox
    - Text Analytics Toolbox
    - Predictive Maintenance Toolbox



- **Link buku Matlab**
- **Link buku Deep Learning with Matlab**
- **Code Matlab Webinar**

Contoh Menggunakan AlexNet dengan MATLAB

Menggunakan **AlexNet** untuk mengklasifikasikan objek dalam gambar apa pun. Dalam contoh ini, **AlexNet** akan digunakan untuk mengklasifikasikan objek dalam gambar dari **webcam** yang diinstal pada desktop. Selain MATLAB®, akan digunakan hal-hal berikut:

- **Deep Learning Toolbox™**
- Support package utk menggunakan **webcam**
- Support package utk menggunakan **AlexNet**

Setelah memuat **AlexNet**, **webcam** terhubung dan dapat menangkap gambar langsung.

```
camera = webcam; % Connect to the camera
nnet = alexnet;  % Load the neural net
picture = camera.snapshot;  % Take a picture
```

Selanjutnya, ukuran gambar diubah 227x227 piksel, ukuran yang dibutuhkan oleh **AlexNet**

```
picture = imresize(picture,[227,227]); %
Resize the picture
```

AlexNet sekarang dapat mengklasifikasikan gambar

```
label = classify(nnet, picture); %
Classify the picture
image(picture); % Show the picture
title(char(label)); % Show the label
```



Pelatihan ulang (retraining) Jaringan yang ada di MATLAB

Pada contoh sebelumnya, digunakan jaringan langsung dari tool-box. **Tidak dimodifikasi** dengan cara apa pun karena **AlexNet** dilatih untuk gambar yang mirip dengan yang akan diklasifikasikan.

Menggunakan **AlexNet** untuk objek ***yang tidak dilatih*** di jaringan asli, **AlexNet** dapat dilatih melalui **transfer pembelajaran**.

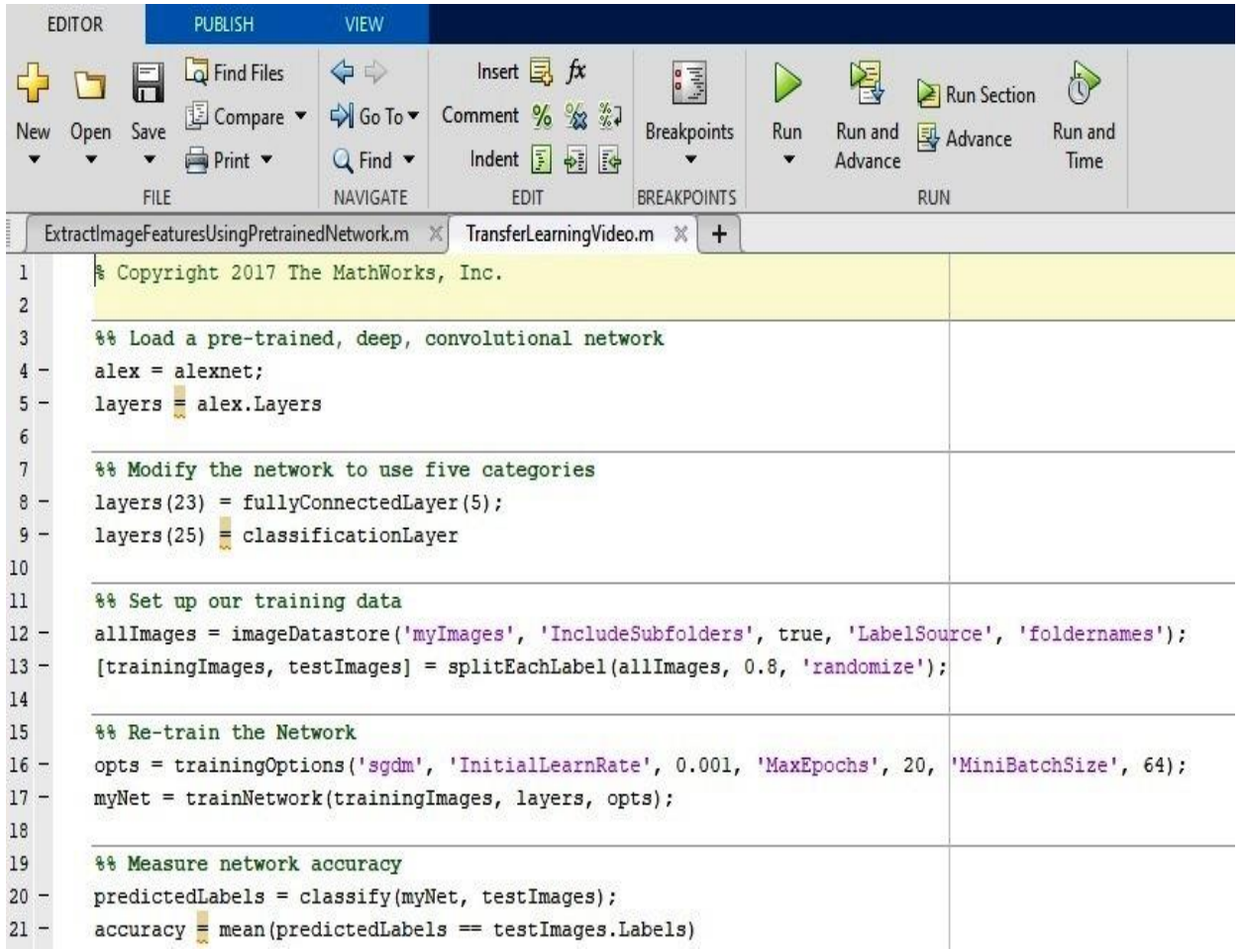
Transfer pembelajaran adalah **pendekatan yang mene-rapkan pengetahuan tentang satu jenis masalah untuk masalah yang berbeda tetapi terkait.**

Dalam hal ini, **cukup memotong 3 lapisan terakhir** dari jaringan dan melatihnya dengan gambar kita sendiri.

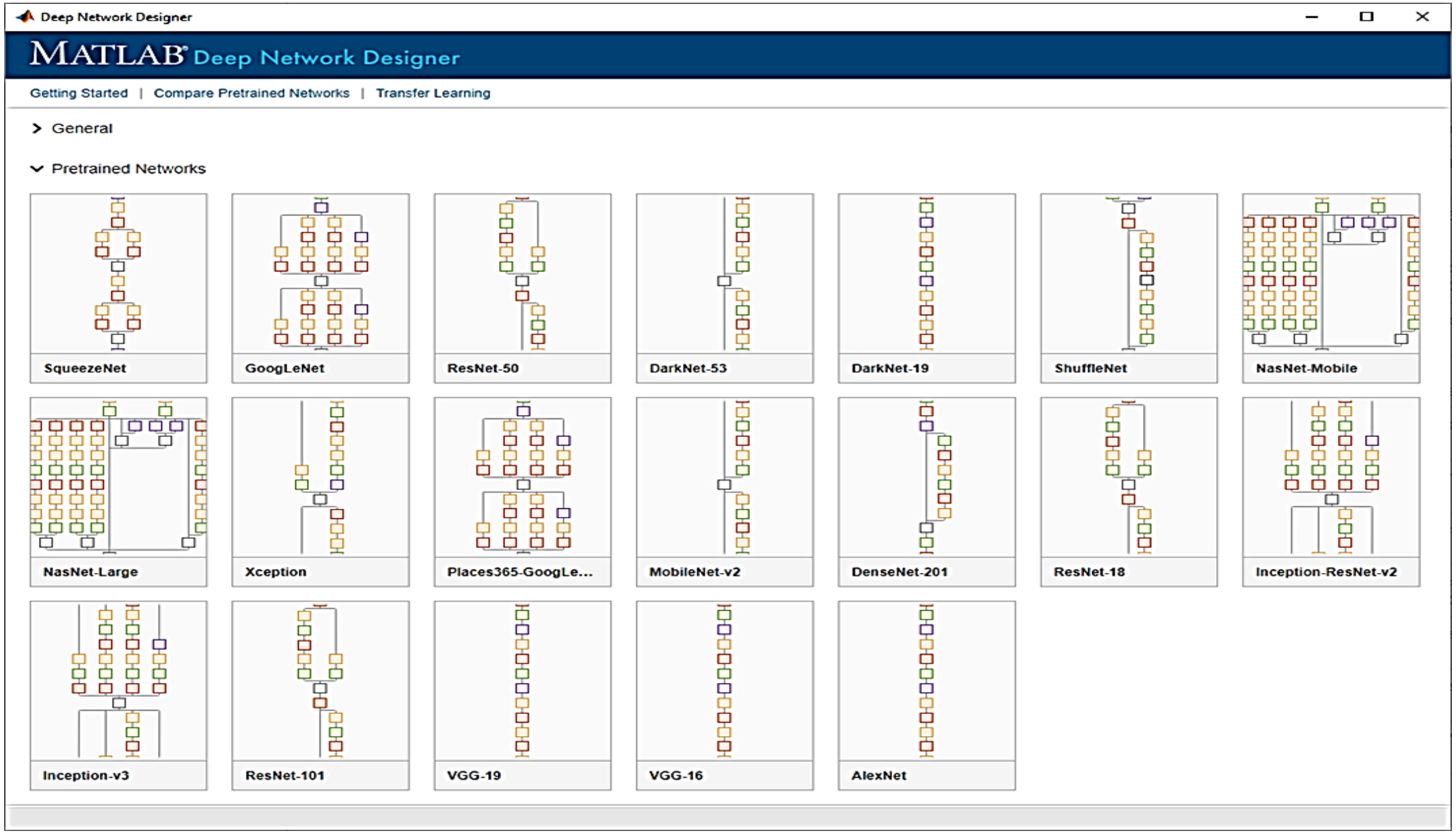
Catatan

Jika transfer belajar tidak sesuai dengan aplikasi kita, kita mungkin perlu melatih jaringan kita sendiri dari awal (**from scratch**)

