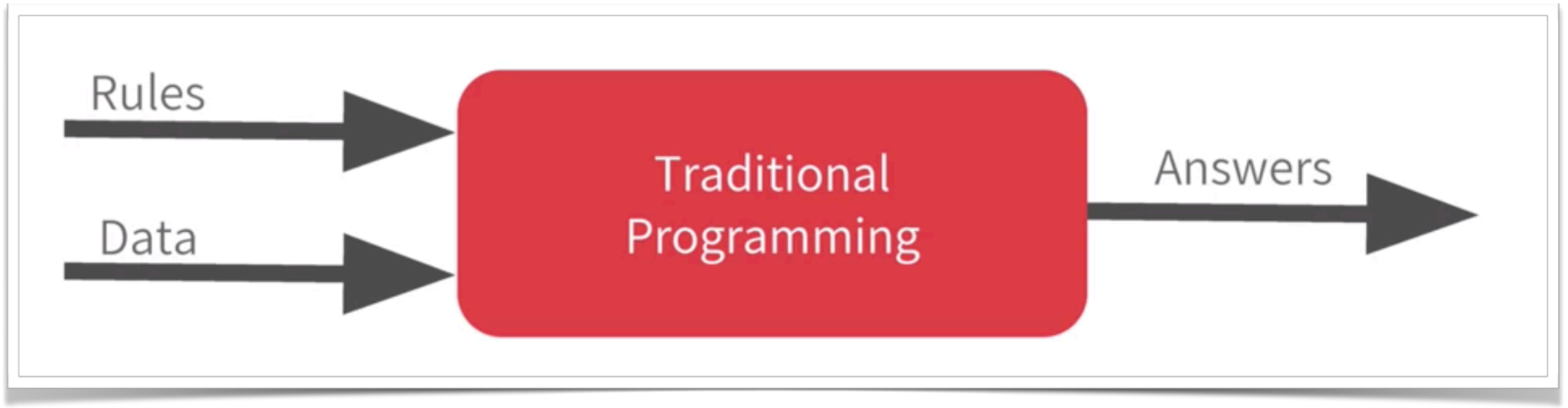
skymind

畳み込みニューラルネットワークの基礎と応用

Agenda

19:00 ～ 19:05 opening
19:05 ～ 19:20 **ニューラルネットワークの基礎**
19:20 ～ 19:30 **walkthrough** Kerasを用いたコーディングサンプル
19:30 ～ 19:45 **畳み込みニューラルネットワークの基礎**
19:45 ～ 20:00 **walkthrough** CNNのフィルタの理解
20:00 ～ 20:10 休憩
20:10 ～ 20:25 **CNNアーキテクチャの紹介**
20:25 ～ 20:40 **Skymind Intelligent Layer (SKIL) の解説**
20:40 ～ 20:55 **walkthrough** SKILを用いたモデルのデプロイ
20:55 ～ 21:00 closing


Rules
Data
Traditional Programming
Answers

Activity Recognition

```
if(speed<4){
    status=WALKING;
}
```

Activity Recognition

```
if(speed<4){
    status=WALKING;
}
```

```
if(speed<4){
    status=WALKING;
} else {
    status=RUNNING;
}
```

Activity Recognition

```
if(speed<4){
    status=WALKING;
}
```

```
if(speed<4){
    status=WALKING;
} else {
    status=RUNNING;
}
```

```
if(speed<4){
    status=WALKING;
} else if(speed<12){
    status=RUNNING;
} else {
    status=BIKING;
}
```

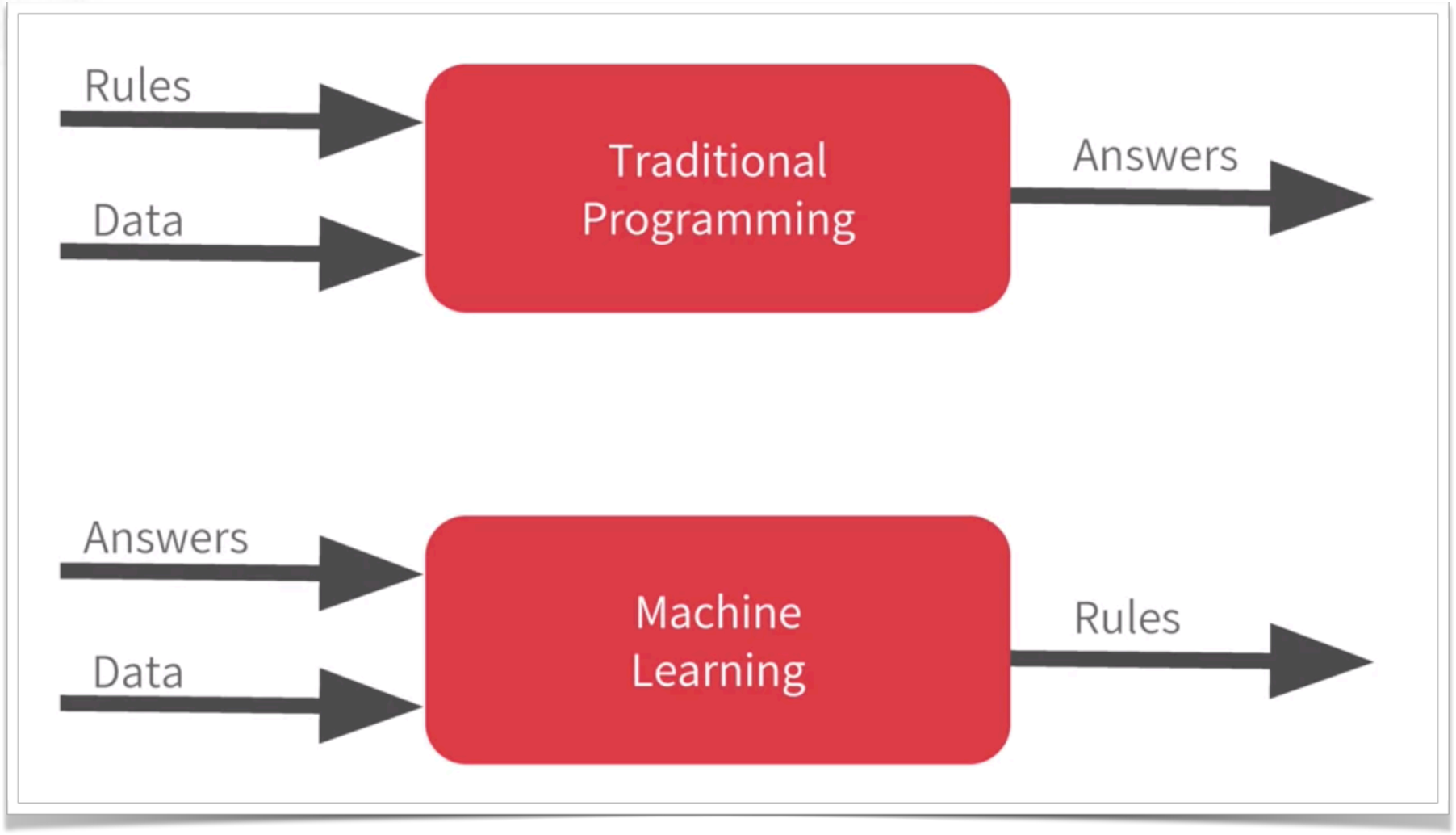
Activity Recognition

```
if(speed<4){
    status=WALKING;
}
```

```
if(speed<4){
    status=WALKING;
} else {
    status=RUNNING;
}
```

```
if(speed<4){
    status=WALKING;
} else if(speed<12){
    status=RUNNING;
} else {
    status=BIKING;
}
```

```
// Oh crap
```


Rules
Data
Traditional Programming
Answers
Answers
Data
Machine Learning
Rules

Activity Recognition

```
01010010101001010101
10010101010010111101
01001010100101010010
01010010101001010101
```

Label = WALKING

```
10101001010010101010
01010100100100100001
00100111110101011111
10101001001111010111
```

Label = RUNNING

```
10010100111110101010
11010101110101011100
10101011110101010110
11111100011110101010
```

Label = BIKING

```
11111111110100111010
00111110101111101010
01011101010101011100
10101010101001111100
```

Label = GOLFING
(Sort of)

機械学習の"Hello World”

```
x = -2, -1, 0, 1, 2, 3, 4
y = -3, -1, 1, 3, 5, 7, 9
```

$$y = f(x)$$

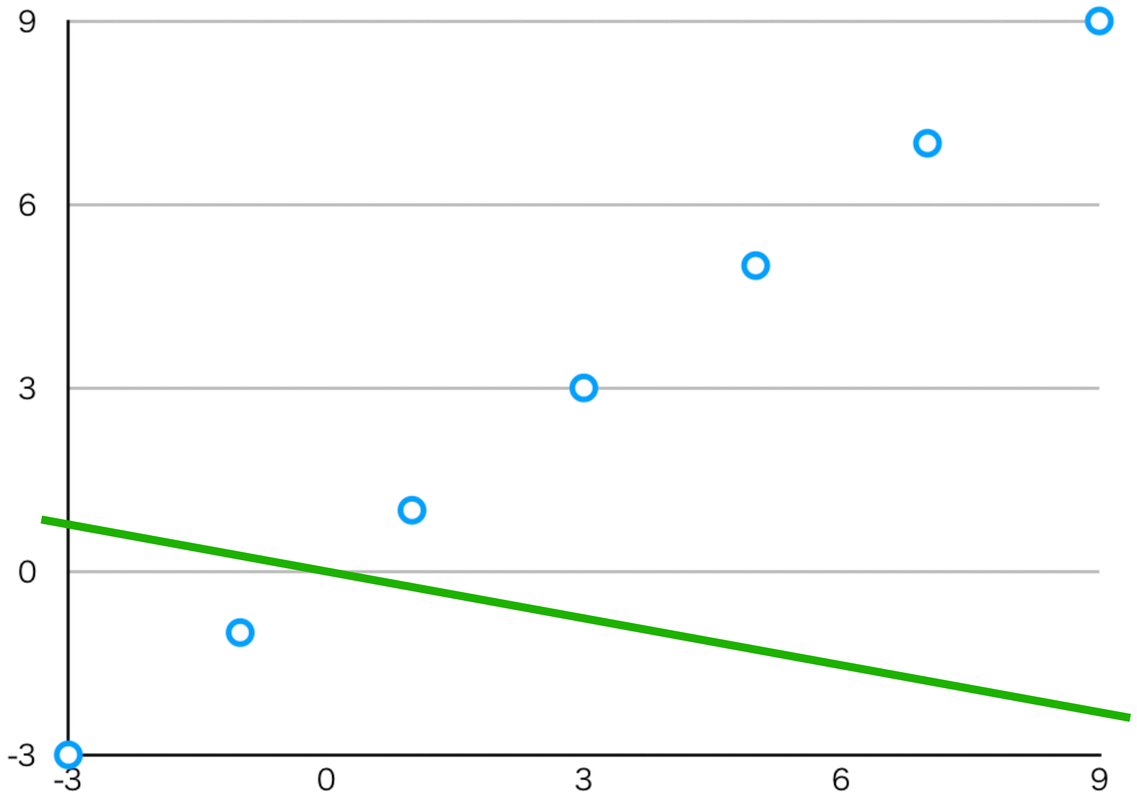
機械学習の"Hello World”

```
x = -2, -1, 0, 1, 2, 3, 4
y = -3, -1, 1, 3, 5, 7, 9
```

