

レポート問題（成績評価用）

https://colab.research.google.com/drive/1prdgUlcJAKpabwGtlm_jk1s_7FpbsNm6

農学生命情報科学特論 I

ICT や IoT 等の先端技術を活用し、効率よく高品質生産を可能にするスマート農業への取り組みは世界的に進められています。その基礎を支えている技術の一つがプログラミング言語。なかでも、習得しやすくかつ応用範囲の広い Python がとくに注目されています。本科目では、農学や分子生物学などの分野で利用される Python の最新事例を紹介しながら、Python の基礎文法の講義を行います。

孫 建強 <https://aabbdd.jp/>

農研機構・農業情報研究センター

OCT
03 13:15–16:30

Python 基礎

第 1 回目の授業では、プログラミング言語の基本であるデータ構造とアルゴリズムを簡単に紹介してから、Python の基本構文を紹介する。Python のスカラー、リスト、ディクショナリ、条件構文と繰り返し構文を取り上げる。

OCT
10 13:15–16:30

テキストデータ処理

バイオインフォマティクスの分野において、塩基配列やアミノ酸配列などの文字列からなるデータを扱うことが多い。第 2 回目の授業では、Python を利用した文字列処理を紹介し、FASTA や GFF などのファイルから情報を抽出する方法を取り上げる。

OCT
17 13:15–16:30

データ分析

第 3 回目の授業では、Python ライブラリー (NumPy や Pandas) を利用して、CSV ファイルの処理などのデータ分析やデータ可視化を中心に取り上げる。

OCT
24 13:15–16:30

スマート農業

Python のライブラリー (PyTorch 等) を利用して、深層学習による物体分類や物体検出モデルを実装する例を示す。

農学生命情報科学特論 I



- 機械学習
- 画像解析・物体認識
- 物体認識モデル実装

教師あり学習

教師あり学習は、訓練データとして特徴量と正解ラベルがペアとして与えられるときに行う学習方法である。機械が特徴量と正解ラベルの関係性を探索し、最適な関数で両者を結びつける。教師あり学習は、分類問題および回帰問題などに応用される。

- ニューラルネットワーク
- ロジスティック回帰
- サポートベクトルマシン (SVM)
- 決定木
- ランダムフォレスト
- k 近傍法
- 線形回帰
- スパース回帰

教師なし学習

教師なし学習は、訓練データとして特徴量のみが与えられるときに行う学習方法である。機械が大量なデータ（特徴量）を解釈し、データに隠されたパターンを抽出して、グループ分けしたりする。教師なし学習は、クラスタリングや次元削減・特徴抽出、外れ値検出などに適用される。











- 階層型クラスタリング
- k-means
- 自己組織化モデル (SOM)
- トピックモデル (pLSI, LDA)
- 主成分分析 (PCA)
- 線形判別分析 (LDA)

強化学習

最適化問題

回帰問題

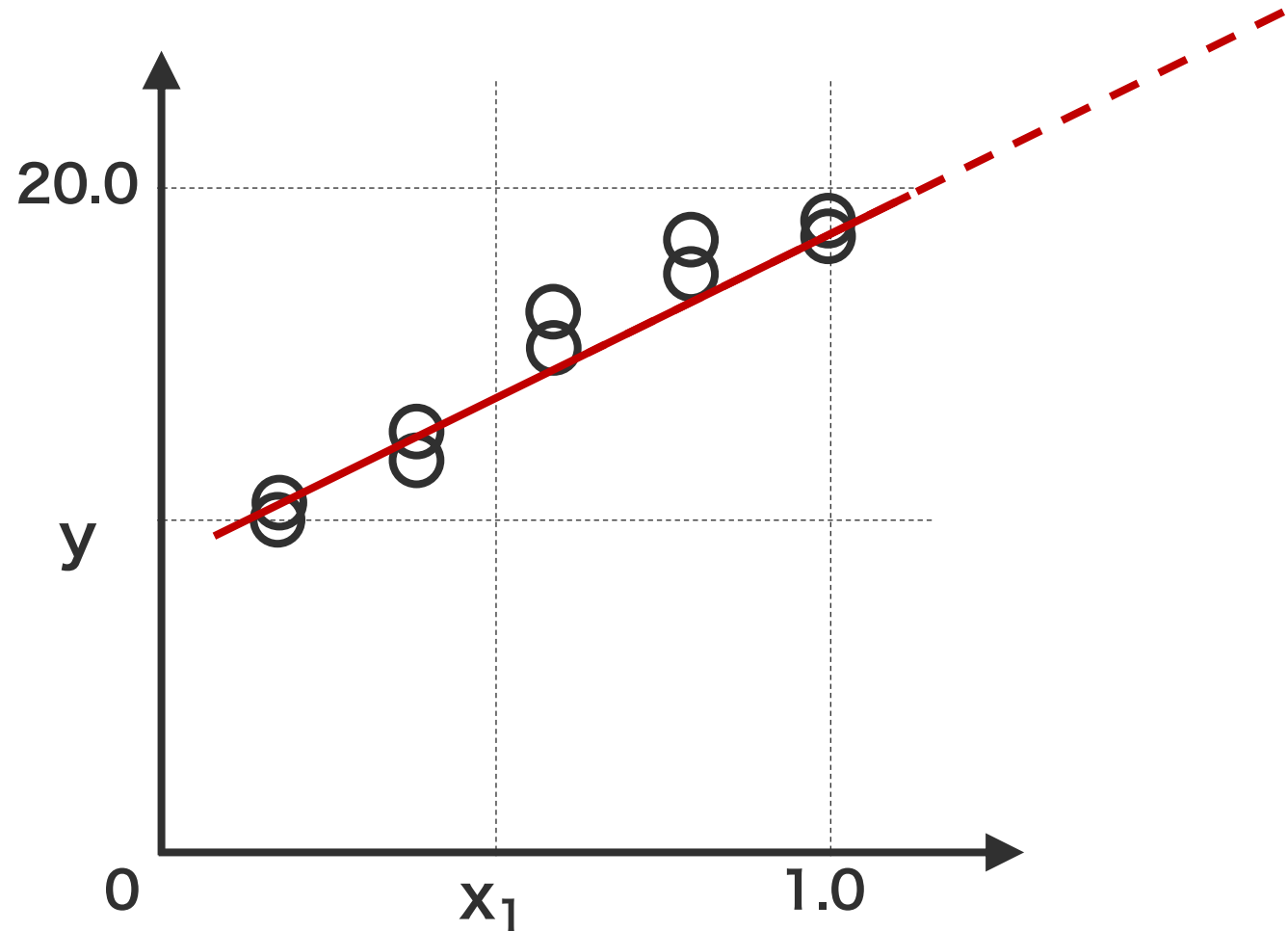
施肥量を利用して収穫量を予測するにはどうすればいいのか？

| 実験区画 | 区画 1 | 区画 2 | 区画 3 | 区画 4 | 区画 5 |
|------|---|--|---|---|---|
| 施肥量 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
| 収穫量 |  10.2 |  12.4 |  16.1 |  17.8 |  18.2 |
| |  10.1 |  11.9 |  17.2 |  18.1 |  18.0 |

回帰問題

施肥量を利用して収穫量を予測するにはどうすればいいのか？

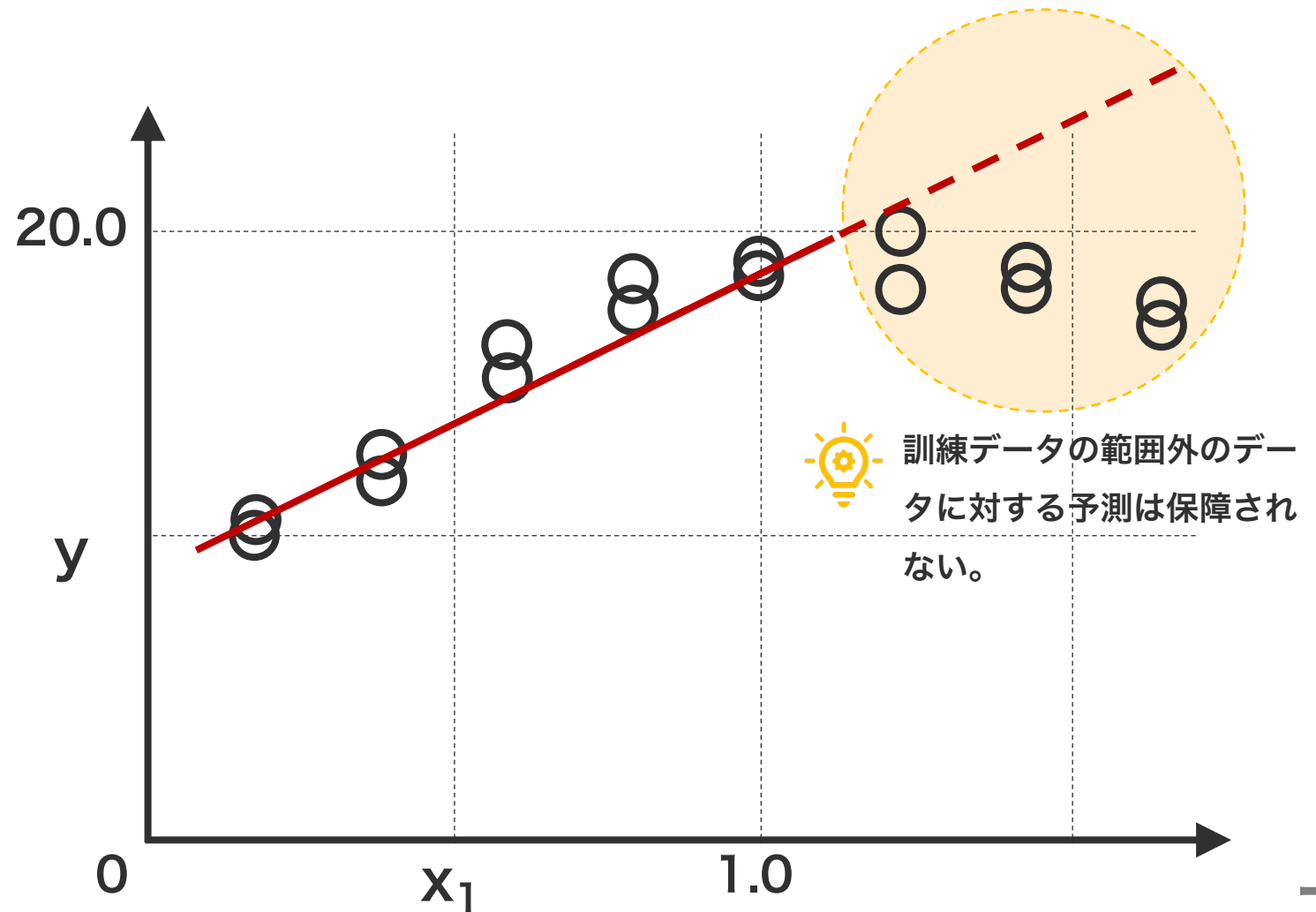
| 収穫量 | 施肥量 |
|------|-----|
| 10.2 | 0.2 |
| 10.1 | 0.2 |
| 12.4 | 0.4 |
| 11.9 | 0.4 |
| 16.1 | 0.6 |
| 17.2 | 0.6 |
| 17.8 | 0.8 |
| 18.1 | 0.8 |
| 18.2 | 1.0 |
| 18.0 | 1.0 |



回帰問題

施肥量を利用して収穫量を予測するにはどうすればいいのか？

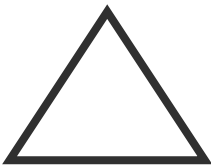
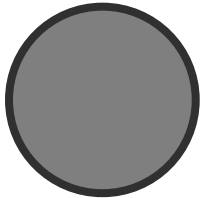

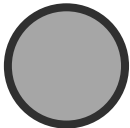
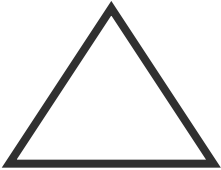
| 収穫量 | 施肥量 |
|------|-----|
| 10.2 | 0.2 |
| 10.1 | 0.2 |
| 12.4 | 0.4 |
| 11.9 | 0.4 |
| 16.1 | 0.6 |
| 17.2 | 0.6 |
| 17.8 | 0.8 |
| 18.1 | 0.8 |
| 18.2 | 1.0 |
| 18.0 | 1.0 |



分類問題

円形と三角形を分類したい場合、どの特徴に着目すればいいのか？

| | | | | | |
|-------|---|---|--|---|---|
| |  |  |  |  |  |
| X_1 | 1.0 | 0.5 | 1.0 | 0.4 | 0.1 |
| X_2 | 0.8 | 1.0 | 0.8 | 0.9 | 1.0 |

| | | | | | |
|-------|--|--|---|--|--|
| |  |  |  |  |  |
| X_1 | 1.0 | 0.3 | 1.0 | 0.5 | 1.0 |
| X_2 | 0.8 | 1.0 | 1.0 | 1.0 | 0.7 |

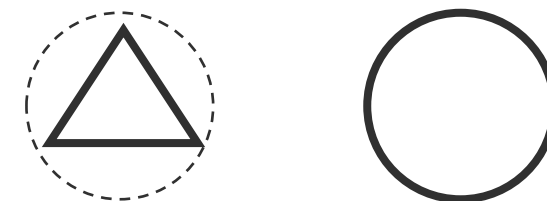
色

色に着目すると、三角形は白が多く、円形は黒い場合が多い。ここで、白を 1、黒を 0 とする色の指標を考える。



外接円との面積比

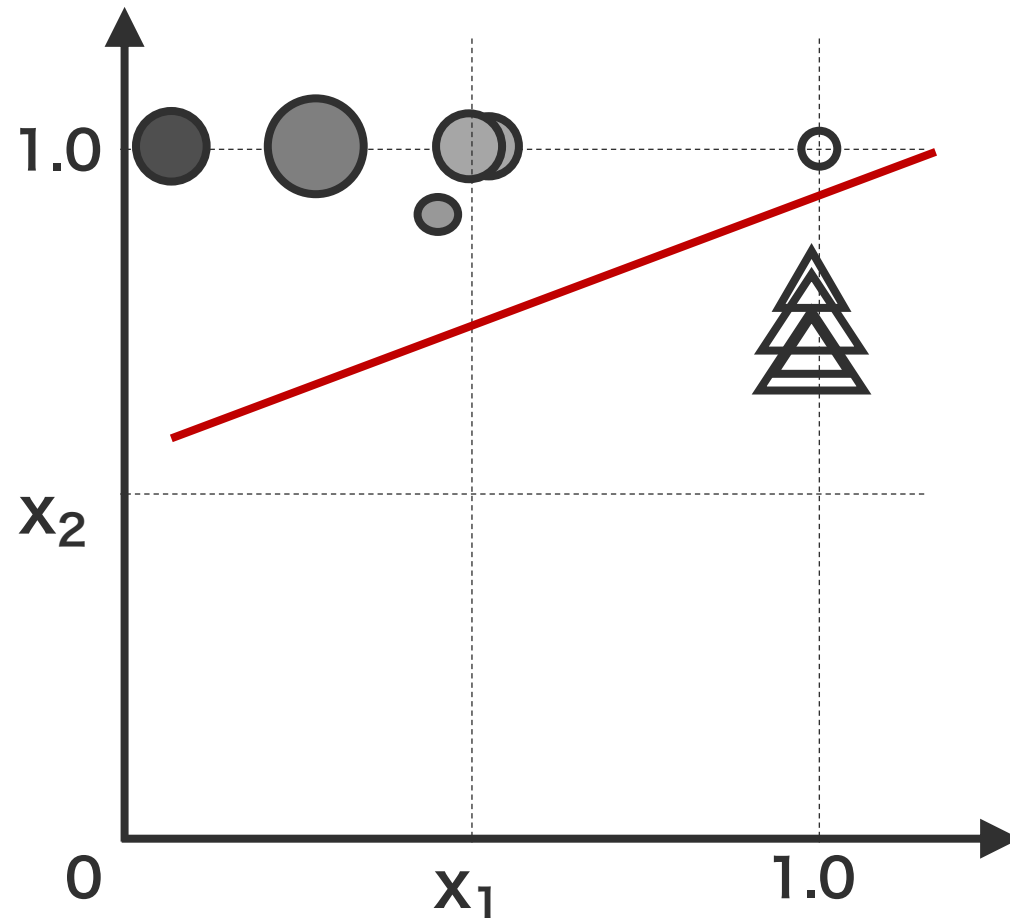
分類したい形とその外接円の面積の比に着目すると、三角形の場合は 1 よりも小さく、円形の場合はほぼ 1 になる。



分類問題

円形と三角形を分類したい場合、どの特徴に着目すればいいのか？

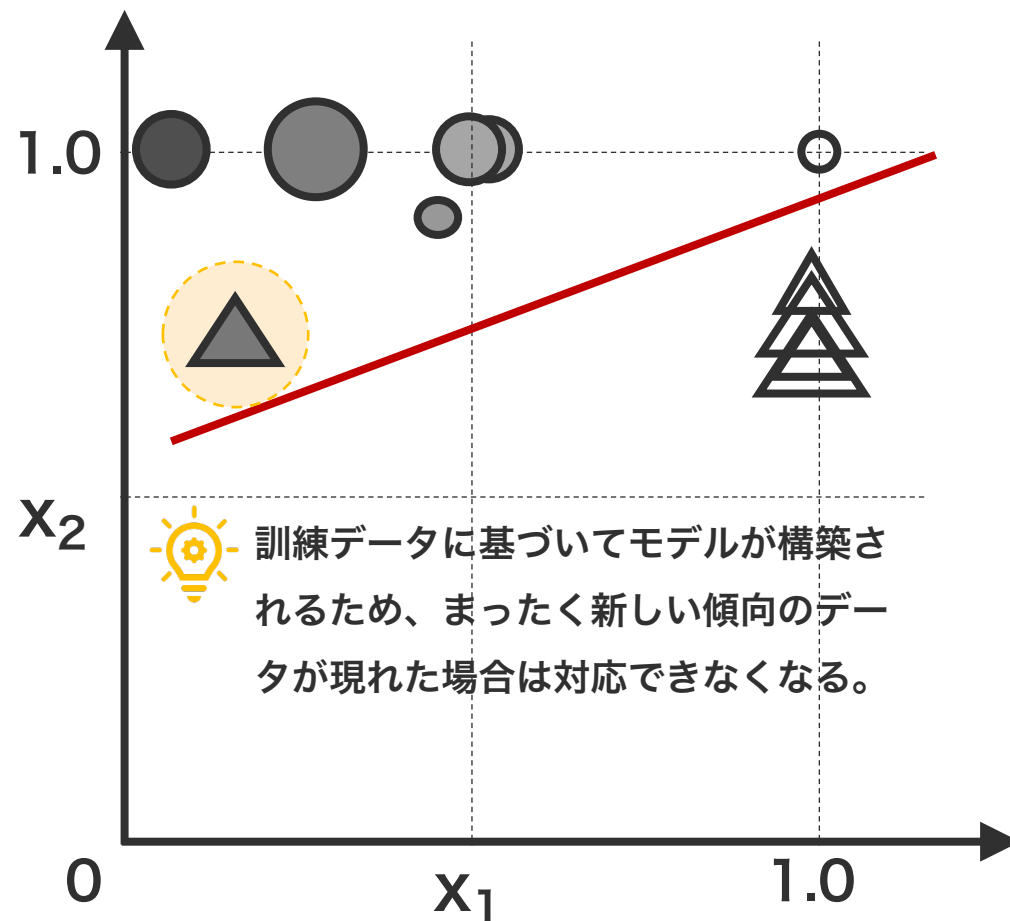
| ラベル | X_1 | X_2 |
|-----|-------|-------|
| 1 | 1.0 | 0.8 |
| 0 | 0.7 | 1.0 |
| 1 | 1.0 | 0.8 |
| 0 | 0.8 | 0.9 |
| 0 | 0.9 | 1.0 |
| 1 | 1.0 | 0.8 |
| 0 | 0.8 | 1.0 |
| 0 | 1.0 | 1.0 |
| 0 | 0.7 | 1.0 |
| 1 | 1.0 | 0.7 |



分類問題

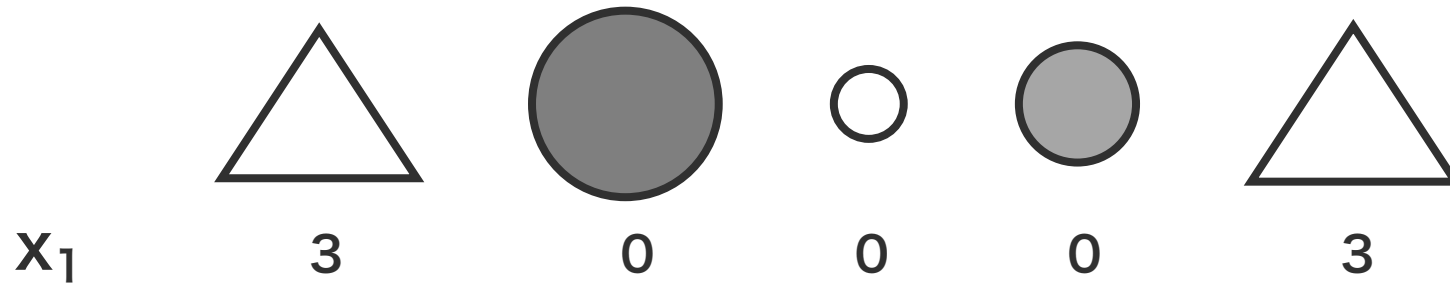
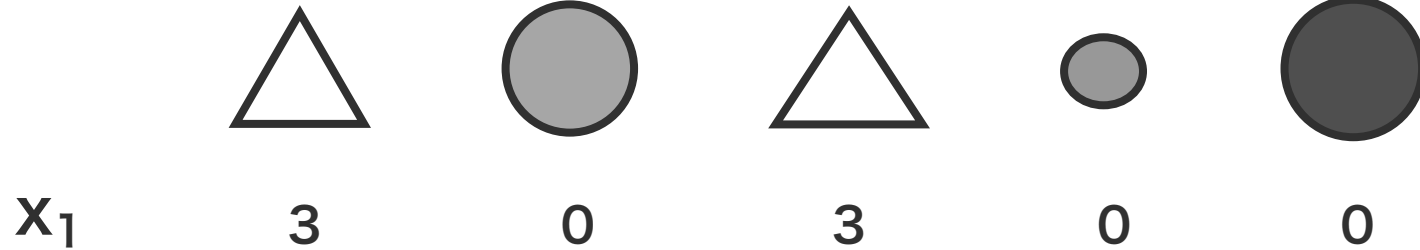
円形と三角形を分類したい場合、どの特徴に着目すればいいのか？

| ラベル | X_1 | X_2 |
|-----|-------|-------|
| 1 | 1.0 | 0.8 |
| 0 | 0.7 | 1.0 |
| 1 | 1.0 | 0.8 |
| 0 | 0.8 | 0.9 |
| 0 | 0.9 | 1.0 |
| 1 | 1.0 | 0.8 |
| 0 | 0.8 | 1.0 |
| 0 | 1.0 | 1.0 |
| 0 | 0.7 | 1.0 |
| 1 | 1.0 | 0.7 |



分類問題

円形と三角形を分類したい場合、どの特徴に着目すればいいのか？



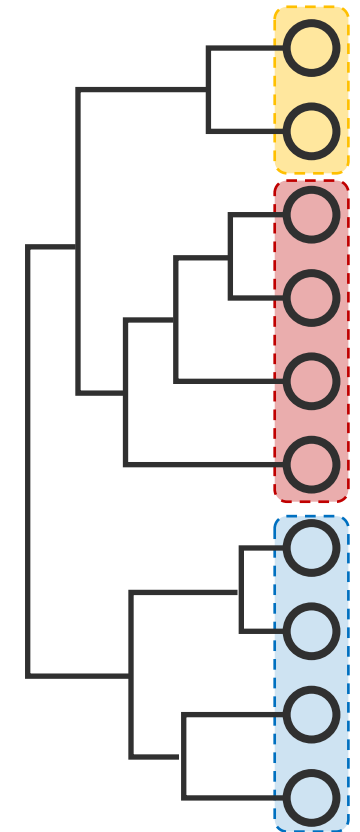
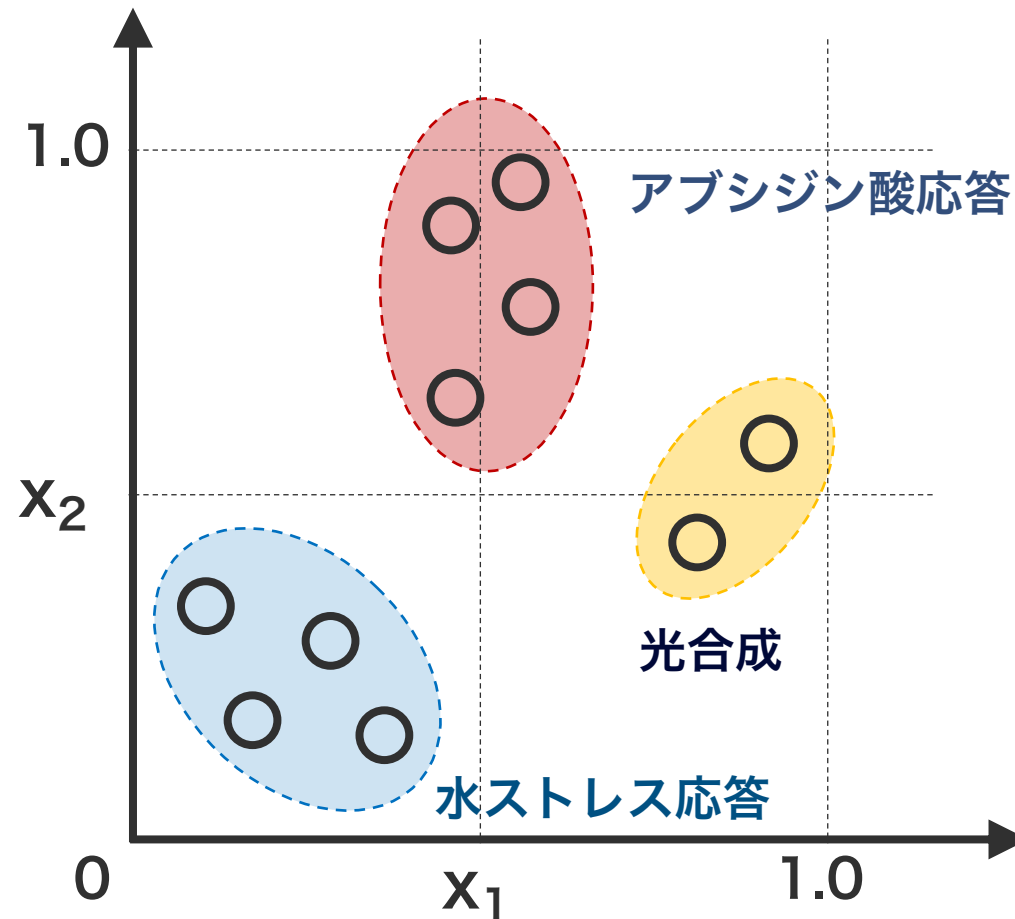
角の数