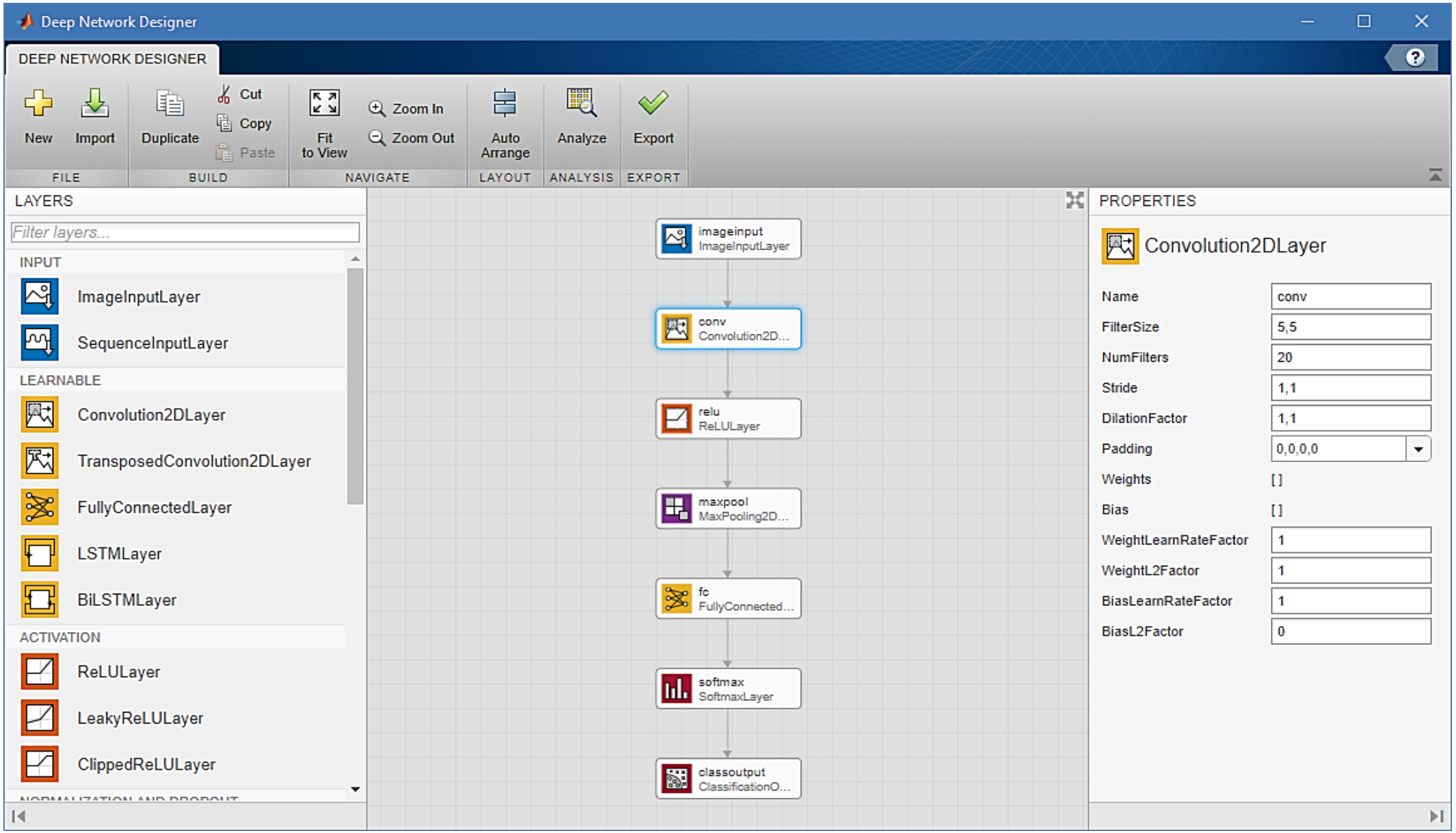
Metode ini menghasilkan hasil yang paling akurat, tetapi umumnya membutuhkan ratusan ribu gambar berlabel dan sumber daya komputasi yang cukup.

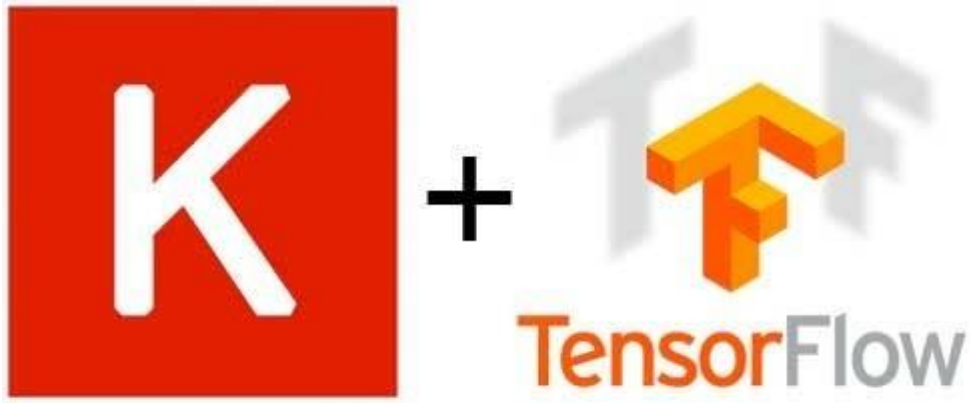


Pre-trained Jaringan yang ada di MATLAB



Desain Jaringan di MATLAB dengan Deep Network Designer

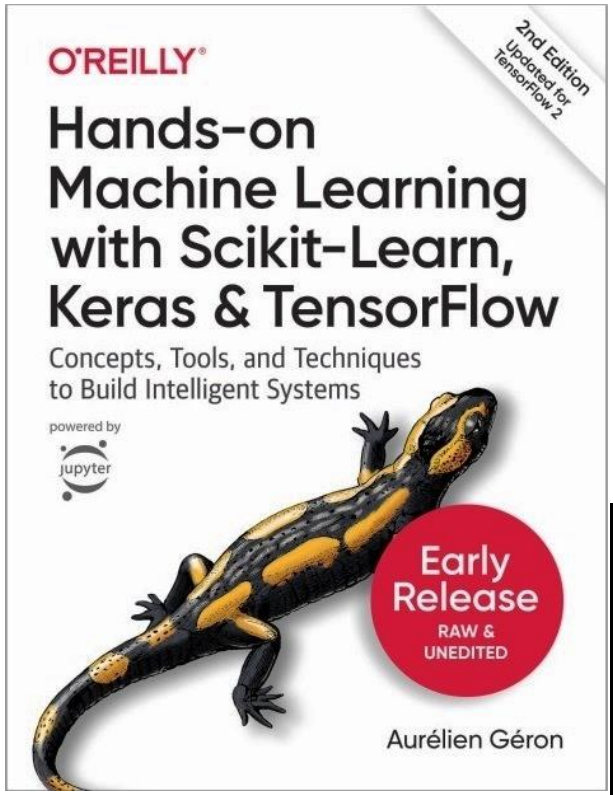


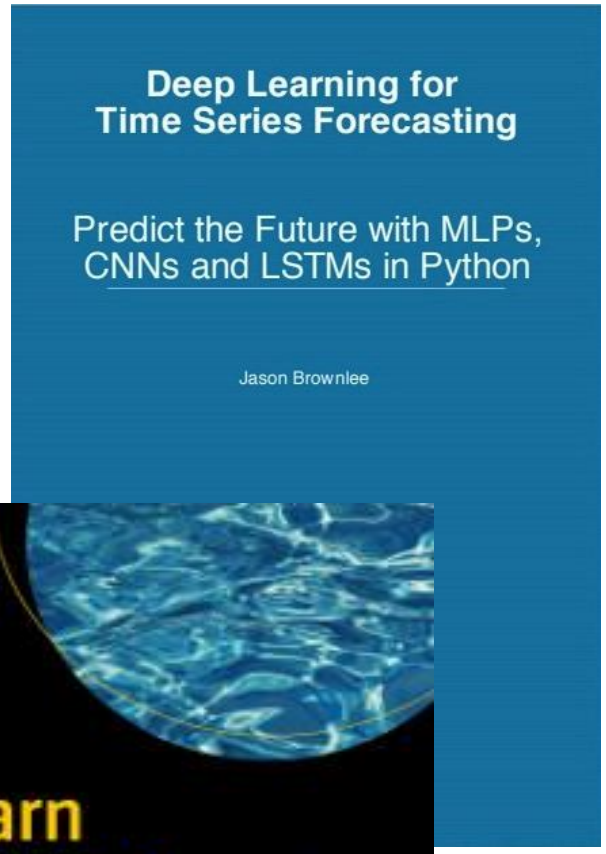


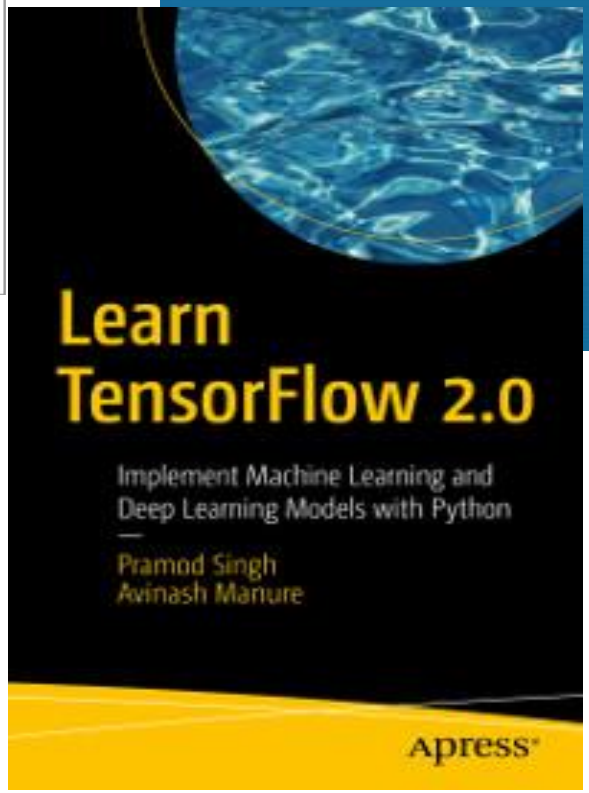
Deep Learning dengan Keras & TensorFlow

Link-Link Buku dan Code

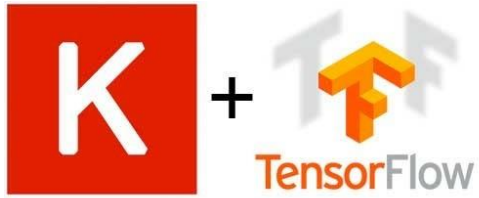
- Buku DL dan ML dgn **Scikit-learn**, **Keras**, **Tensorflow** :
- Buku Mastery ML dan DL dari ***Jason Brownlee***
- Buku Python
- Code + Buku "***Learn TensorFlow 2.0 Implement Machine Learning and Deep Learning Models with Python***"
- Code Keras TF webinar
- Keras Tutorial







Apa itu Keras dan TensorFlow?



Deep Learning with Keras

- **Keras** adalah Open-Source API **Deep Learning** yang ditulis dengan Python, berjalan di atas platform **Machine Learning TensorFlow**.
- Dikembangkan, oleh François Chollet, Google engineer, dengan fokus utk memungkinkan eksperimen cepat.
  - Mampu beralih dari ide ke hasil secepat mungkin adalah kunci untuk melakukan penelitian yang baik.
- **Keras** adalah API tingkat tinggi dari **TensorFlow 2.0**: antarmuka yang dapat didekati dan sangat produktif untuk memecahkan masalah **Machine Learning** , dengan fokus pada **Deep Learning** modern.
- **Keras** menyediakan abstraksi penting dan blok bangunan untuk mengembangkan dan mengirim solusi **Machine Learning** dengan kecepatan iterasi tinggi.