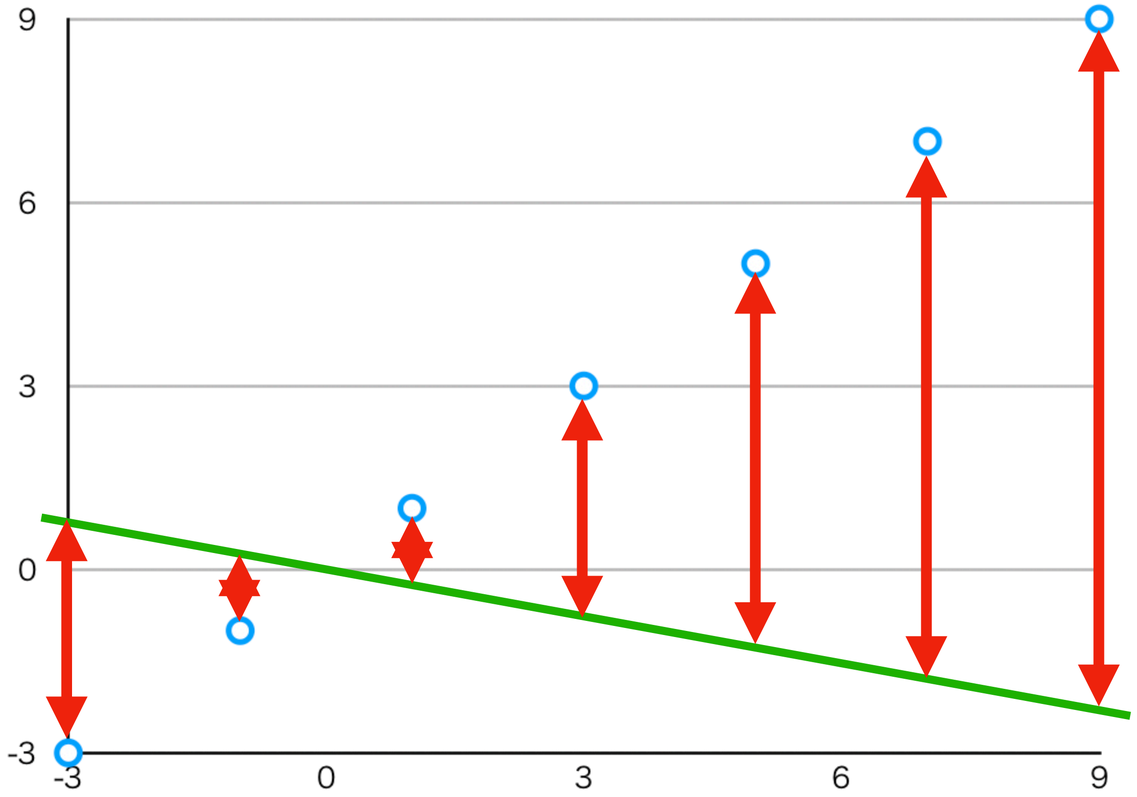
$$y = f(x) = 2x + 1$$

機械学習のアプローチ

- 適当にモデルを初期化 (y_=ax+b)

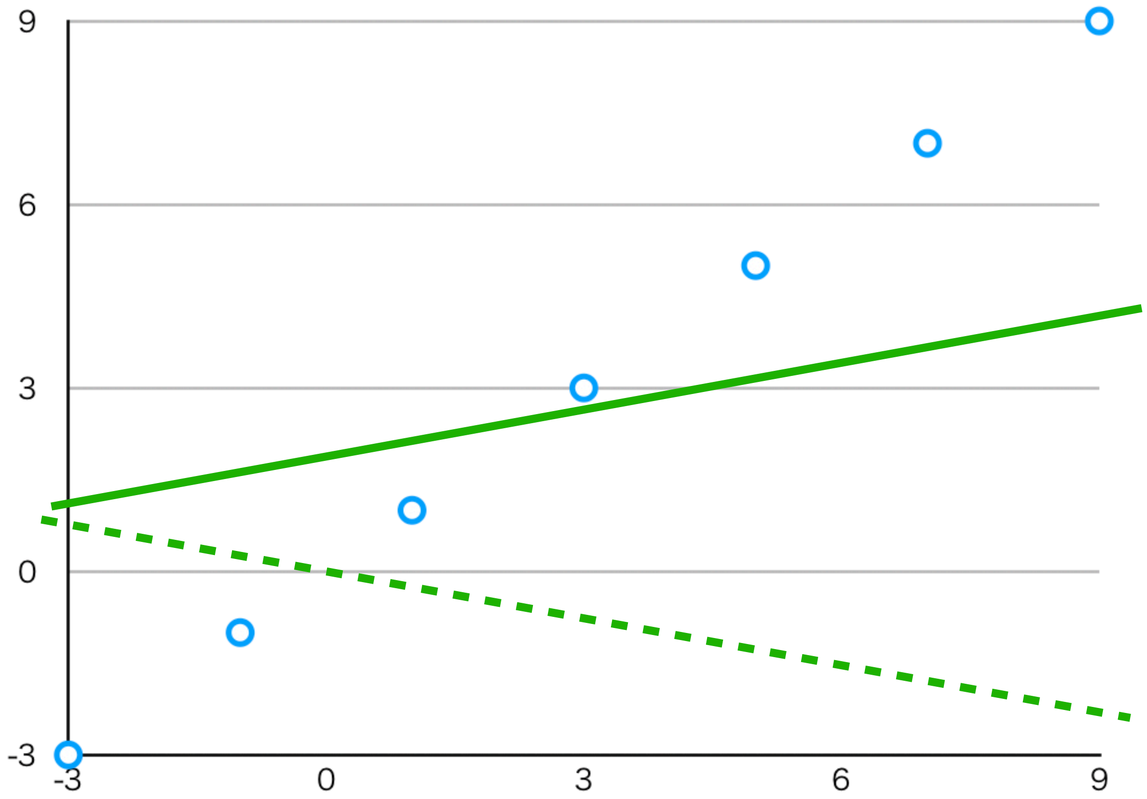


機械学習のアプローチ



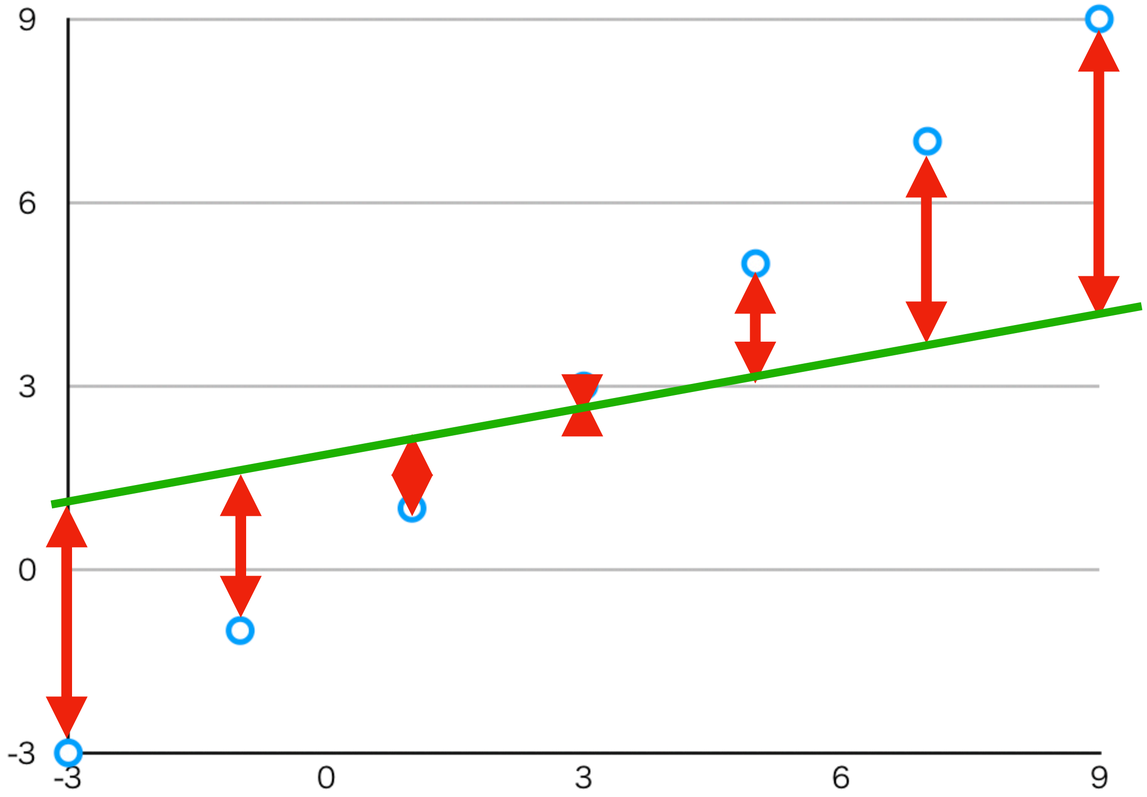
- 適当にモデルを初期化 ($y\_=ax+b$)
- **誤差（損失）を計算 ($L=1/N\ \Sigma(y-y\_)^2$)**

機械学習のアプローチ



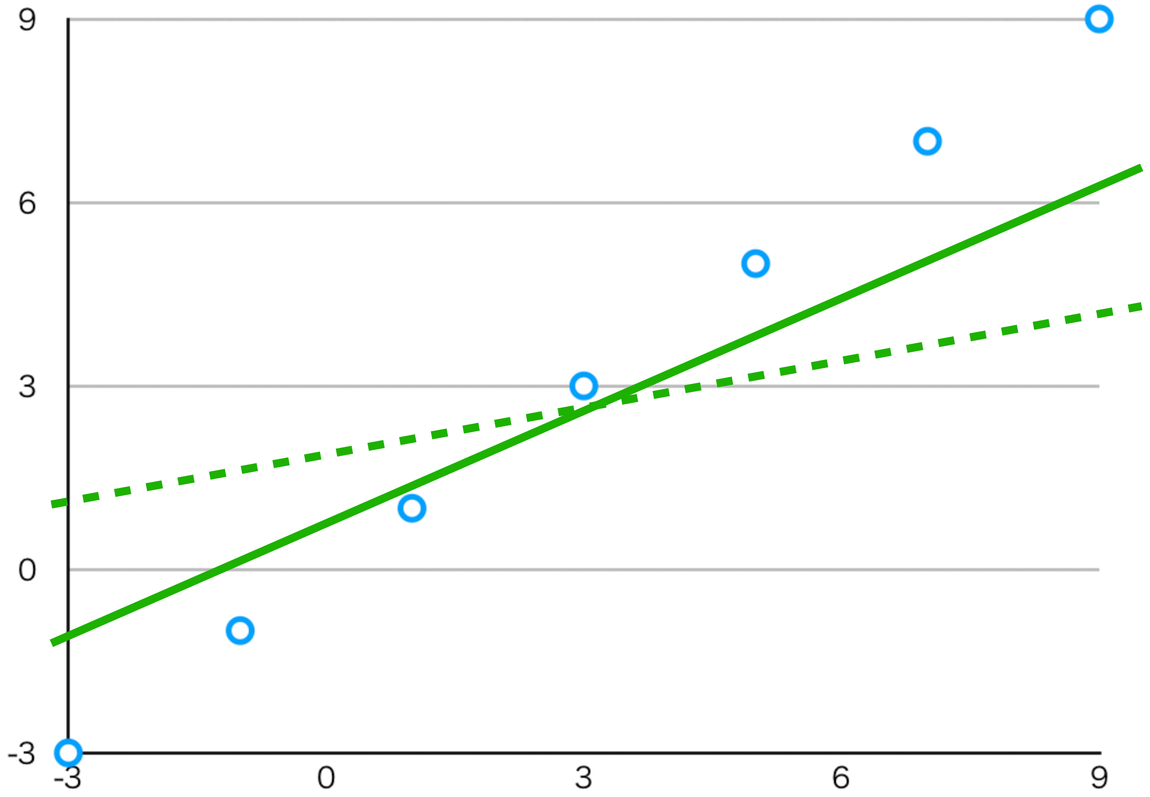
- 適当にモデルを初期化 ($y\_=ax+b$)
- 誤差（損失）を計算 ($L=1/N\ \Sigma(y-y\_)^2$)
- **誤差が小さくなるようにパラメータ (a, b) を少し更新 ($a \leftarrow a - \eta\ \partial L/\partial a$)**

機械学習のアプローチ



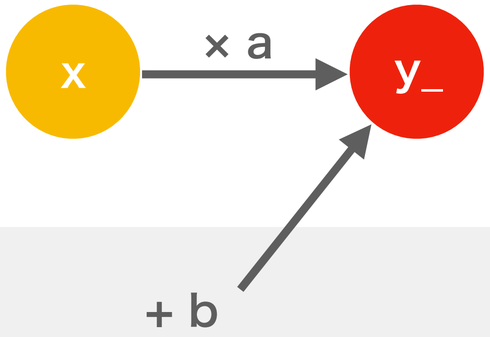
- 適当にモデルを初期化 ($y\_=ax+b$)
- 誤差（損失）を計算 ($L=1/N\ \Sigma(y-y\_)^2$)
- 誤差が小さくなるようにパラメータ (a, b) を少し更新 ($a \leftarrow a - \eta\ \partial L/\partial a$)
- **誤差（損失）を計算 ($L=1/N\ \Sigma(y-y\_)^2$)**