円形の角は 0 個で、三角形の角の数は 3 個である。そのため、角の数を特徴量として使えば、この 1 つの特徴量だけで円形と三角形を分けられるようになる。

クラスタリング

10 個の遺伝子に対して、乾燥ストレス処理前 X_1 と処理後 X_2 の発現量を測定した。遺伝子間の関係を調べるにはどうすればいいのか？

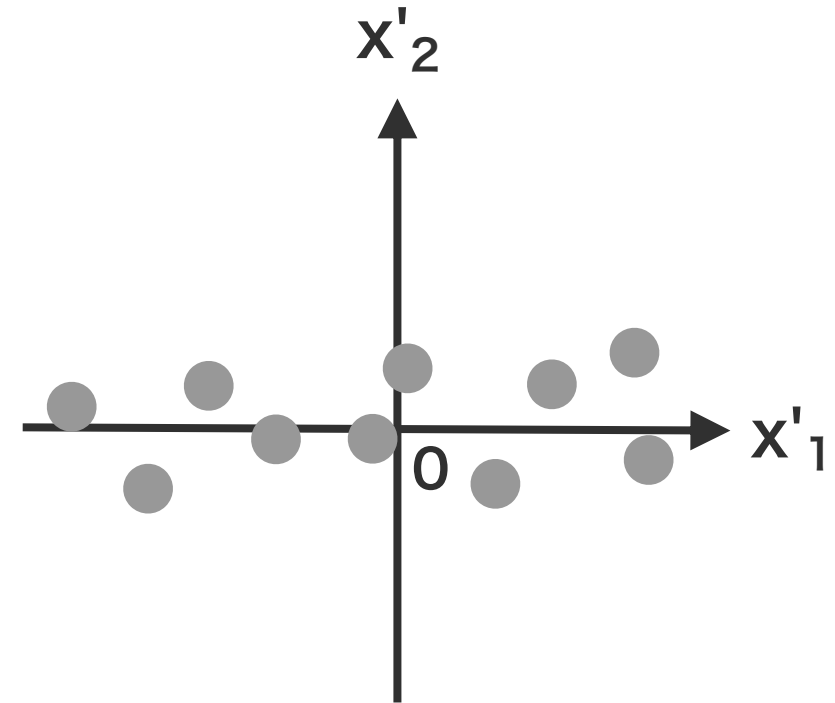
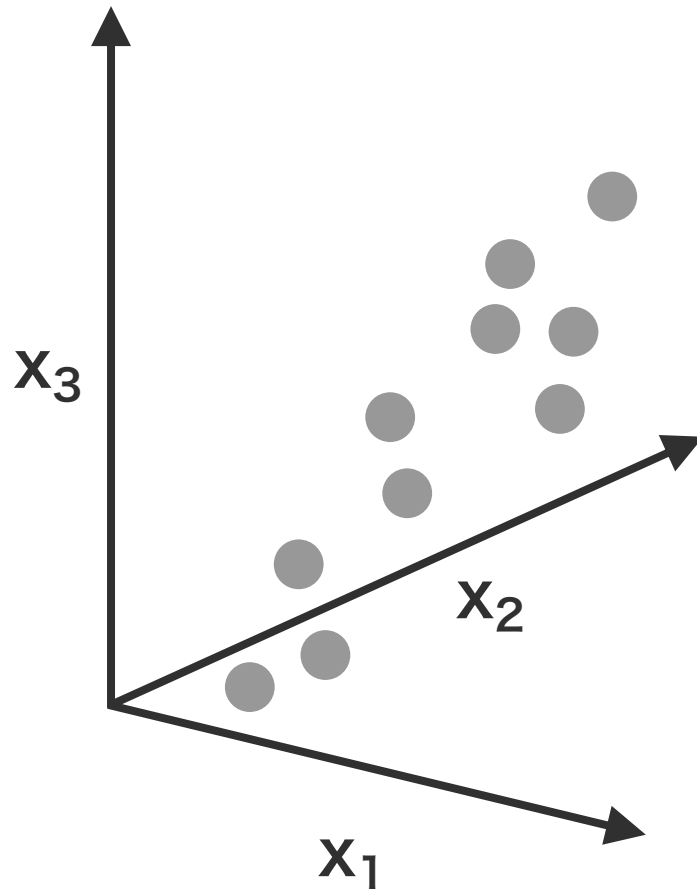
| X_1 | X_2 |
|-------|-------|
| 0.12 | 0.38 |
| 0.92 | 0.57 |
| 0.39 | 0.21 |
| 0.52 | 0.98 |
| 0.49 | 0.88 |
| 0.26 | 0.25 |
| 0.34 | 0.36 |
| 0.49 | 0.61 |
| 0.53 | 0.73 |
| 0.82 | 0.46 |



次元削減・特徴量抽出

多次元の情報を二次元の情報に落として図示したい。

| X_1 | X_2 | X_3 |
|-------|-------|-------|
| 0.12 | 0.38 | 0.54 |
| 0.92 | 0.57 | 0.12 |
| 0.39 | 0.21 | 0.42 |
| 0.52 | 0.98 | 0.85 |
| 0.49 | 0.88 | 0.24 |
| 0.26 | 0.25 | 0.53 |
| 0.34 | 0.36 | 0.87 |
| 0.49 | 0.61 | 0.42 |
| 0.53 | 0.73 | 0.13 |
| 0.82 | 0.46 | 0.56 |



農学生命情報科学特論 I



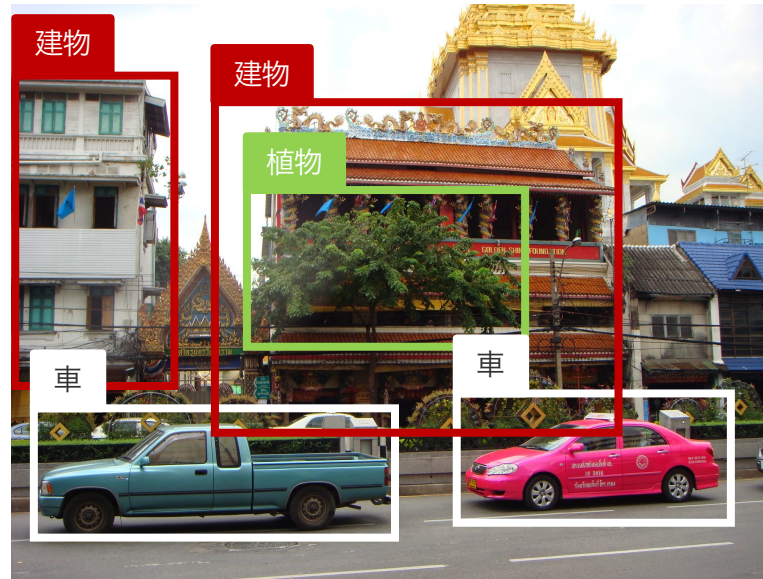
- 機械学習
- 画像解析・物体認識
- 物体認識モデル実装

画像解析

物体分類



物体検出



セグメンテーション



簡単

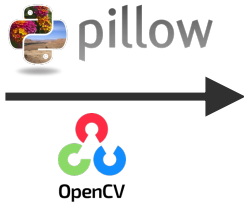
- アルゴリズムが簡単で実装しやすい。
- モデルも簡単で少ない時間で訓練できる。
- 応用範囲が狭い。

複雑

- アルゴリズムが複雑である。
- モデルも複雑で訓練コストがかかる。
- 応用範囲が広い。

物体分類

入力



```
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 009 029 224
025 061 037 103 053 243
031 051 053 081 041 217
093 049 058 109 036 236
054 101 061 002 104 216
109 025 026 105 022 220
111 042 024 021 024 231
151 149 123 149 254 221
251 254 255 245 253
254 253 250 249 251
```



PyTorch



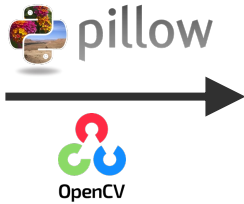
出力

猫

物体分類

入力

出力



```
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 032 004 029 224
025 061 037 193 053 243
031 051 053 081 041 217
093 049 058 092 036 236
054 101 061 024 101 216
109 025 026 105 022 220
111 042 024 021 024 231
151 149 249 249 254 221
251 254 255 245 253
254 253 250 249 251
```

x



PyTorch



猫

y



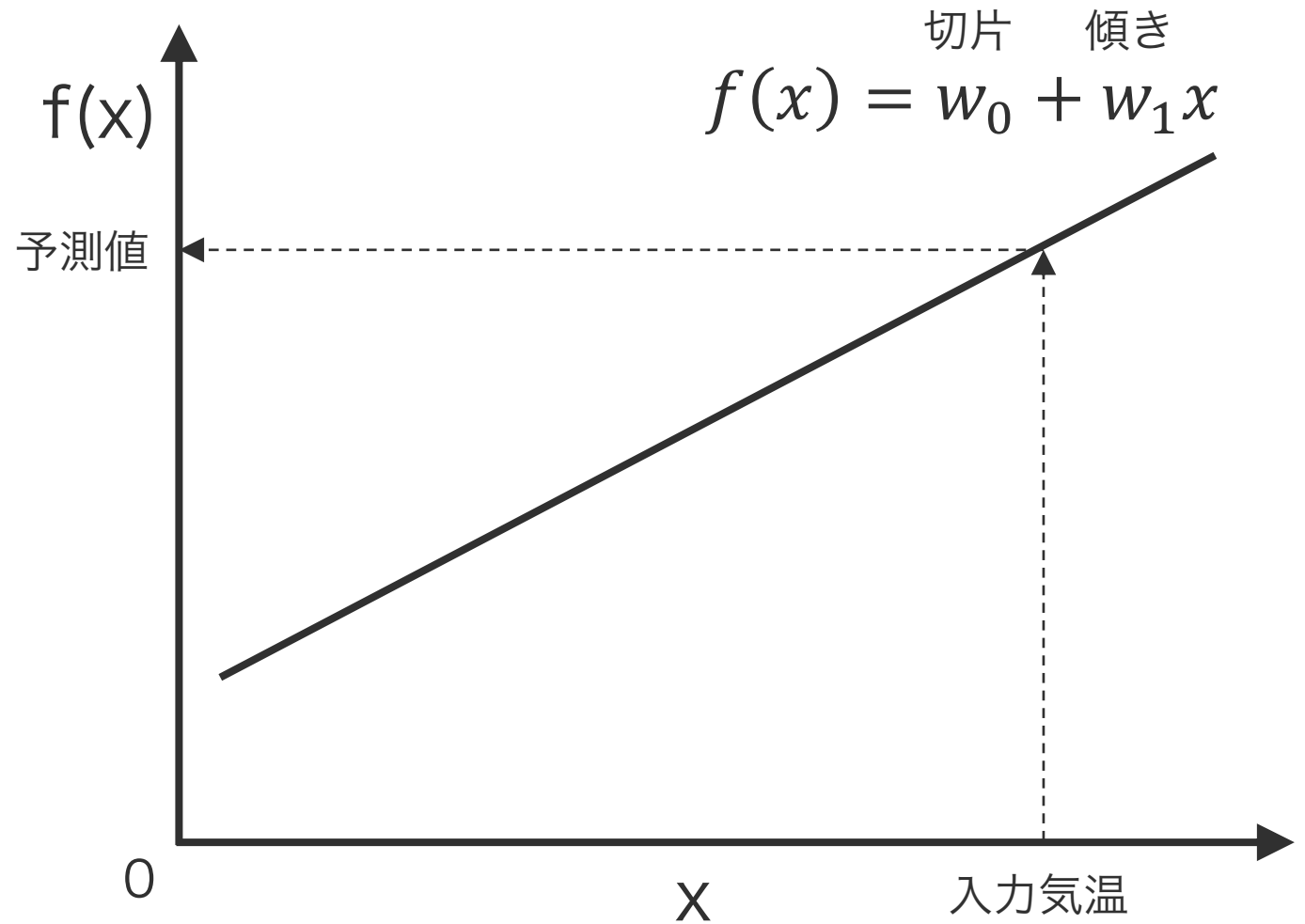
$$y = f(x)$$

関数

ある作物の生産量が気温のみに影響されると仮定する。この仮定のもとで、気温を使って生産量を予測できる。例えば、

$$f(x) = w_0 + w_1 x$$

▲ 生産量 ▲ 気温



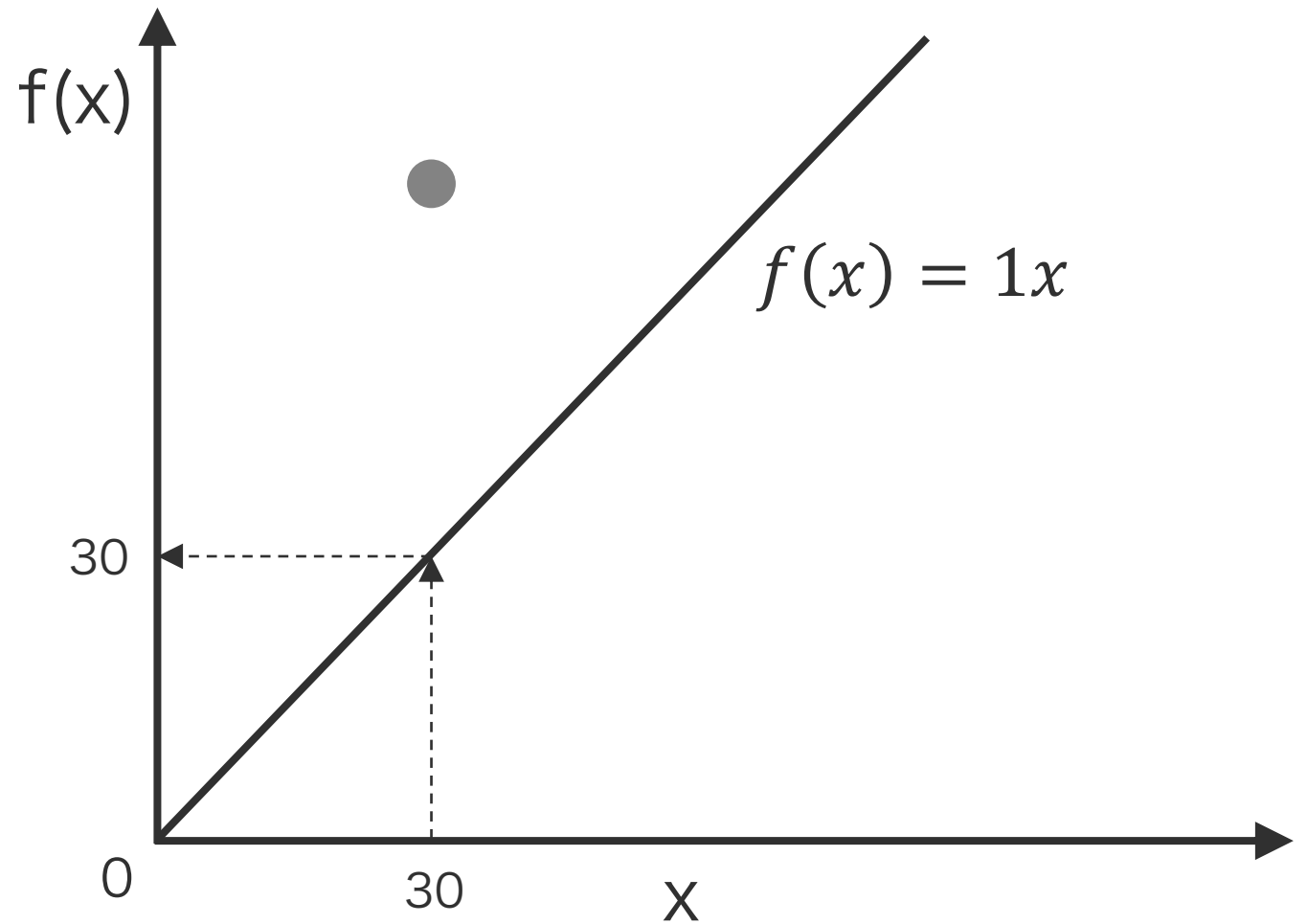
関数

ある作物の生産量が気温のみに影響されると仮定する。この仮定のもとで、気温を使って生産量を予測できる。例えば、

$$f(x) = w_0 + w_1 x$$

▲ ▲
生産量 気温

| 生産量 | 気温 | w_0 | w_1 | 予測値 |
|-----|----|-------|-------|-----|
| 70 | 30 | 0 | 1 | 30 |
| | | | | |
| | | | | |
| | | | | |
| | | | | |



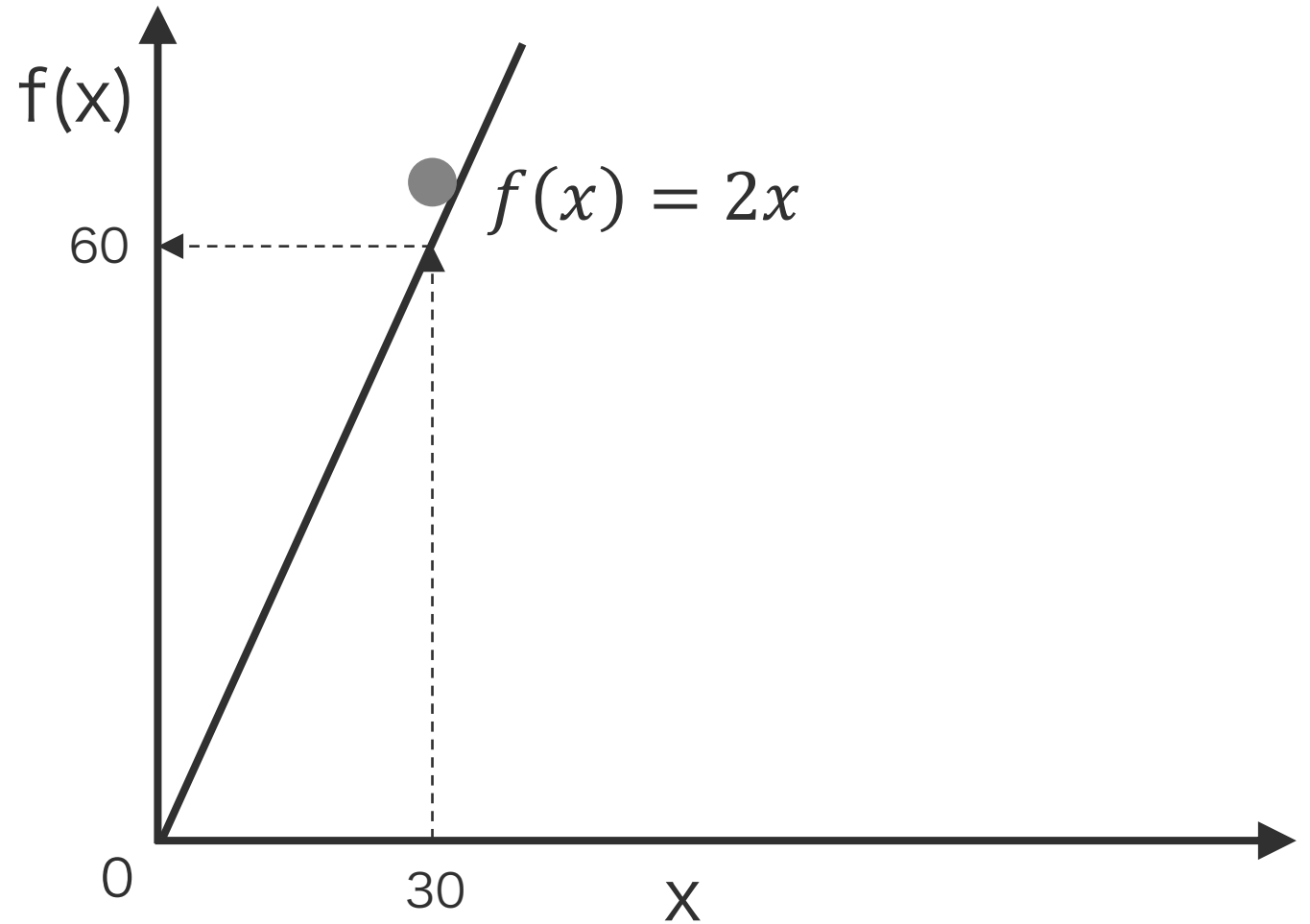
関数

ある作物の生産量が気温のみに影響されると仮定する。この仮定のもとで、気温を使って生産量を予測できる。例えば、

$$f(x) = w_0 + w_1 x$$

▲ ▲
生産量 気温

| 生産量 | 気温 | w_0 | w_1 | 予測値 |
|-----|----|-------|-------|-----|
| 70 | 30 | 0 | 1 | 30 |
| | | 0 | 2 | 60 |
| | | | | |
| | | | | |
| | | | | |



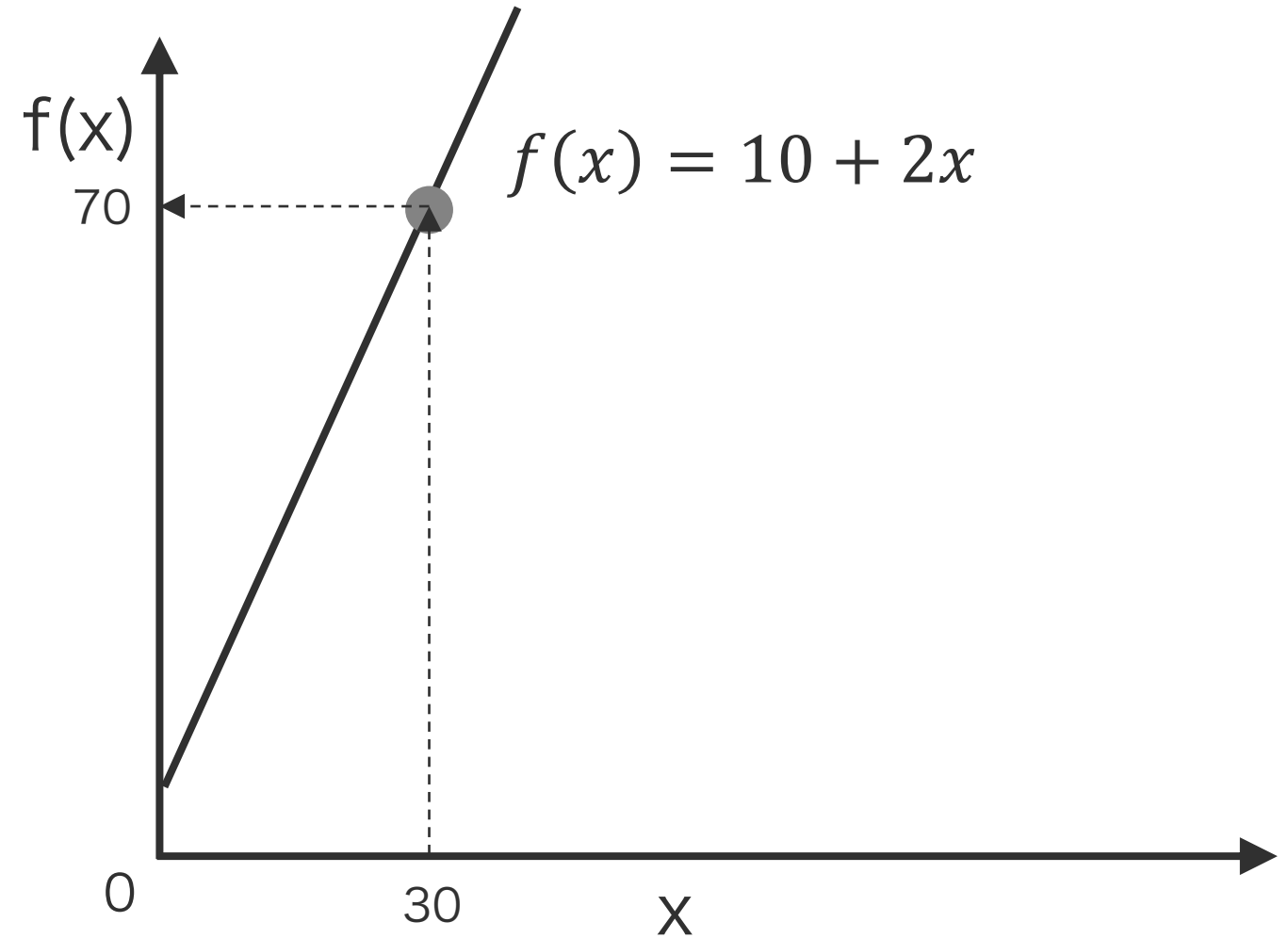
関数

ある作物の生産量が気温のみに影響されると仮定する。この仮定のもとで、気温を使って生産量を予測できる。例えば、

$$f(x) = w_0 + w_1 x$$

▲ ▲
生産量 気温

| 生産量 | 気温 | w_0 | w_1 | 予測値 |
|-----|----|-------|-------|-----|
| 70 | 30 | 0 | 1 | 30 |
| | | 0 | 2 | 60 |
| | | 10 | 2 | 70 |
| | | | | |
| | | | | |



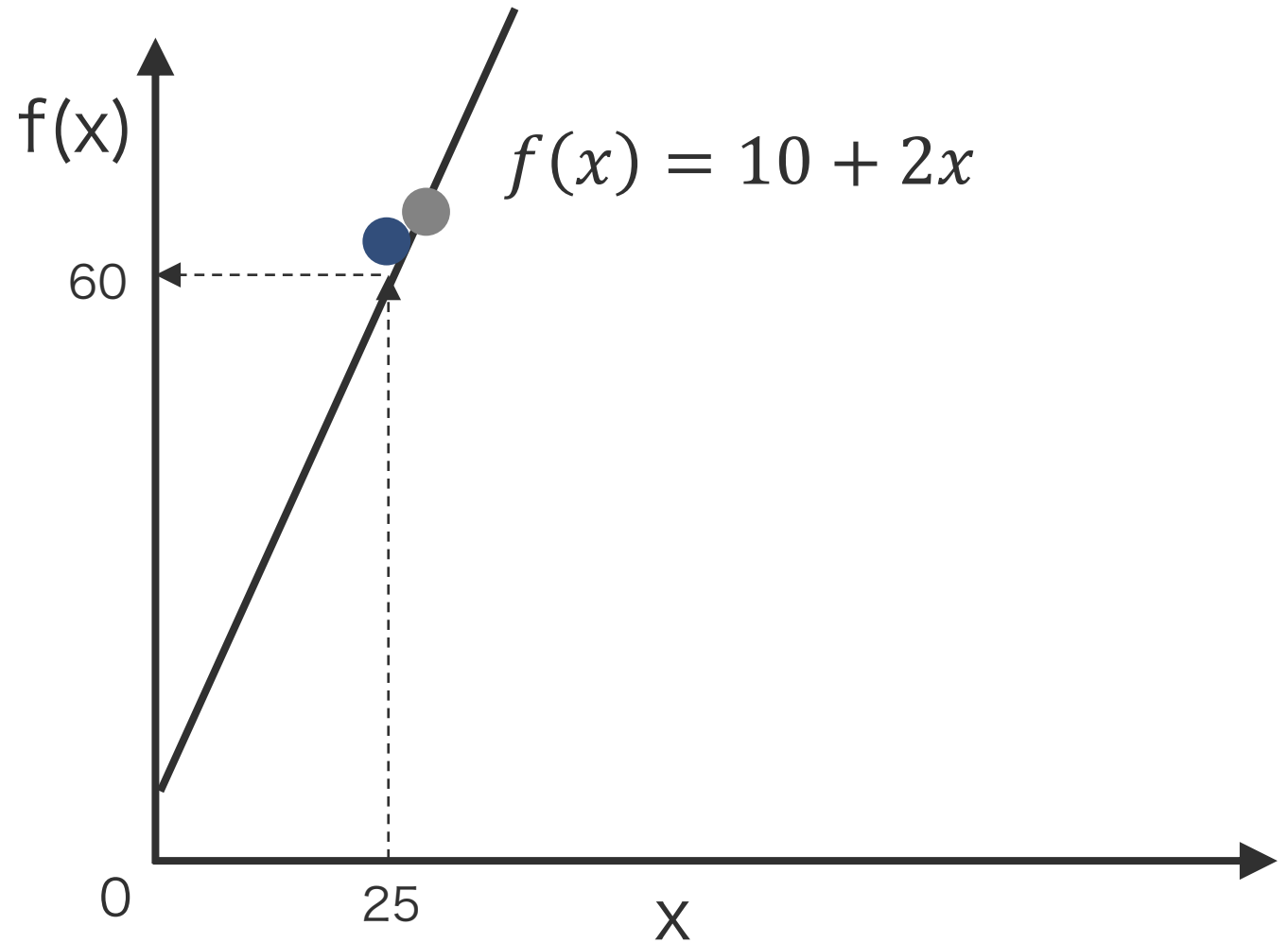
関数

ある作物の生産量が気温のみに影響されると仮定する。この仮定のもとで、気温を使って生産量を予測できる。例えば、

$$f(x) = w_0 + w_1 x$$

▲ ▲
生産量 気温

| 生産量 | 気温 | w_0 | w_1 | 予測値 |
|-----|----|-------|-------|-----|
| 70 | 30 | 0 | 1 | 30 |
| | | 0 | 2 | 60 |
| | | 10 | 2 | 70 |
| 65 | 25 | 10 | 2 | 60 |
| | | | | |