Apa itu Keras dan TensorFlow?



- TensorFlow **2.0** adalah ***end-to-end, open-source*** platform **Machine Learning** (deep learning library yang dikembang-kan oleh Google)
- Kita dapat menganggapnya sebagai lapisan infrastruktur untuk ***differentiable programming***. yaitu menggabungkan empat kemampuan utama:
  - Menjalankan operasi tensor tingkat rendah secara efisien pada CPU, GPU, atau TPU.
  - Menghitung gradien ekspresi yang dapat dibedakan secara bebas
  - Menskalakan perhitungan untuk banyak perangkat (mis. **Super-komputer Summit di Oak Ridge National Lab**, yang terdiri dari 27.000 GPU).
  - Mengekspor program ("graph") ke runtime eksternal seperti server, peramban, perangkat seluler dan tertanam.

Apa itu Backend?



- **Keras** tidak menangani perhitungan tingkat rendah. Sebagai gantinya, ia menggunakan pustaka lain untuk melakukannya, yang disebut **"Backend"**.
- Dengan demikian, **Keras** adalah **wrapper** API tingkat tinggi untuk API tingkat rendah, dan mampu berjalan di atas **TensorFlow**, **CNTK**, atau **Theano**.
- **Backend** adalah istilah dalam **Keras** yang melakukan semua perhitungan tingkat rendah seperti produk tensor, konvolusi, dan banyak hal lainnya dengan bantuan perpustakaan lain seperti **TensorFlow** atau **Theano**.
  - Jadi, "mesin backend" akan melakukan perhitungan dan pengembangan model.
  - Tensorflow adalah "mesin backend" default tetapi kita dapat mengubahnya dalam konfigurasi.

Fitur-Fitur: Keras vs TensorFlow

Fitur-Fitur Keras

- Fokus pada pengalaman pengguna.
- Multi-backend dan multi-platform.
- Produksi model yang mudah
- Memungkinkan pembuatan prototipe yang mudah dan cepat
- Dukungan **Convolutional networks**
- Dukungan **Recurrent networks**
- Ekspresif, fleksibel, dan tepat untuk penelitian inovatif.
- Framework berbasis Python yang membuatnya mudah untuk debug dan meng-**explore**.
- **Neural networks library** yang sangat modular ditulis dengan Python
- Dikembangkan dengan fokus aktif memungkinkan eksperimen cepat

Fitur-fitur Tensorflow

- Melakukan debug lebih cepat dengan tool Python
- Model dinamis dengan aliran kontrol Python
- Dukungan untuk gradien khusus dan tingkat tinggi
- Menawarkan berbagai tingkat abstraksi, yang membantu dalam membuat dan melatih model.
- Memungkinkan untuk melatih dan menggunakan model dengan cepat, apa pun bahasa atau platform yang digunakan.
- Menyediakan fleksibilitas dan kontrol dengan fitur seperti API dan Model Keras Functional
- Didokumentasikan dengan baik sehingga mudah dimengerti
- Paling populer dan mudah digunakan dengan Python

Perbedaan: Keras vs TensorFlow

Keras

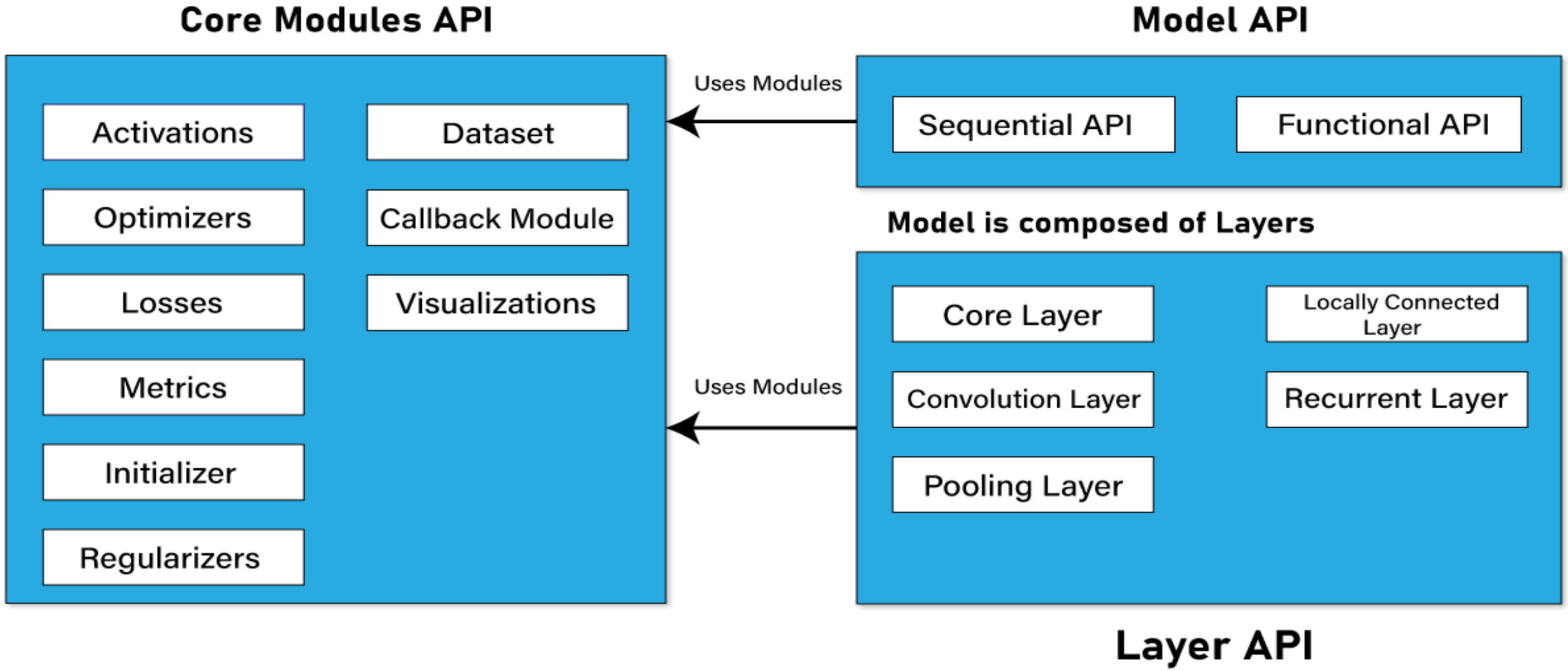
- API tingkat tinggi yang berjalan di atas TensorFlow, CNTK, dan Theano.
- Mudah digunakan jika tahu bahasa Python.
- Sempurna untuk implementasi cepat.
- Menggunakan tool debug API lain seperti TFDBG
- Arsitektur sederhana yang mudah dibaca dan ringkas.
- Biasanya digunakan untuk dataset kecil.
- Dukungan komunitas.
- Dapat digunakan untuk model berkinerja rendah.

Tensorflow

- Framework yang menawarkan API level tinggi dan level rendah.
- Perlu mempelajari sintaks menggunakan berbagai fungsi Tensorflow.
- Ideal untuk penelitian **DL**, jaringan kompx.
- Dapat menggunakan tool visualisasi Tensor board untuk debugging
- Tidak terlalu mudah digunakan.
- Digunakan untuk model kinerja tinggi dan dataset besar.
- Didukung oleh komunitas besar dari perusahaan teknologi.
- Dapat digunakan untuk model kinerja tinggi.

Arsitektur Keras

Keras API dibagi tiga kategori utama: **Model**, **Layer**, **Core Modules**



Model

Model Keras : 2 Jenis

Model Sekuensial **-**

Model Sekuensial pada dasarnya adalah komposisi linier Keras Layers. Model ini mudah, minimal serta memiliki kemampuan untuk mewakili hampir semua yang tersedia jaringan saraf.

Contoh sederhana:

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

model = Sequential()
model.add(Dense(512, activation='relu',
input_shape=(784,)))
```

- Baris 1 impor Sequential model dari Keras models
- Baris 2 impor Dense layer dan Activation module
- Baris 4 membuat sequential model baru dgn menggunakan Sequential API
- Baris 5 menambah dense layer (Dense API) dgn fungsi aktivasi relu (Activation module)

Model Sekuensial memaparkan kelas Model untuk membuat model yang disesuaikan juga. Kita bisa menggunakan konsep sub-kelas untuk membuat model kompleks kita sendiri.

API Fungsional: API Fungsional pada dasarnya digunakan untuk membuat model yang kompleks.

Layer dan Core Modules Keras

Layer

Setiap lapisan Keras dalam model Keras mewakili lapisan yang sesuai (lapisan input, disembunyikan layer dan output layer) dalam model jaringan saraf yang diusulkan sebenarnya.

Keras menyediakan banyak hal lapisan pre-built se-hingga jaringan saraf kompleks dapat dengan mudah dibuat. Beberapa Lapisan Keras yang penting ditentukan di bawah ini,

- **Core Layers**
- **Convolution Layers**
- **Pooling Layers**
- **Recurrent Layers**

Core Modules

Keras juga menyediakan banyak fungsi built-in ter-kait jaringan saraf untuk membuat Model keras dan lapisan Keras. Beberapa fungsi adalah sebagai beri-kut:

- **Activations module** - Fungsi aktivasi adalah konsep penting dalam JST dan modul aktivasi menyediakan banyak fungsi aktivasi seperti **softmax**, **relu**, dll
- **Loss module** – menyediakan fungsi-fungsi **loss** seperti ***mean_squared_error***, ***mean_absolute_error***, ***poisson***, dll,
- **Optimizer module** – menyediakan fungsi-fungsi optimizer seperti **adam**, **sgd**, dll,
- **Regularizers** – menyediakan fungsi-fungsi Regularizer seperti **L1 regularizer**, **L2 regularizer**, dll,

Cheat-Sheet for AI



Table of Content

Neural Networks

03 Neural Networks Basics

04 Neural Network Graphs

Machine Learning

06 Machine Learning Basics

07 Scikit Learn with Python

08 Scikit Learn Algorithm

09 Choosing ML Algorithm

Data Science with Python

11 Tensor Flow

12 Python Basics

13 PySpark Basics

14 Numpy Basics

15 Bokeh

16 Karas

17 Pandas

18 Data Wrangling with Pandas

19 Data Wrangling with dplyr & tidyr

20 SciPi

21 MatPlotLib

22 Data Visualization with ggplot

23 Big-O

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Download: Cheat-Sheet for AI

Cheat Sheet: Keras

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Inspect Model