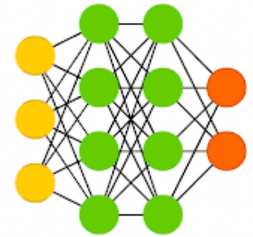
機械学習のアプローチ



- 適当にモデルを初期化 ($y\_=ax+b$)
- 誤差（損失）を計算 ($L=1/N\ \Sigma(y-y\_)^2$)
- 誤差が小さくなるようにパラメータ (a, b) を少し更新 ($a \leftarrow a - \eta\ \partial L/\partial a$)
- 誤差（損失）を計算 ($L=1/N\ \Sigma(y-y\_)^2$)
- **誤差が小さくなるようにパラメータ (a, b) を少し更新 ($a \leftarrow a - \eta\ \partial L/\partial a$)**
- …

```
import tensorflow as tf


model = tf.keras.Sequential([tf.keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')

xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model.fit(xs, ys, epochs=500)

>>>print(model.predict([10.0]))
[[18.977331]]
```



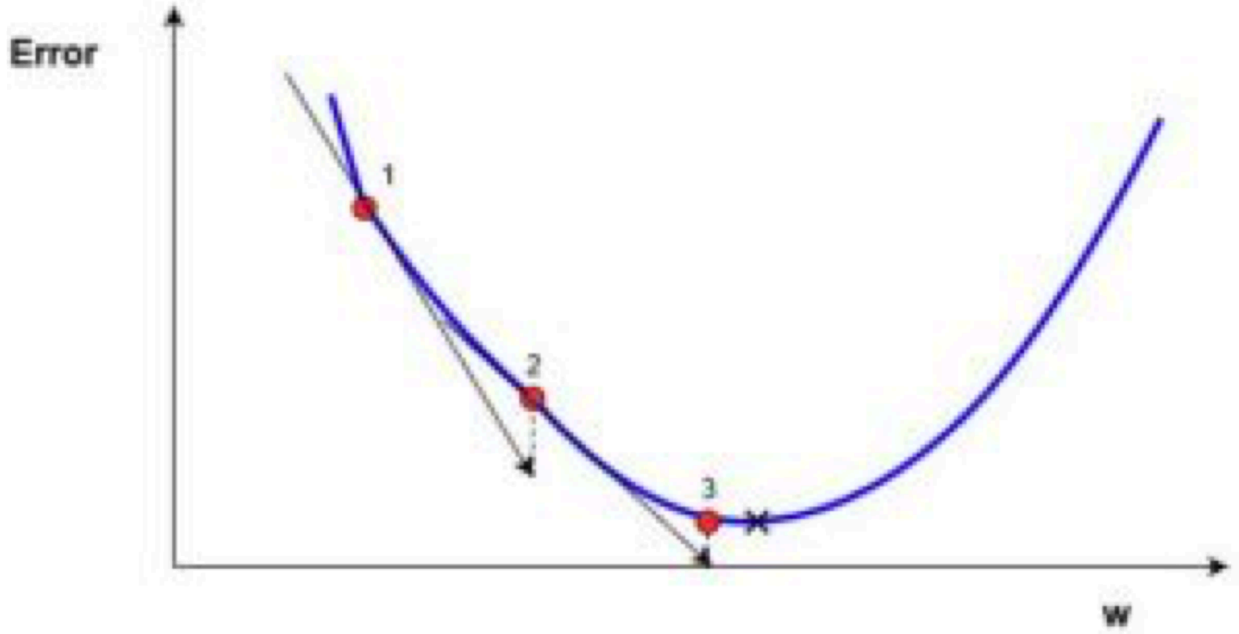


```
1  import tensorflow as tf
2
3
4  model = tf.keras.Sequential([tf.keras.layers.Dense(units=1, input_shape=[1])])
5  model.compile(optimizer='sgd', loss='mean_squared_error')
6
7  xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
8  ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)
9
10 model.fit(xs, ys, epochs=500)
11
12 >>>print(model.predict([10.0]))
13 [[18.977331]]
```

```
import tensorflow as tf


model = tf.keras.Sequential([tf.keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')

xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model.fit(xs, ys, epochs=500)

>>>print(model.predict([10.0]))
[[18.977331]]
```

skymind

確率的勾配降下法 SGD, stochastic gradient descent.
デファクトスタンダード
（改良版：momentum, Nesterovの加速法, RMSProp, Adam, ...）



$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

勾配降下法

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

「目隠しで足元の勾配情報のみを使って山の頂上を目指すようなもの」

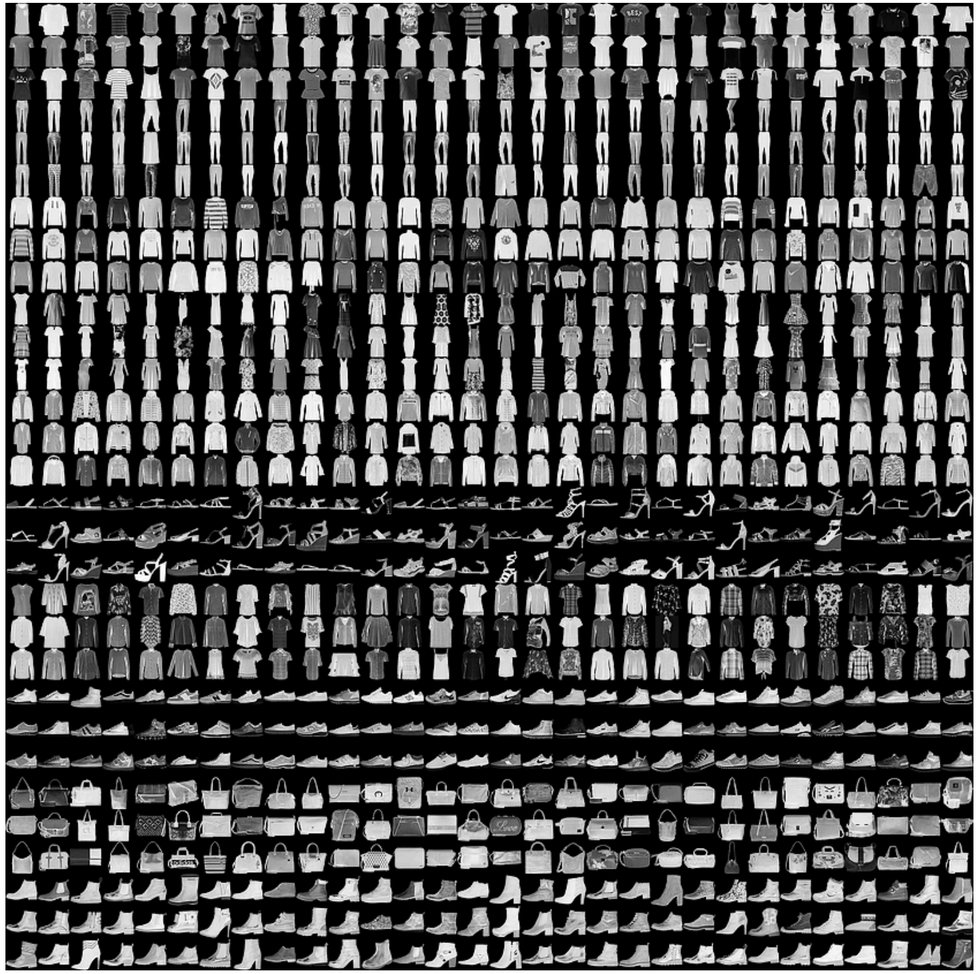
https://twitter.com/momiji_fullmoon/status/1110316960611368960

学習率 η は歩幅のイメージ（η小＝すり足、η大＝巨人の一歩）

Fashion MNIST Dataset

- 7万画像
- 10カテゴリ
- 28×28 pixels
- 実験用データセット

ラベル	記述
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

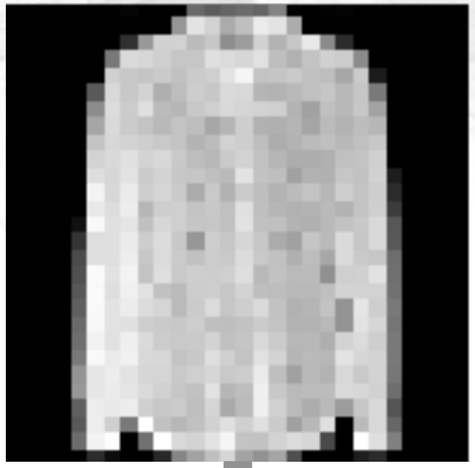


https://github.com/zalandoresearch/fashion-mnist

```
import tensorflow as tf
import matplotlib.pyplot as plt

fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
train_images = train_images / 255.
test_images = test_images / 255.
```

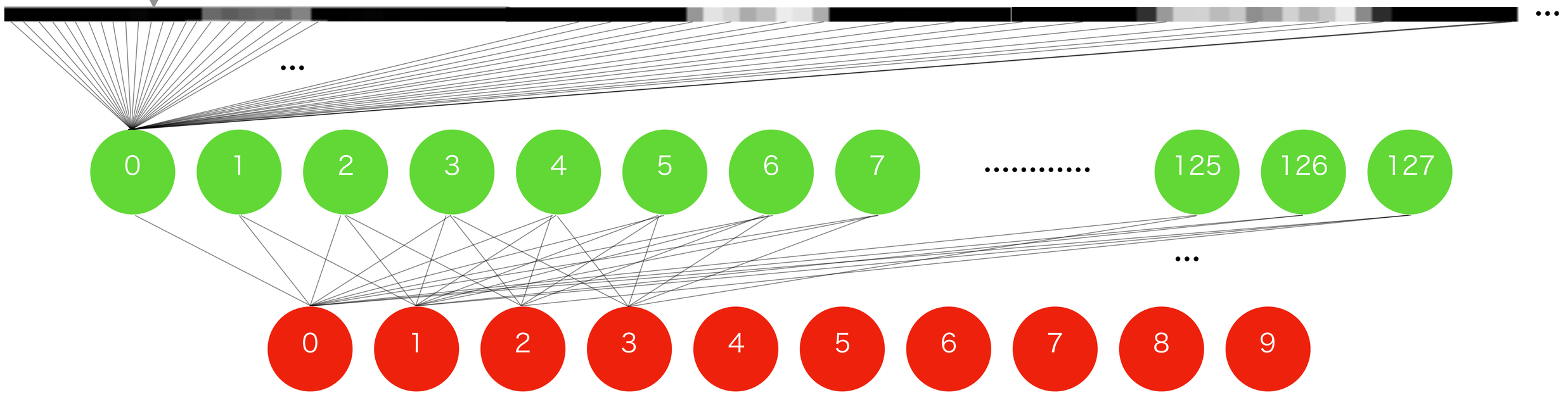
```
import tensorflow as tf
import matplotlib.pyplot as plt

fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
train_images = train_images / 255.
test_images = test_images / 255.

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

```
import tensorflow as tf
import matplotlib.pyplot as plt

fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
train_images = train_images / 255.
test_images = test_images / 255.