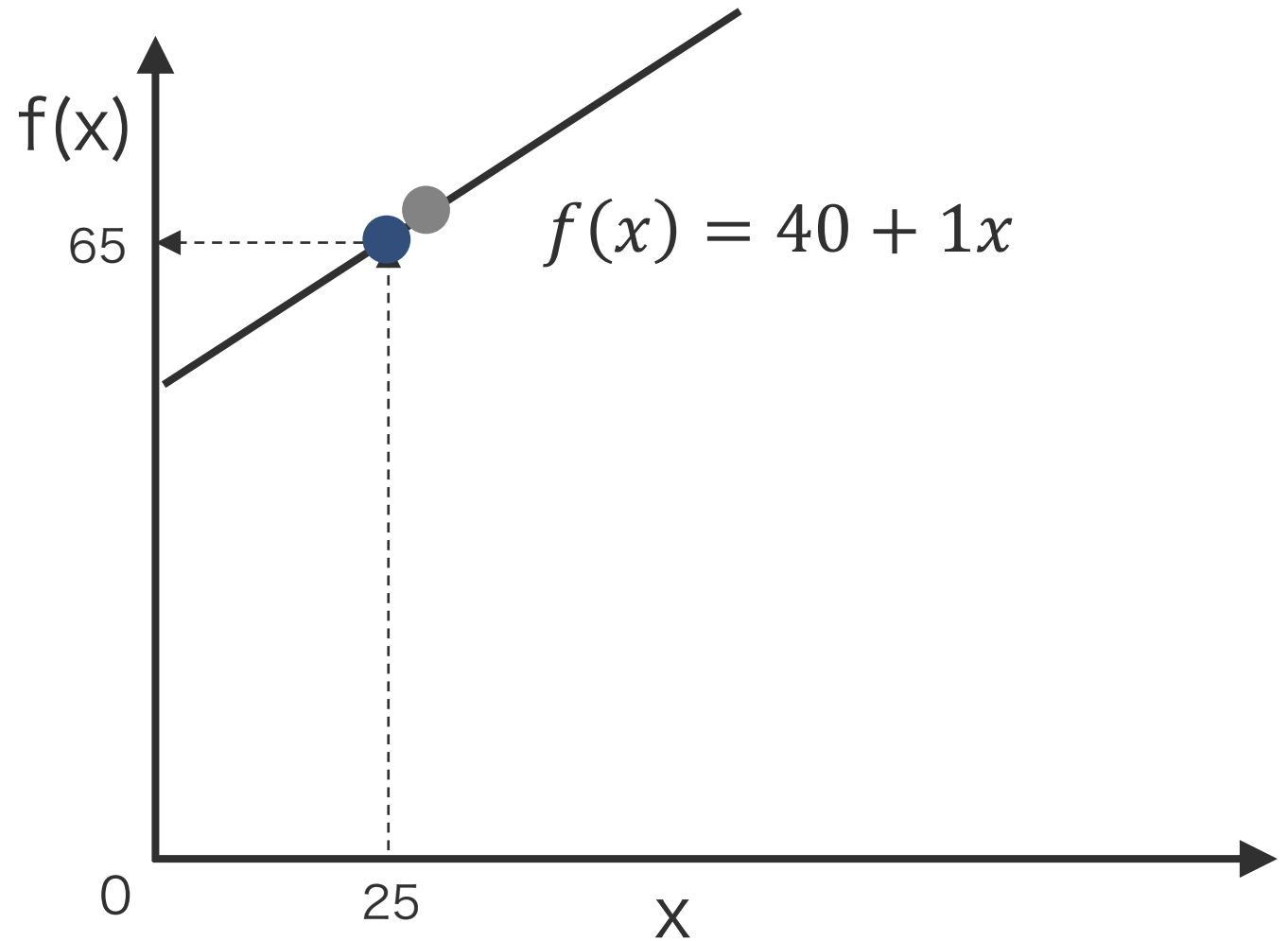
関数

ある作物の生産量が気温のみに影響されると仮定する。この仮定のもとで、気温を使って生産量を予測できる。例えば、

$$f(x) = w_0 + w_1 x$$

▲ ▲
生産量 気温

| 生産量 | 気温 | w_0 | w_1 | 予測値 |
|-----|----|-------|-------|-----|
| 70 | 30 | 0 | 1 | 30 |
| | | 0 | 2 | 60 |
| | | 10 | 2 | 70 |
| 65 | 25 | 10 | 2 | 60 |
| | | 40 | 1 | 65 |



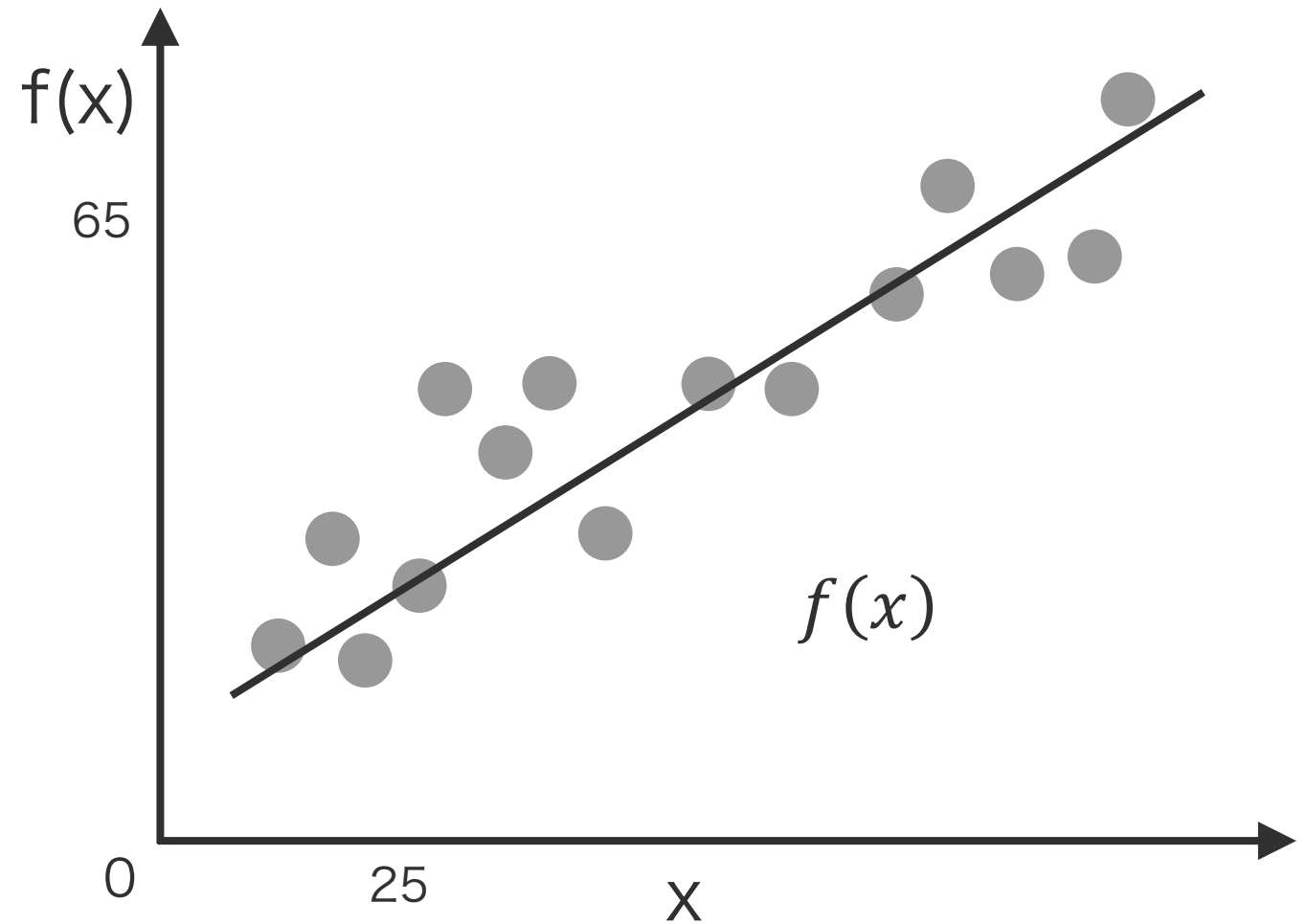
関数

ある作物の生産量が気温のみに影響されると仮定する。この仮定のもとで、気温を使って生産量を予測できる。例えば、

$$f(x) = w_0 + w_1 x$$

▲ 生産量 ▲ 気温

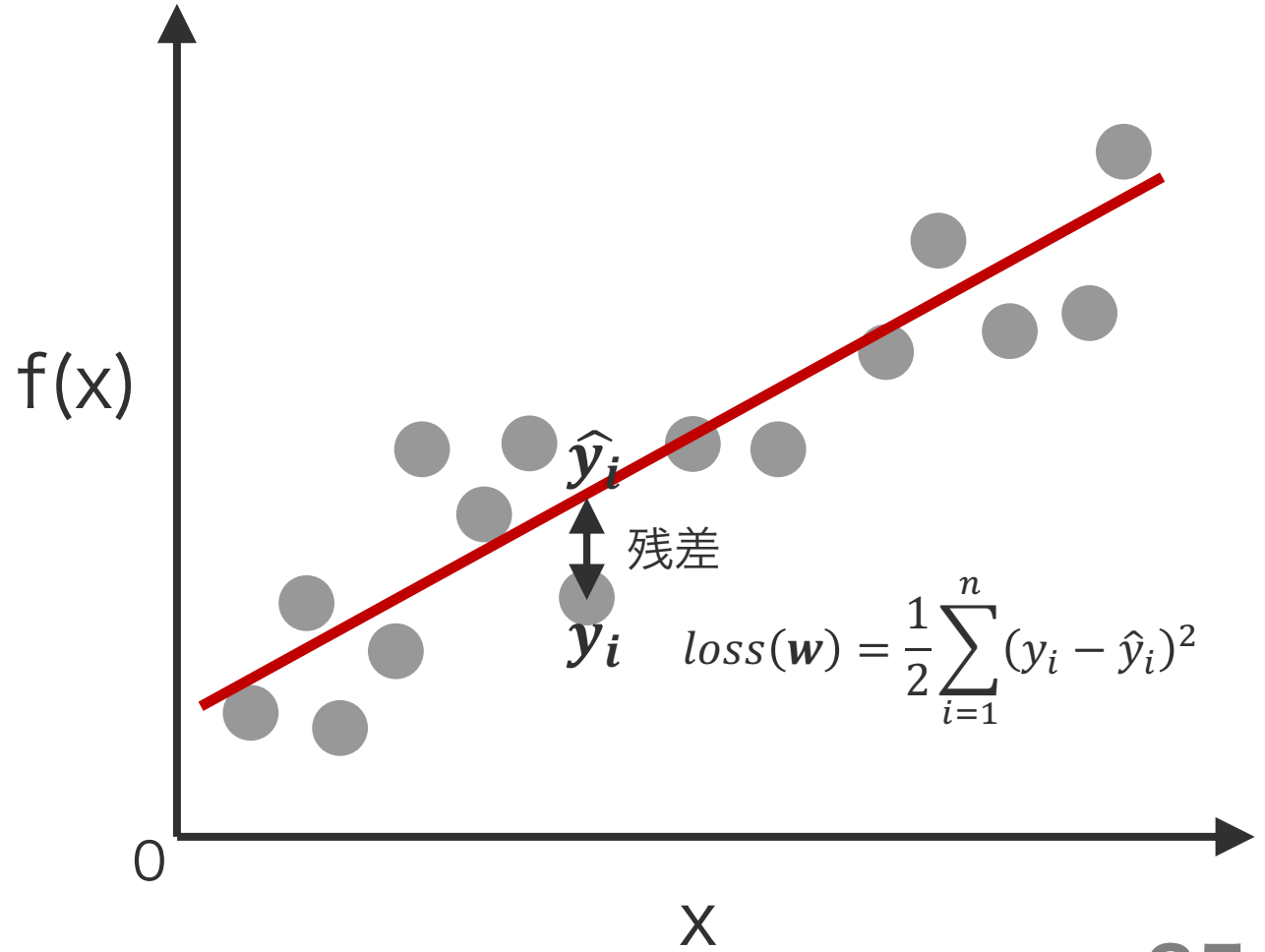
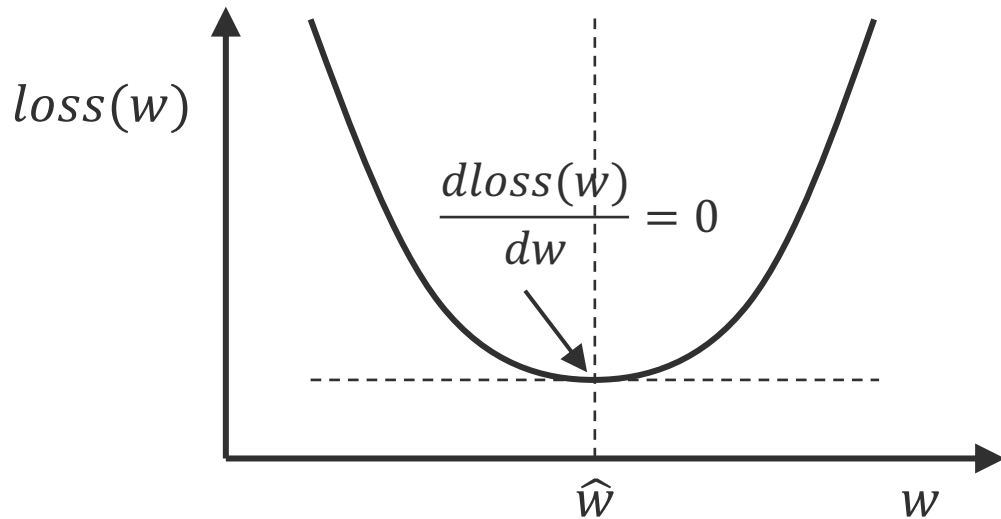
データがたくさんあったら・・・ $f(x)$ の係数をどのようにして決定すればよいのか。



勾配降下法

最適な重み \mathbf{w} をを見つけるには、 $\text{loss}(\mathbf{w})$ を最小にする \mathbf{w} を見つければよい。

$$\text{loss}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{x}_i \mathbf{w})^2$$



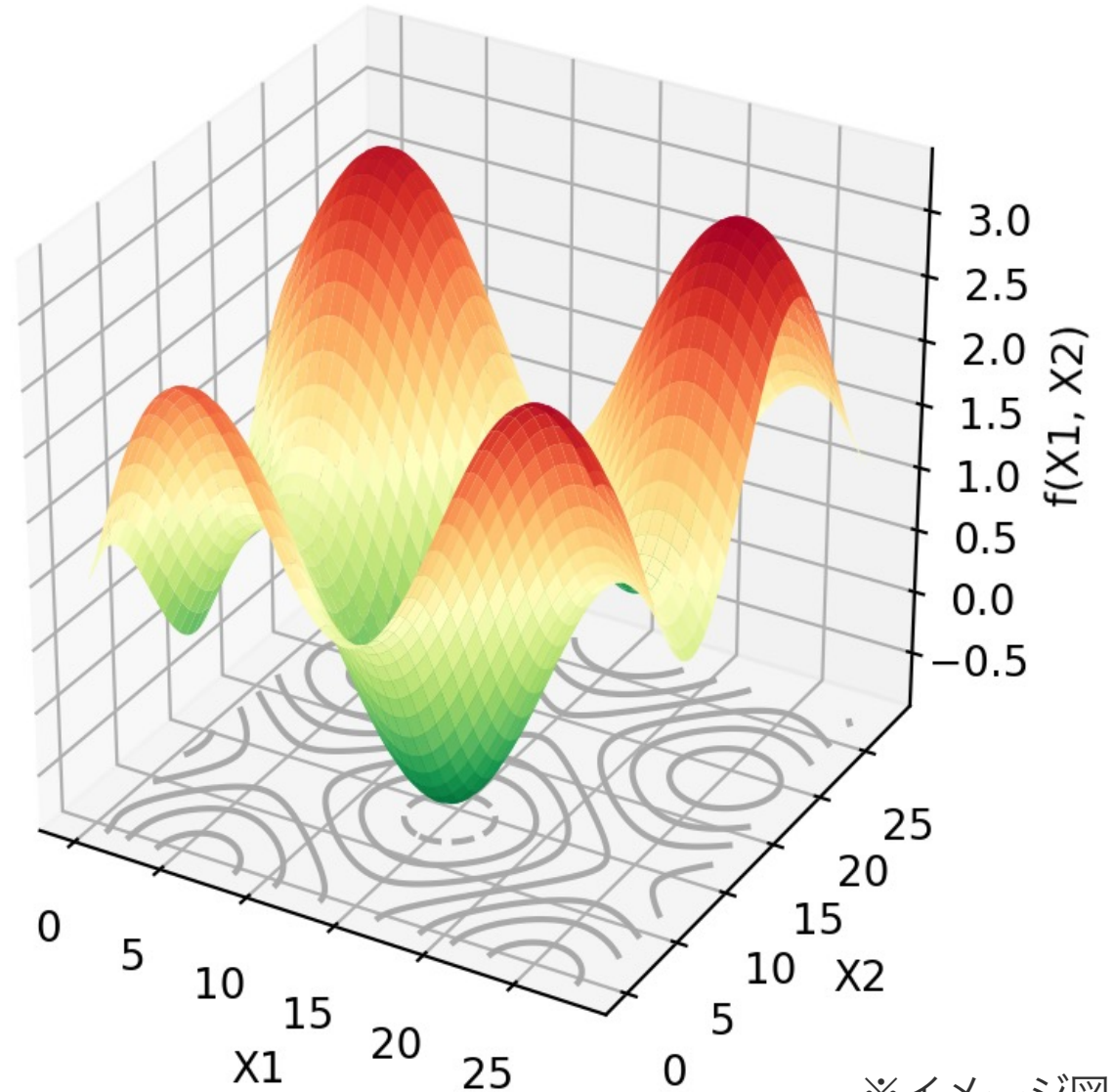
多変量関数

説明変数が多くなると、求める \mathbf{w} も多くなる。
そのため、 $\text{loss}(\mathbf{w})$ 関数の形は複雑になり、最小値を求めるのがますます困難になる。

$$y = w_0 + w_1 x_1 + w_2 x_2$$

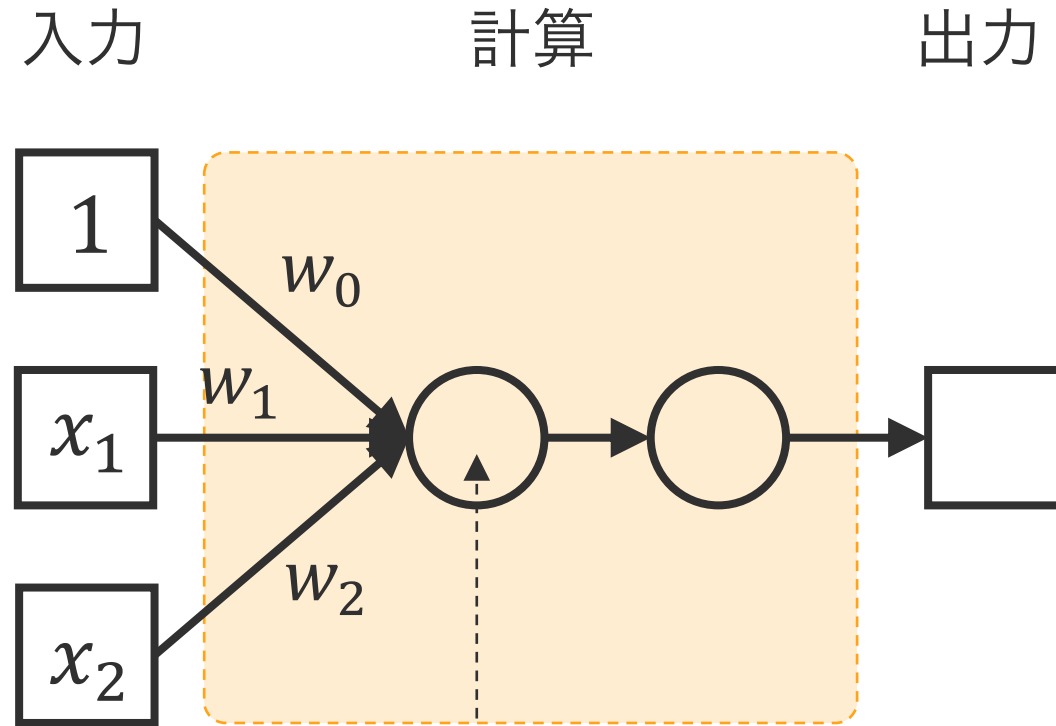
▲ 乾燥重量 ▲ 施肥量 ▲ 気温

$$\text{loss}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{x}_i \mathbf{w})^2$$



※イメージ図

ニューラルネットワーク



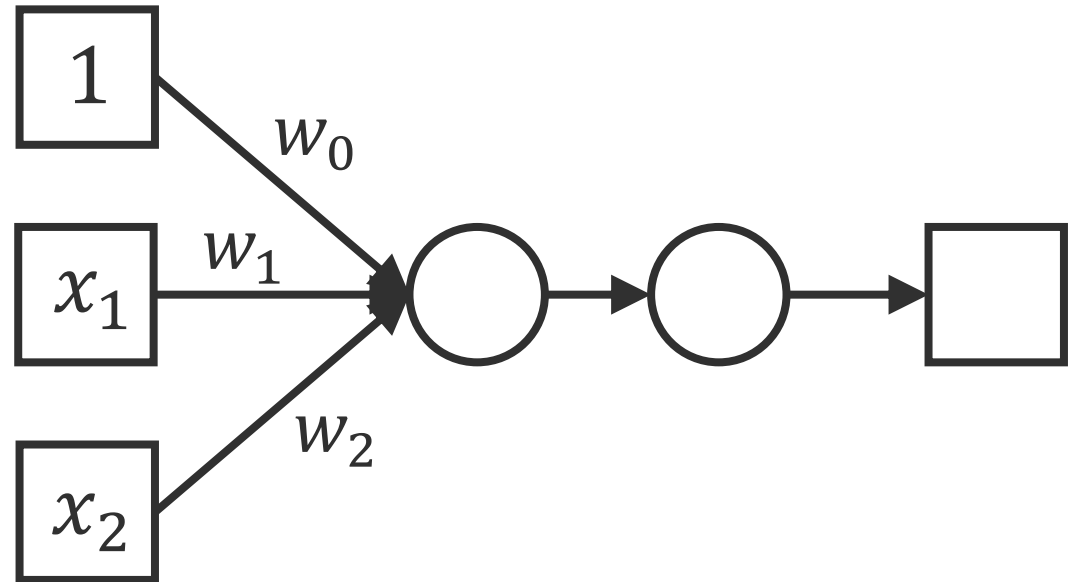
回帰分析と同じ計算を行なっている。

$$f(x) = w_0 + w_1x_1 + w_2x_2$$

ニューラルネットワーク



入力 x_1 x_2

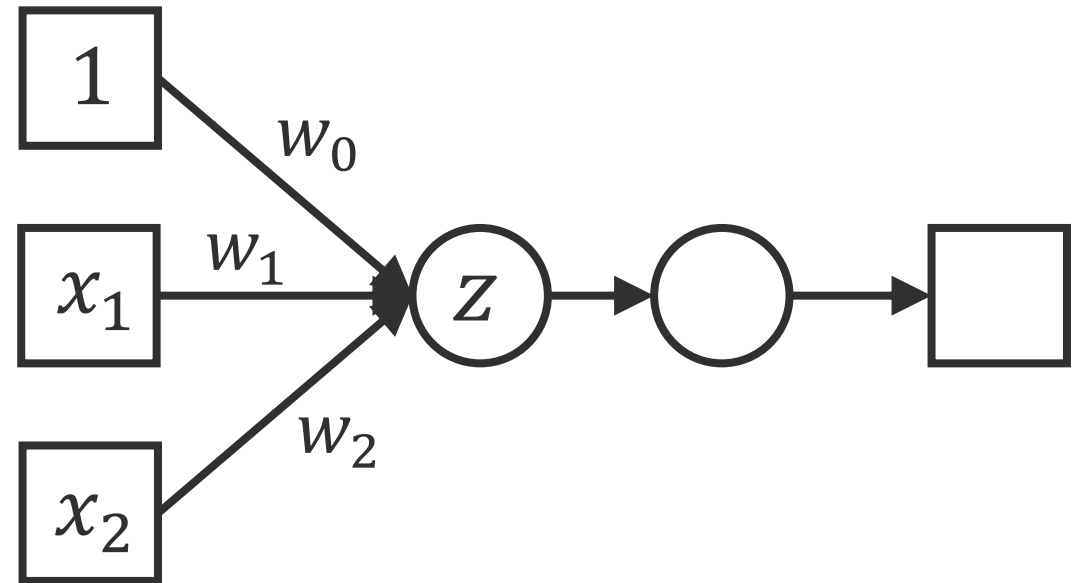


ニューラルネットワーク



入力 x_1 x_2

処理 $z = w_0 + w_1x_1 + w_2x_2$



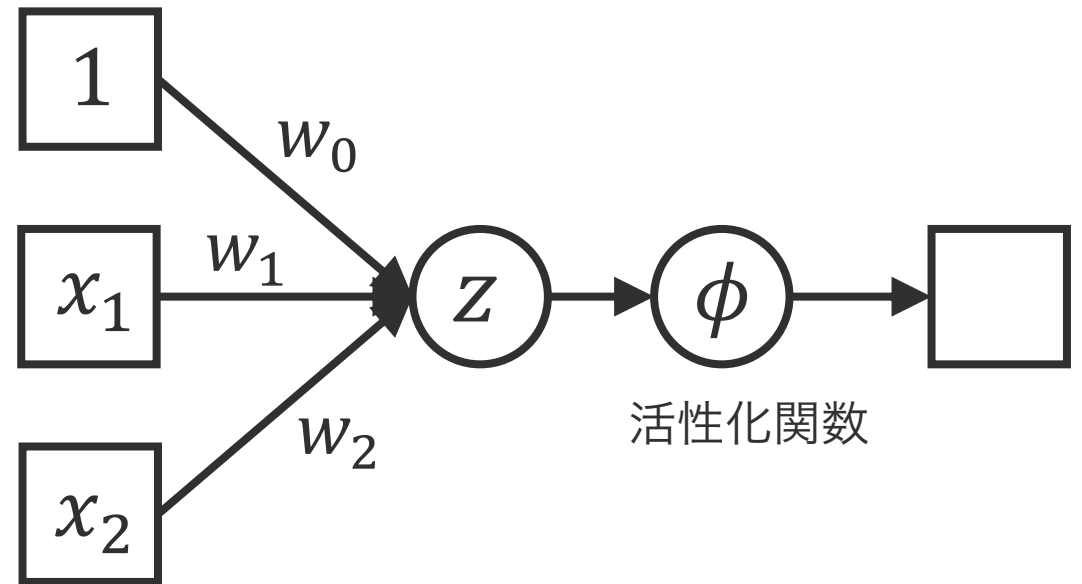
ニューラルネットワーク



入力 x_1 x_2

処理 $z = w_0 + w_1x_1 + w_2x_2$

処理 $\phi(z) = z$



ニューラルネットワーク

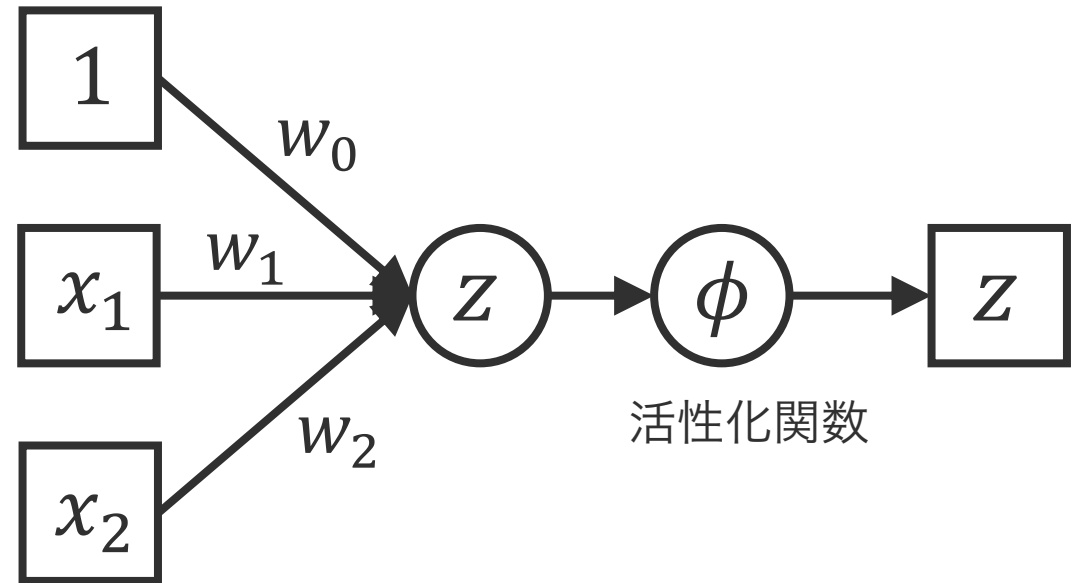


入力 x_1 x_2

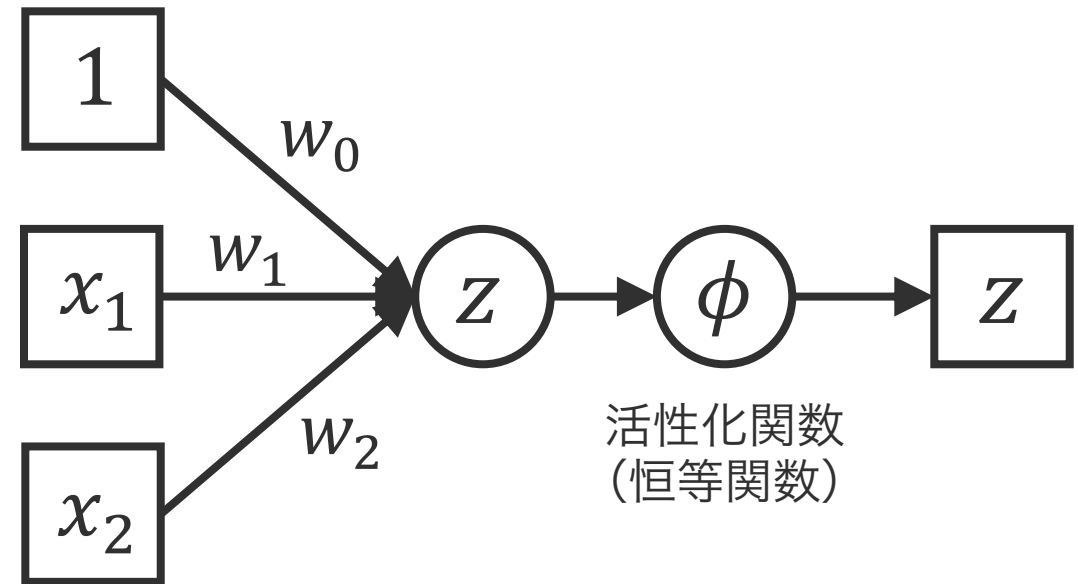
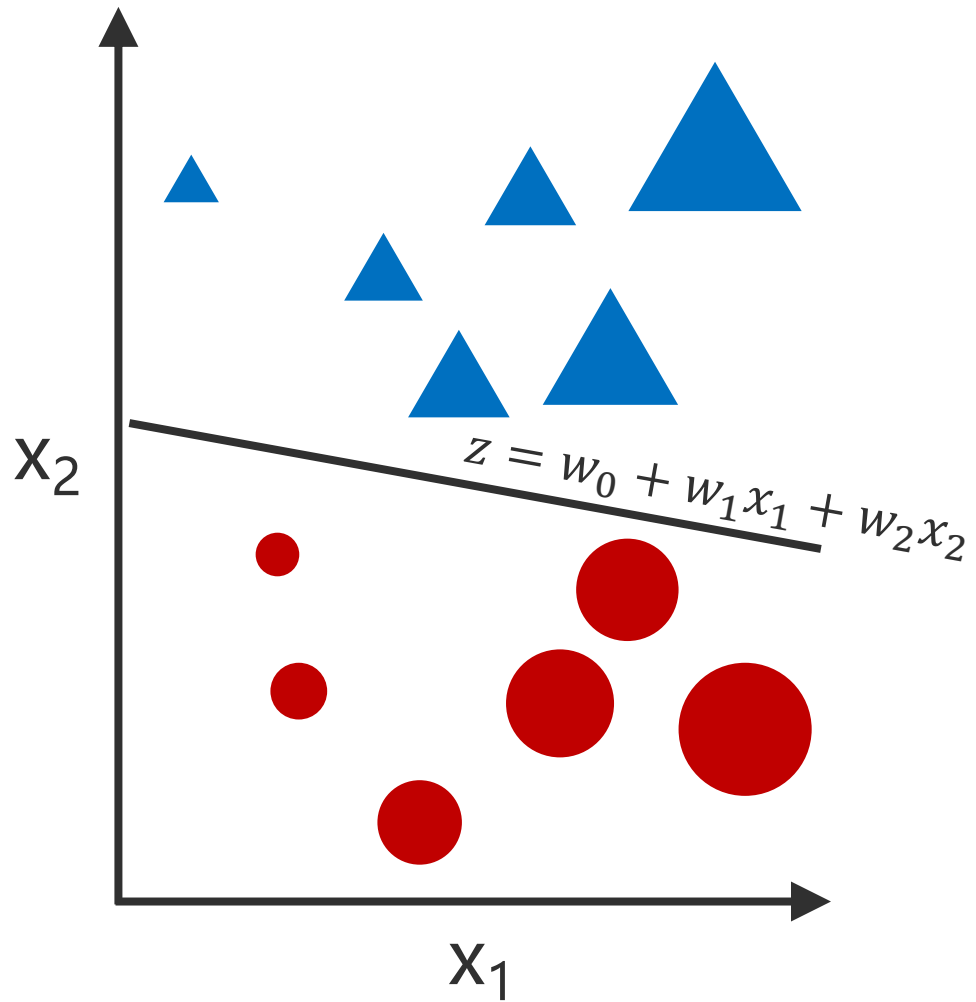
処理 $z = w_0 + w_1x_1 + w_2x_2$