```
>>> model.output_shape          Model output shape
>>> model.summary()             Model summary representation
>>> model.get_config()          Model configuration
>>> model.get_weights()         List all weight tensors in the model
```

Compile Model

MLP: Binary Classification

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

MLP: Multi-Class Classification

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

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

DataCamp
Learn Python for Data Science Interactively

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Cheat Sheet: TensorFlow



Tensor Flow Cheat Sheet

BecomingHuman.AI

TensorFlow

In May 2017 Google announced the second-generation of the TPU, as well as the availability of the TPUs in Google Compute Engine.[12] The second-generation TPUs deliver up to 180 teraflops of performance, and when organized into clusters of 64 TPUs provide up to 11.5 petaflops.

Info

TensorFlow

TensorFlow™ is an open source software library created by Google for numerical computation and large scale computation. Tensorflow bundles together Machine Learning. Deep learning models and frameworks and makes them useful by way of common metaphor.

Keras

Keras is an open sourced neural networks library, written in Python and is built for fast experimentation via deep neural networks and modular design. It is capable of running on top of TensorFlow, Theano, Microsoft Cognitive Toolkit, or PlaidML.

Skflow

Scikit Flow is a high level interface base on tensorflow which can be used like sklearn. You can build you own model on your own data quickly without rewriting extra code.provides a set of high level model classes that you can use to easily integrate with your existing Scikit-learn pipeline code.

Installation

How to install new package in Python

pip install <package-name>

Example: pip install requests

How to install tensorflow?

device = cpu/gpu

python_version = cp27/cp34

sudo pip install https://storage.googleapis.com/tensorflow/linux/$device/tensorflow-0.8.0-$python_version-none-linux_x86_64.whl
sudo pip install

How to install Skflow

pip install sklearn

How to install Keras

pip install keras

update ~/.keras/keras.json – replace "theano" by "tensorflow"

Helpers

Python helper Important functions

type(object)
Get object type

help(object)
Get help for object (list of available methods, attributes, signatures and so on)

dir(object)
Get list of object attributes (fields, functions)

str(object)
Transform an object to string object?
Shows documentations about the object

globals()
Return the dictionary containing the current scope's global variables.

locals()
Update and return a dictionary containing the current scope's local variables.

id(object)
Return the identity of an object. This is guaranteed to be unique among simultaneously existing objects.

import _builtin_
dir(_builtin_)
Other built-in functions

Tensor Flow

Main classes

tf.Graph()
tf.Operation()
tf.Tensor()
tf.Session()

Some useful functions

tf.get_default_session()
tf.get_default_graph()
tf.reset_default_graph()
ops.reset_default_graph()
tf.device("/cpu:0")
tf.name_scope(value)
tf.convert_to_tensor(value)

TensorFlow Optimizers

GradientDescentOptimizer
AdadeltaOptimizer
AdagradOptimizer
MomentumOptimizer
AdamOptimizer
FtrlOptimizer
RMSPropOptimizer

Reduction

reduce_sum
reduce_prod
reduce_min
reduce_max
reduce_mean
reduce_all
reduce_any
accumulate_n

Activation functions

tf.nn?
relu
relu6
elu
softplus
softsign
dropout
bias_add
sigmoid
tanh
sigmoid_cross_entropy_with_logits
softmax
log_softmax
softmax_cross_entropy_with_logits
sparse_softmax_cross_entropy_with_logits
weighted_cross_entropy_with_logits
etc.

Skflow

Main classes

TensorFlowClassifier
TensorFlowRegressor
TensorFlowDNNClassifier
TensorFlowDNNRegressor
TensorFlowLinearClassifier
TensorFlowLinearRegressor
TensorFlowRNNClassifier
TensorFlowRNNRegressor
TensorFlowEstimator

Each classifier and regressor have following fields n_classes=0 (Regressor), n_classes are expected to be input (Classifier)

batch_size=32,
steps=200, // except
TensorFlowRNNClassifier - there is 50
optimizer='Adagrad',
learning_rate=0.1,

Each class has a method fit

fit(X, y, monitor=None, logdir=None)

X: matrix or tensor of shape [n_samples, n_features...]. Can be iterator that returns arrays of features. The training input samples for fitting the model.

Y: vector or matrix [n_samples] or [n_samples, n_outputs]. Can be iterator that returns array of targets. The training target values (class labels in classification, real numbers in regression).

monitor: Monitor object to print training progress and invoke early stopping

logdir: the directory to save the log file that can be used for optional visualization.

predict (X, axis=1, batch_size=None)
Args:

X: array-like matrix, [n_samples, n_features...] or iterator.
axis: Which axis to argmax for classification.
By default axis 1 (next after batch) is used. Use 2 for sequence predictions.

batch_size: If test set is too big, use batch size to split it into mini batches. By default the batch_size member variable is used.
Returns:
y: array of shape [n_samples]. The predicted classes or predicted value.

Originally created by Altoros. **See Original here.**

Dasar-Dasar Keras untuk Deep Learning

Struktur utama dalam **Keras** adalah **Model yang mendefinisikan graph lengkap dari suatu jaringan**.

Dapat ditambahkan lebih banyak lapisan ke model yang ada untuk membangun model khusus yang dibutuhkan.

Berikut cara membuat **Model Sequential** dan beberapa layer yang biasa digunakan dalam **DL**

1. Sequential Model

```
from keras.models import Sequential
from keras.layers import Dense,Activation,
     Conv2D,MaxPooling2D,Flatten,Dropout

model = Sequential()
```

2. Convolutional Layer

Contoh lapisan konvolusional sebagai lapisan input dengan bentuk input **320x320x3**, dengan **48** filter ukuran **3x3** dan menggunakan **ReLU** sebagai fungsi aktivasi.

```
input_shape=(320,320,3) #this is the input
shape of an image 320x320x3
model.add(Conv2D(48,(3,3),activation='relu',
input_shape= input_shape))
```

tipe lainnya adalah

```
model.add(Conv2D(48,(3,3),activation='relu')
```

3. MaxPooling Layer

Untuk **downsample** representasi input, gunakan ***MaxPool2d*** dan tentukan ukuran kernel

```
model.add(MaxPooling2D(pool_size=(2, 2))
```

4. Dense Layer

menambahkan **Fully Connected Layer** dengan hanya menentukan Ukuran output

```
model.add(Dense(256, activation='relu'))
```

5. Dropout Layer

Menambahkan ***dropout layer*** dengan probabilitas 50%

```
model.add(Dropout(0.5))
```

Kompilasi, Pelatihan, dan Evaluasi

Setelah model di tetapkan, model dilatih. Namun model jaringan perlu **dikompilasi** terlebih dahulu dengan **fungsi loss** dan **fungsi pengoptimal**. Ini akan memungkinkan model jaringan untuk mengubah **bobot** dan meminimalkan **loss**.

```
model.compile(loss='mean_squared_error',
optimizer='adam')
```

Untuk **memulai pelatihan**, gunakan **fit** untuk memasukkan **data pelatihan** dan **validasi** ke model. Ini akan memungkinkan melatih jaringan secara batch dan mengatur **Epochs**.

```
model.fit(X_train, X_train, batch_size=32,
epochs=10, validation_data=(x_val, y_val))
```

Langkah terakhir adalah mengevaluasi model dengan data uji.

```
score = model.evaluate(x_test, y_test,
batch_size=32)
```

Dasar-Dasar Keras untuk Deep Learning

Percobaan: Regresi Linier Sederhana

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
import numpy as np
import matplotlib.pyplot as plt

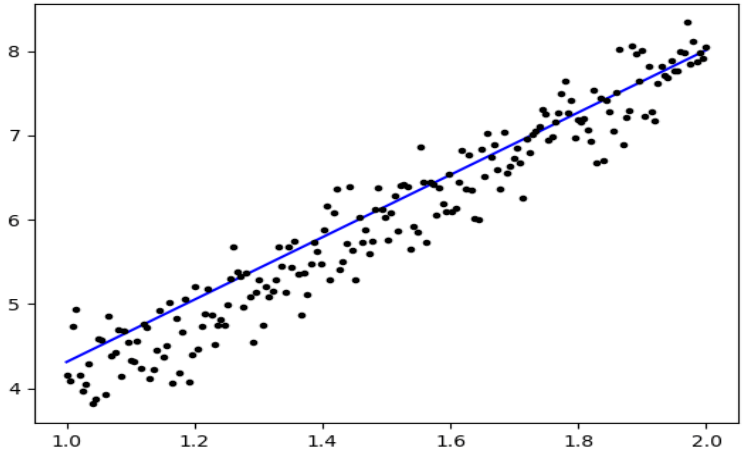
x = data = np.linspace(1,2,200)
y = x*4 + np.random.randn(*x.shape) * 0.3

model = Sequential()
model.add(Dense(1, input_dim=1,
activation='linear'))
model.compile(optimizer='sgd', loss='mse',
metrics=['mse'])

weights = model.layers[0].get_weights() w_init
= weights[0][0][0]
b_init = weights[1][0]
print('Linear regression model is initialized
with weights w: %.2f, b: %.2f' % (w_init,
b_init))
```

```
model.fit(x,y, batch_size=1, epochs=30,
shuffle=False)
weights = model.layers[0].get_weights() w_final
= weights[0][0][0]
b_final = weights[1][0]
print('Linear regression model is trained to
have weight w: %.2f, b: %.2f' % (w_final,
b_final)) predict = model.predict(data)
plt.plot(data, predict, 'b', data , y, 'k.')
plt.show()
```

Stlh pelatihan data, output:

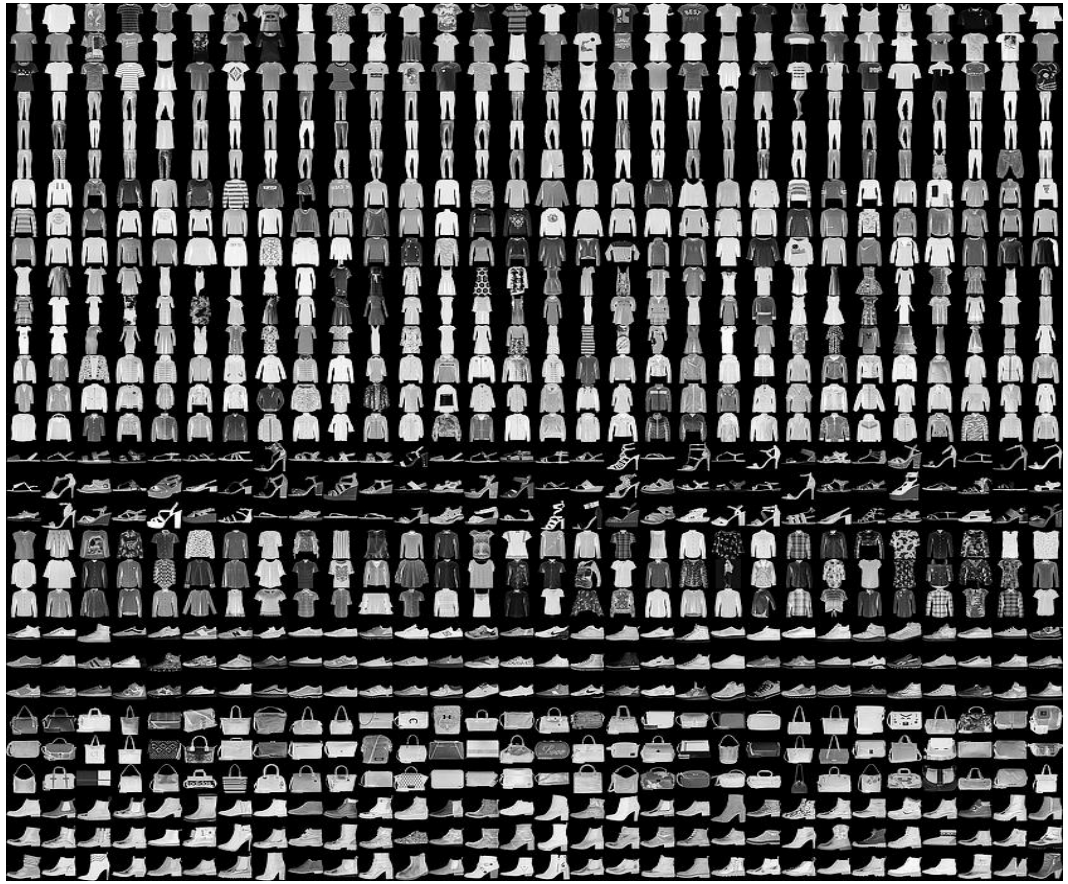


dengan bobot awal: **w: 0.37, b: 0.00**

Dan bobot akhir: **w: 3.70, b: 0.61**

Contoh-Contoh Implementasi Deep Learning dgn Keras

Dataset Fashion-MNIST, Zalando (The MIT License [MIT] Copyright © [2017] Zalando SE, https://tech.zalando.com/), memuat 70,000 gambar (grayscale) dlm 10 kategori yang berbeda. Ukuran 28 × 28 pixels setiap artikel pakaian, dengan nilai dari 0 sd 255



Dari total 70.000 gambar, **60.000** digunakan untuk **pelatihan** dan **10.000 sisanya** untuk pengujian. Label adalah bilangan bulat array mulai dari 0 hingga 9. Nama kelas bukan bagian dari dataset dan karenanya kita perlu menyertakan pemetaan di bawah ini saat pelatihan / prediksi.

Label -> Description: 0 -> T-shirt/top, 1 -> Trouser, 2 -> Pullover, 3 -> Dress, 4 -> Coat, 5 -> Sandal, 6 -> Shirt, 7 -> Sneaker, 8 -> Bag, 9 -> Ankle boot

- **Deep Neural Network** (three hidden layers) Fashion MNIST:
  - *Training Accuracy* **0.93**, *Test Accuracy* **0.88**
- **CNN** Fashion MNIST:
  - *Training Accuracy* **0.97**, *Test Accuracy* 0.92

▪Code Keras TF webinar

Fine-Tune Pre-Trained Models di Keras dan Bgmn menggunakannya

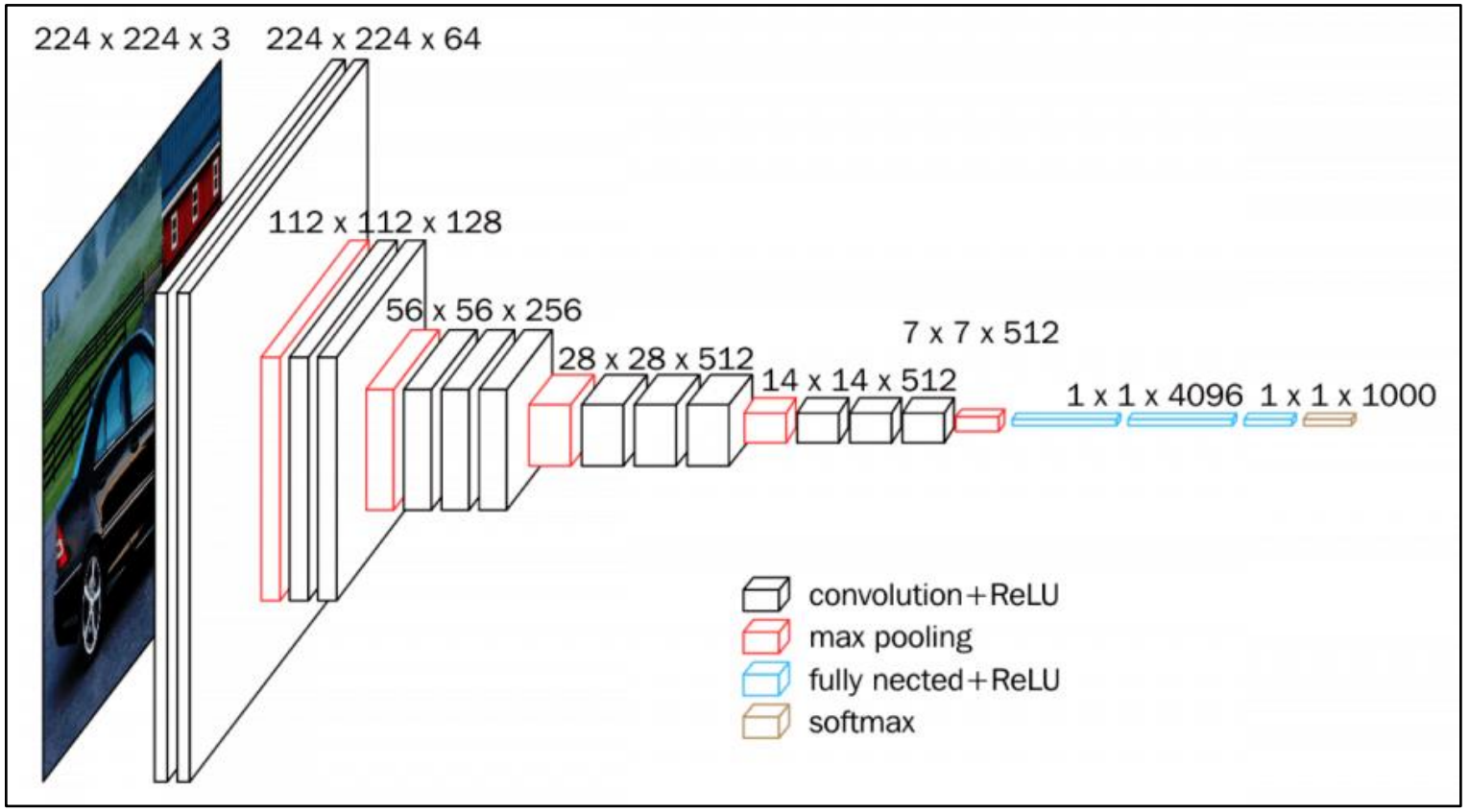
- **Fine-tuning** adalah tugas **untuk men-tweak** model pra-terlatih (pre-trained) sehingga parameter akan beradaptasi dengan model baru.
  - Jika melatih dari awal pada model baru, dibutuhkan sejumlah besar data, sehingga jaringan dapat menemukan semua parameter.
  - Untuk mempermudah, dalam kasus ini, digunakan **model pra-terlatih** sehingga **parameter sudah dipelajari** dan **memiliki bobot**.
- Misalnya, Ingin melatih model untuk menyelesaikan masalah klasifikasi tetapi tersedia hanya sedikit data, maka dapat diselesaikan dengan metode **Transfer Pembelajaran** + **Fine-Tuning.**
- Dengan menggunakan jaringan pra-terlatih & bobot, **tidak perlu** melatih seluruh jaringan. **Yang diperlukan hanya melatih lapisan terakhir** yang digunakan untuk menyelesaikan tugas (ini disebut **metode Fine-Tuning**).

Fine-Tune Pre-Trained Models di Keras dan Bgmn menggunakannya

Persiapan Model Jaringan

- Untuk model pra-terlatih, dapat dimuat berbagai model yang sudah dimiliki Keras di library-nya seperti: ***VGG16, InceptionV3, ResNet, MobileNet, Xception, InceptionResNetV2***
- Sebagai contoh, digunakan model jaringan **VGG16** dan **imageNet** sebagai bobot untuk model tsb. Jaringan akan disempurnakan untuk mengklasifikasikan **8 jenis kelas** yang berbeda menggunakan Images dari **Kaggle Natural Images Dataset** (https://www.kaggle.com/prasunroy/natural-images)

Model Arsitektur VGG16





APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Fine-Tune Pre-Trained Models di Keras dan Bgmn menggunakannya

1. Persiapan Data

Membangkitkan data training data menggunakan ***Image-DataGenerator*** Keras. Pertama **download** Images dari Kaggle, kemudian memulai pelatihan

```
from keras.preprocessing.image import
ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt

train_path = 'images/train/'
test_path = 'images/test/'
batch_size = 16
image_size = 224
num_class = 8

train_datagen = ImageDataGenerator(
validation_split= 0.3, shear_range=0.2,
zoom_range=0.2, horizontal_flip= True)
```

```
train_generator = train_datagen.flow_from_
directory( directory= train_path, target_size=
(image_size,image_size), batch_size=batch_size,
class_mode= 'categorical', color_mode='rgb',
shuffle=True)
```

ImageDataGenerator akan membuat data ***X_training*** dari direktori.