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
    loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=128, verbose=1,
    validation_data=(test_images, test_labels))
```



```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

Flatten: (28, 28) => (784)

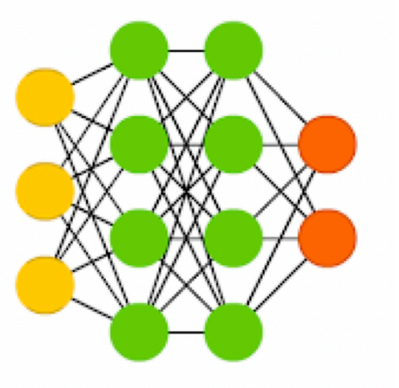


Dense Layer の欠点

- 入力のベクトルの**全要素**の相関をみている

  > 住宅価格予測みたいな話ならまだいい

  > もう作ってる特徴量と特徴量の組み合わせ

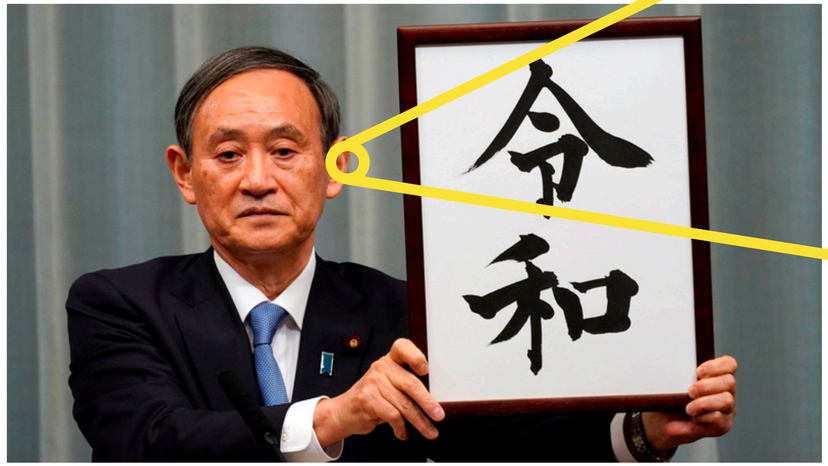


  - 例）東京墨田区 & 床面積 30m^2 & 1K & 風呂トイレ別 & 新築 => 家賃月10万円

- **「画像の特徴量」を抽出**してからDense Layerに渡せば効率的

畳込みニューラルネットワーク (Convolutional Neural Network; CNN)

畳み込み (Convolution)

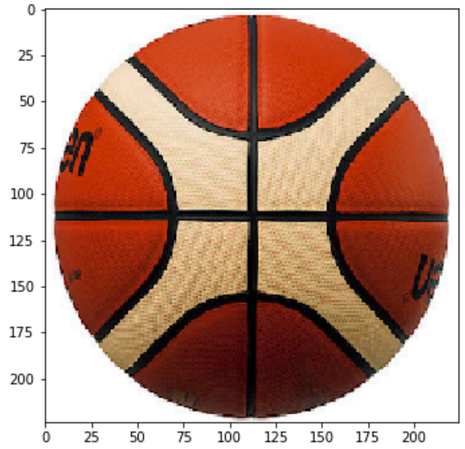


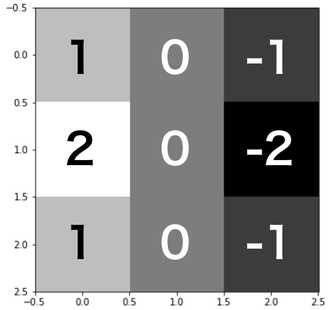
144	60	19
188	82	32
156	55	27

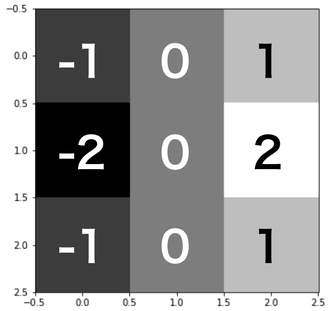
-1	0	-2
0.5	4.5	-1.5
1.5	2	-3

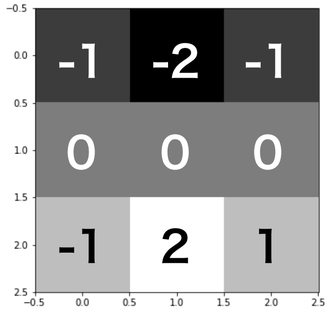
$$u_{ijm} = \sum_{k=0}^{K-1} \sum_{p=0}^{W-1} \sum_{q=0}^{H-1} z_{i+p,j+q,k}^{(l-1)} h_{pqkm} + b_{ijm}$$

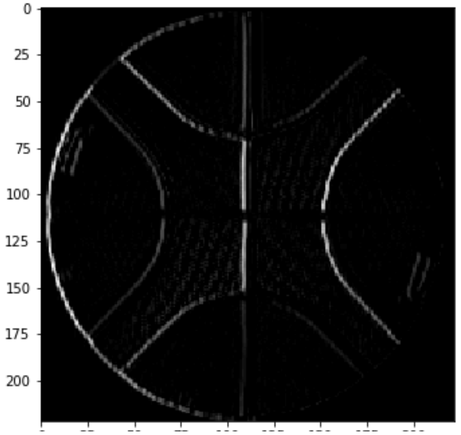
```
CURRENT_PIXEL_VALUE = 82

NEW_PIXEL_VALUE =

   (-1 * 144) + (0 * 60) + (-2 * 19)

+ (0.5 * 188) + (4.5 * 82) + (-1.5 * 32)

+ (1.5 * 156) + (2 * 55) + (-3 * 27)
```

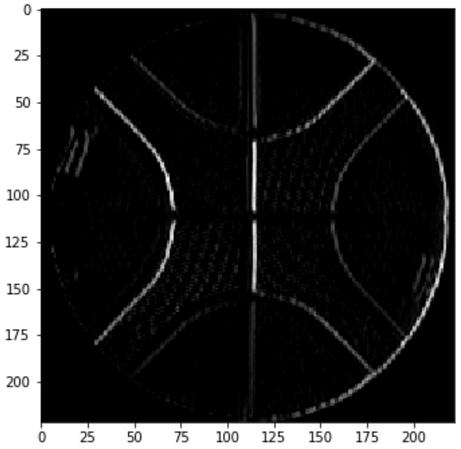


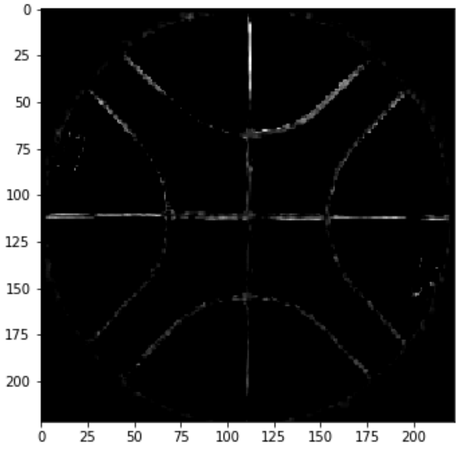

1 0 -1
2 0 -2
1 0 -1


-1 0 1
-2 0 2
-1 0 1


-1 -2 -1
0 0 0
-1 2 1







stride=(1, 1), padding='valid'

stride=(1, 1), padding='valid'

stride=(1, 1), padding='valid'

stride=(1, 1), padding='valid'

stride=(1, 1), padding='valid'

stride=(1, 1), padding='valid'

stride=(1, 1), padding='valid'

stride=(2, 2), padding='valid'

stride=(2, 2), padding='valid'

stride=(2, 2), padding='valid'

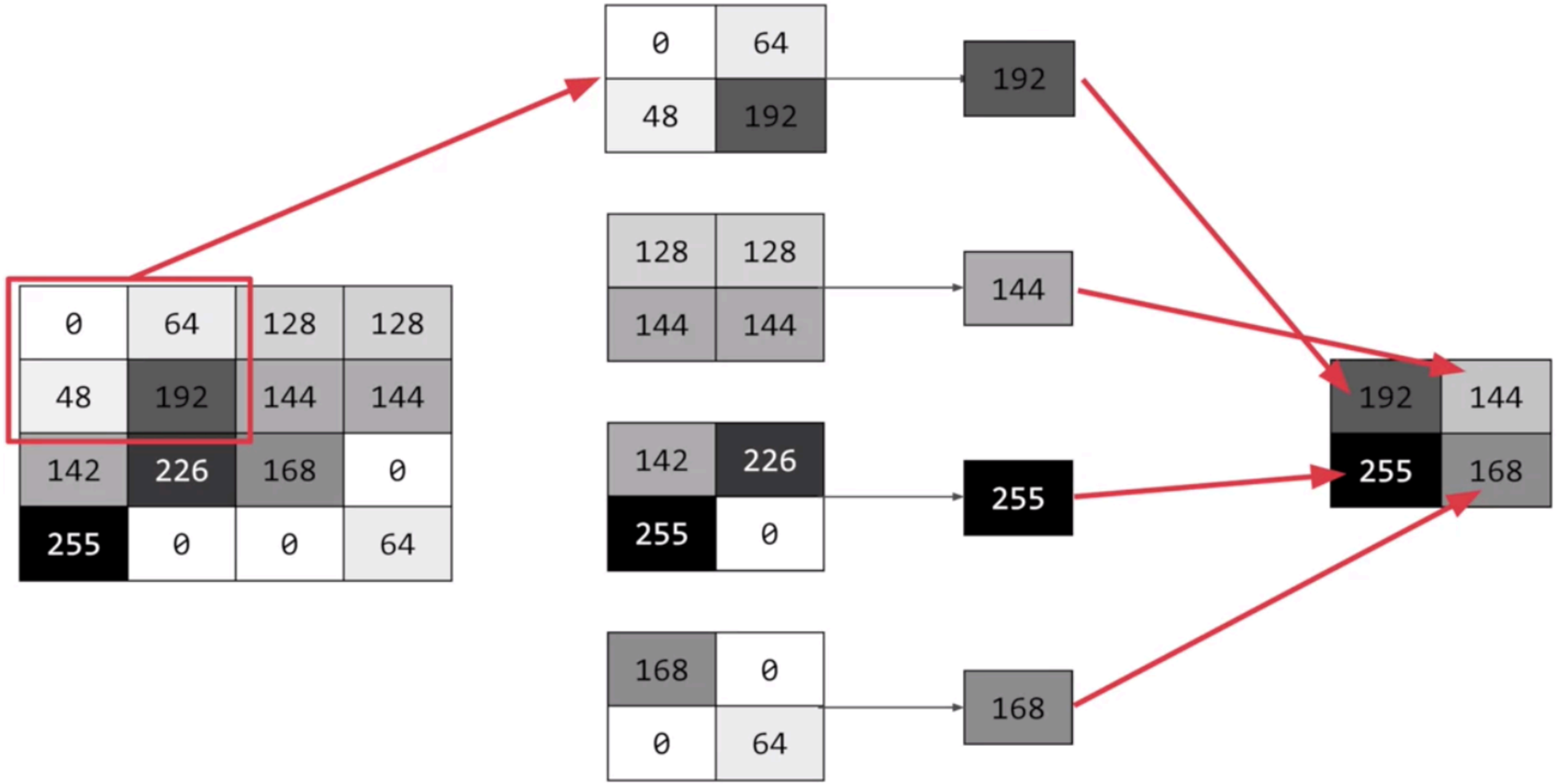
stride=(1, 1), padding='same'

stride=(1, 1), padding='same'

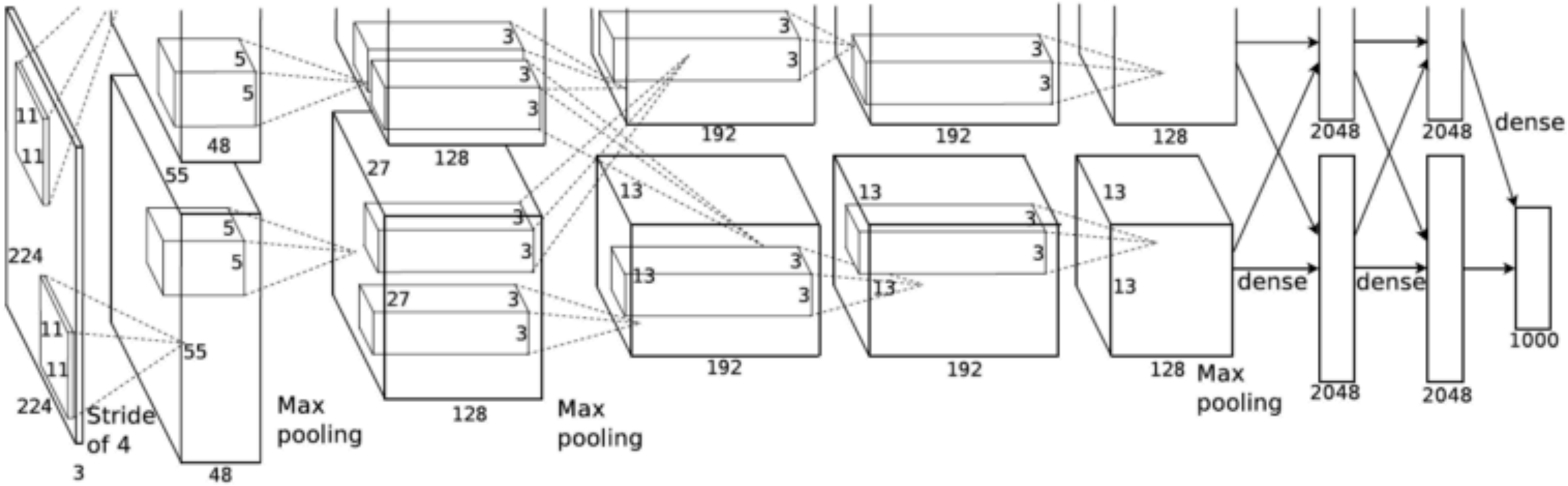
stride=(1, 1), padding='same'