処理 $\phi(z) = z$

出力 z

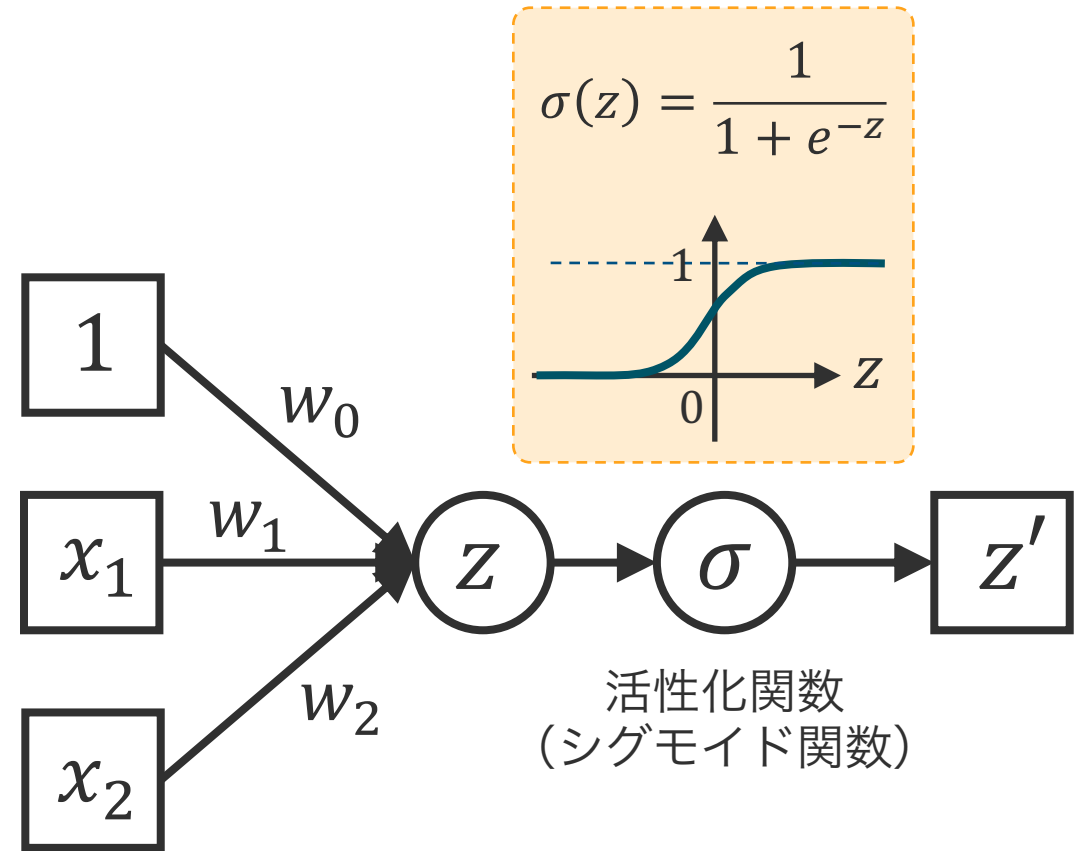
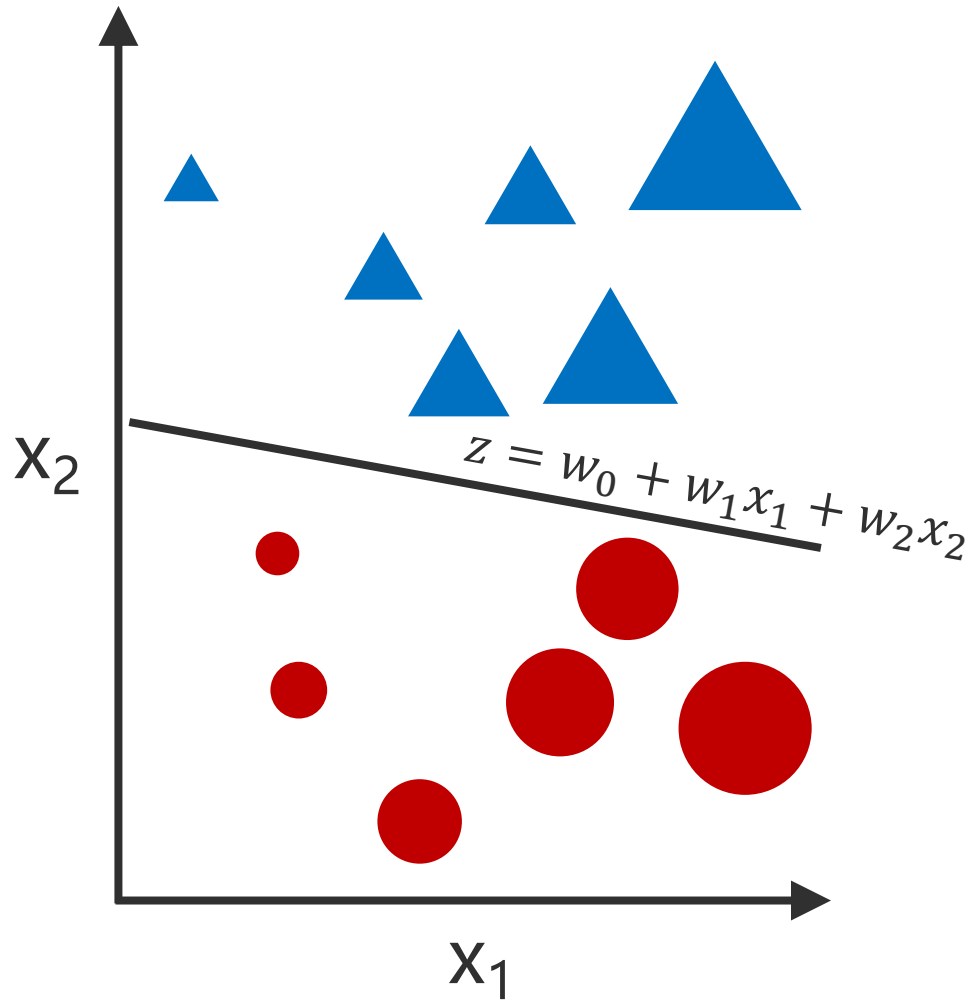


線形分離

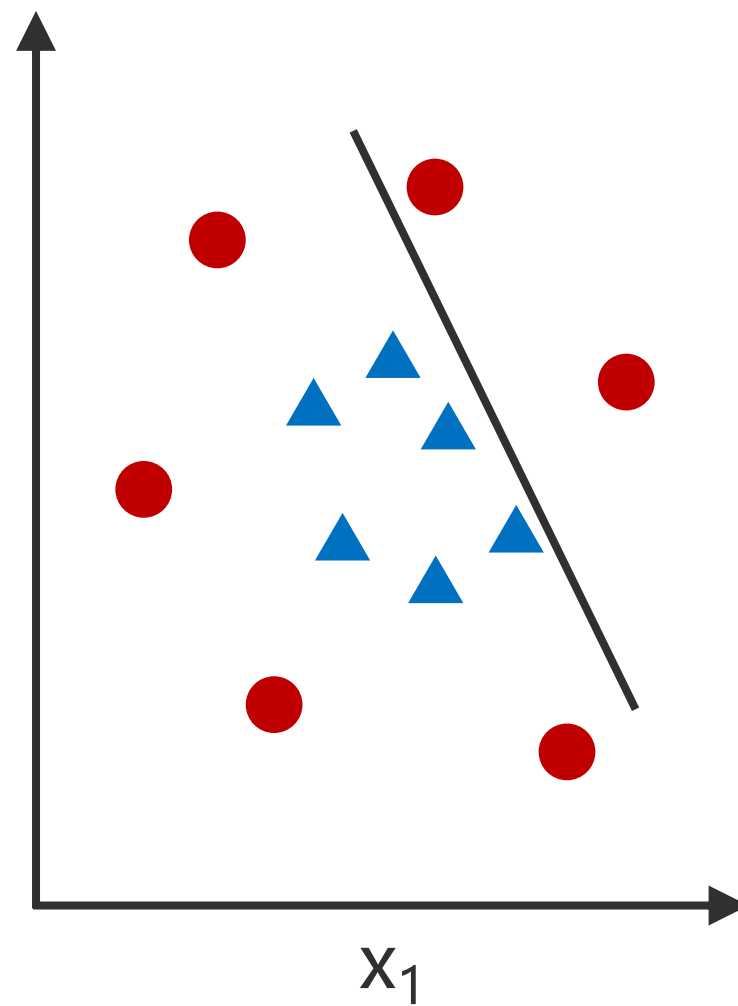
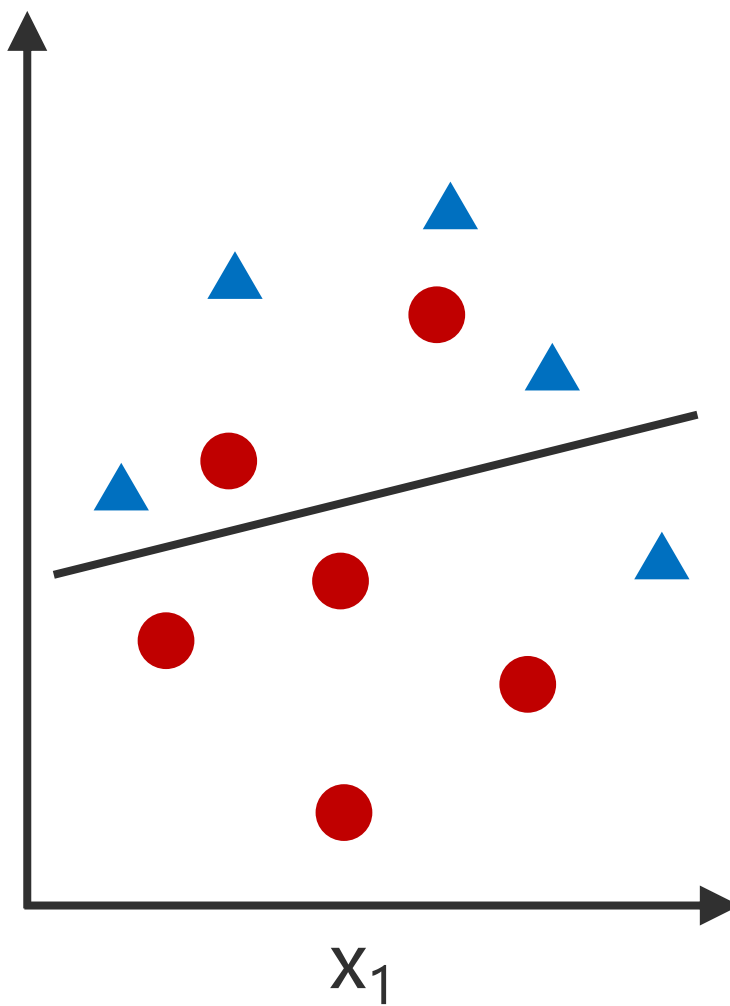
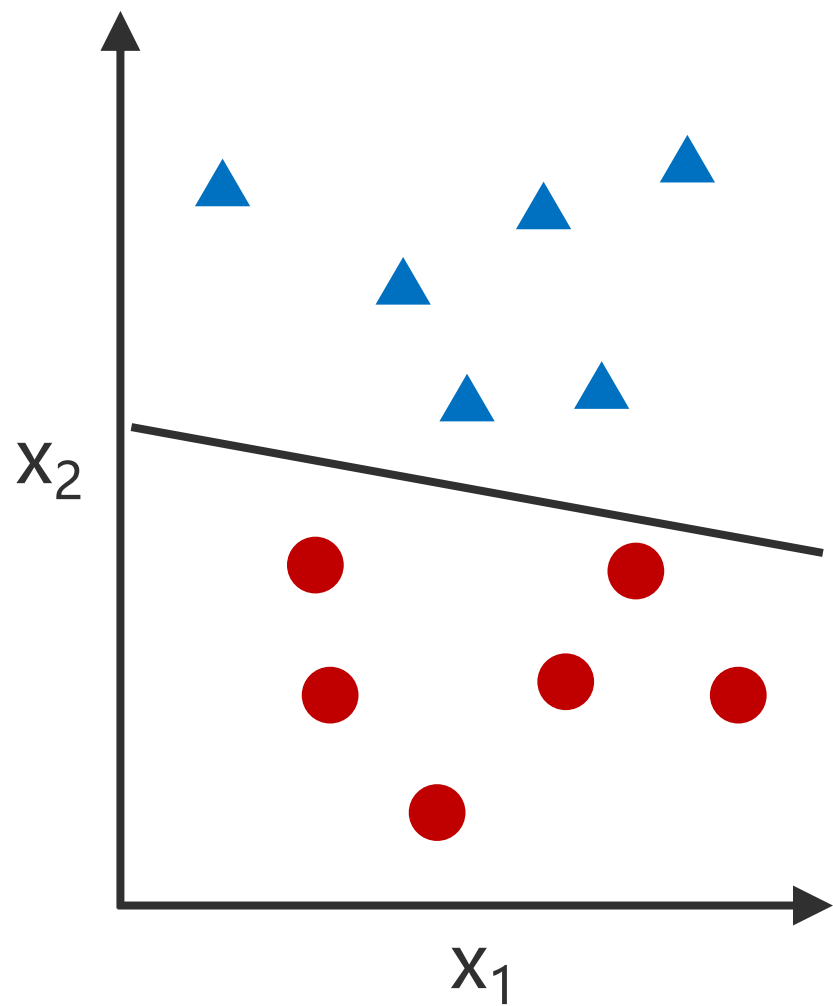


活性化関数
(恒等関数)

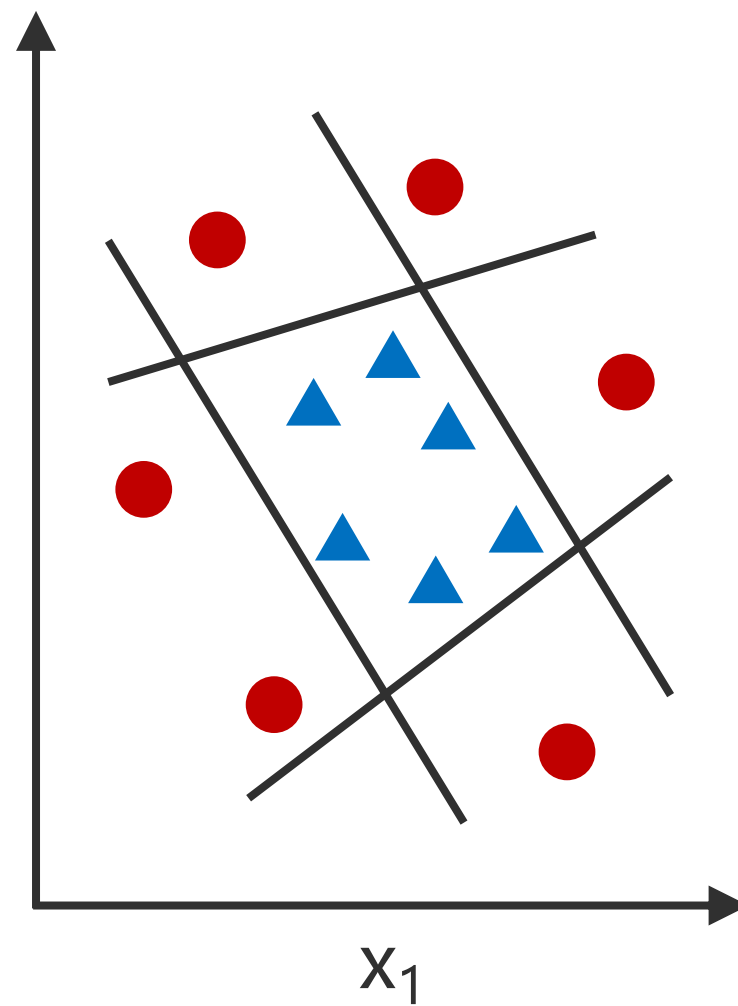
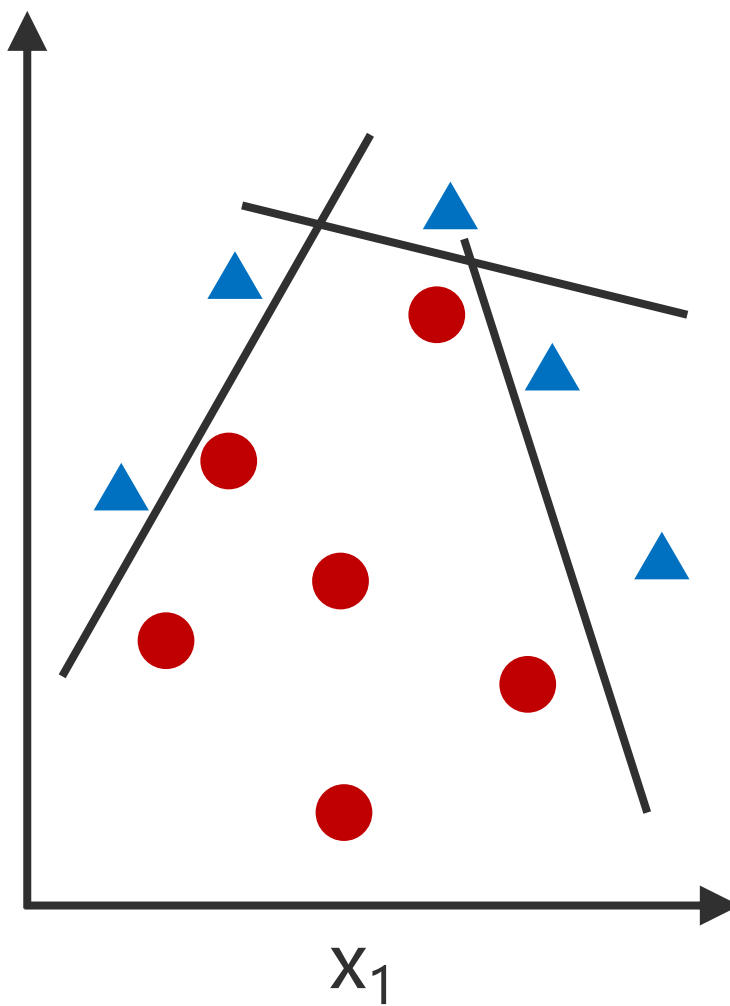
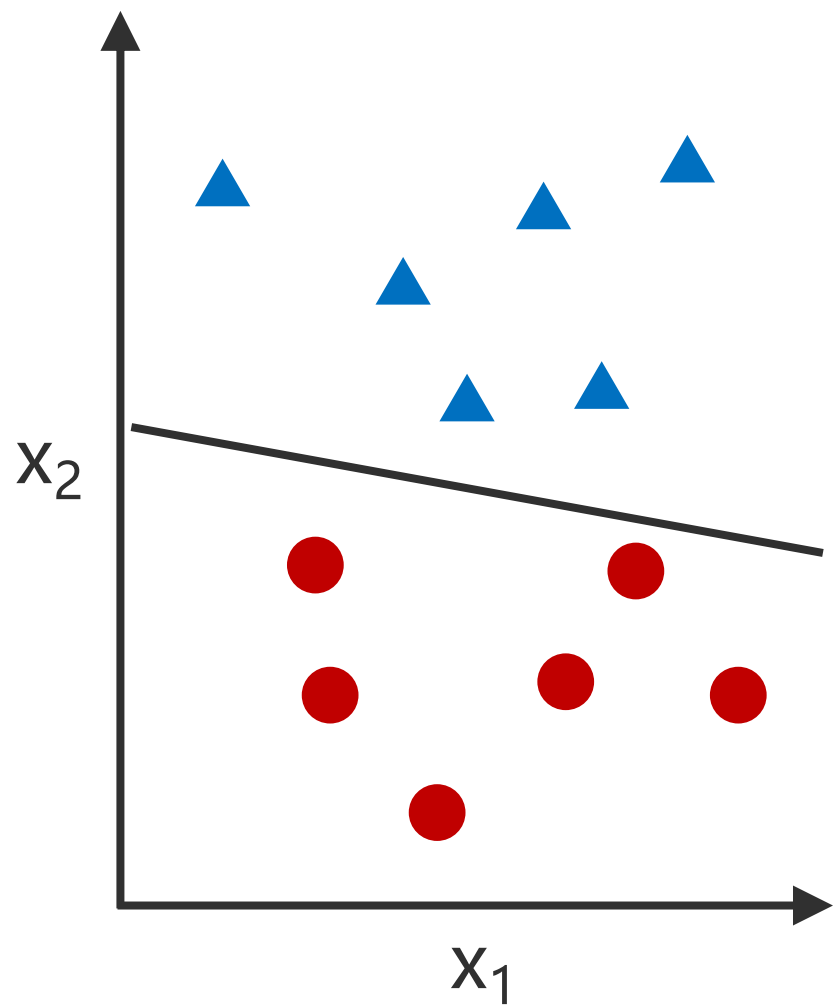
線形分離