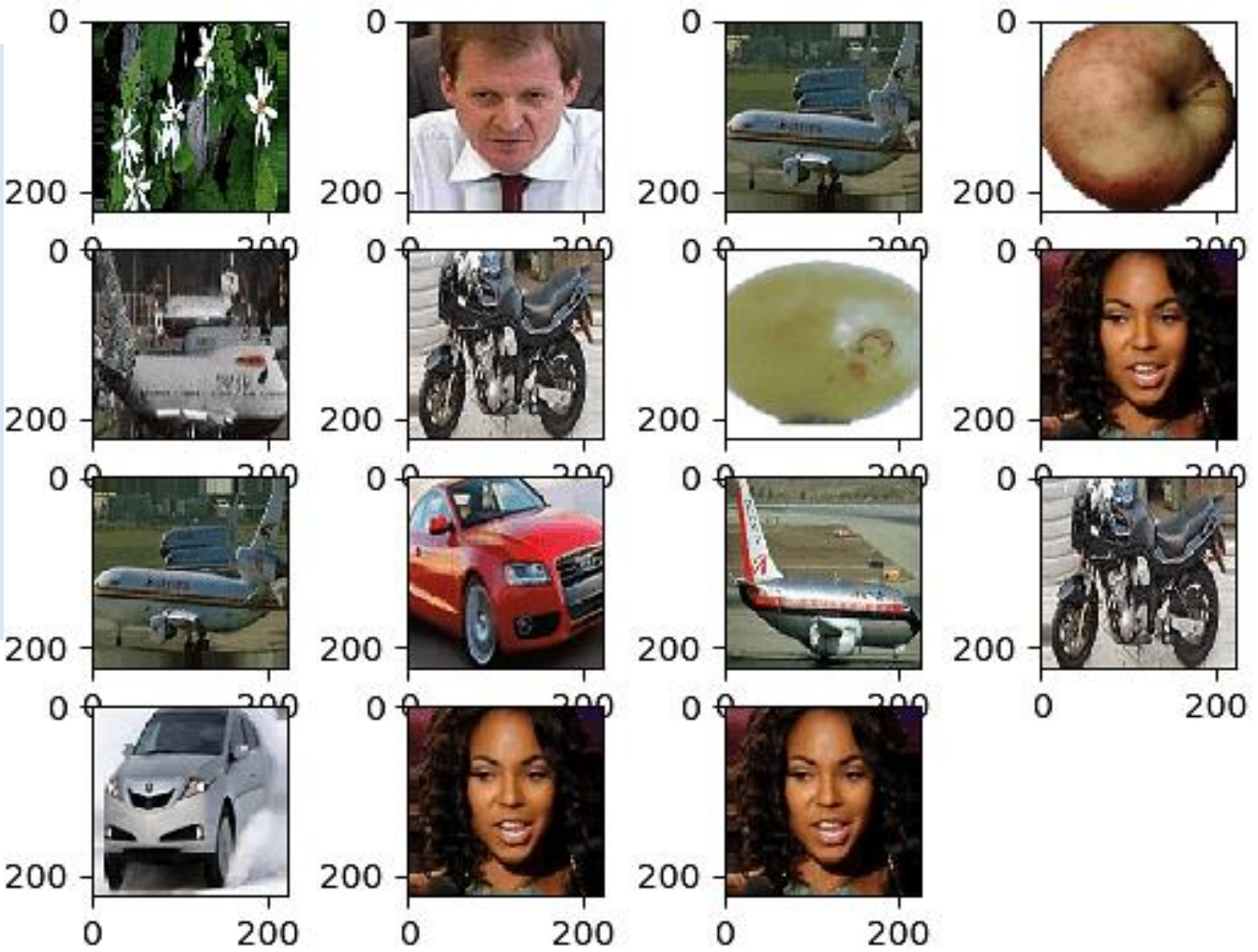
Sub-direktori dalam direktori tersebut akan digunakan sebagai **kelas** untuk setiap objek.

Gambar akan dimuat dengan mode warna RGB, dengan mode kelas kategorikal untuk data ***Y_training***, dengan ukuran batch 16. Akhirnya, data di shuffle.

Fine-Tune Pre-Trained Models di Keras dan Bgmn menggunakannya

Menampilkan gambar secara acak dengan memplot-nya dengan ***matplotlib***

```
x_batch, y_batch = train_generator.next()
fig=plt.figure()
columns = 4
rows = 4
for i in range(1, columns*rows):
      num = np.random.randint(batch_size)
      image = x_batch[num].astype(np.int)
      fig.add_subplot(rows, columns, i)
      plt.imshow(image)
plt.show()
```



2. Membuat Model dari VGG16

```
import keras
from keras.models import Model,load_model
from keras.layers import Activation,
     Dropout, Flatten, Dense
from keras.preprocessing.image import
     ImageDataGenerator
from keras.applications.vgg16 import VGG16
#Load the VGG model
base_model = VGG16(weights='imagenet',
 include_top=False, input_shape=(image_size,
 image_size, 3))
print(base_model.summary())
# Freeze the layers
for layer in base_model.layers:
     layer.trainable = False
```

```
# # Create the model
model = keras.models.Sequential()
# # Add the vgg convolutional base model
model.add(base_model)
# # Add new layers
model.add(Flatten())
model.add(Dense(1024,activation='relu'))
model.add(Dense(1024,activation='relu'))
model.add(Dense(num_class, activation=
 'softmax'))
# # Show a summary of the model. Check the
number of trainable parameters
print(model.summary())
```

Fine-Tune Pre-Trained Models di Keras dan Bgmn menggunakannya

Gambar di bawah, ringkasan model jaringannya. Dari input dari output **VGG16** Layers, kemudian ditambah-kan **2 Fully Connected** Layer yang akan mengekstrak **1024** fitur dan layer output yang akan menghitung **8** kelas dengan aktivasi **softmax**.

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 7, 7, 512)	14714688
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 1024)	25691136
dense_2 (Dense)	(None, 1024)	1049600
dense_3 (Dense)	(None, 8)	8200

Total params: 41,463,624
Trainable params: 26,748,936
Non-trainable params: 14,714,688

3. Pelatihan

```
# # Compile the model
from keras.optimizers import SGD

model.compile(loss='categorical_crossentropy',
     optimizer=SGD(lr=1e-3),metrics=['accuracy'])
# # Start the training process
# model.fit(x_train, y_train, validation_split=0.30,
          batch_size=32, epochs=50, verbose=2)

# # #save the model # model.save('catdog.h5')

history = model.fit_generator( train_generator,
    steps_per_epoch=train_generator.n/batch_size,
    epochs=10)

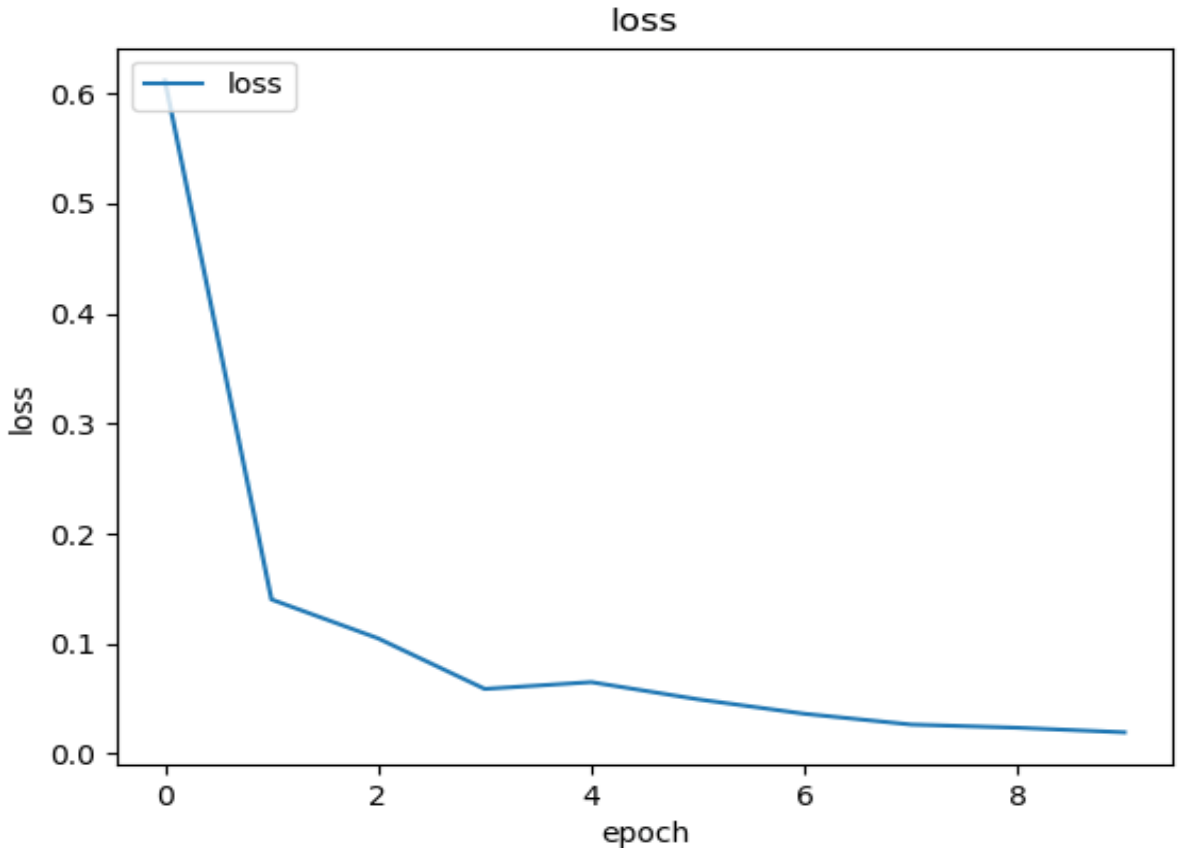
model.save('fine_tune.h5')
# summarize history for accuracy
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.title('loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['loss'], loc='upper left')
plt.show()
```

Fine-Tune Pre-Trained Models di Keras dan Bgmn menggunakannya

Hasilnya:

```
Epoch 1/10
432/431 [==============================] - 53s 123ms/step - loss: 0.5524 - acc:
0.9474
Epoch 2/10
432/431 [==============================] - 52s 119ms/step - loss: 0.1571 - acc:
0.9831
Epoch 3/10
432/431 [==============================] - 51s 119ms/step - loss: 0.1087 - acc:
0.9871
Epoch 4/10
432/431 [==============================] - 51s 119ms/step - loss: 0.0624 - acc:
0.9926
Epoch 5/10
432/431 [==============================] - 51s 119ms/step - loss: 0.0591 - acc:
0.9938
Epoch 6/10
432/431 [==============================] - 51s 119ms/step - loss: 0.0498 - acc:
0.9936
Epoch 7/10
432/431 [==============================] - 51s 119ms/step - loss: 0.0403 - acc:
0.9958
Epoch 8/10
432/431 [==============================] - 51s 119ms/step - loss: 0.0248 - acc:
0.9959
Epoch 9/10
432/431 [==============================] - 51s 119ms/step - loss: 0.0466 - acc:
0.9942
Epoch 10/10
432/431 [==============================] - 52s 120ms/step - loss: 0.0338 - acc:
0.9947
```

Dari gambar, **loss** turun secara signifikan dan akurasinya hampir 100%. Untuk menguji model, diambil gambar **secara acak** dari internet dan meletakkannya di **folder pengujian** dengan kelas yang berbeda untuk menguji.