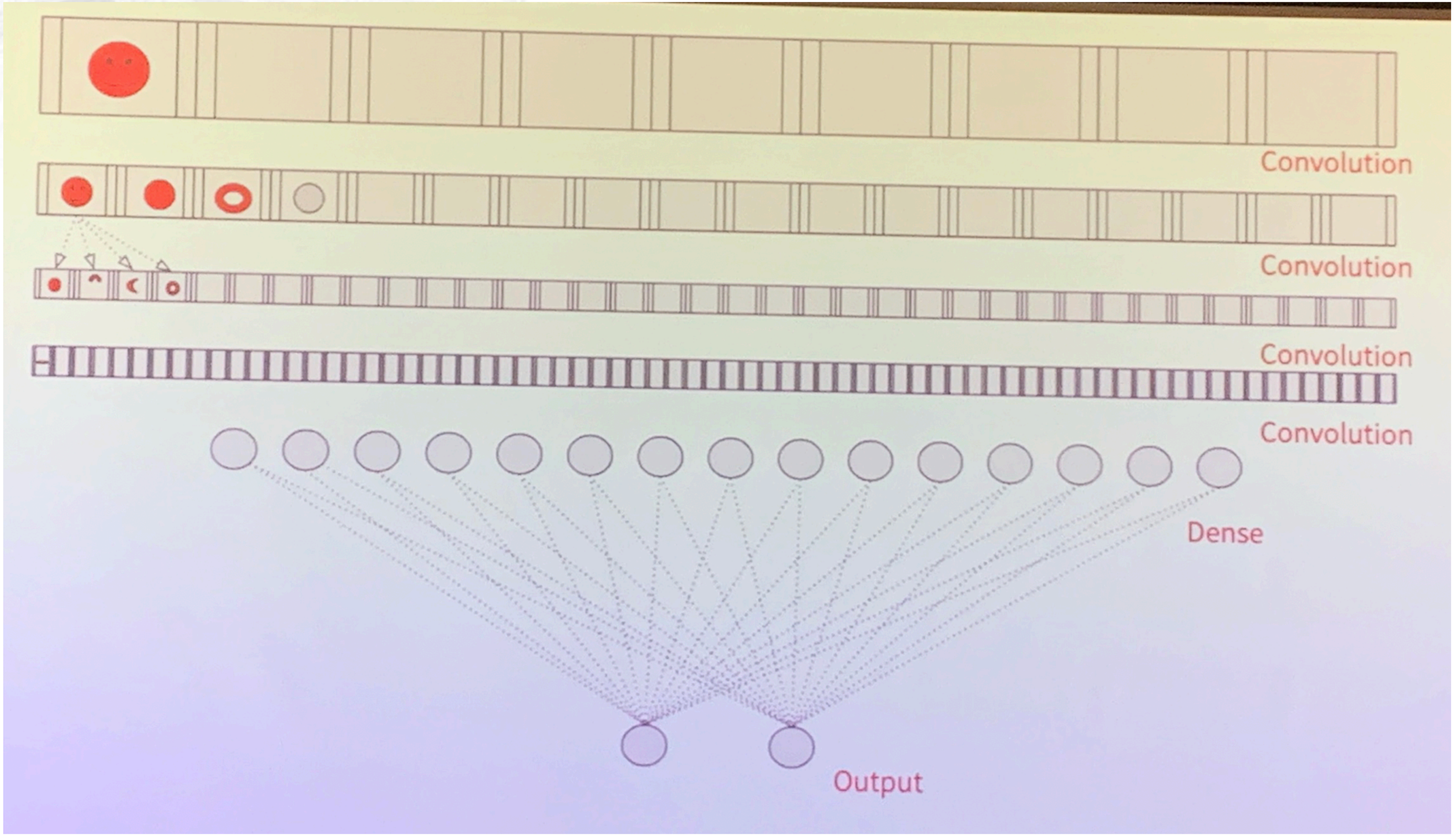
stride=(1, 1), padding='same'

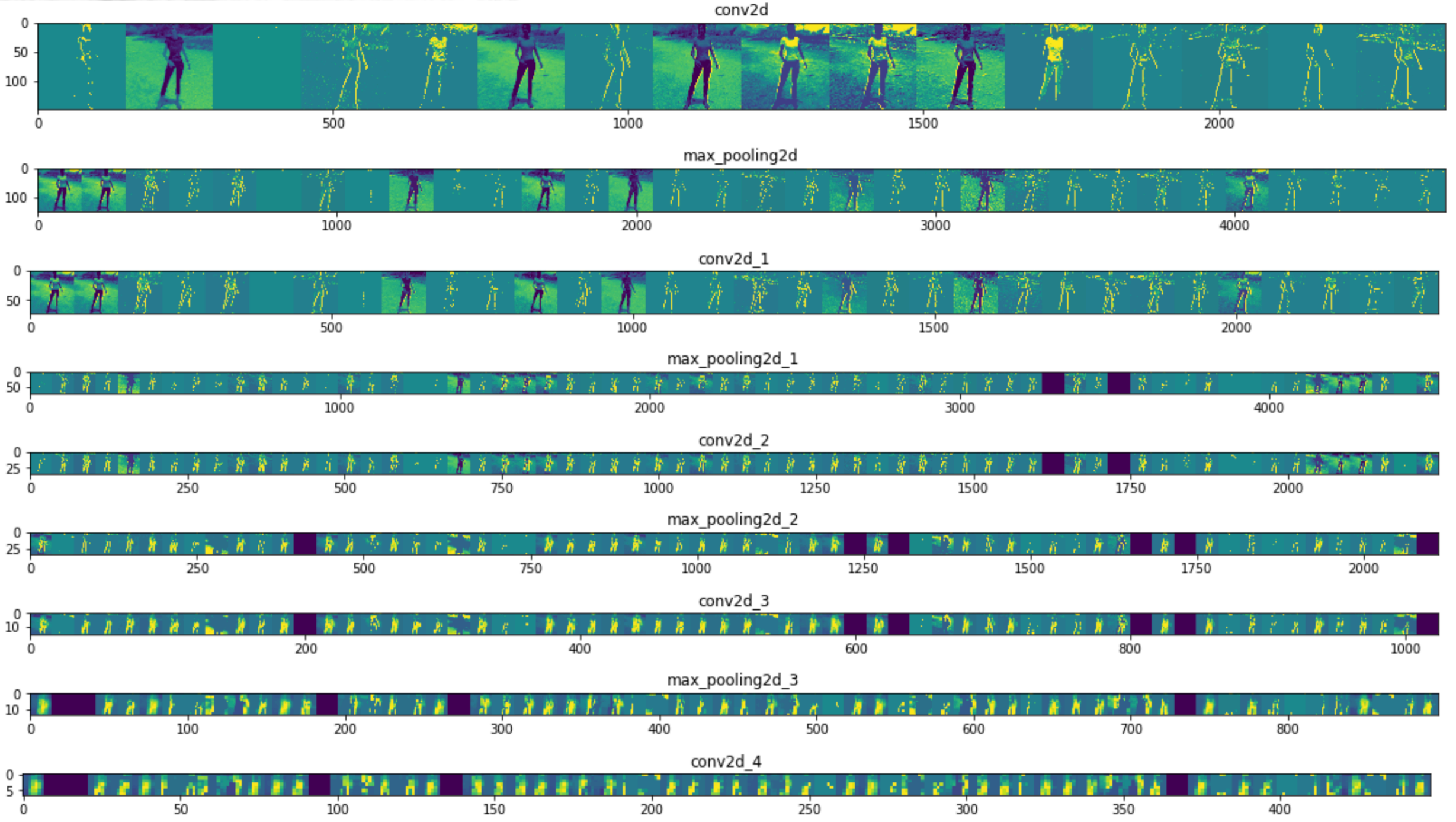
Max Pooling

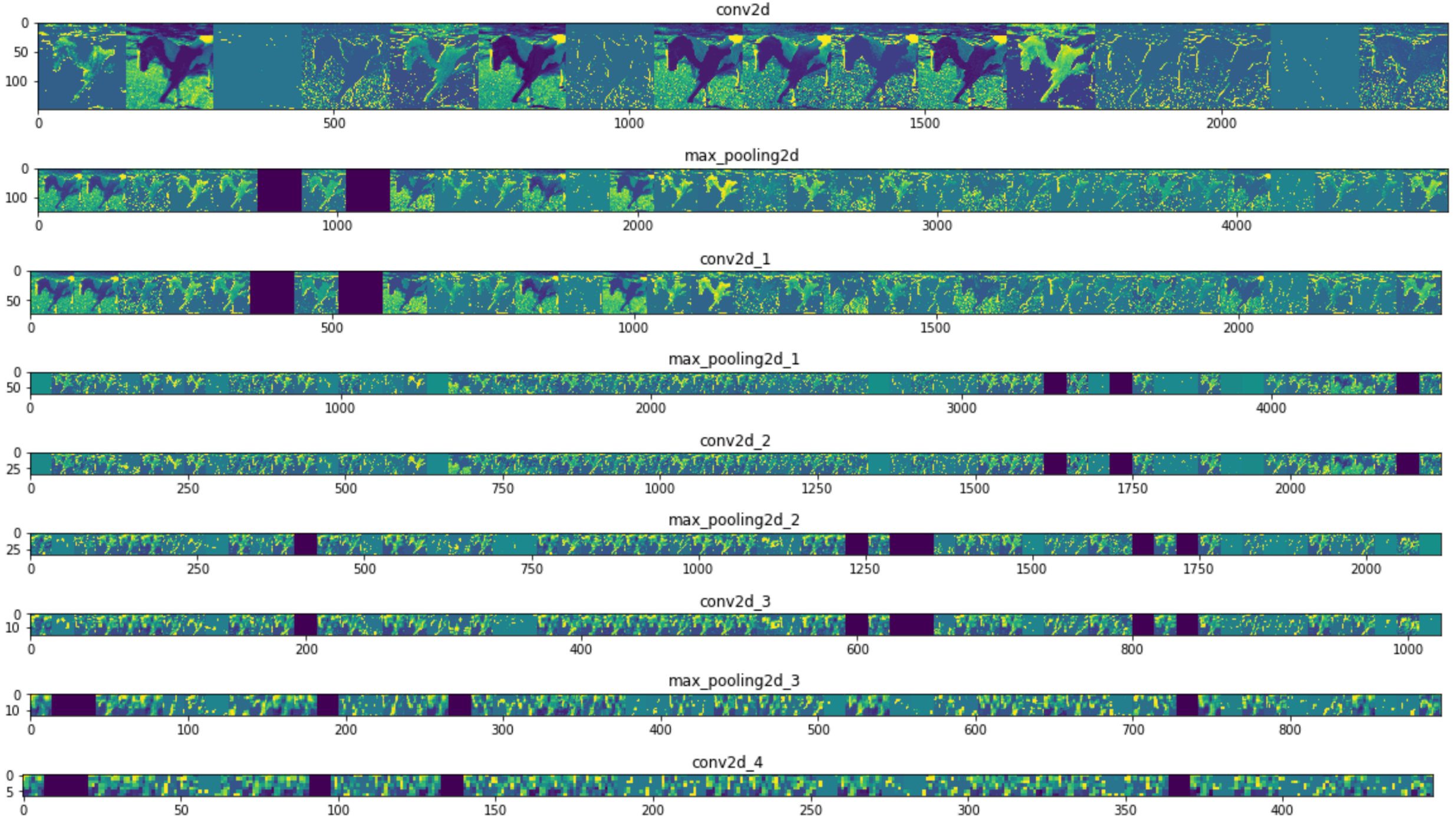

0 64 128 128
48 192 144 144
142 226 168 0
255 0 0 64
0 64
48 192
192
128 128
144 144
144
142 226
255 0
255
168 0
0 64
168
192 144
255 168