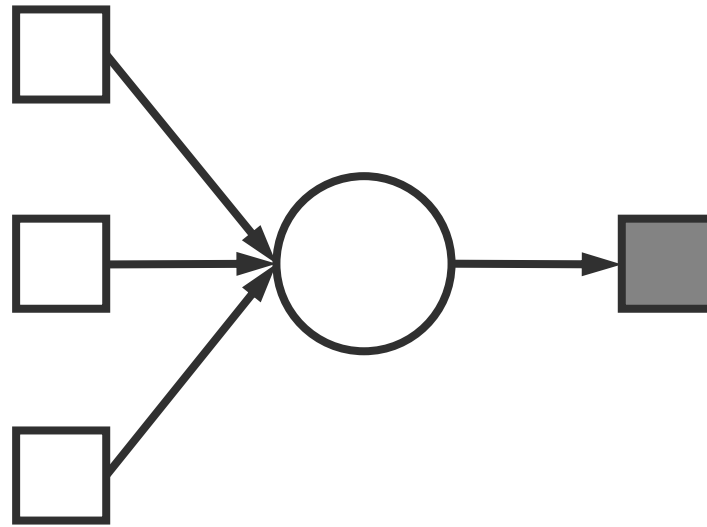
線形分離



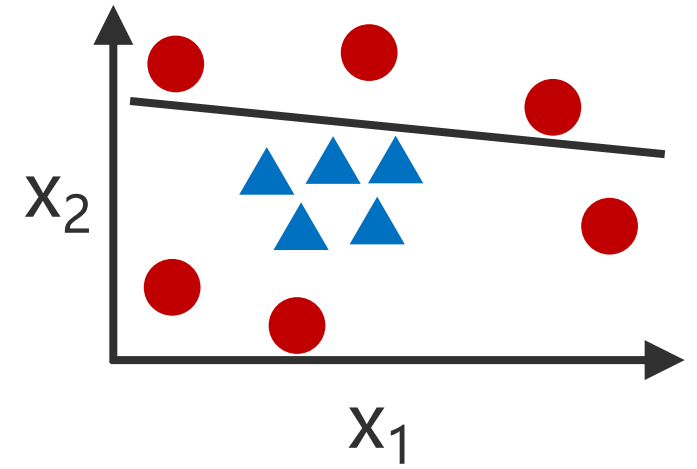
線形分離



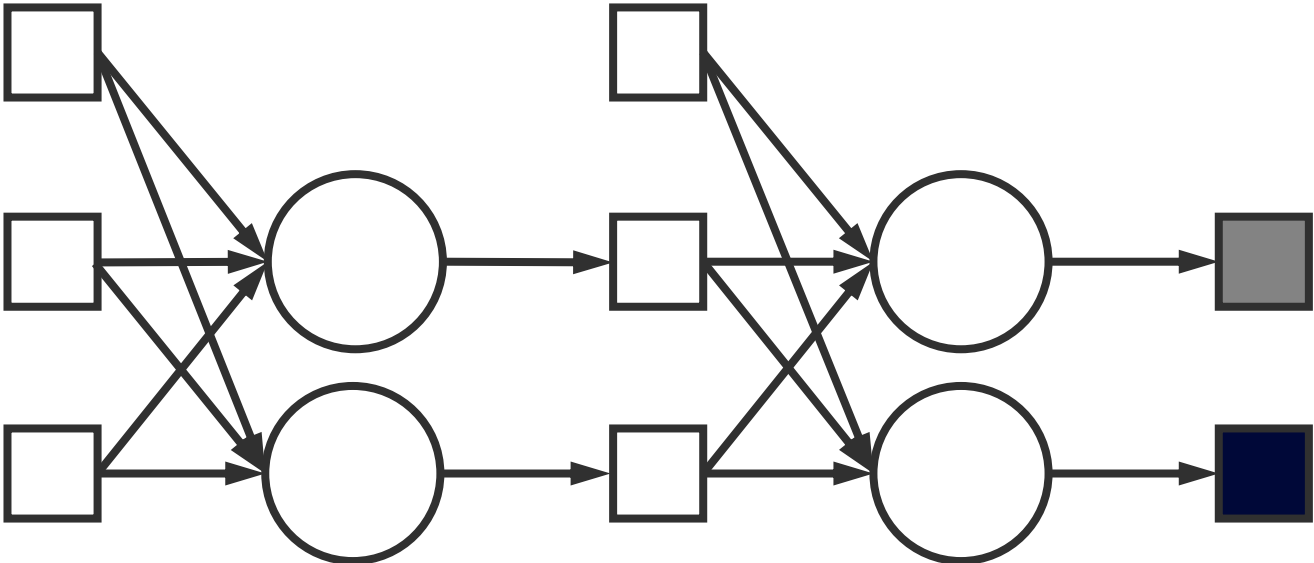
ニューラルネットワーク



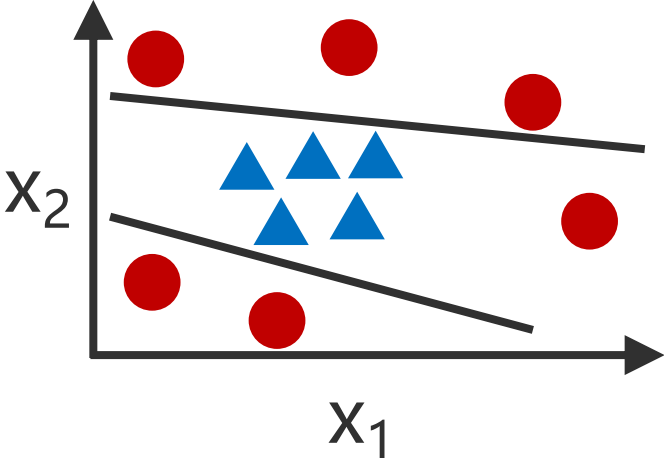
※ 図はイメージである。



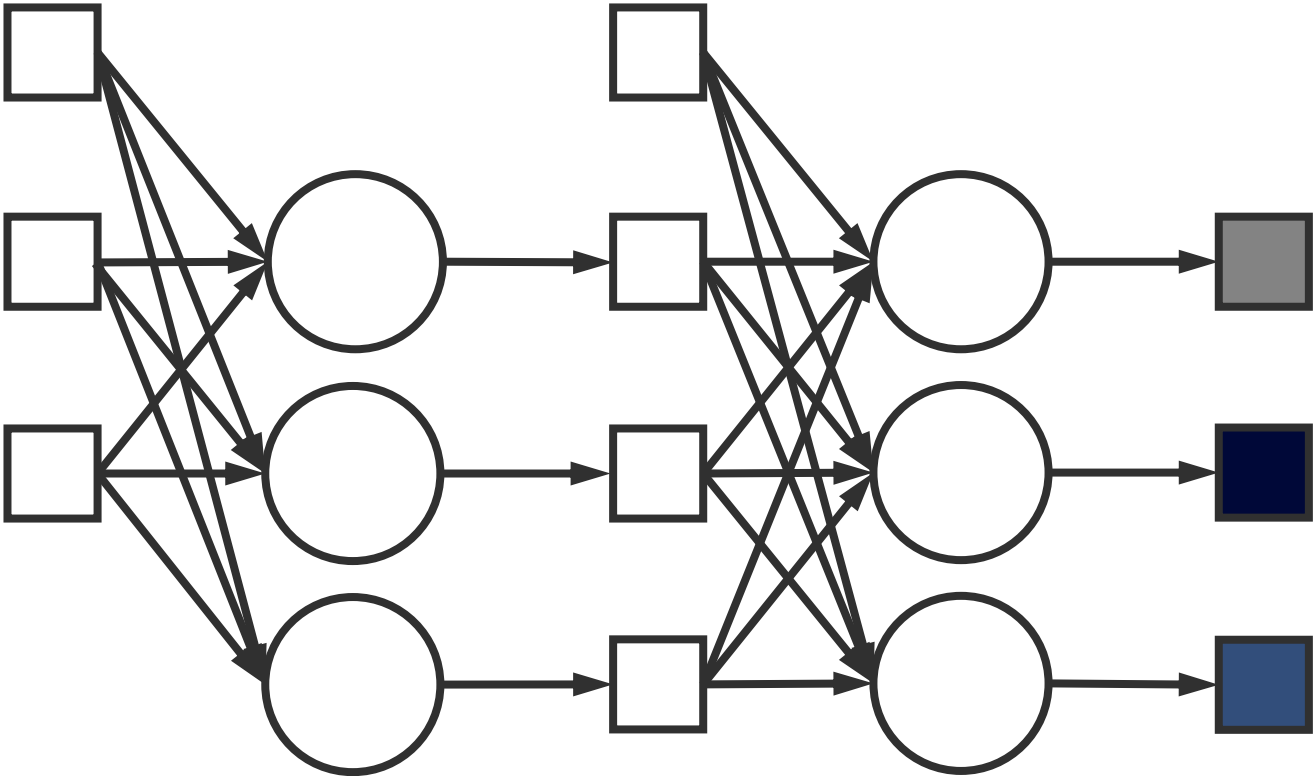
ニューラルネットワーク



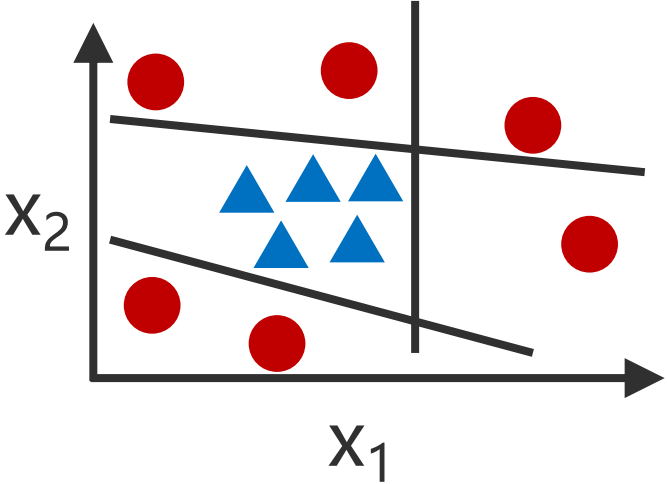
※ 図はイメージである。



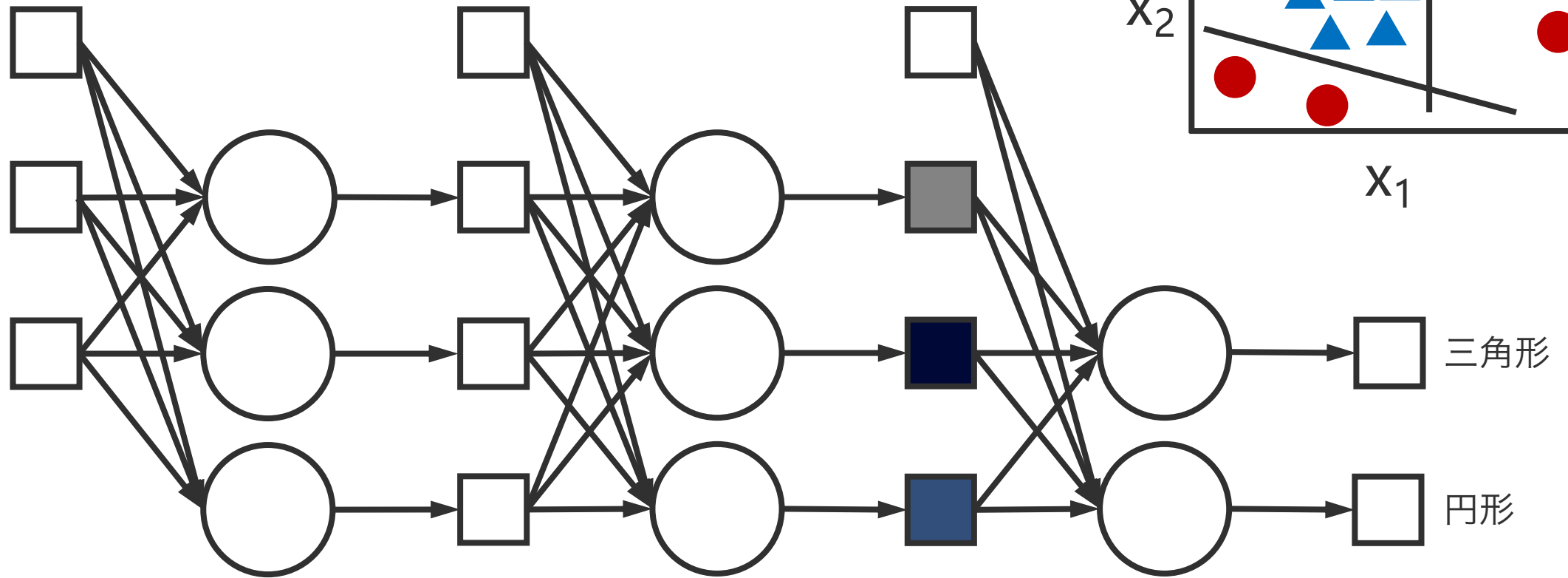
ニューラルネットワーク



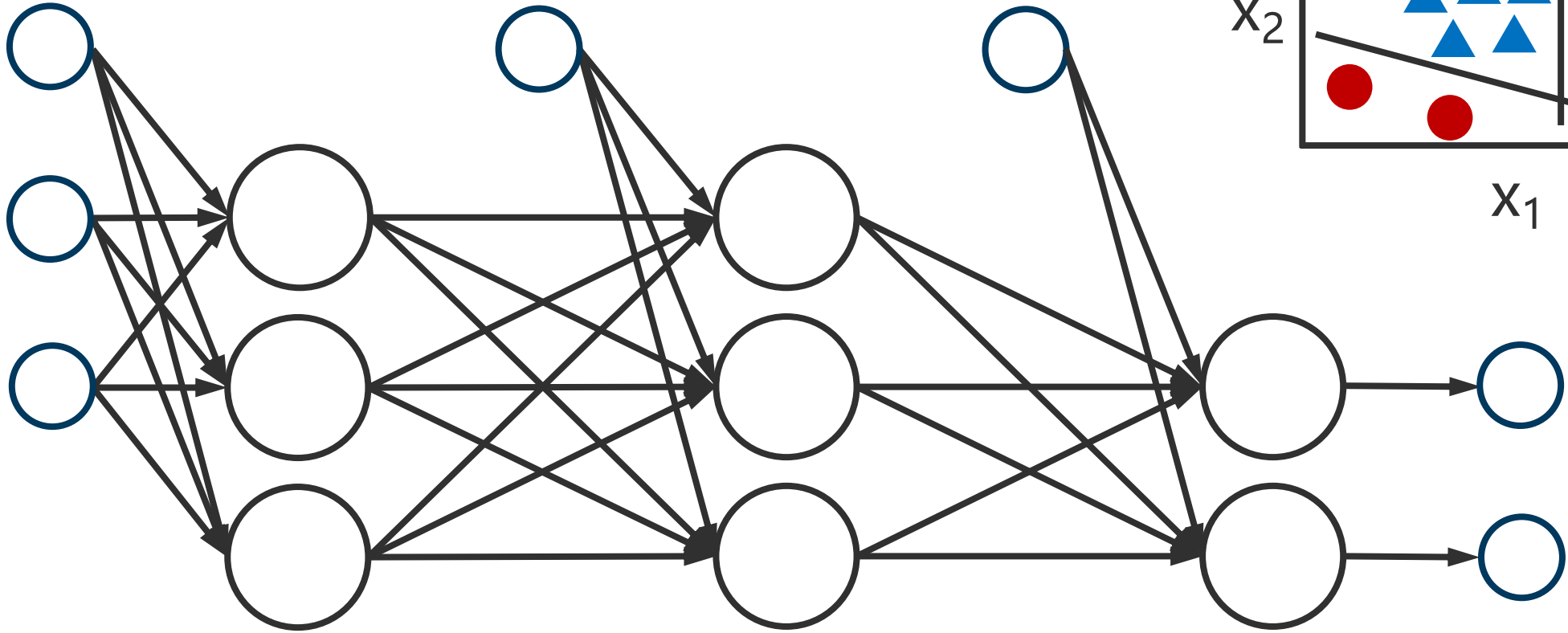
※ 図はイメージである。



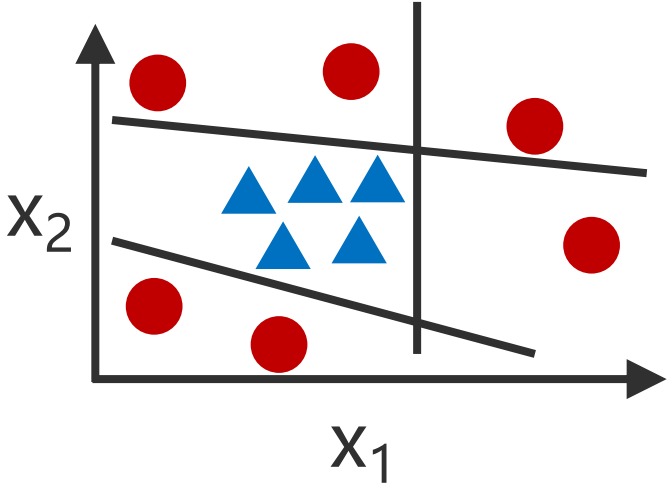
ニューラルネットワーク



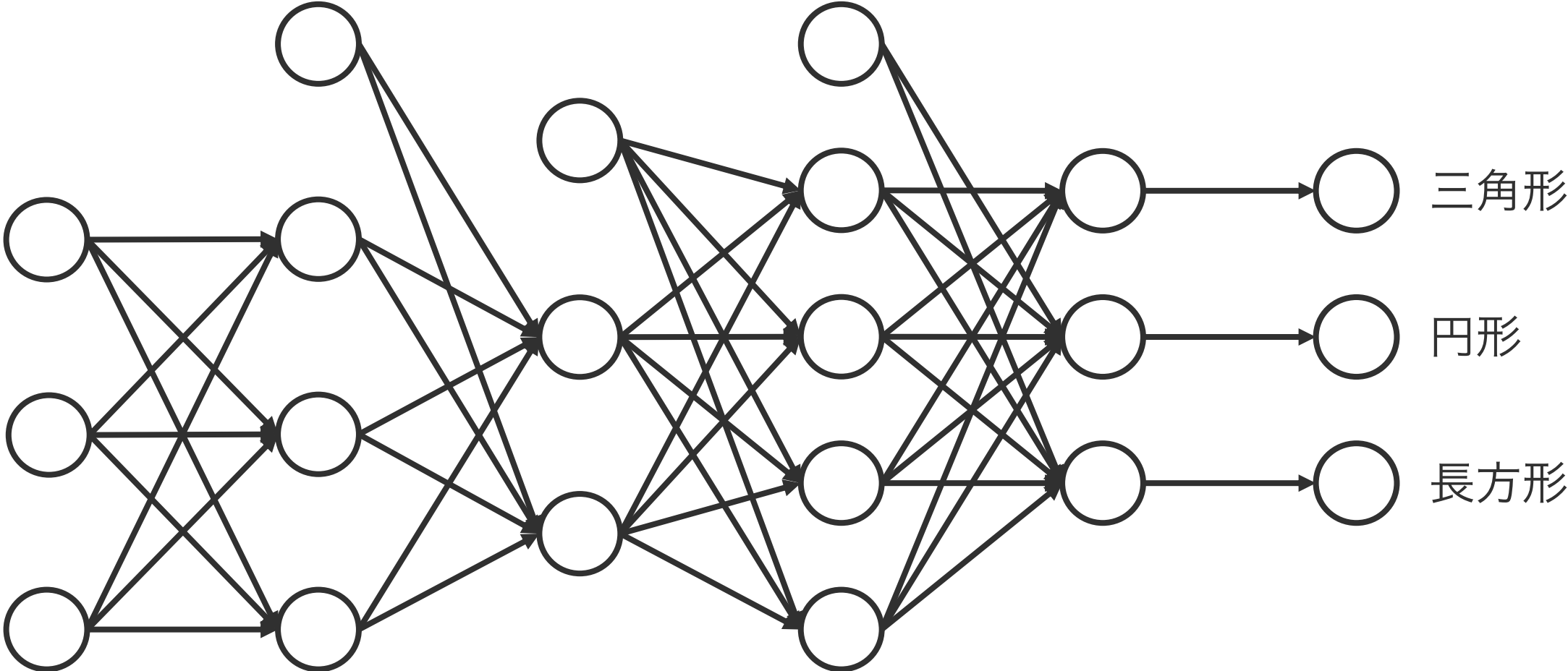
ニューラルネットワーク



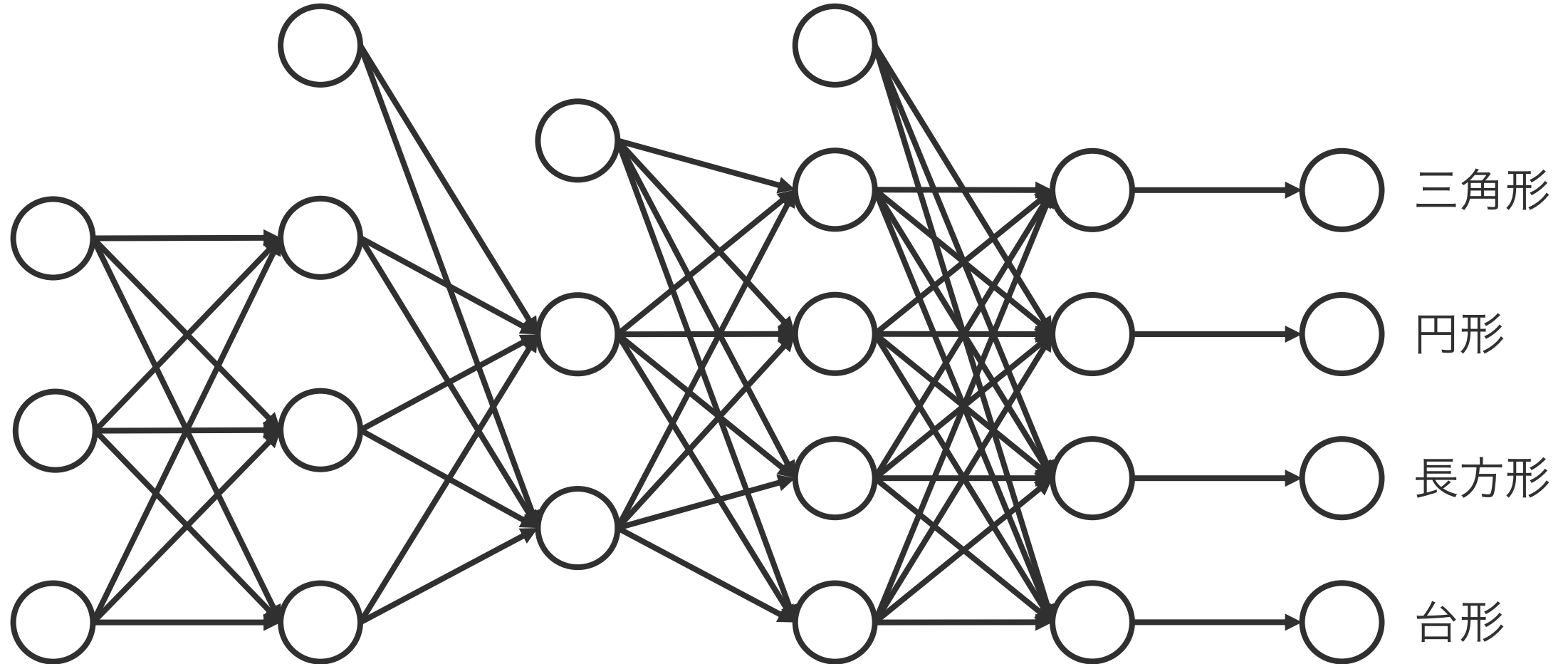
※ 図はイメージである。



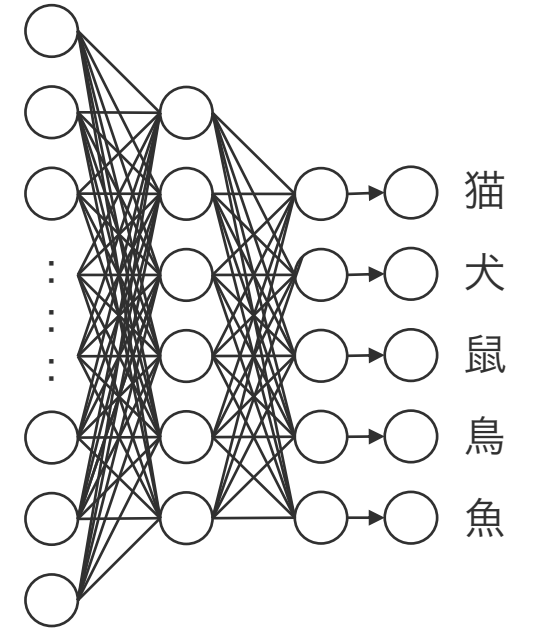
ニューラルネットワーク



ニューラルネットワーク



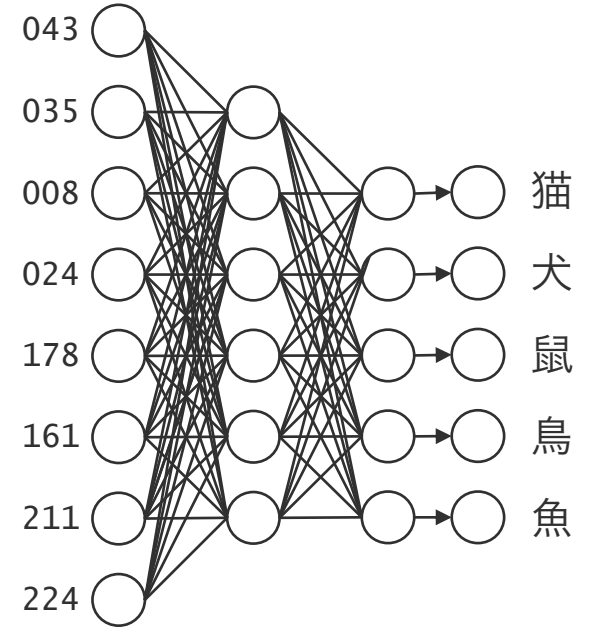
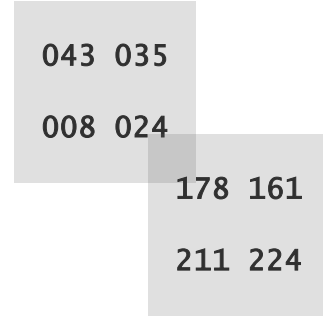
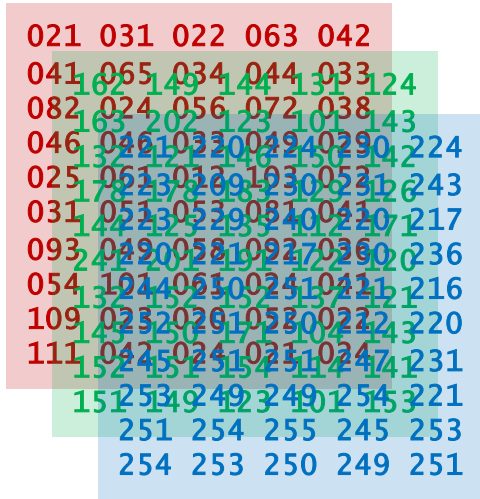
画像解析 (物体分類)



画像の RGB 輝度をそのままニューラルネットワークに入力すると・・・

- 画像サイズとが大きい ($225 \times 225 \times 3 = 151,875$)
- 画像にノイズが含まれている
- オブジェクトがズレると、行列の各位置の値が大きくズれる

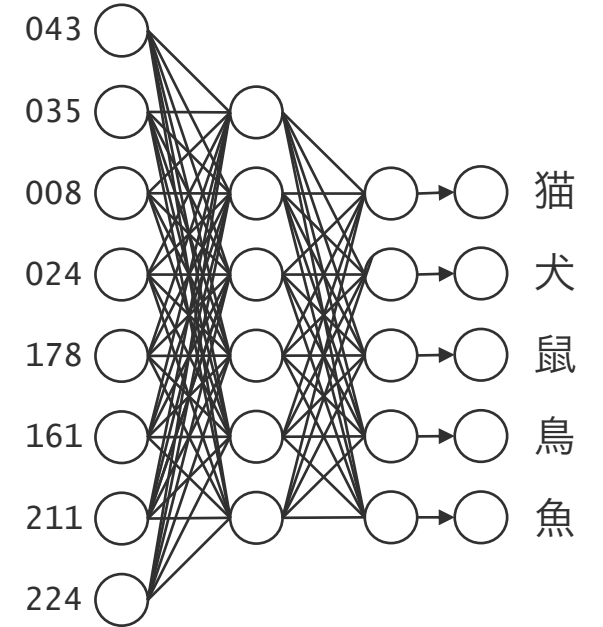
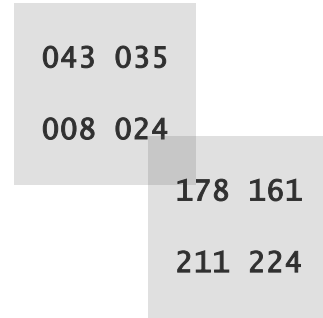
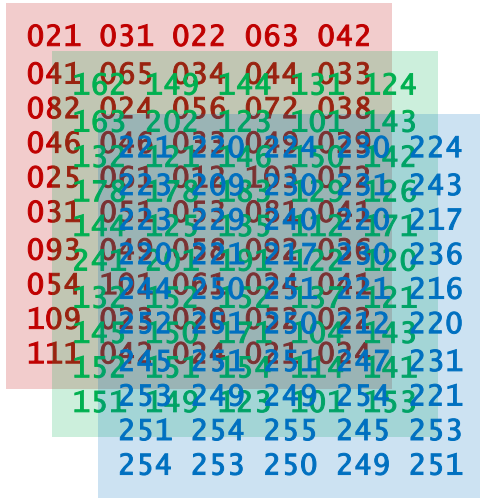
画像解析 (物体分類)



画像から重要な特徴だけを取り出して、その特徴量のみをニューラルネットワークに代入して分類を行う。

畳み込みニューラルネットワーク (CNN)

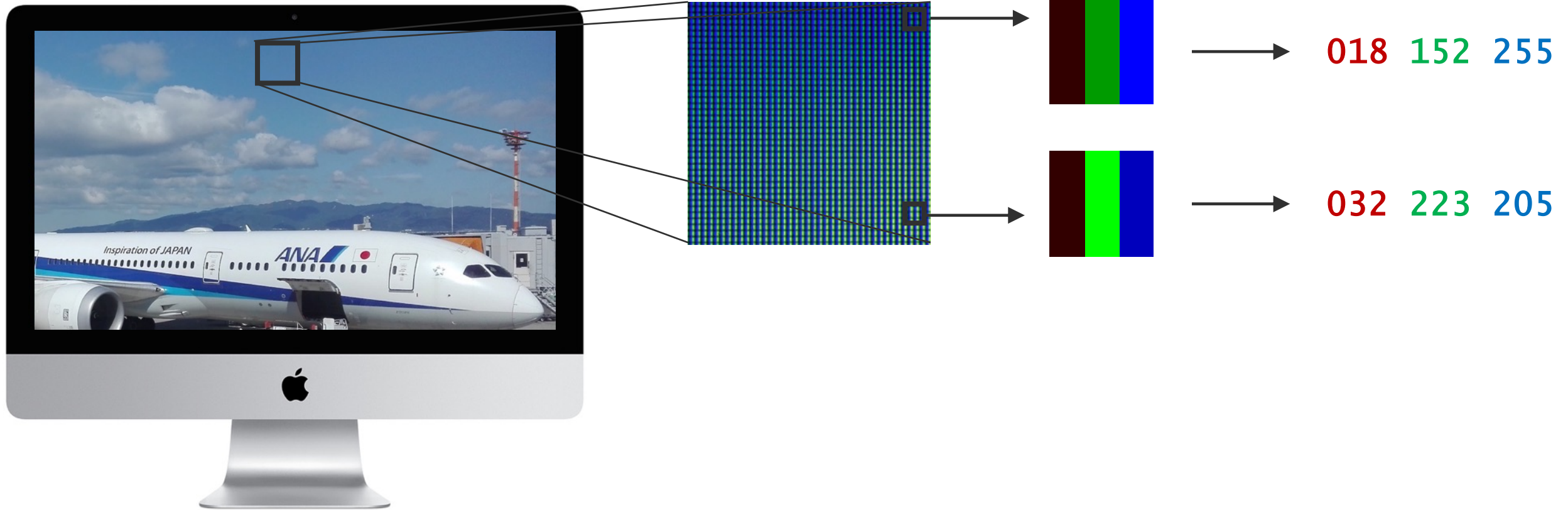
画像解析 (物体分類)



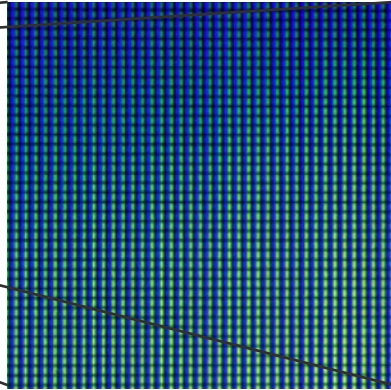
画像から重要な特徴だけを取り出して、その特徴量のみをニューラルネットワークに代入して分類を行う。

畳み込みニューラルネットワーク (CNN)