Fine-Tune Pre-Trained Models di Keras dan Bgmn menggunakannya

4. Pengujian Model

```
model = load_model('fine_tune.h5')
test_datagen = ImageDataGenerator()

train_generator = train_datagen.flow_from_
   directory( directory=train_path,target_
   size=(image_size,image_size),batch_size=
   batch_size, class_mode='categorical',
   color_mode='rgb', shuffle=True)

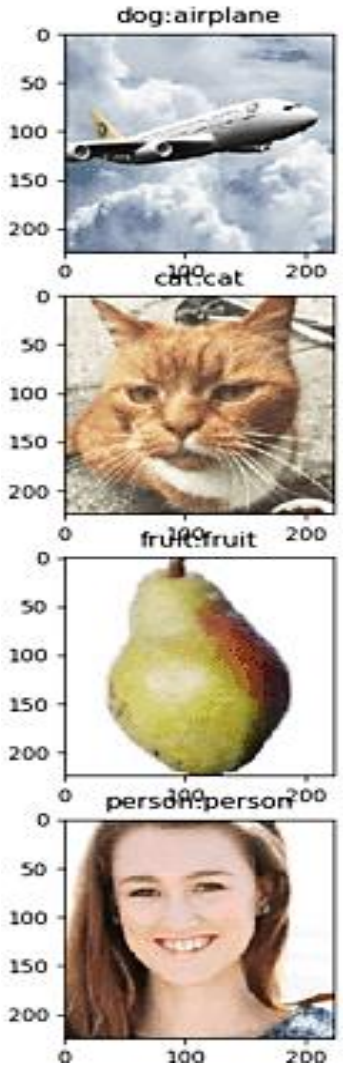
test_generator =test_datagen.flow_from_
   directory( directory=test_path, target_
   size=(image_size, image_size), color_
   mode='rgb', shuffle=False, class_mode=
   'categorical', batch_size=1)

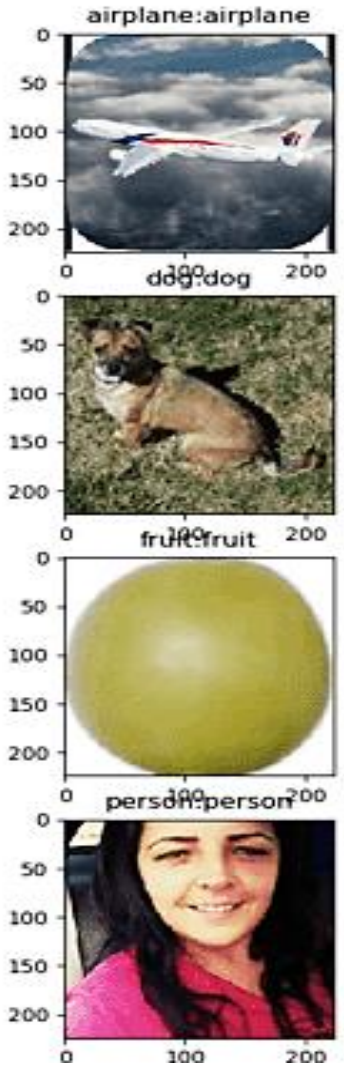
filenames = test_generator.filenames
nb_samples = len(filenames)
```

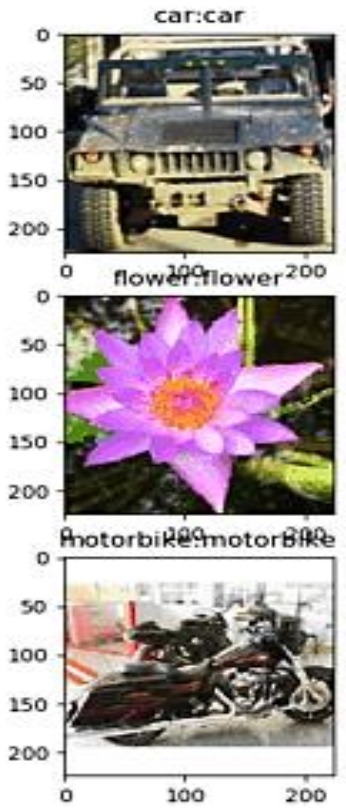
```
fig=plt.figure()
columns = 4
rows = 4
for i in range(1, columns*rows -1):
   x_batch, y_batch=test_generator.next()
   name = model.predict(x_batch)
   name = np.argmax(name, axis=-1)
   true_name = y_batch
   true_name = np.argmax(true_name, axis=-1)
   label_map = (test_generator.class_indices)
   label_map = dict((v,k) for k,v in
   label_map.items()) #flip k,v
   predictions = [label_map[k] for k in name]
   true_value = [label_map[k] for k in
       true_name]
   image = x_batch[0].astype(np.int)
   fig.add_subplot(rows, columns, i)
   plt.title(str(predictions[0]) + ':' +
       str(true_value[0]))
   plt.imshow(image)
plt.show()
```

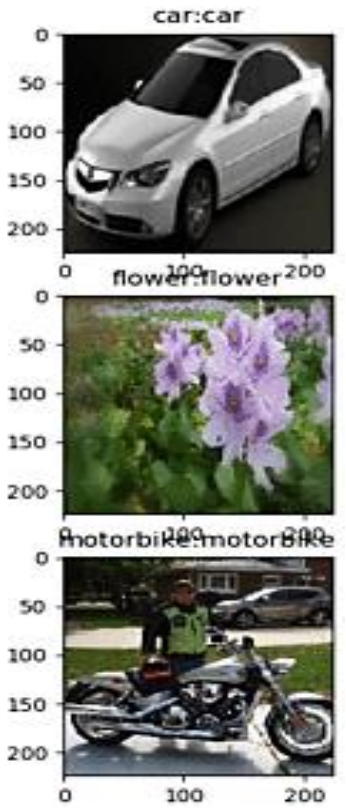
Fine-Tune Pre-Trained Models di Keras dan Bgmn menggunakannya

Hasil pengujian: Hanya 1 gambar yang diprediksi **salah** dari pengujian **14** gambar!







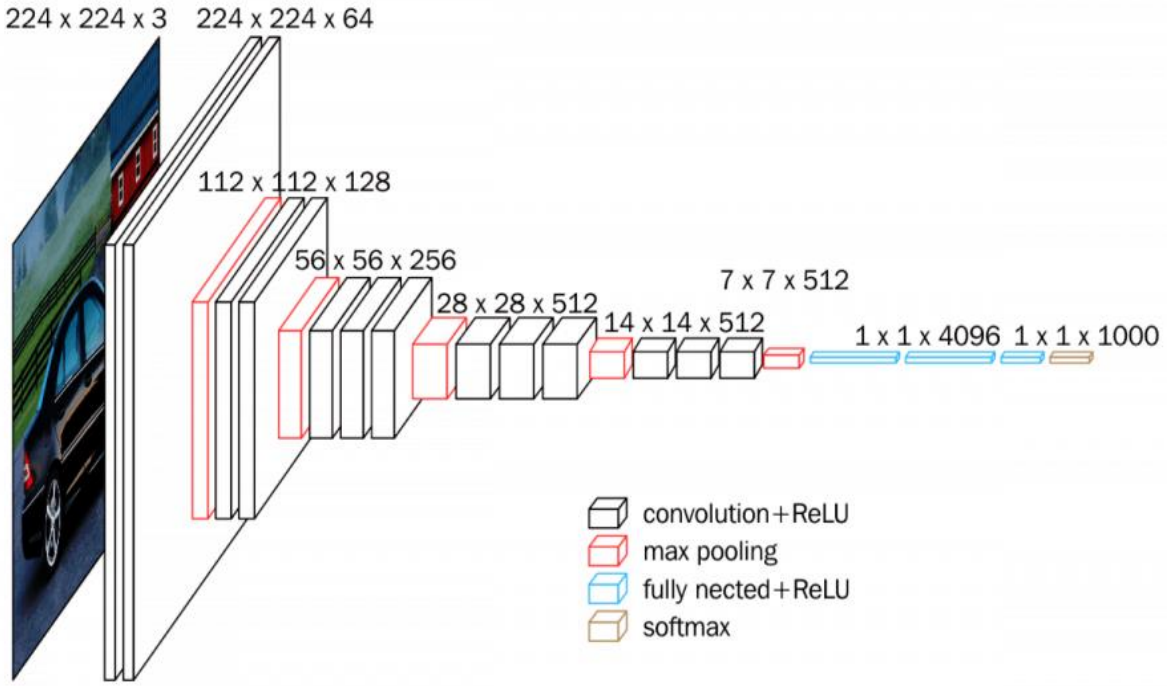


APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Face Recognition Neural Network dengan Keras

Mengapa kita membutuhkan Pengenalan ?

Kita perlu Pengenalan untuk membuat lebih mudah bagi kita untuk mengenali atau mengidentifikasi wajah seseorang, jenis objek, perkiraan usia seseorang dari wajahnya, atau bahkan tahu ekspresi wajah orang itu.



Model Jaringan

digunakan Model Jaringan **VGG16** tetapi dengan **bobot VGGFace.**

VGGFace adalah implementasi Keras dari **Deep Face Recognition** yang diperkenalkan oleh ***Parkhi, Omkar M. et al.*** "**Deep Face Recognition**." BMVC (2015). Framework menggunakan VGG16 sebagai arsitektur jaringan.

VGGFace dapat di download di **github** (https://github.com/rcmalli/keras-vggface)

```
from keras.applications.vgg16 import VGG16
from keras_vggface.vggface import VGGFace
face_model = VGGFace(model='vgg16', weights=
          'vggface', input_shape=(224,224,3))
face_model.summary()
```

Face Recognition Neural Network dengan Keras

Ringkasan Model Jaringan

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
conv1_1 (Conv2D)	(None, 224, 224, 64)	1792
conv1_2 (Conv2D)	(None, 224, 224, 64)	36928
pool1 (MaxPooling2D)	(None, 112, 112, 64)	0
conv2_1 (Conv2D)	(None, 112, 112, 128)	73856
conv2_2 (Conv2D)	(None, 112, 112, 128)	147584
pool2 (MaxPooling2D)	(None, 56, 56, 128)	0
conv3_1 (Conv2D)	(None, 56, 56, 256)	295168
conv3_2 (Conv2D)	(None, 56, 56, 256)	590080
conv3_3 (Conv2D)	(None, 56, 56, 256)	590080
pool3 (MaxPooling2D)	(None, 28, 28, 256)	0
conv4_1 (Conv2D)	(None, 28, 28, 512)	1180160
conv4_2 (Conv2D)	(None, 28, 28, 512)	2359808
conv4_3 (Conv2D)	(None, 28, 28, 512)	2359808
pool4 (MaxPooling2D)	(None, 14, 14, 512)	0
conv5_1 (Conv2D)	(None, 14, 14, 512)	2359808
conv5_2 (Conv2D)	(None, 14, 14, 512)	2359808
conv5_3 (Conv2D)	(None, 14, 14, 512)	2359808
pool5 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc6 (Dense)	(None, 4096)	102764544
fc6/relu (Activation)	(None, 4096)	0
fc7 (Dense)	(None, 4096)	16781312
fc7/relu (Activation)	(None, 4096)	0
fc8 (Dense)	(None, 2622)	10742334
fc8/softmax (Activation)	(None, 2622)	0

Total params: 145,002,878
Trainable params: 145,002,878
Non-trainable params: 0

Traceback (most recent call last):