AlexNet

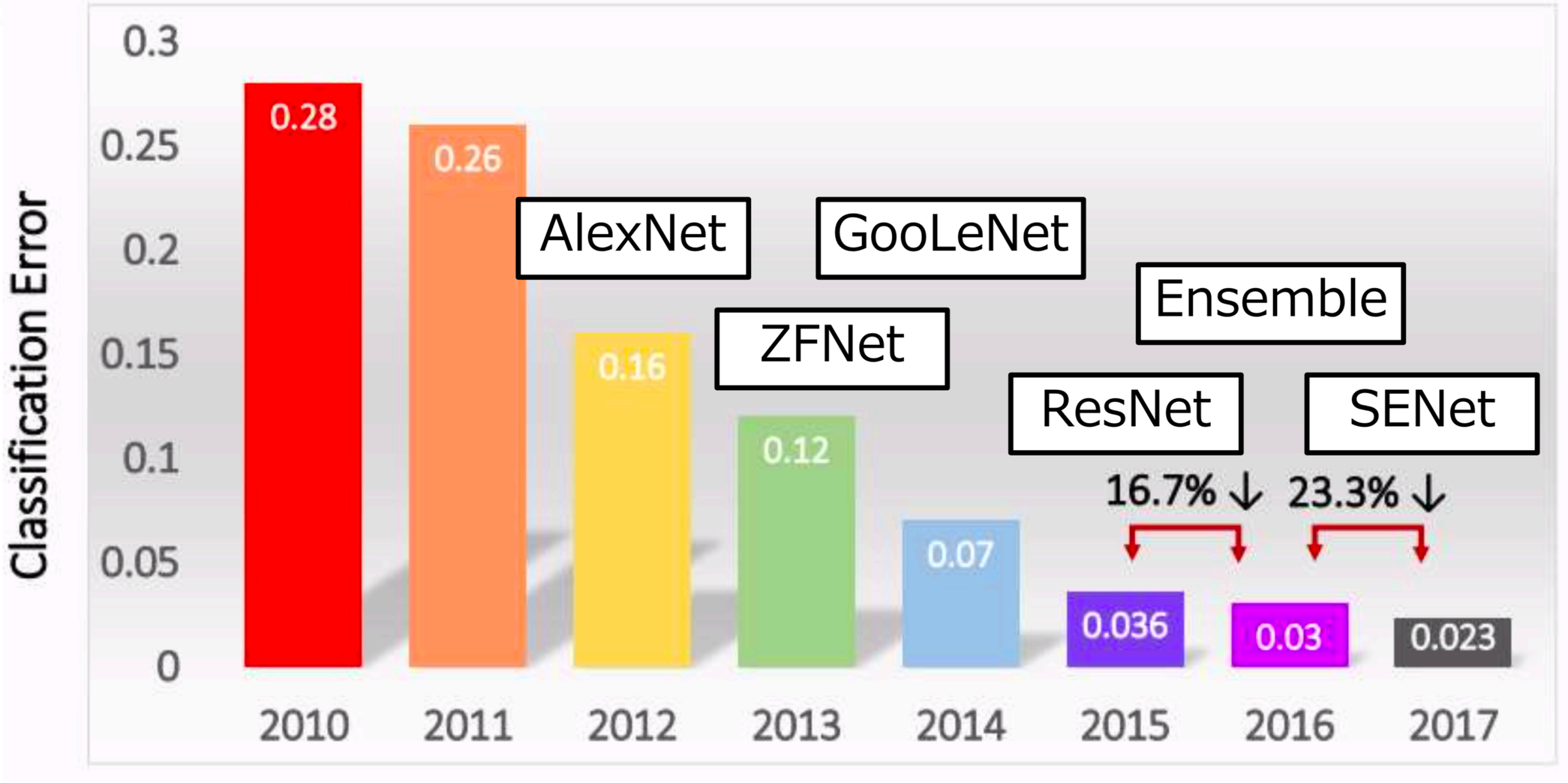

11
11
224
224
3
Stride
of 4
5
5
48
55
48
5
5
55
11
11
Max
pooling
3
3
3
128
27
128
3
3
27
3
3
128
Max
pooling
3
3
192
3
3
192
13
13
13
192
3
3
13
3
3
192
13
13
128
Max
pooling
dense
2048
2048
dense
dense
2048
2048
dense
1000


Convolution
Convolution
Convolution
Convolution
Dense
Output


conv2d
max_pooling2d
conv2d_1
max_pooling2d_1
conv2d_2
max_pooling2d_2
conv2d_3
max_pooling2d_3
conv2d_4


conv2d
max_pooling2d
conv2d_1
max_pooling2d_1
conv2d_2
max_pooling2d_2
conv2d_3
max_pooling2d_3
conv2d_4