物体分類



物体分類

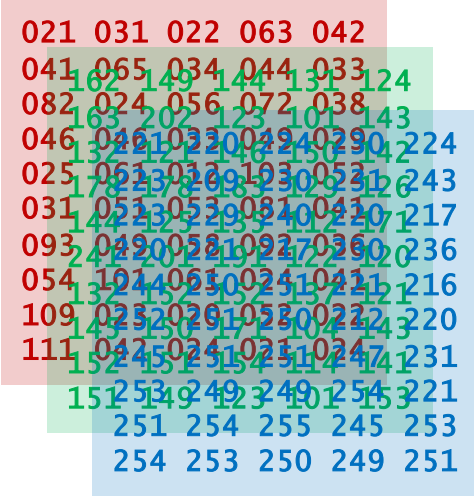


| | | | | |
|-----|-----|-----|-----|-----|
| 021 | 031 | 022 | 063 | 042 |
| 041 | 065 | 034 | 044 | 033 |
| 082 | 024 | 056 | 072 | 038 |
| 046 | 046 | 033 | 049 | 029 |
| 025 | 061 | 012 | 103 | 052 |
| 031 | 051 | 053 | 081 | 041 |
| 093 | 049 | 058 | 092 | 036 |
| 054 | 101 | 061 | 024 | 041 |
| 109 | 023 | 020 | 052 | 022 |
| 111 | 042 | 024 | 021 | 024 |

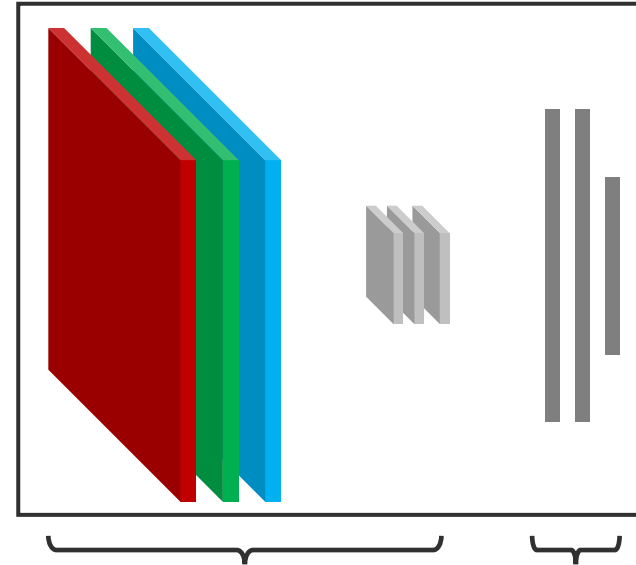
| | | | | |
|-----|-----|-----|-----|-----|
| 162 | 149 | 144 | 131 | 124 |
| 163 | 202 | 123 | 101 | 143 |
| 132 | 121 | 146 | 150 | 142 |
| 178 | 178 | 183 | 129 | 126 |
| 144 | 125 | 135 | 112 | 171 |
| 241 | 201 | 191 | 122 | 120 |
| 132 | 152 | 152 | 137 | 121 |
| 145 | 150 | 171 | 104 | 143 |
| 152 | 151 | 154 | 114 | 141 |
| 151 | 149 | 123 | 101 | 153 |

| | | | | |
|-----|-----|-----|-----|-----|
| 221 | 220 | 224 | 230 | 224 |
| 223 | 209 | 230 | 231 | 243 |
| 223 | 229 | 240 | 220 | 217 |
| 220 | 221 | 217 | 230 | 236 |
| 244 | 250 | 251 | 221 | 216 |
| 252 | 251 | 250 | 212 | 220 |
| 245 | 251 | 251 | 247 | 231 |
| 253 | 249 | 249 | 254 | 221 |
| 251 | 254 | 255 | 245 | 253 |
| 254 | 253 | 250 | 249 | 251 |

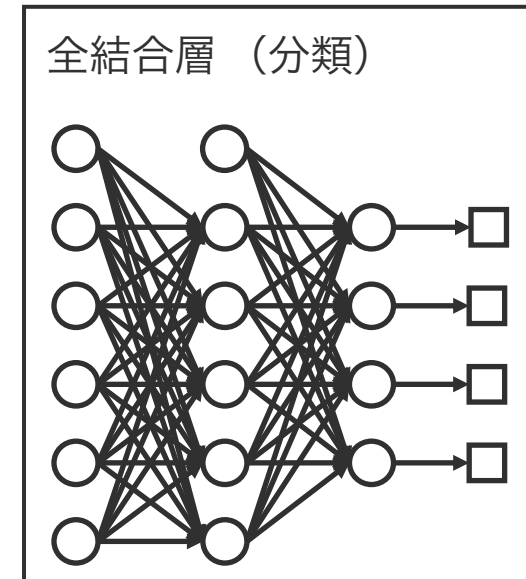
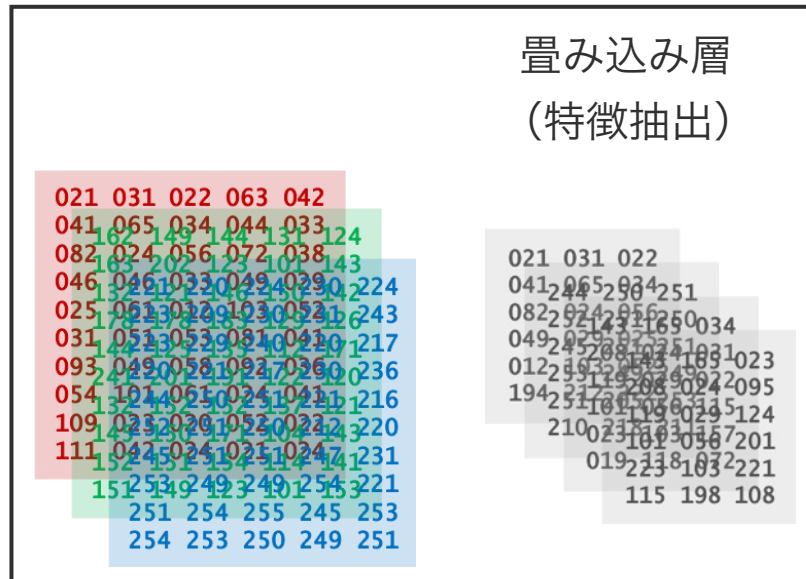
畳み込みニューラルネットワーク



畳み込みニューラルネットワーク



airplane 56.1%
 sky 25.0%
 cloud 12.4%
 mountain 6.5%



モノクロ画像



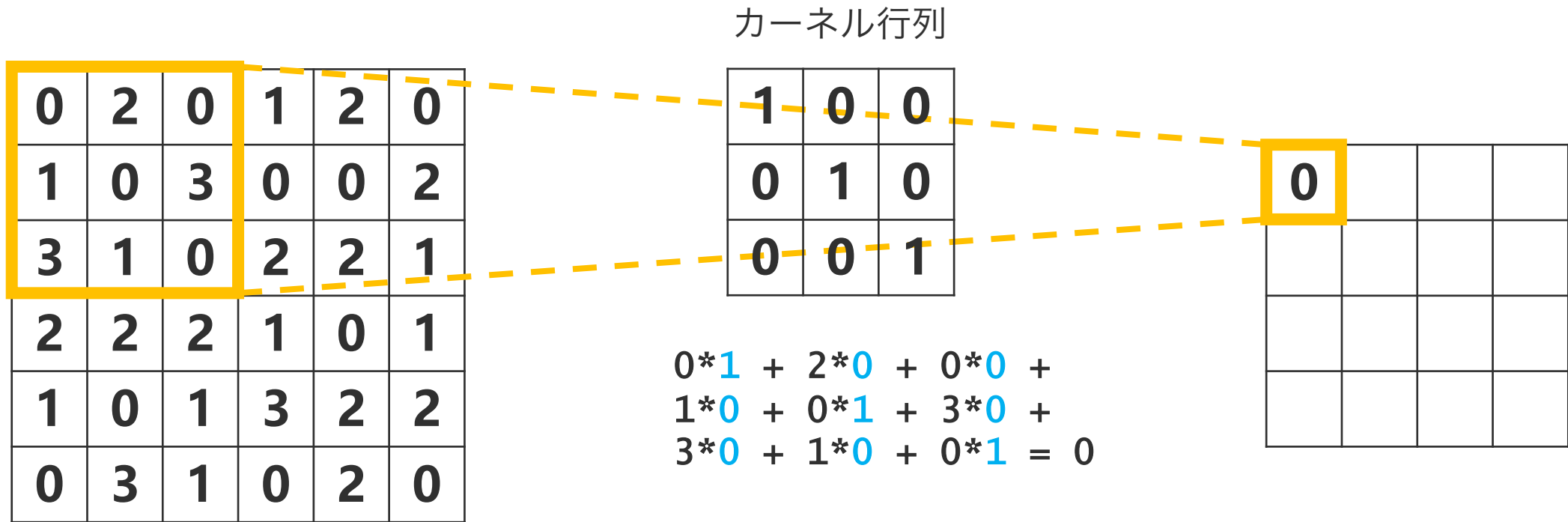
```
162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153
```

カラー画像

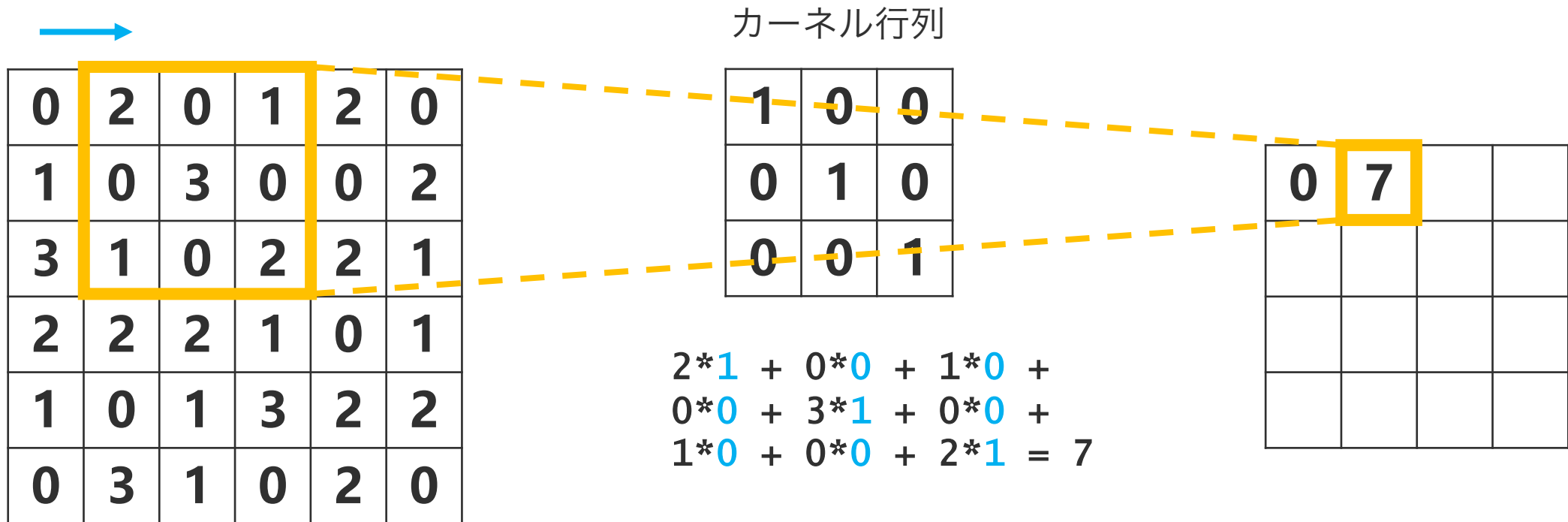


```
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 004 029 0224
025 061 012 013 052
031 051 053 081 041 0243
093 019 058 092 036 0217
054 104 061 025 104 11 216
109 025 026 105 0222 220
111 042 024 021 024 231
152 151 154 114 141
151 149 123 101 153
251 254 255 245 253
254 253 250 249 251
```

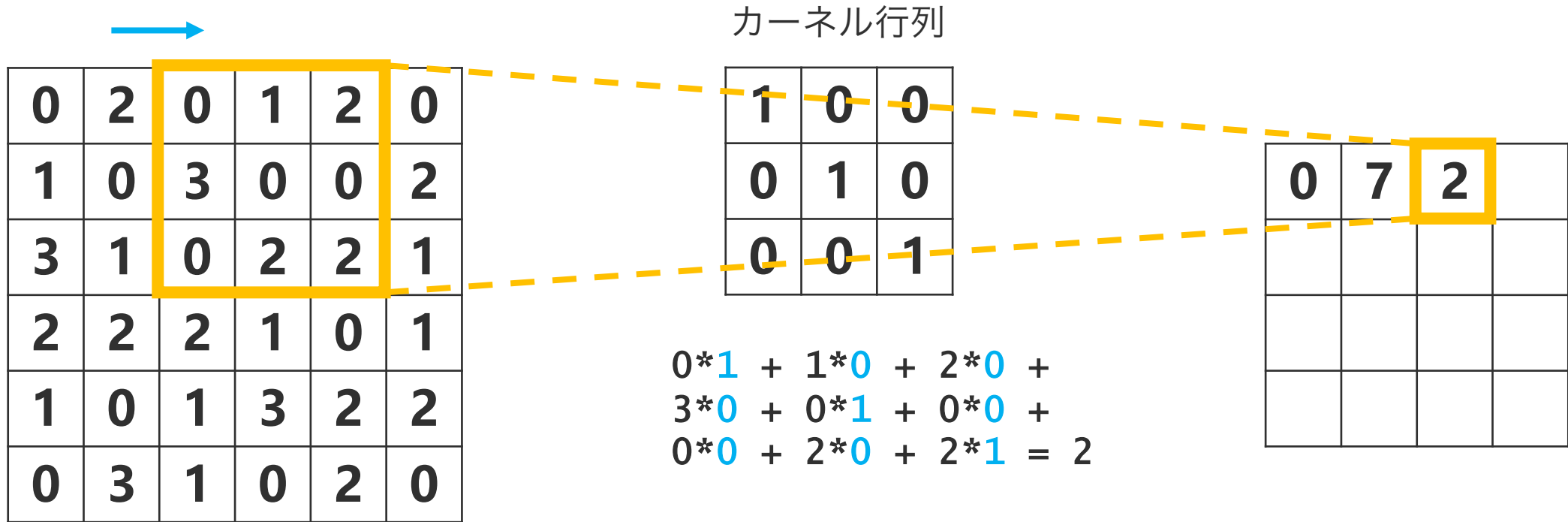
畳み込み演算



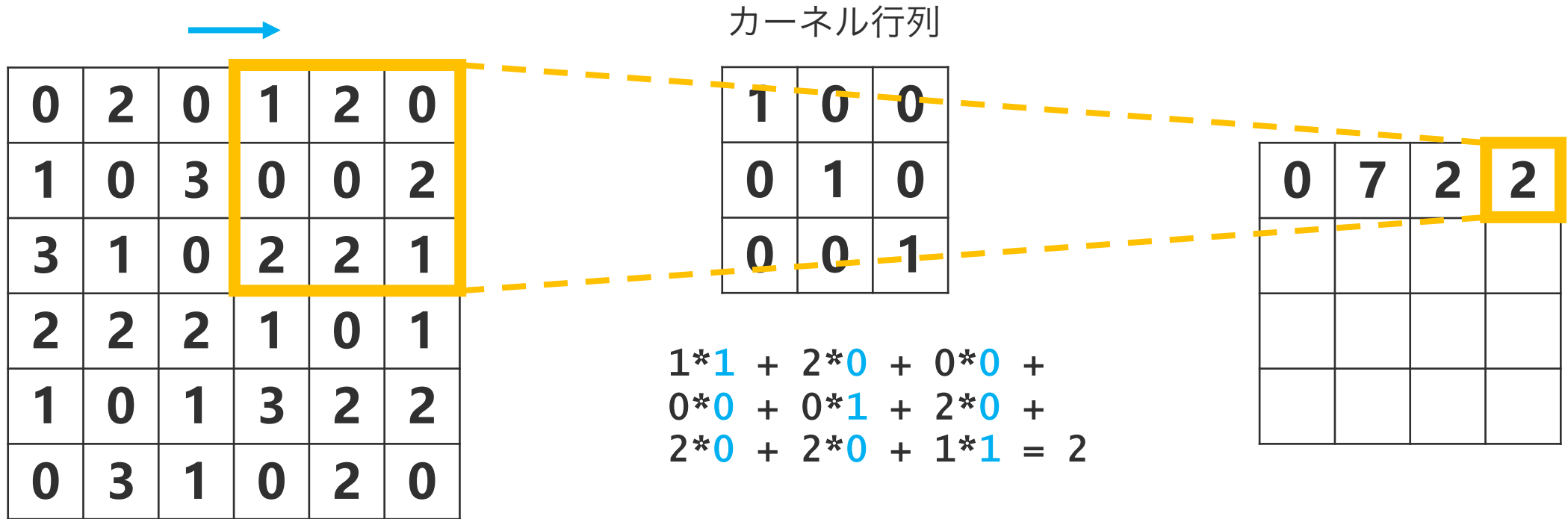
畳み込み演算



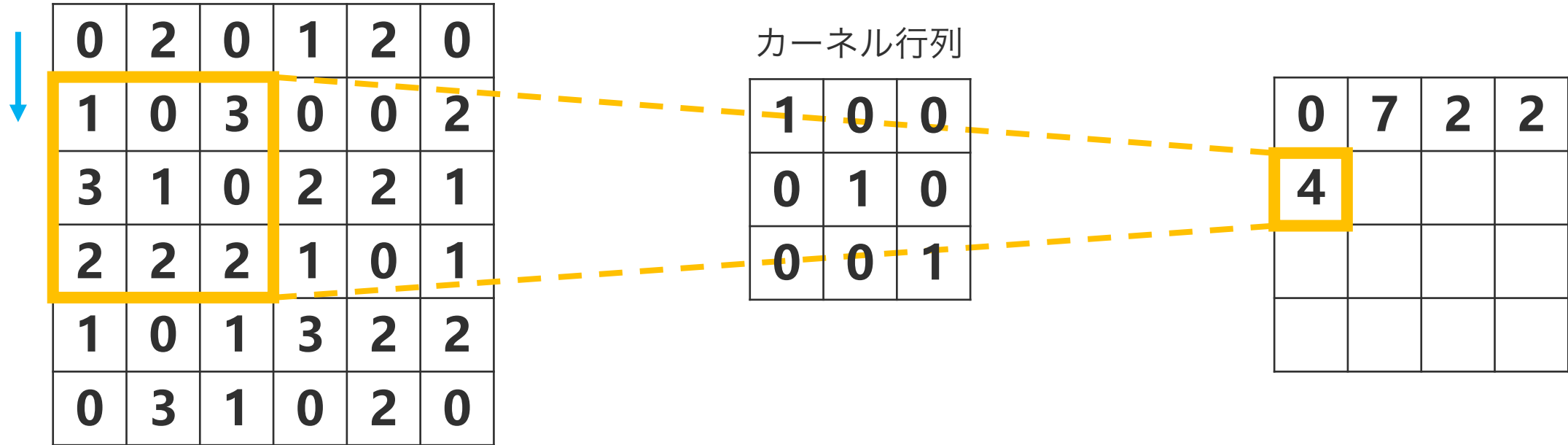
畳み込み演算



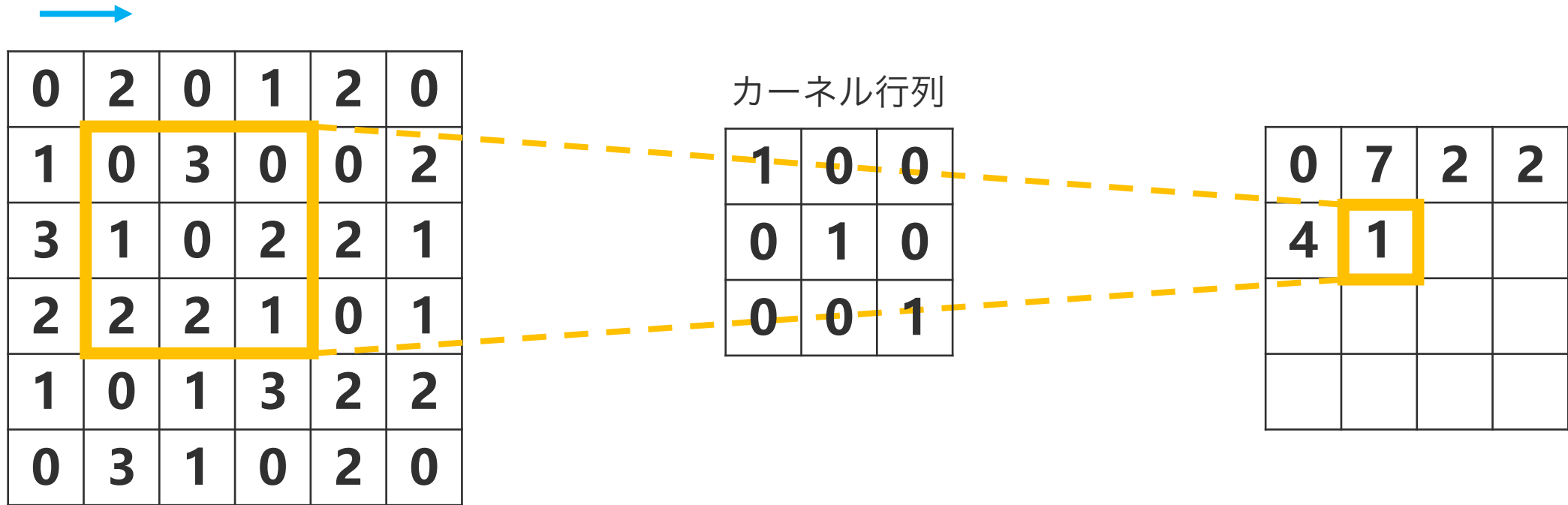
畳み込み演算



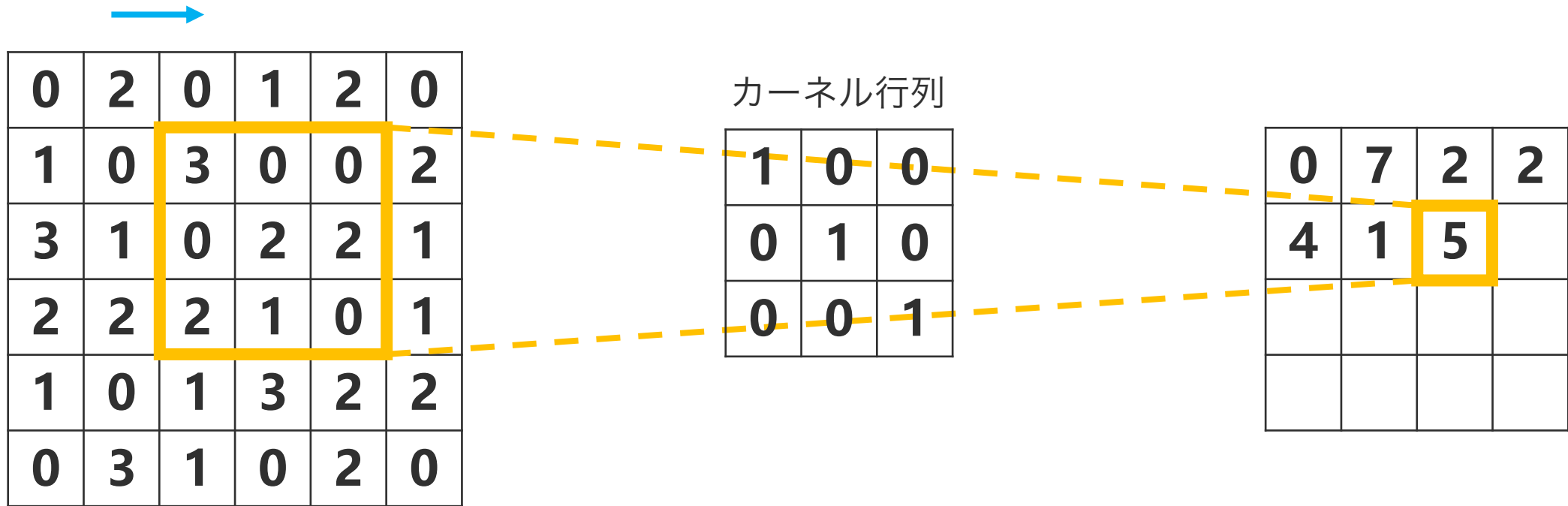
畳み込み演算



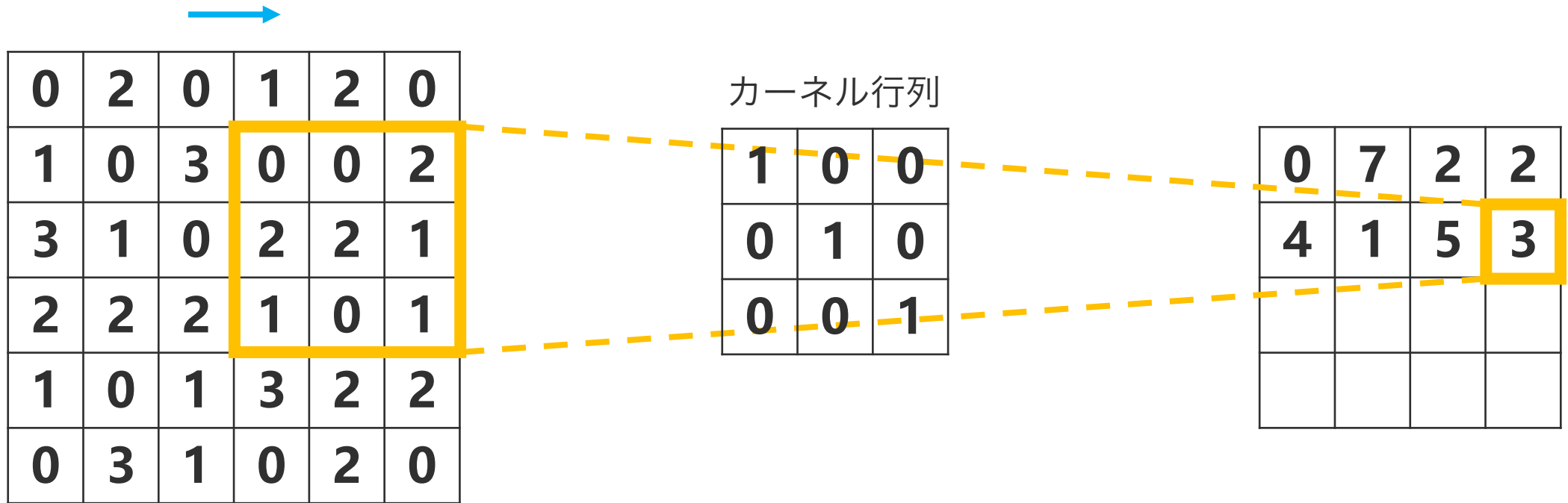
畳み込み演算



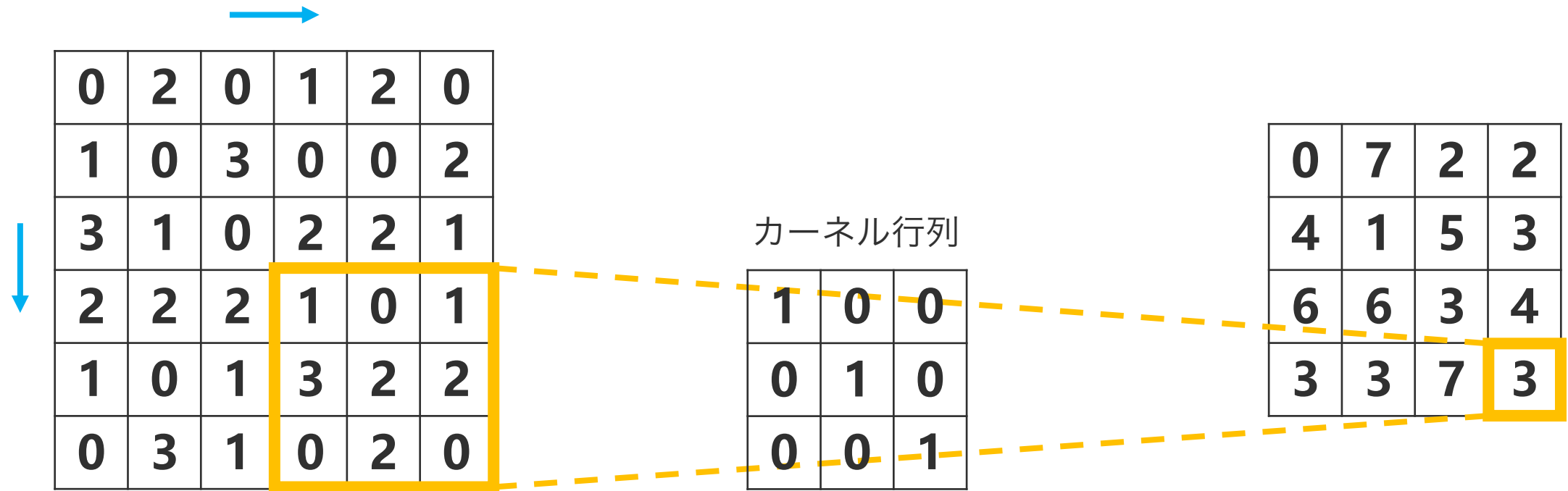
畳み込み演算



畳み込み演算



畳み込み演算



畳み込み演算

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 2 | 0 | 1 | 2 | 0 | 0 |
| 0 | 1 | 0 | 3 | 0 | 0 | 2 | 0 |
| 0 | 3 | 1 | 0 | 2 | 2 | 1 | 0 |
| 0 | 2 | 2 | 2 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 3 | 2 | 2 | 0 |
| 0 | 0 | 3 | 1 | 0 | 2 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