Face Recognition Neural Network dengan Keras

Akan dilakukan **Transfer Learning + Fine Tuning** untuk membuat pelatihan lebih cepat dengan data-set kecil. Pertama, kita akan membekukan lapisan dasar sehingga lapisan tidak bisa dilatih.

```
for layer in face_model.layers: layer.trainable = False
```

lalu ditambahkan layer untuk mengenali wajah pengujian. Akan ditambahkan 2 lapisan yang terhubung sepenuhnya dan lapisan keluaran dengan 5 orang untuk dideteksi.

```
from keras.models import Model, Sequential
from keras.layers import Input,Convolution-
  2D, ZeroPadding2D, MaxPooling2D, Flatten,
  Dense, Dropout, Activation

person_count = 5
last_layer = face_model.get_layer('pool5').
             output
x = Flatten(name='flatten')(last_layer)
x = Dense(1024, activation='relu',name=
    'fc6')(x)
```

```
x = Dense(1024, activation='relu', name=
    'fc7')(x)
out = Dense(person_count, activation=
    'softmax', name='fc8')(x)
custom_face = Model(face_model.input, out)
```

Face Recognition Neural Network dengan Keras

Ringkasan Model Jaringan Fine-Tuned

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
conv1_1 (Conv2D)	(None, 224, 224, 64)	1792
conv1_2 (Conv2D)	(None, 224, 224, 64)	36928
pool1 (MaxPooling2D)	(None, 112, 112, 64)	0
conv2_1 (Conv2D)	(None, 112, 112, 128)	73856
conv2_2 (Conv2D)	(None, 112, 112, 128)	147584
pool2 (MaxPooling2D)	(None, 56, 56, 128)	0
conv3_1 (Conv2D)	(None, 56, 56, 256)	295168
conv3_2 (Conv2D)	(None, 56, 56, 256)	590080
conv3_3 (Conv2D)	(None, 56, 56, 256)	590080
pool3 (MaxPooling2D)	(None, 28, 28, 256)	0
conv4_1 (Conv2D)	(None, 28, 28, 512)	1180160
conv4_2 (Conv2D)	(None, 28, 28, 512)	2359808
conv4_3 (Conv2D)	(None, 28, 28, 512)	2359808
pool4 (MaxPooling2D)	(None, 14, 14, 512)	0
conv5_1 (Conv2D)	(None, 14, 14, 512)	2359808
conv5_2 (Conv2D)	(None, 14, 14, 512)	2359808
conv5_3 (Conv2D)	(None, 14, 14, 512)	2359808
pool5 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc6 (Dense)	(None, 1024)	25691136
fc7 (Dense)	(None, 1024)	1049600
fc8 (Dense)	(None, 5)	5125

Total params: 41,460,549
Trainable params: 26,745,861
Non-trainable params: 14,714,688

Tampak dari gbr, setelah **layer pool5**, akan diratakan menjadi **vektor fitur tunggal** yang akan digunakan oleh **dense layer** untuk pengenalan akhir.

Face Recognition Neural Network dengan Keras

1. Mempersiapkan File Wajah

Membuat direktori yang terdiri dari, misal 5 orang terkenal: ***Ben Afflek , Elton John , Jerry Seinfeld Madonna , Mandy Kaling***.

Setiap folder berisi 10 gambar, untuk setiap pelatihan dan proses evaluasi.

Menggunakan tool **Keras** untuk menyiapkan data. Fungsi ini akan diulang dalam folder dataset dan kemudian disiapkan sehingga dapat digunakan dalam pelatihan.

```
from keras.preprocessing.image import
     ImageDataGenerator
batch_size = 5
train_path = 'data/'
eval_path = 'eval/'
image_size = 224
```

```
train_datagen = ImageDataGenerator(rescale=
     1./255, shear_range=0.2,zoom_range=0.2,
     horizontal_flip=True)

valid_datagen = ImageDataGenerator(rescale=
    1./255, shear_range=0.2, zoom_range=0.2,
    horizontal_flip=True)

train_generator = train_datagen.flow_from_
    directory( train_path,  target_size=
    (image_size,image_size), batch_size=
    batch_size, class_mode= 'sparse',
    color_mode='rgb')

valid_generator = valid_datagen.flow_from_
    directory(directory=eval_path, target_
    size=(224, 224),color_mode='rgb', batch_
    size=batch_size, class_mode='sparse',
    shuffle=True,)
```

Face Recognition Neural Network dengan Keras

2. Melatih Model

Memulai proses pelatihan dengan **mengkompilasi** jaringan dengan fungsi **loss** dan **pengoptimal**. Di sini, digunakan **sparse_categorical_crossentropy** sebagai fungsi **loss** , dengan bantuan **SGD** sebagai pengoptimal pembelajaran .

```
from keras.optimizers import SGD
custom_face.compile(loss='sparse_categorical
      _crossentropy', optimizer=SGD(lr=1e-4,
      momentum=0.9), metrics=['accuracy'])

history = custom_face.fit_generator( train_
     generator, validation_data=valid_
     generator,steps_per_epoch=49/batch_
     size, validation_steps=valid_
     generator.n, epochs=50)

custom_face.evaluate_generator(generator=val
     id_generator)

custom_face.save('vgg_face.h5')
```

3. Pengujian Model dgn gbr uji

Menggunakan gambar ***Ben Afflek.*** sebagai gambar pengujian, dan menunjukkan bahwa wajah yang diprediksi benar

```
import numpy as np
from keras.models import load_model
from keras.preprocessing.image import load_img, save_img,
     img_to_array
from keras_vggface.utils import preprocess_input

model = load_model('vgg_face.h5')

test_img = image.load_img('test.jpg', target_size=(224,
      224))
img_test = image.img_to_array(test_img)
img_test = np.expand_dims(img_test, axis=0)
img_test = utils.preprocess_input(img_test)
predictions = model.predict(img_test)
predicted_class=np.argmax(predictions,axis=1)

labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class]
print(predictions) ['Ben_Afflek']
```

Deploying Model sbg Layanan REST Service

Menggunakan **Flask** untuk men-deploy model sebagai layanan **REST** **(Representational State Transfer)**

Asumsi Model (Model Regresi Linier) sdh dibangun

Flask adalah framework web yang ringan, dibangun dengan Python, untuk men-deploy aplikasi pada server

Contoh main **app.py**

```
! pip install Flask

import pandas as pd
import numpy as np
import sklearn
import joblib
from flask import Flask,render_template,
     request
app=Flask(__name__)
```

```
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict',methods=['GET','POST']
)
def predict():
    if request.method =='POST':
        print(request.form.get('var_1'))
        print(request.form.get('var_2'))
        print(request.form.get('var_3'))
        print(request.form.get('var_4'))
        print(request.form.get('var_5'))
        try:

         var_1=float(request.form['var_1'])
         var_2=float(request.form['var_2'])
         var_3=float(request.form['var_3'])
         var_4=float(request.form['var_4'])
         var_5=float(request.form['var_5'])
```

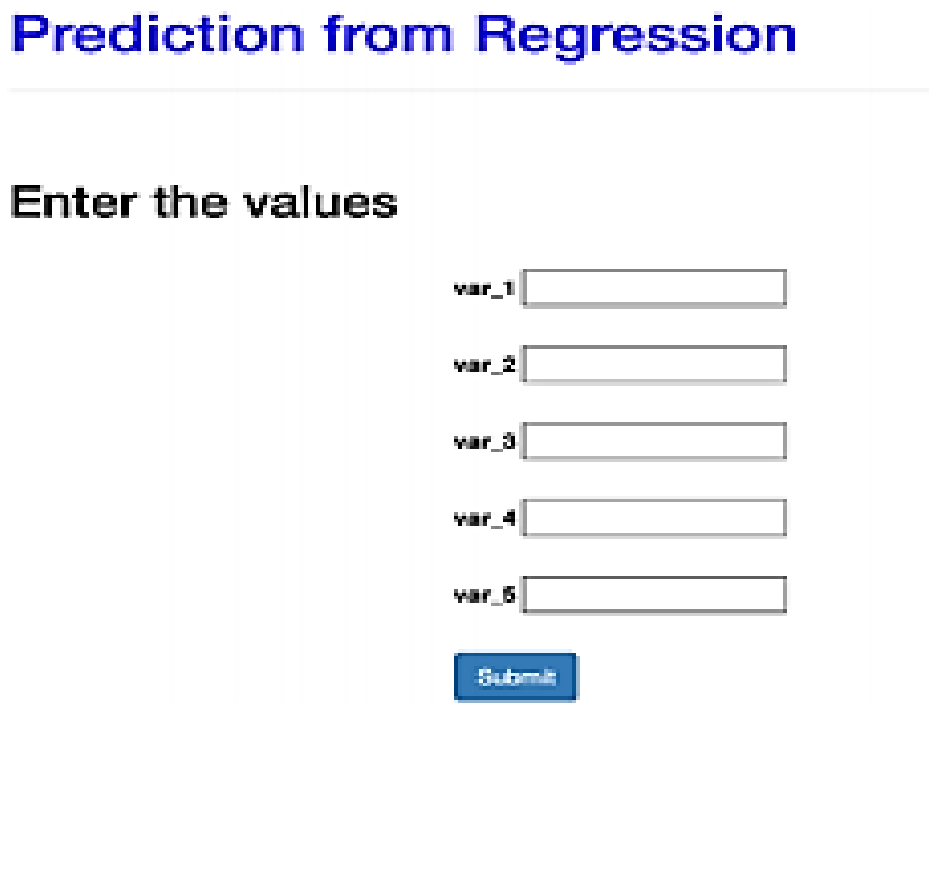
Deploying Model sbg Layanan REST Service

```
pred_args=[var_1,var_2,var_3,var_4,var_5]
pred_arr=np.array(pred_args)
print(pred_arr)
preds=pred_arr.reshape(1,-1)

model=open("linear_regression_model.pkl",
           "rb")
lr_model=joblib.load(model)
model_prediction=lr_model.predict(preds)
model_prediction=round(float(model_predictio
n),2)
except ValueError:
return "Please Enter valid values"
return
render_template('predict.html',prediction=mo
del_prediction)

if __name__=='__main__':
    app.run(host='localhost')
```

Tampilan Deploy Model



Model Regresi Linier Yg di-Deploy

Model Regresi Linier

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import joblib

df=pd.read_csv('Linear_regression_dataset.csv',header='infer')
df.sample(10)

#create X ( input variables with all columns except for outout/target column )
X=df.loc[:,df.columns !='output']

#create the y data with only output column
y=df['output']

#Train the Linear Regression model on training dataset
```

```
lr = LinearRegression().fit(X, y)
lr.score(X,y)

#save model using joblib
joblib.dump(lr,'linear_regression_model.pkl')

#test model with data
test_data=[600,588,90,0.358,0.333]
pred_arr=np.array(test_data)
print(pred_arr)
preds=pred_arr.reshape(1,-1)
print(preds)
model=open("linear_regression_model.pkl",
           "rb")
lr_model=joblib.load(model)
model_prediction=lr_model.predict(preds)
print(model_prediction)
```

Terima Kasih