ImageNet コンペの優勝モデル

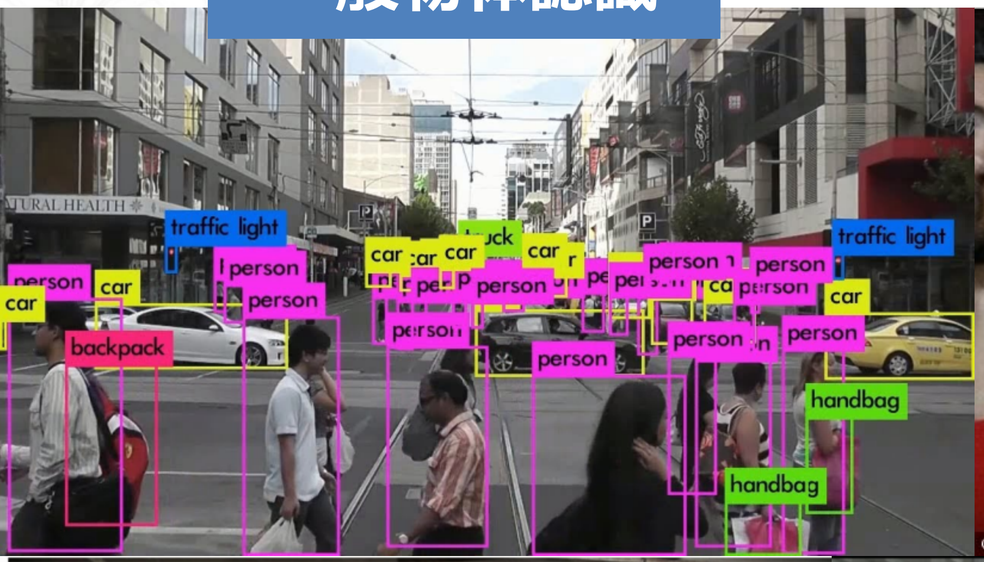


ResNet

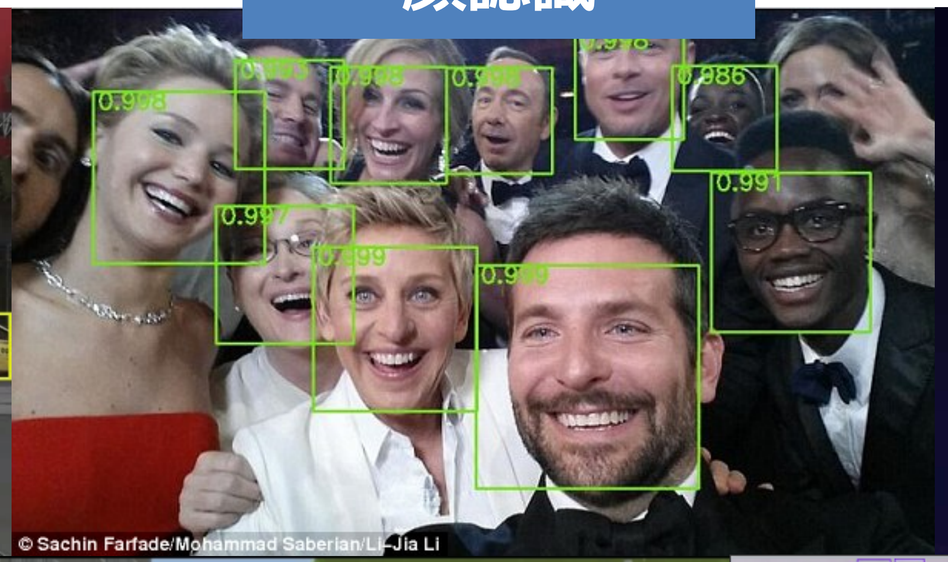
YOLO

skymind | 画像認識のアプリケーション

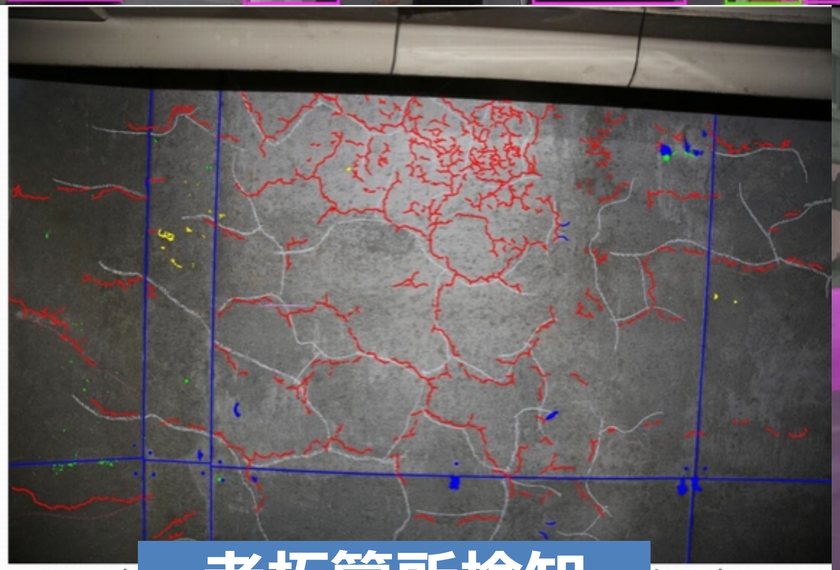
一般物体認識



顔認識

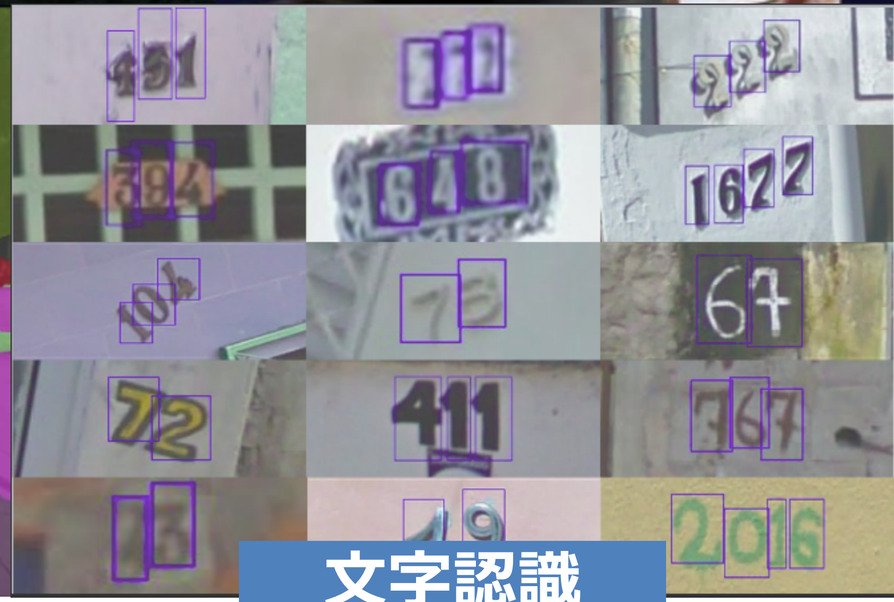


年齢推定



老朽箇所検知

セグメンテーション

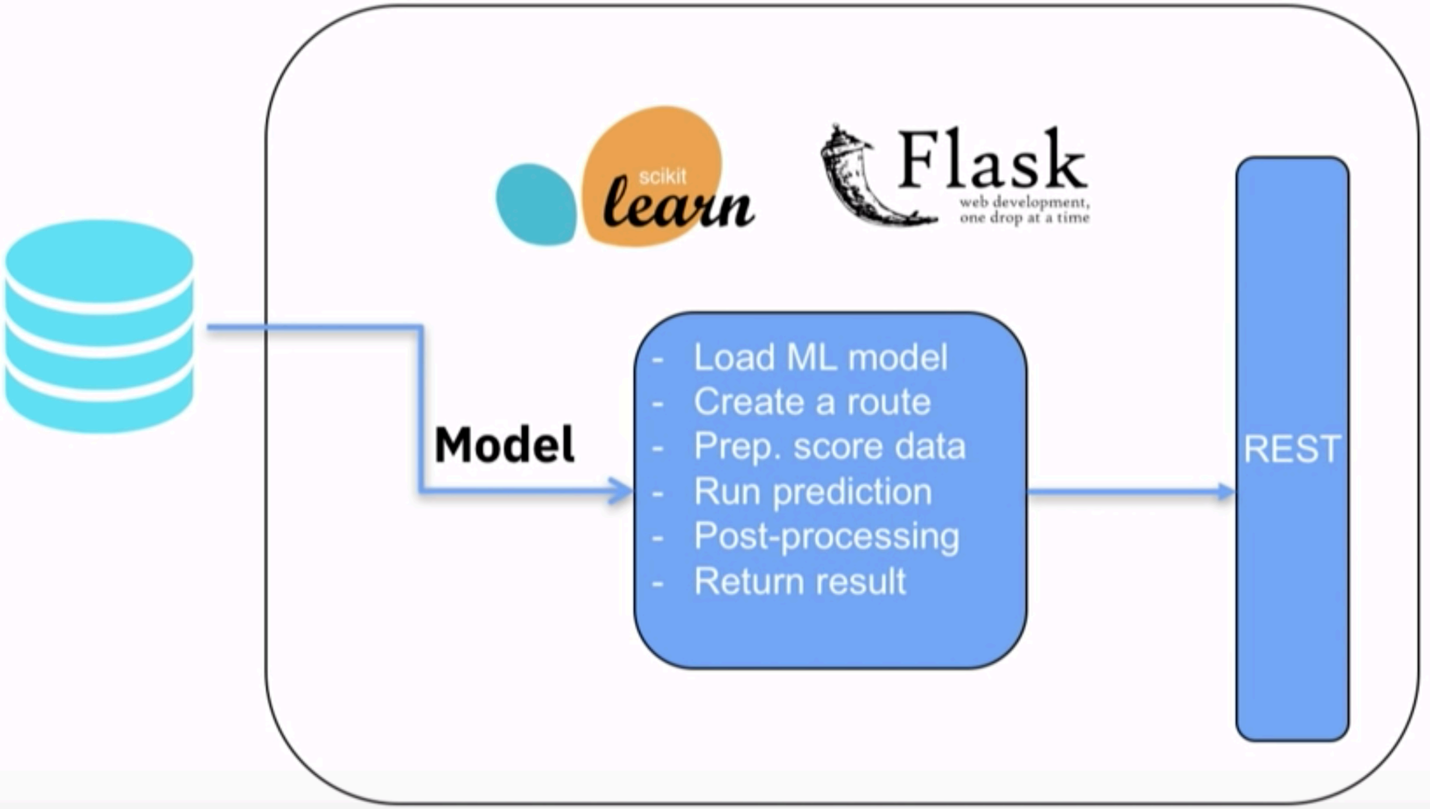


文字認識

Deployment with Cloud Foundry

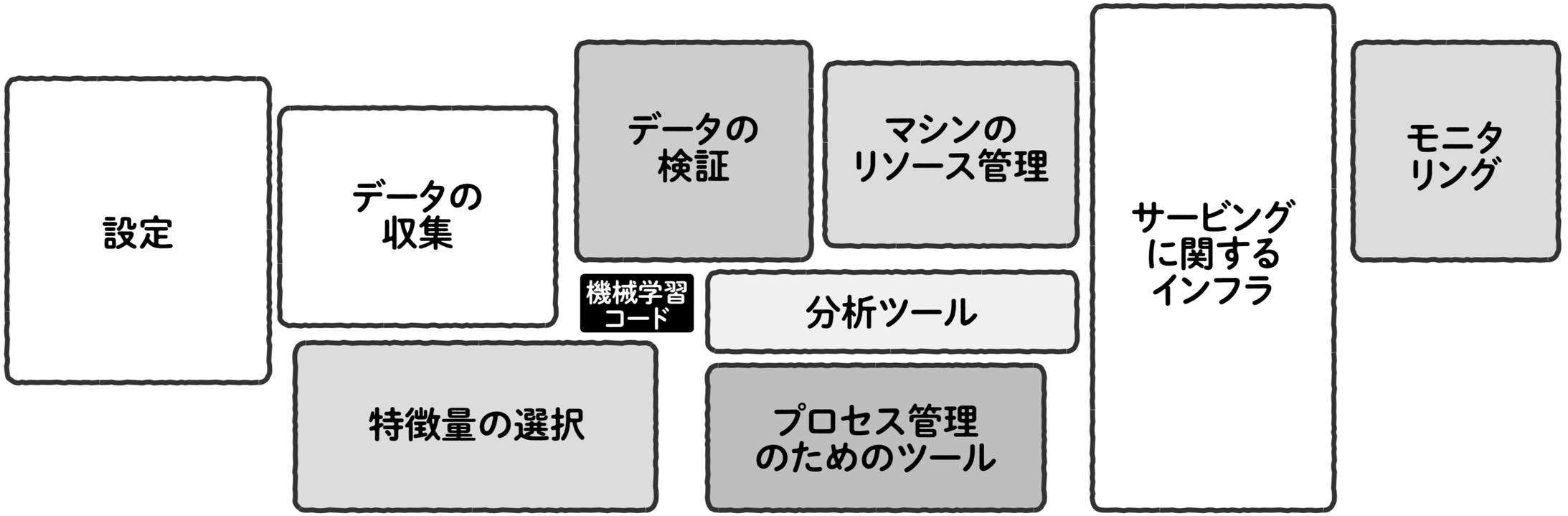
Python App - REST API
Endpoint: /v1/xyz/prediction : POST

Questions
- How to configure the app ?
- How many instances ?
- How much RAM ?
- Zero downtime deployment ?
- Connection to DB?
- Logging ?
- Crashes and recovery?
- Routing ?
- Workload scaling ?
- Integration with other tools?
- ...



https://www.youtube.com/watch?v=-UYyyeYJAoQ

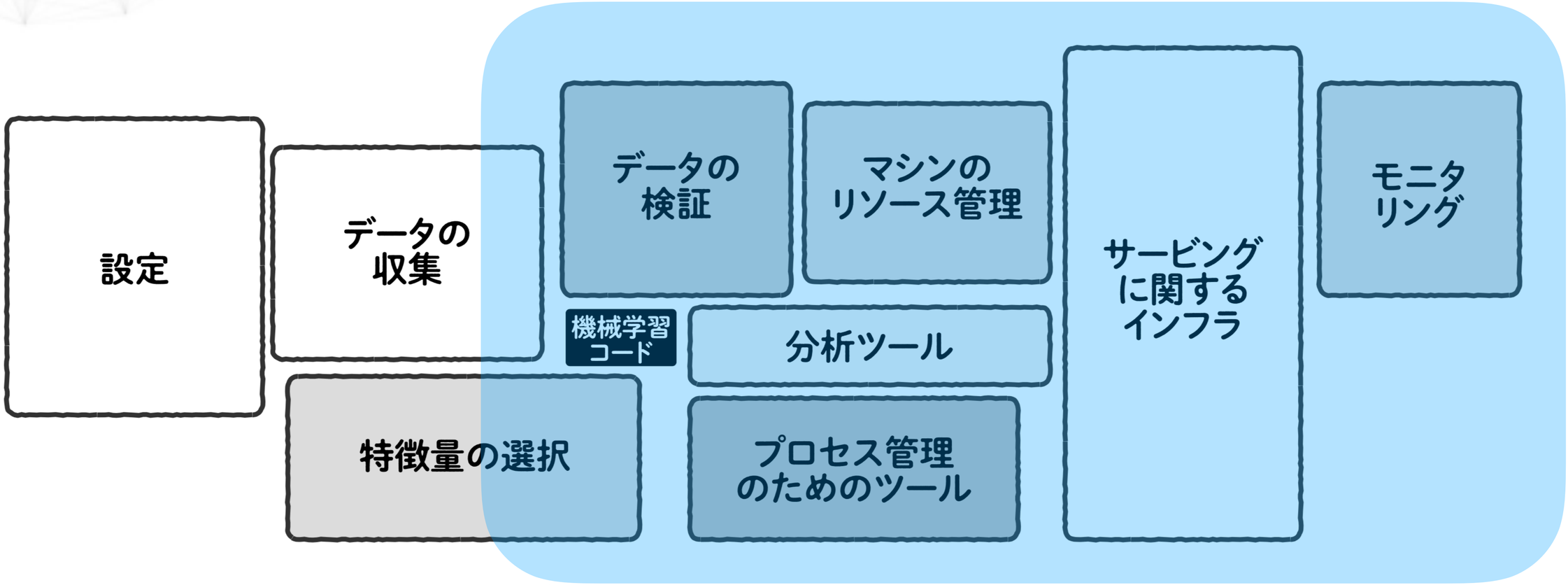
モデルの利用



▶ 図 3.1　機械学習のモデリングのコードの割合は少ない（[2] 中の図を翻訳して引用）

n月間ラムダノートVol.1, No.1より

モデルの利用

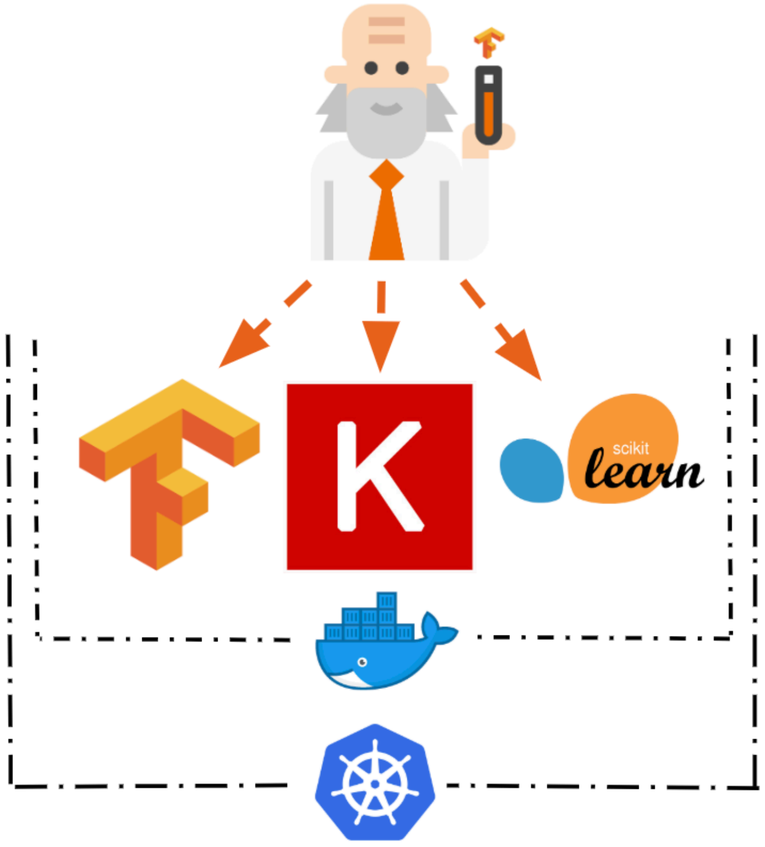


▶ 図 3.1　機械学習のモデリングのコードの割合は少ない（[2] 中の図を翻訳して引用）

skymind

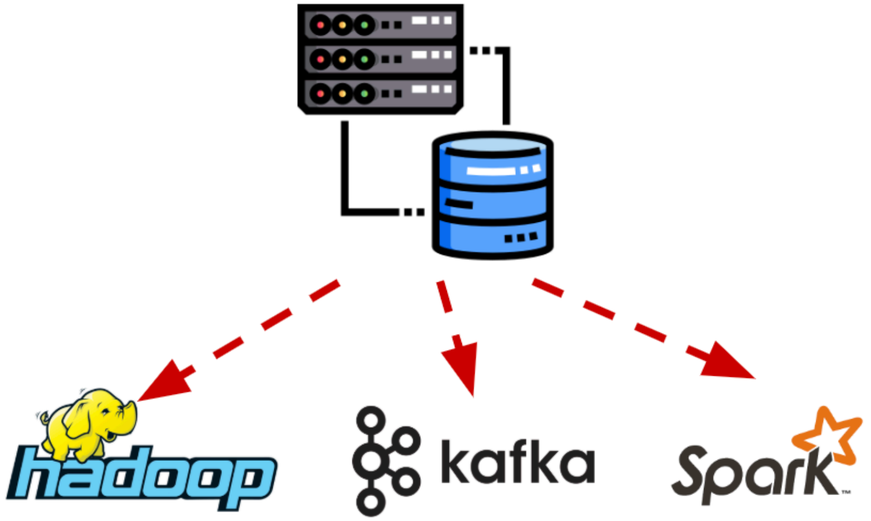
Most AI Frameworks Not Built for Enterprise