カーネル行列

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

| | | | | | |
|---|---|---|---|---|---|
| 0 | 5 | 0 | 1 | 2 | 0 |
| 2 | 0 | 7 | 2 | 2 | 4 |
| 5 | 4 | 1 | 5 | 3 | 1 |
| 2 | 6 | 6 | 3 | 4 | 3 |
| 4 | 3 | 3 | 7 | 3 | 2 |
| 0 | 4 | 1 | 1 | 5 | 0 |

モノクロ画像



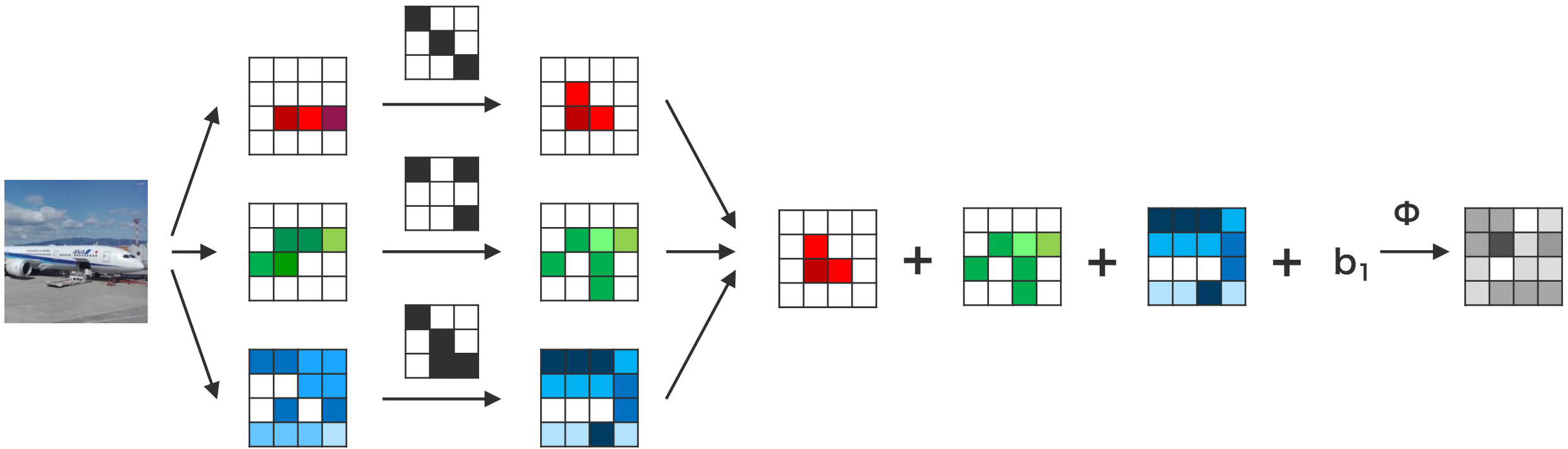
```
162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153
```

カラー画像

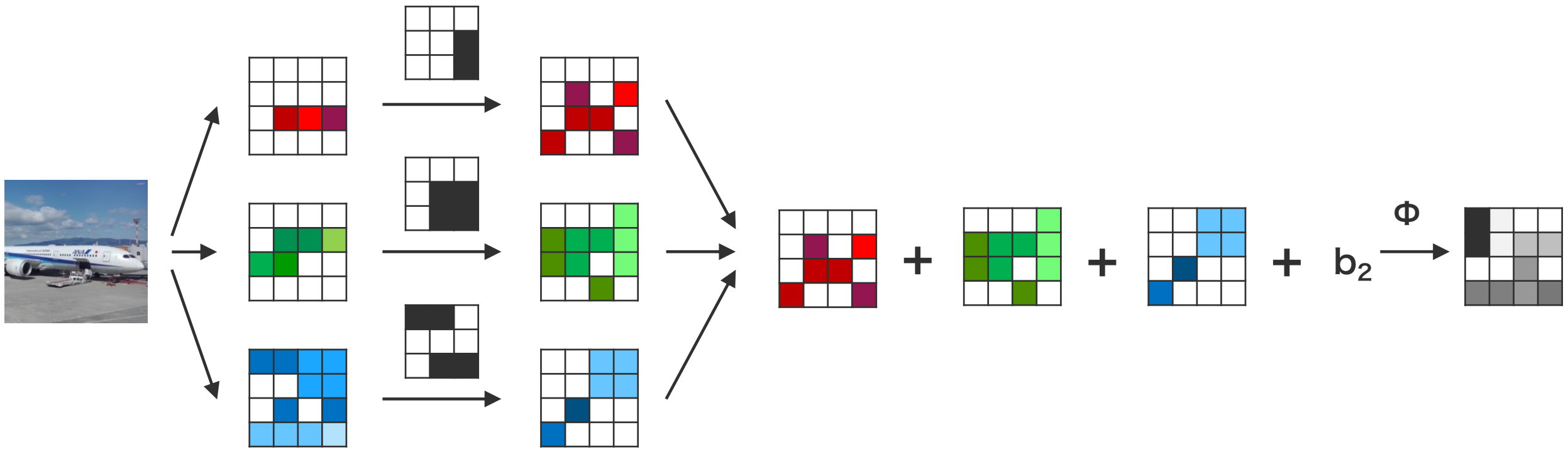


```
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 004 029 0224
025 051 012 013 052 1
178 178 183 129 126
031 051 053 081 041 0217
093 019 058 092 036 036
054 104 061 025 041 1216
109 025 026 053 022 220
111 042 024 021 024 231
152 151 154 114 141
152 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153
251 254 255 245 253
254 253 250 249 251
```

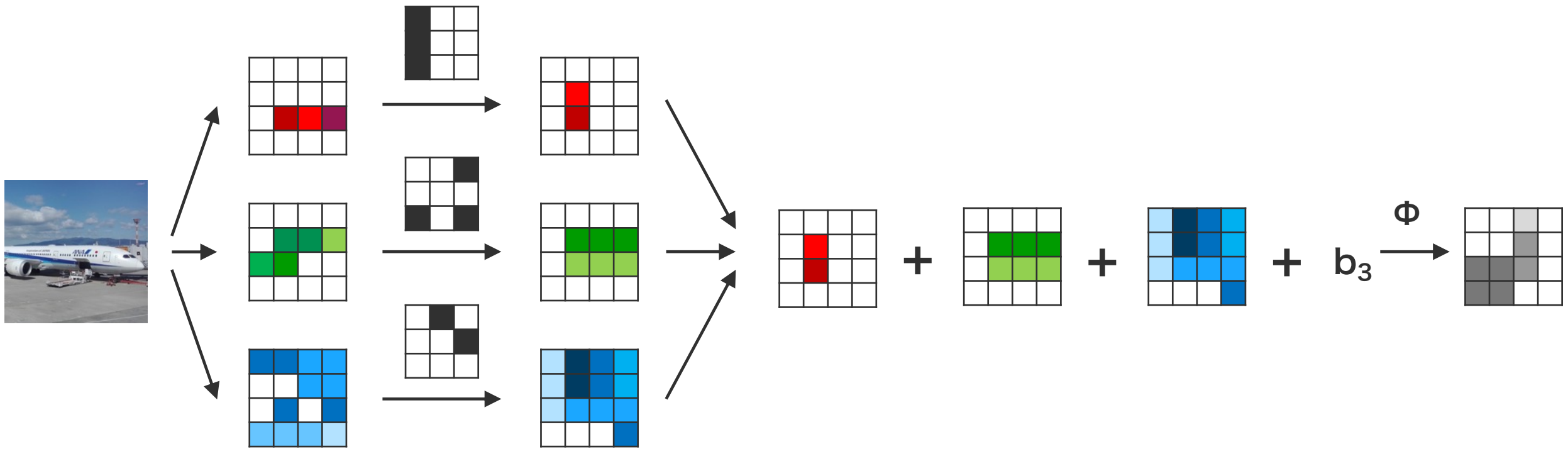
畳み込み演算



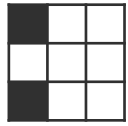
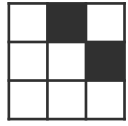
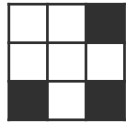
畳み込み演算



畳み込み演算

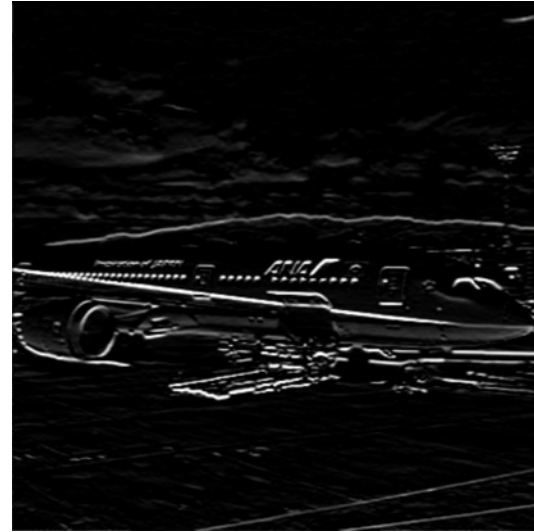
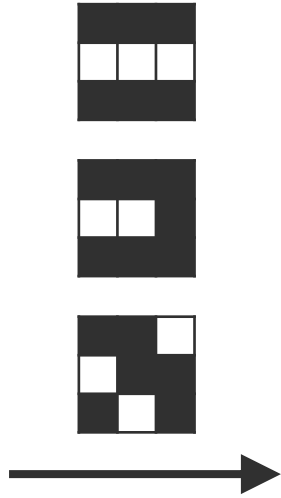


畳み込み演算



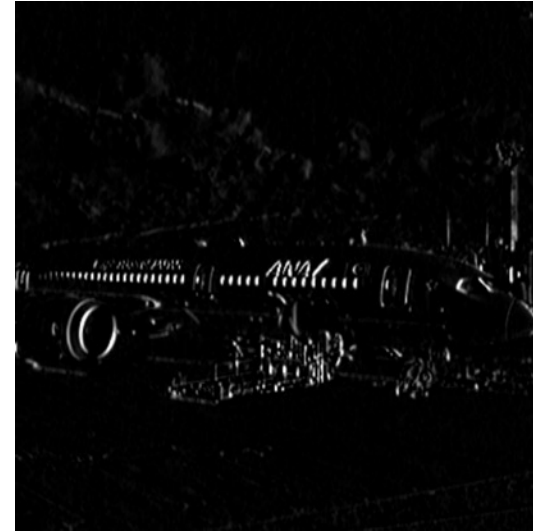
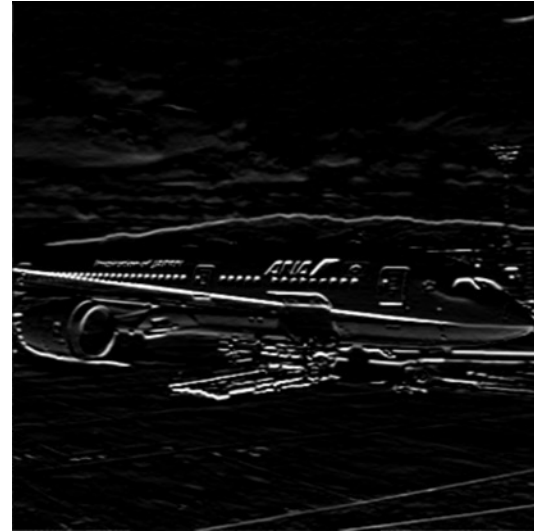
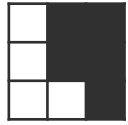
カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

畳み込み演算



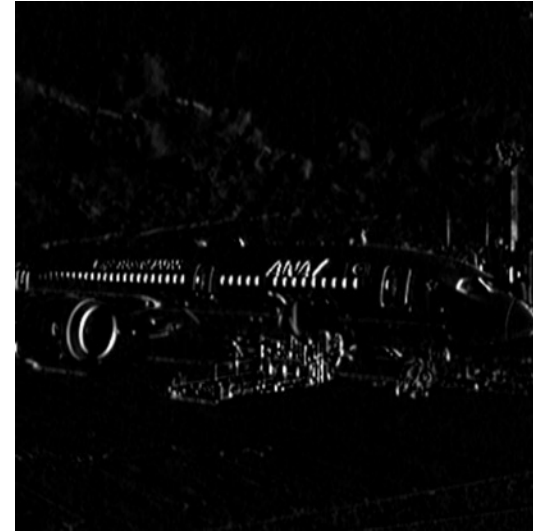
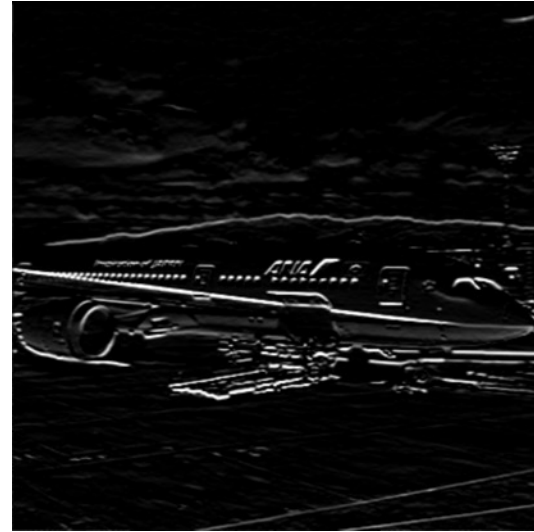
カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

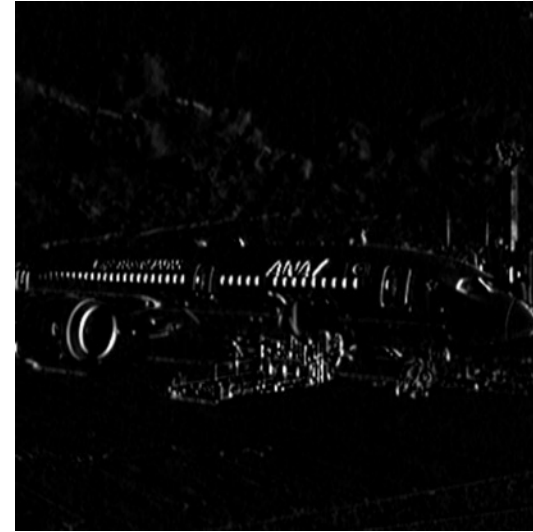
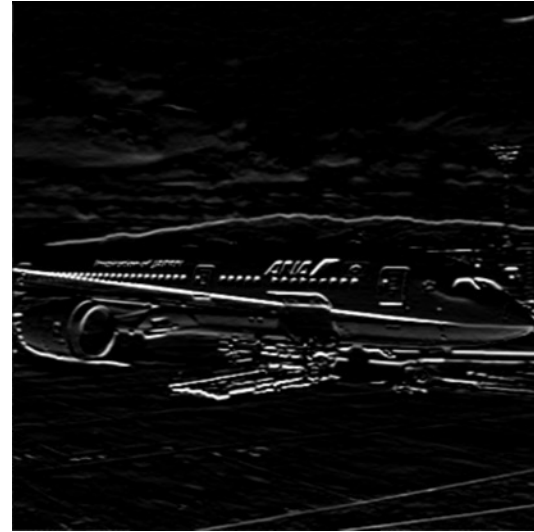
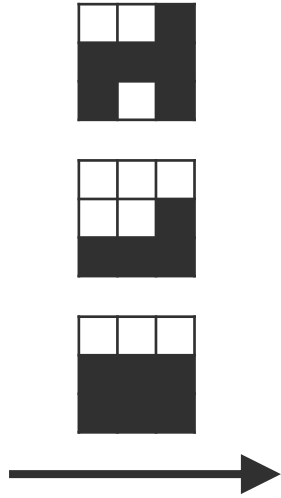
畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。



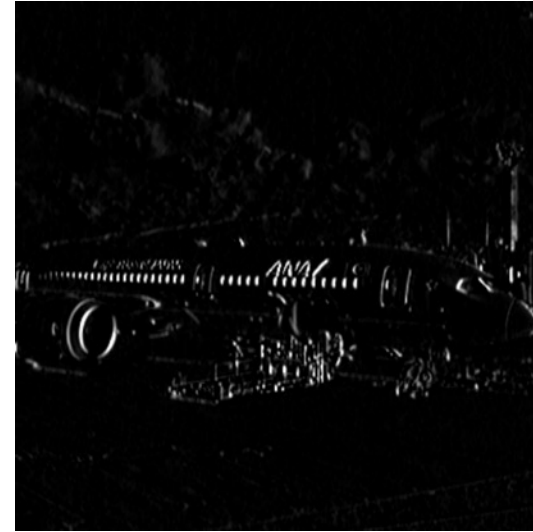
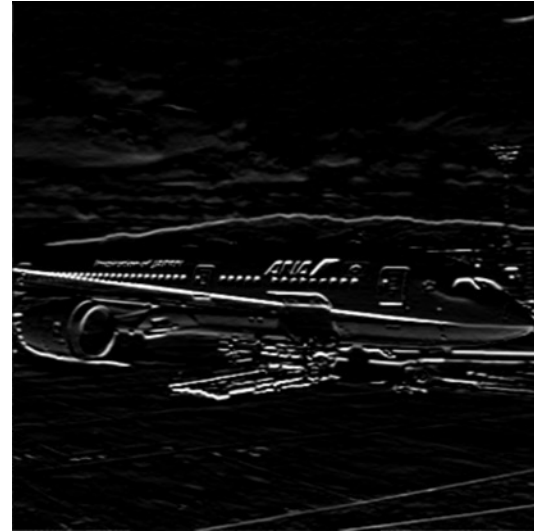
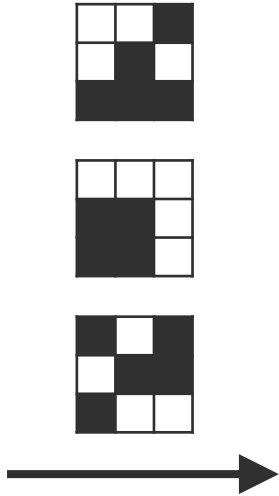
畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。



畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

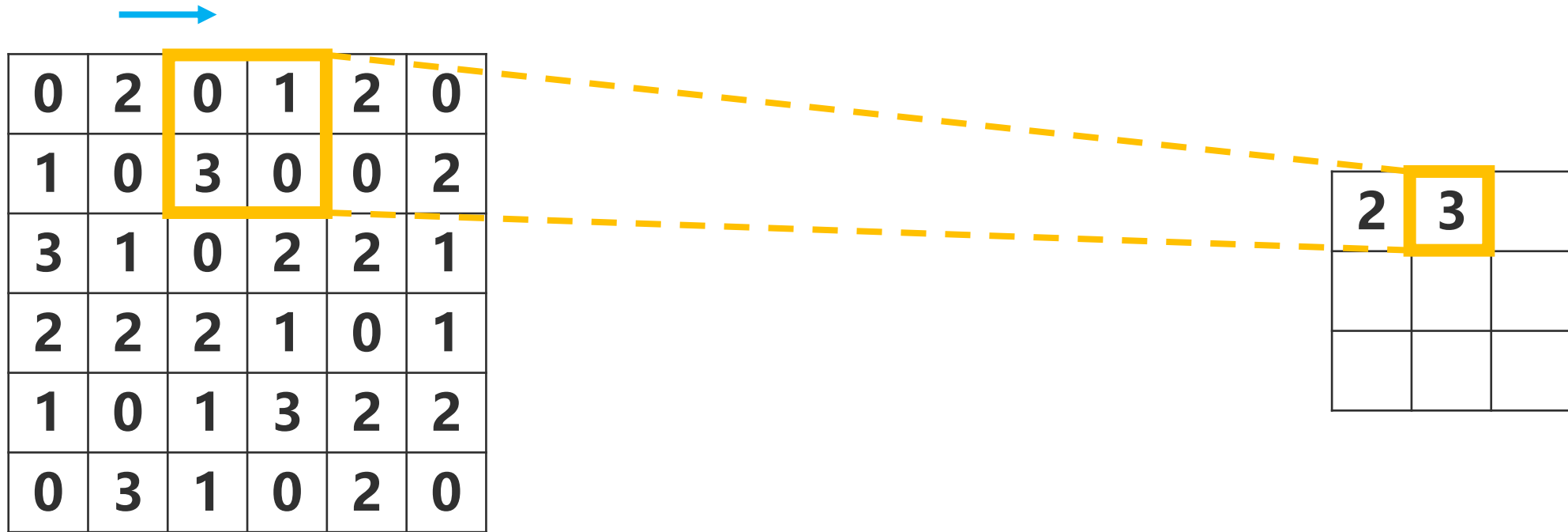


最大値プーリング演算

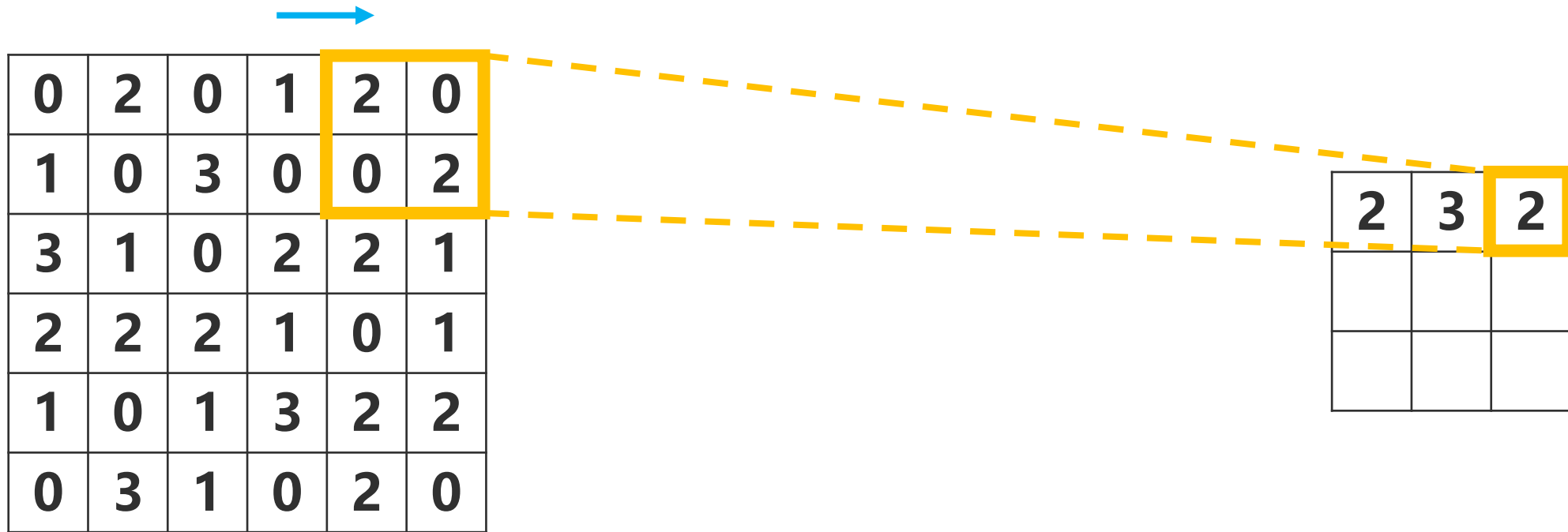
| | | | | | |
|---|---|---|---|---|---|
| 0 | 2 | 0 | 1 | 2 | 0 |
| 1 | 0 | 3 | 0 | 0 | 2 |
| 3 | 1 | 0 | 2 | 2 | 1 |
| 2 | 2 | 2 | 1 | 0 | 1 |
| 1 | 0 | 1 | 3 | 2 | 2 |
| 0 | 3 | 1 | 0 | 2 | 0 |

| | | |
|---|--|--|
| 2 | | |
| | | |
| | | |

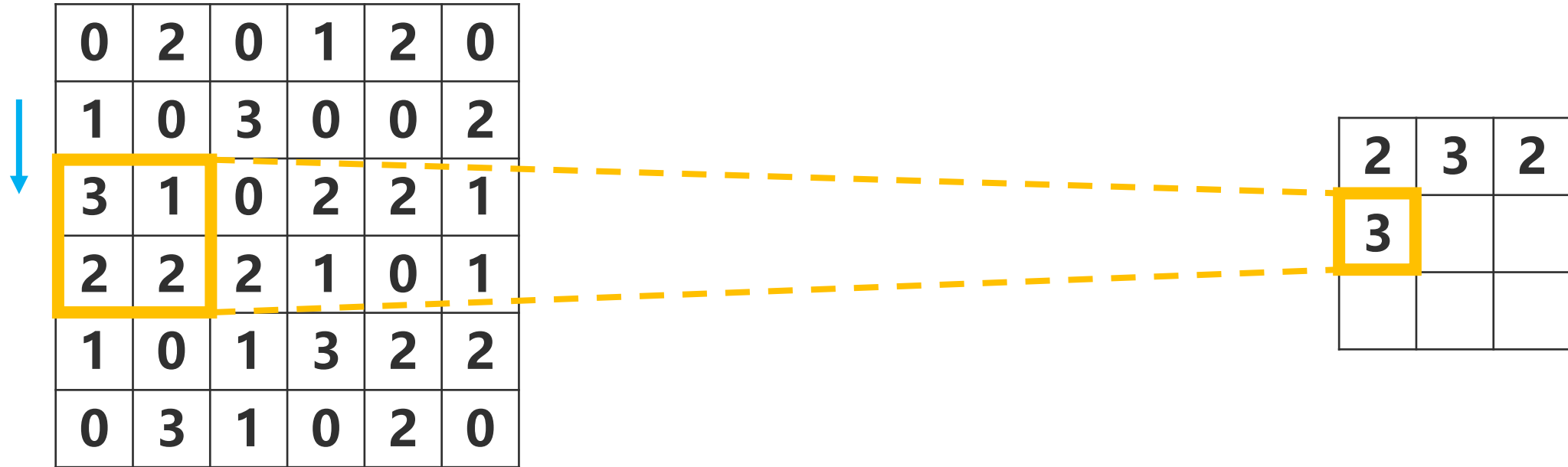
最大値プーリング演算



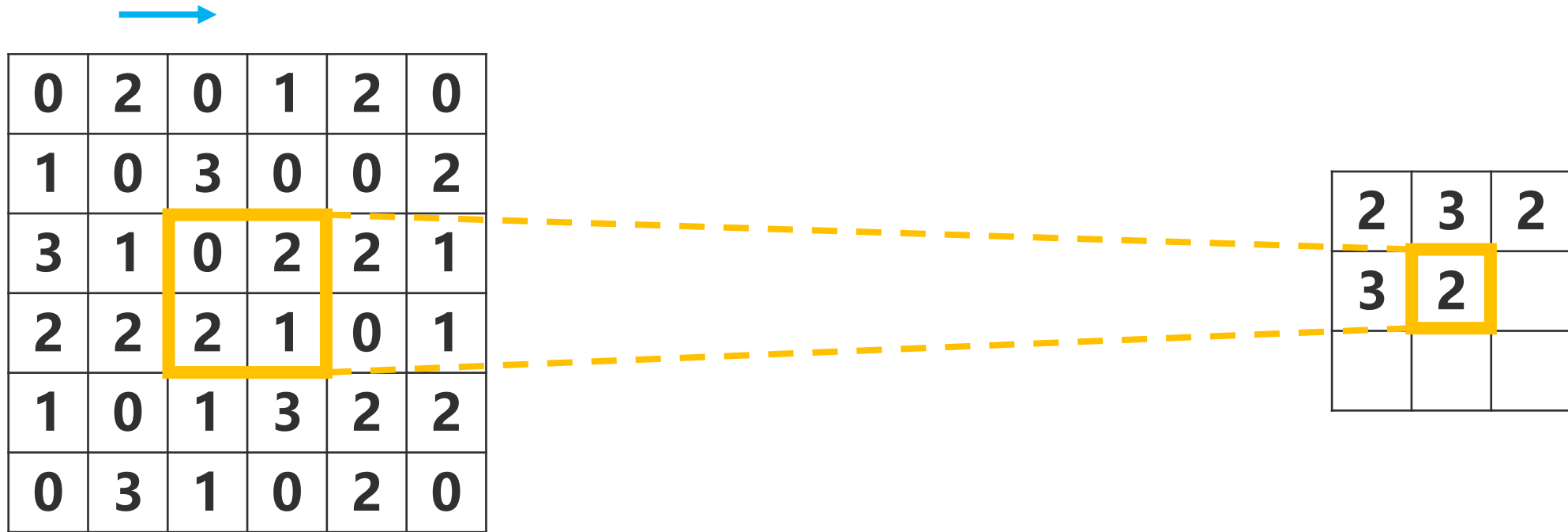
最大値プーリング演算



最大値プーリング演算



最大値プーリング演算



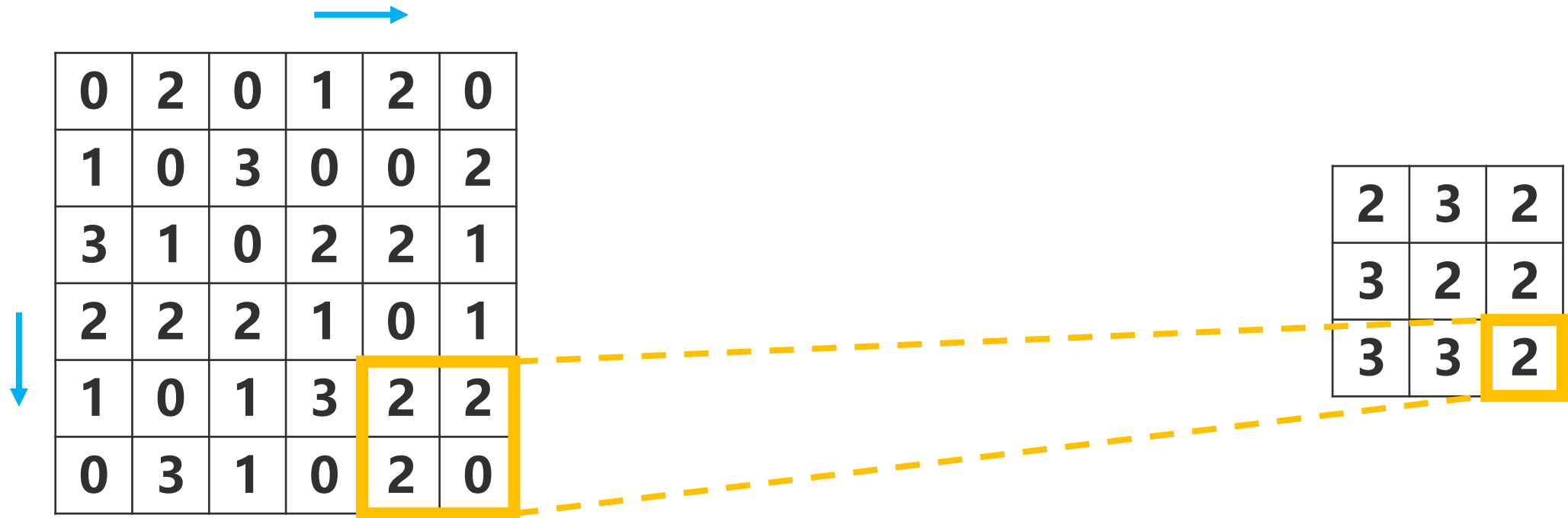
最大値プーリング演算



| | | | | | |
|---|---|---|---|---|---|
| 0 | 2 | 0 | 1 | 2 | 0 |
| 1 | 0 | 3 | 0 | 0 | 2 |
| 3 | 1 | 0 | 2 | 2 | 1 |
| 2 | 2 | 2 | 1 | 0 | 1 |
| 1 | 0 | 1 | 3 | 2 | 2 |
| 0 | 3 | 1 | 0 | 2 | 0 |