Development



≠

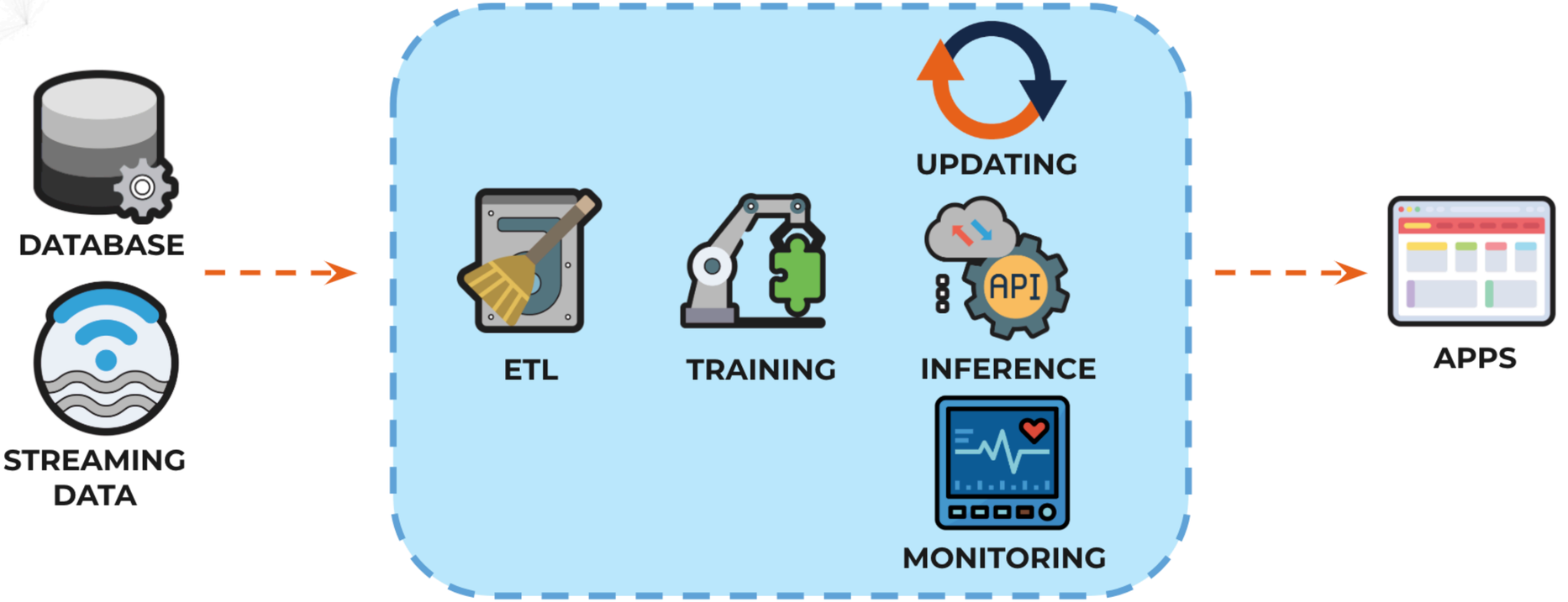
Production



- Petabytes of data
- Distributed clusters for data processing
- SLAs mandatory
  - Uptime guarantees!

skymind

SKIL provides a solution



Standardized AI infrastructure lets teams unify practices.

Configurable clusters: Adapts to opinionated ops teams and messy environments

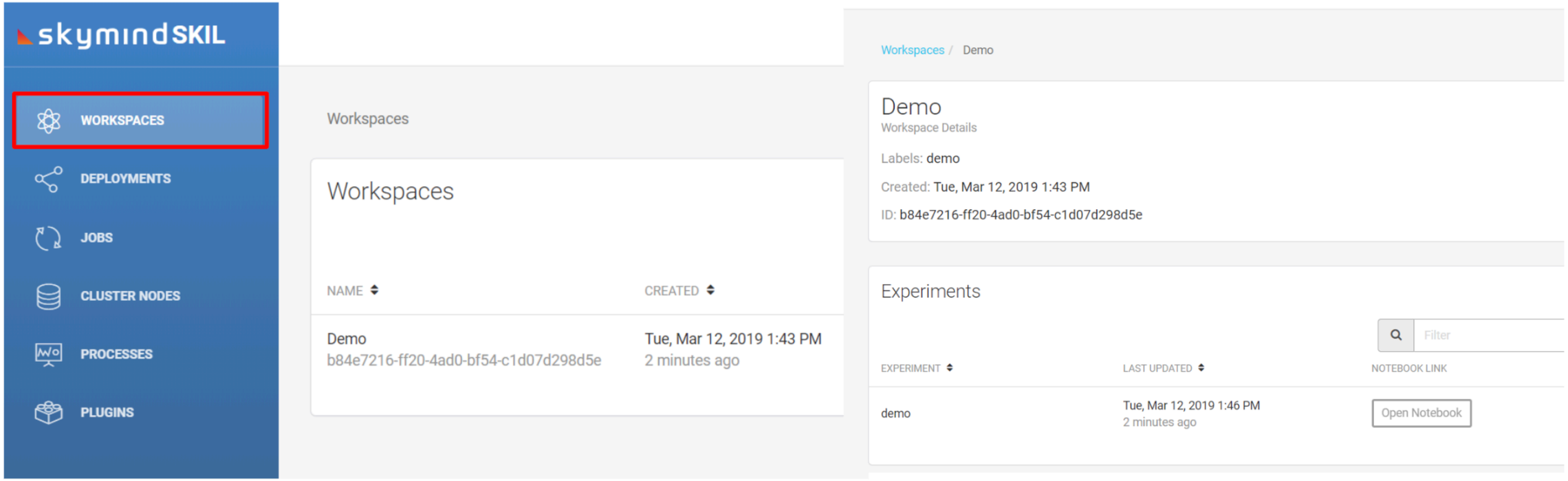
A managed software distribution for popular ML/AI frameworks

Ingests AI models from any ML vendor/tool, allowing data scientists to keep favorite tools

skymind | SKIL User Interface

Workspaces, Experiments

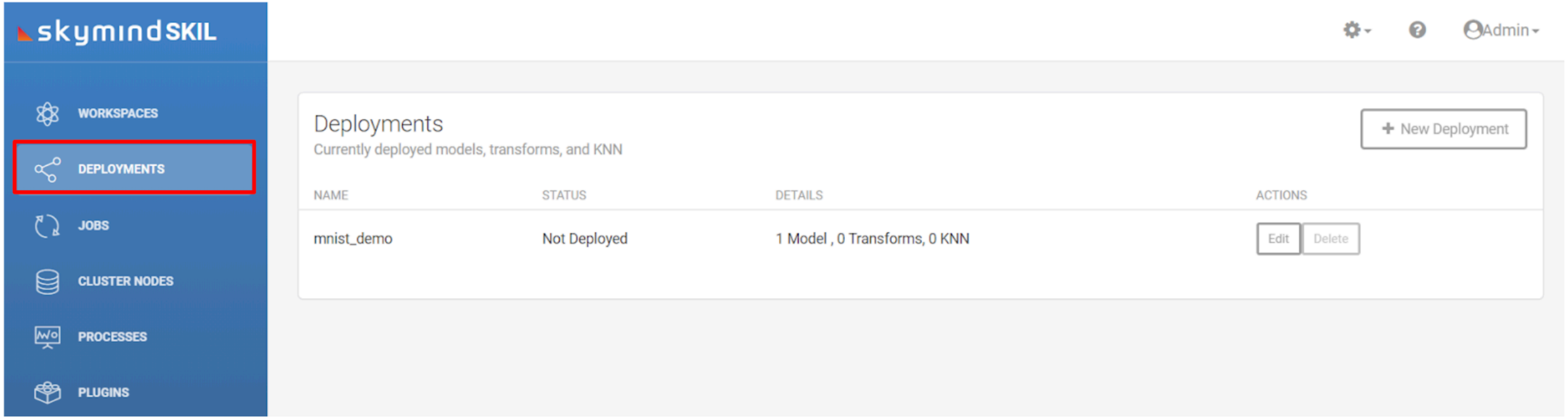
- **Workspaces** are a shared lab to conduct a set of “experiments” centered around a particular project.
- **Experiments** are different configurations of neural net models and data pipelines applied to a given problem



skymind | SKIL User Interface

Deployments

- When mark a model as “deployed”, it serve as **REST API endpoint** to use by client applications
- Endpoint example: *http://[host]:9008/endpoints/deployment/model/kerasmodel/default*



skymind | Key Components of Model Management

SKIL Job System

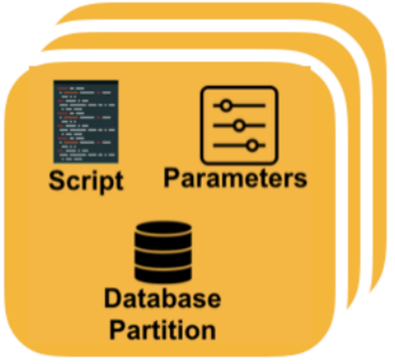
SKIL servers inside a tenant are triggered to run job/scripts with specific parameters on tenant resources.

Model History Server

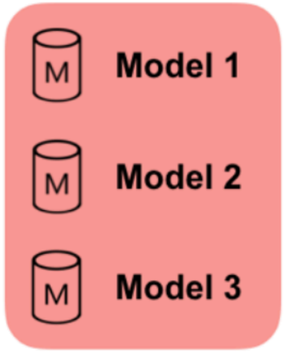
Keeps lists of models with performance results. APIs allow them to be compared to report the best models for deployment.

Deployment System

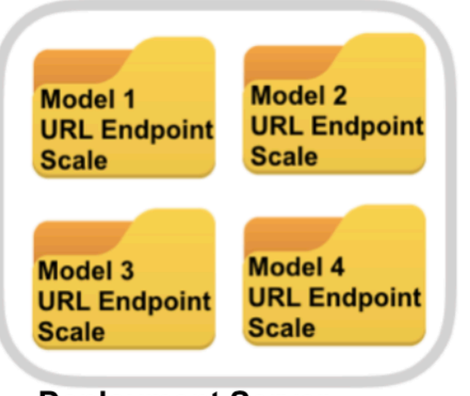
Deployment server handles deployment, scaling and versioning of models.



Jobs



Model History Server



Deployment Server

Best model on real data can Be used with transfer learning To fine tune model with latest data.

Real-time feedback requests are stored back in DB to monitor model performance on real data stream

YOLOのデプロイ

まとめ