| | | |
|---|---|---|
| 2 | 3 | 2 |
| 3 | 2 | 2 |
| | | |

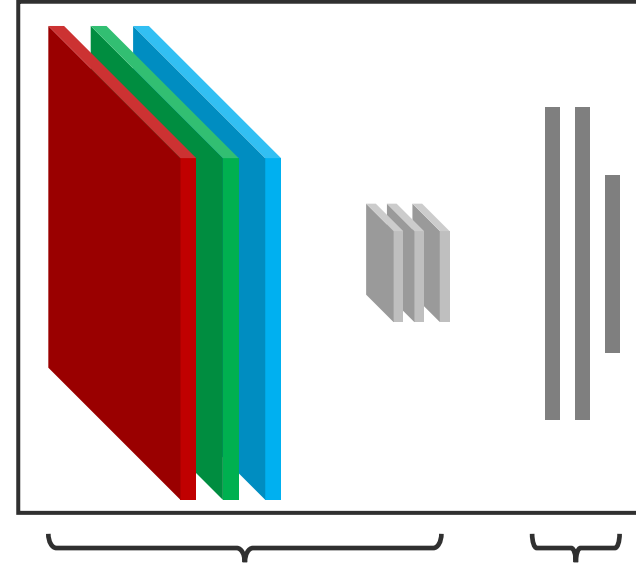
最大値プーリング演算



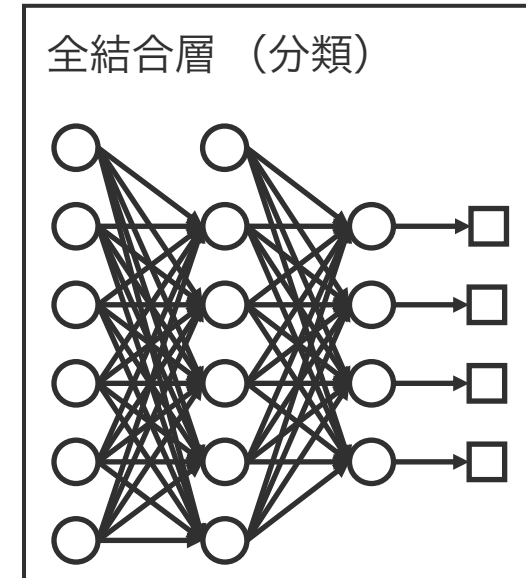
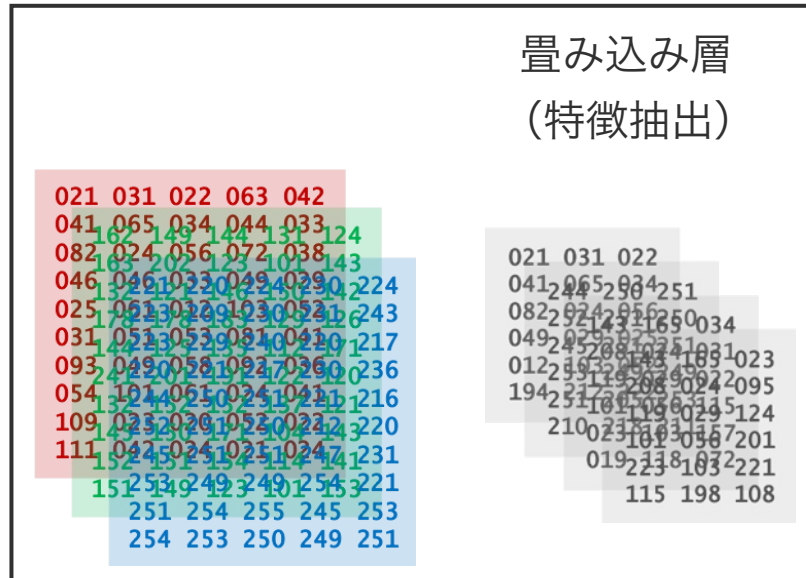
畳み込みニューラルネットワーク



畳み込みニューラルネットワーク



airplane 56.1%
 sky 25.0%
 cloud 12.4%
 mountain 6.5%



畳み込みニューラルネットワーク

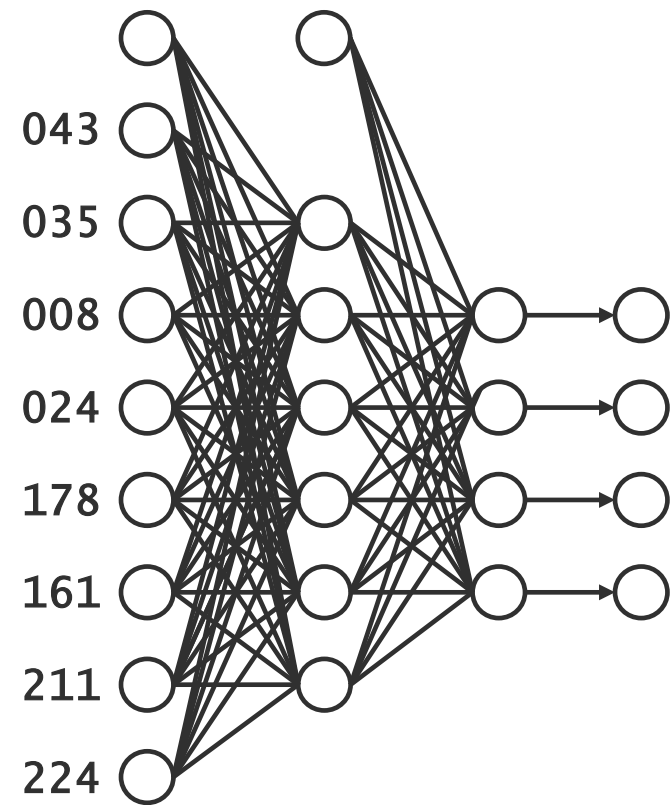
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 031 049 029 024
025 061 010 030 053 024
031 051 053 081 041 021
093 049 058 092 036 026
054 104 061 024 041 021
109 025 020 058 002 022
111 044 024 021 024 021
151 149 123 101 153 221
251 254 255 245 253
254 253 250 249 251



021 031 022
041 065 034 251
082 024 056 250 034
049 029 02 251 03 1023
012 103 05 249
194 251 10 08 024 095
210 023 10 096 157 201
019 218 072 221
115 198 108



043 035
008 024
178 161
211 224

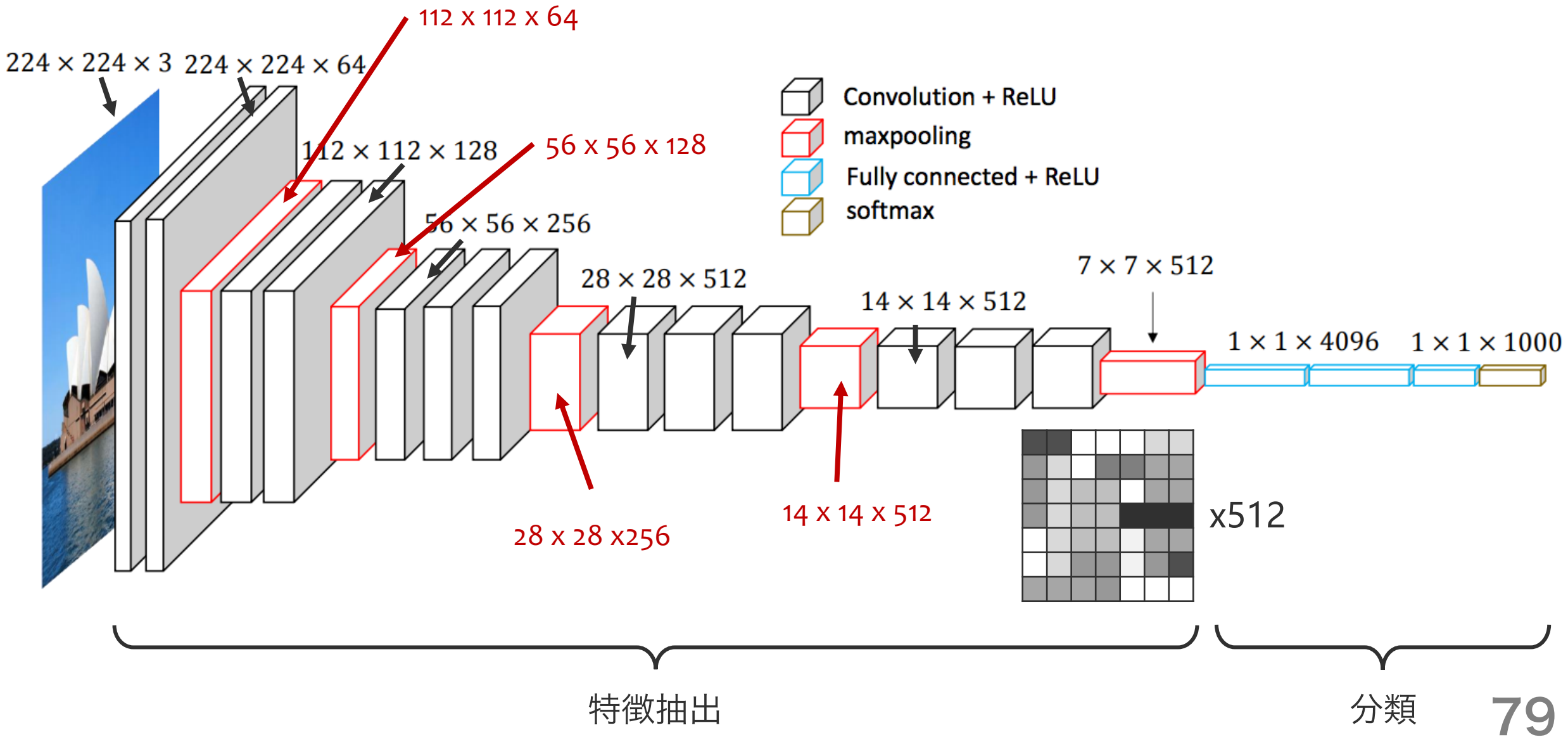


畳み込み層

全結合層

VGG16

<https://doi.org/10.1145/3038912.3052638>

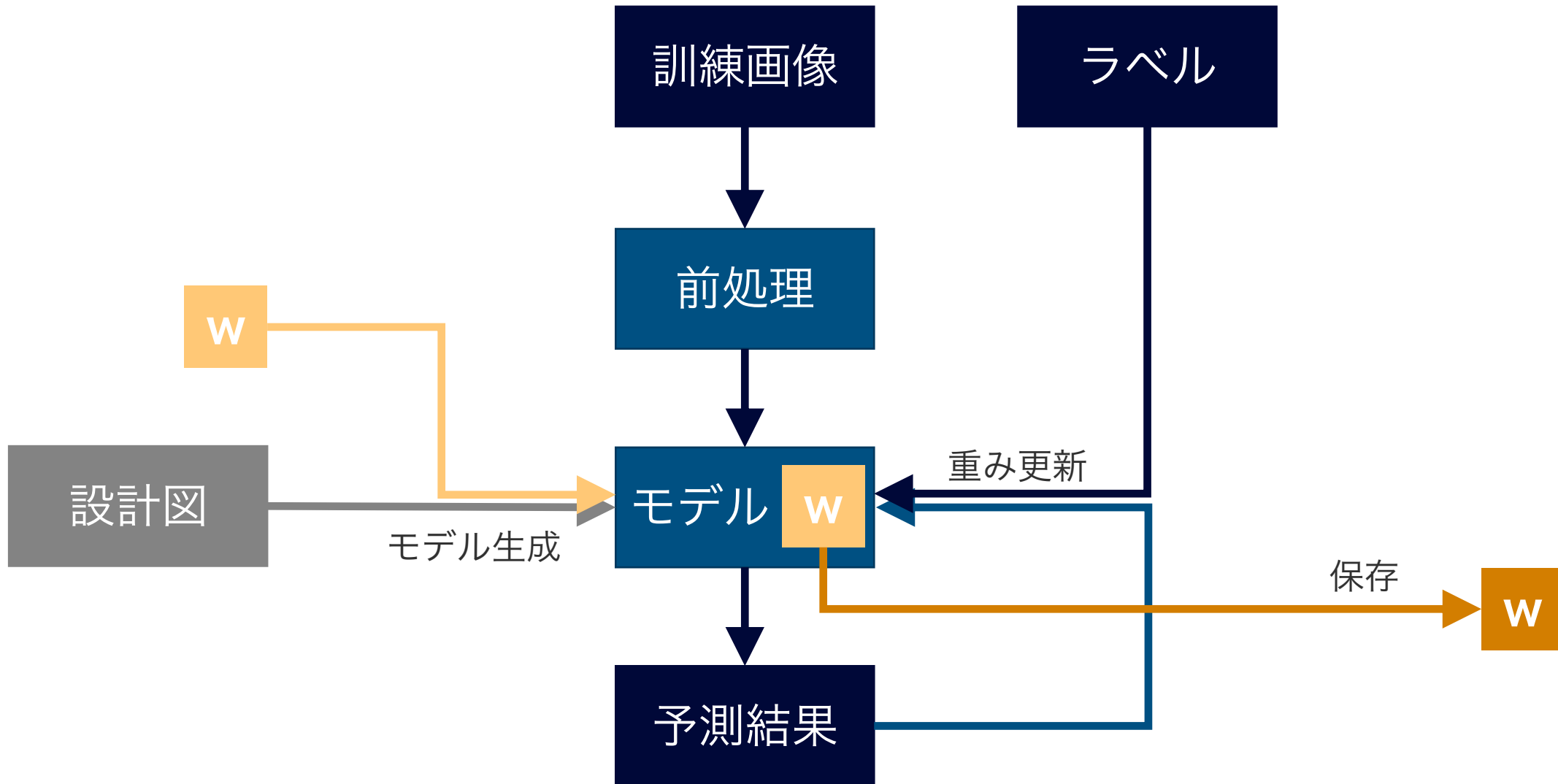


農学生命情報科学特論 I

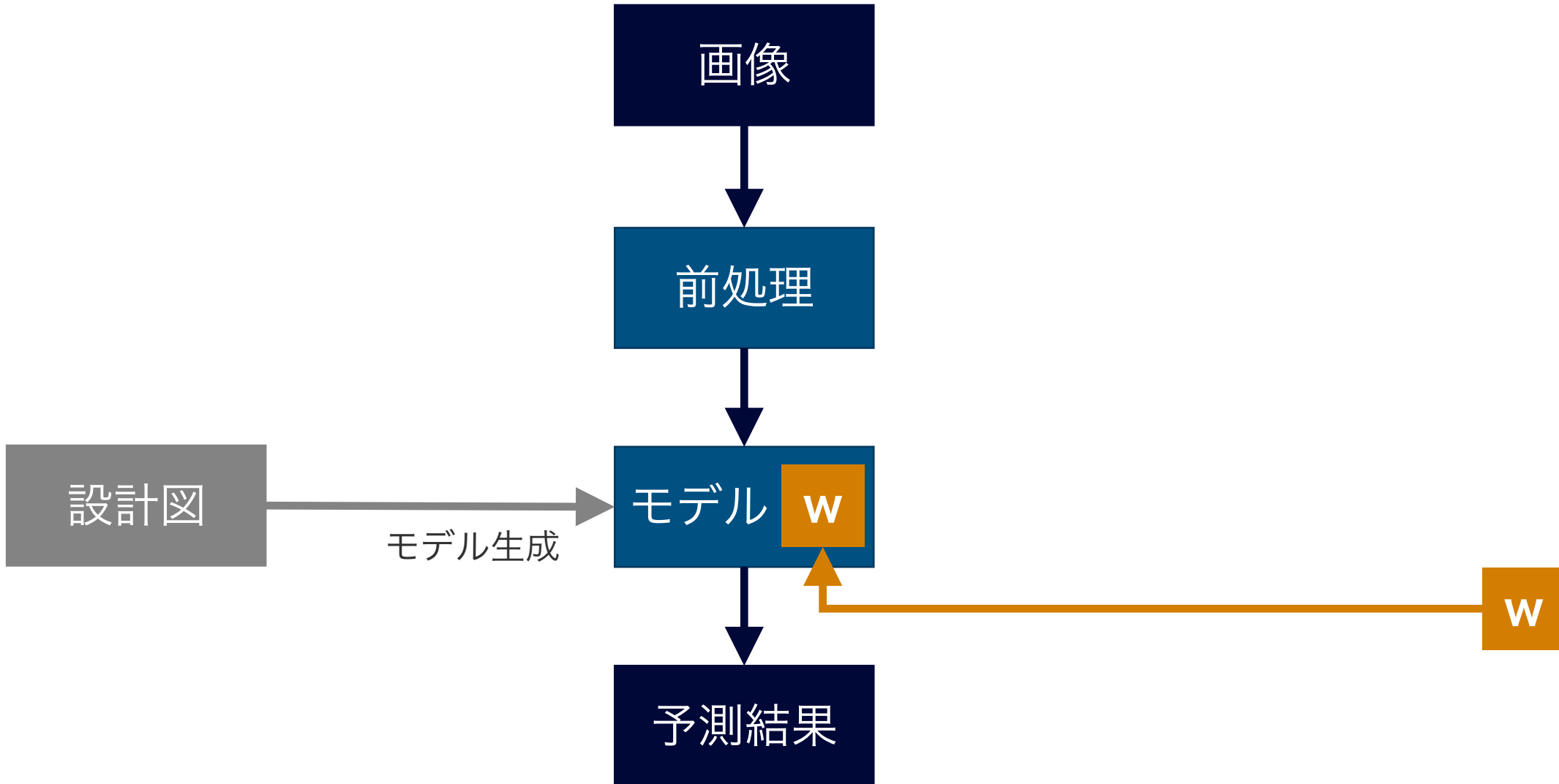


- 機械学習
- 画像解析・物体認識
- 物体認識モデル実装

訓練プロセス



推論プロセス



モデル設計

```
class SimpleNet(torch.nn.Module):

    def __init__(self):
        super().__init__()
        self.nn_inputs = (((((224-5+1)/2)-5+1)/2)^2)*32

        self.conv1 = torch.nn.Conv2d(3, 16, 5)
        self.pool1 = torch.nn.MaxPool2d(2, 2)
        self.conv2 = torch.nn.Conv2d(16, 32, 5)
        self.pool2 = torch.nn.MaxPool2d(2, 2)
        self.fc1 = torch.nn.Linear(self.nn_inputs, 512)
        self.fc2 = torch.nn.Linear(512, 64)
        self.fc3 = torch.nn.Linear(64, 5)

    def forward(self, x):
        x = torch.nn.functional.relu(self.conv1(x))
        x = self.pool1(x)
        x = torch.nn.functional.relu(self.conv2(x))
        x = self.pool2(x)
        x = x.view(-1, self.nn_inputs)
        x = torch.nn.functional.relu(self.fc1(x))
        x = torch.nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

モデル設計

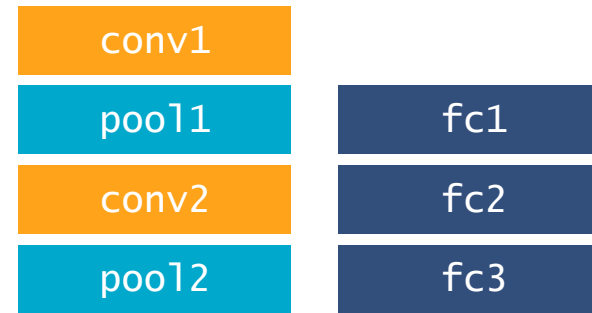
```
class SimpleNet(torch.nn.Module):
```

```
def __init__(self):
    super().__init__()
    self.nn_inputs = (((((224-5+1)/2)-5+1)/2)^2)*32

    self.conv1 = torch.nn.Conv2d(3, 16, 5)
    self.pool1 = torch.nn.MaxPool2d(2, 2)
    self.conv2 = torch.nn.Conv2d(16, 32, 5)
    self.pool2 = torch.nn.MaxPool2d(2, 2)
    self.fc1 = torch.nn.Linear(self.nn_inputs, 512)
    self.fc2 = torch.nn.Linear(512, 64)
    self.fc3 = torch.nn.Linear(64, 5)
```

```
def forward(self, x):
    x = torch.nn.functional.relu(self.conv1(x))
    x = self.pool1(x)
    x = torch.nn.functional.relu(self.conv2(x))
    x = self.pool2(x)
    x = x.view(-1, self.nn_inputs)
    x = torch.nn.functional.relu(self.fc1(x))
    x = torch.nn.functional.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

モデル構築に必要な部品を準備



モデル設計

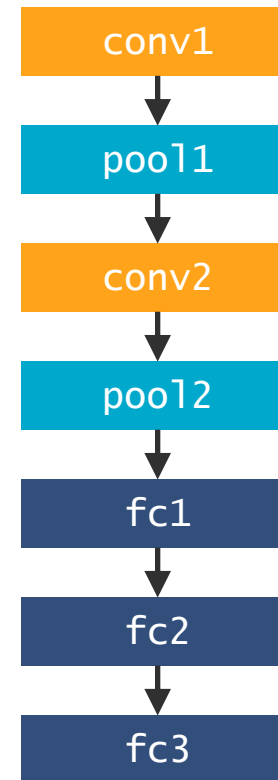
```
class SimpleNet(torch.nn.Module):
```

```
    def __init__(self):
        super().__init__()
        self.nn_inputs = (((((224-5+1)/2)-5+1)/2)^2)*32

        self.conv1 = torch.nn.Conv2d(3, 16, 5)
        self.pool1 = torch.nn.MaxPool2d(2, 2)
        self.conv2 = torch.nn.Conv2d(16, 32, 5)
        self.pool2 = torch.nn.MaxPool2d(2, 2)
        self.fc1 = torch.nn.Linear(self.nn_inputs, 512)
        self.fc2 = torch.nn.Linear(512, 64)
        self.fc3 = torch.nn.Linear(64, 5)
```

```
    def forward(self, x):
        x = torch.nn.functional.relu(self.conv1(x))
        x = self.pool1(x)
        x = torch.nn.functional.relu(self.conv2(x))
        x = self.pool2(x)
        x = x.view(-1, self.nn_inputs)
        x = torch.nn.functional.relu(self.fc1(x))
        x = torch.nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

部品同士を繋ぎデータの流れを定義



モデル設計

```
class SimpleNet(torch.nn.Module):
```

```
    def __init__(self):
        super().__init__()
        self.nn_inputs = (((((224-5+1)/2)-5+1)/2)^2)*32

        self.conv1 = torch.nn.Conv2d(3, 16, 5)
        self.pool1 = torch.nn.MaxPool2d(2, 2)
        self.conv2 = torch.nn.Conv2d(16, 32, 5)
        self.pool2 = torch.nn.MaxPool2d(2, 2)
        self.fc1 = torch.nn.Linear(self.nn_inputs, 512)
        self.fc2 = torch.nn.Linear(512, 64)
        self.fc3 = torch.nn.Linear(64, 5)
```

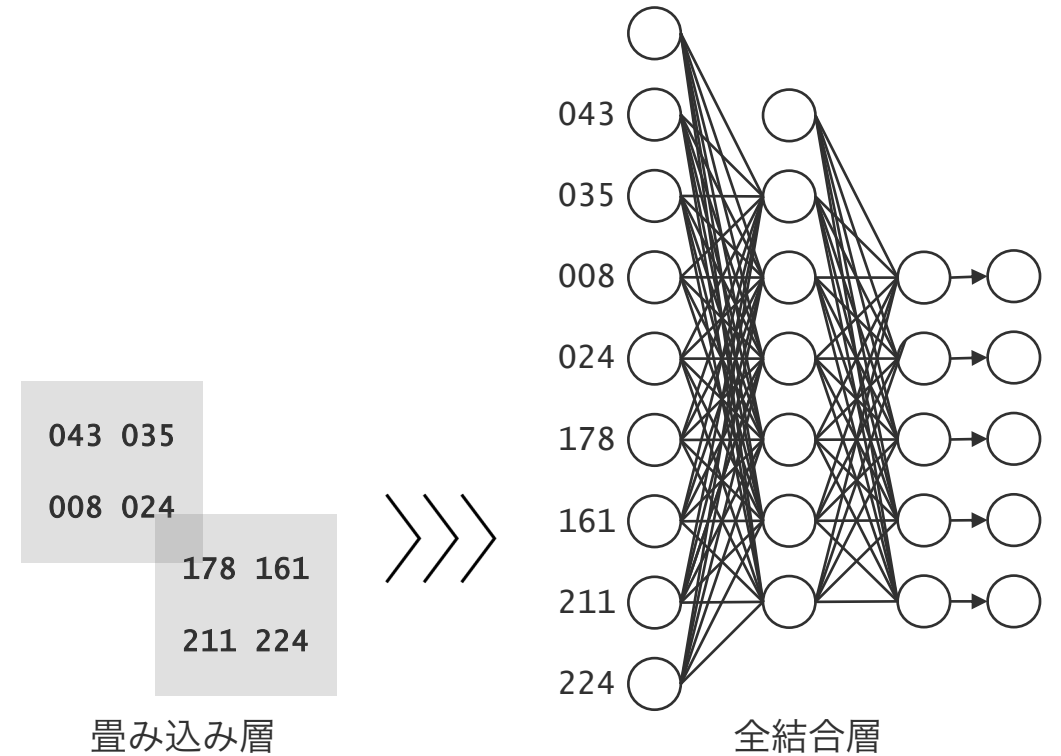
畳み込み層の出力数を計算



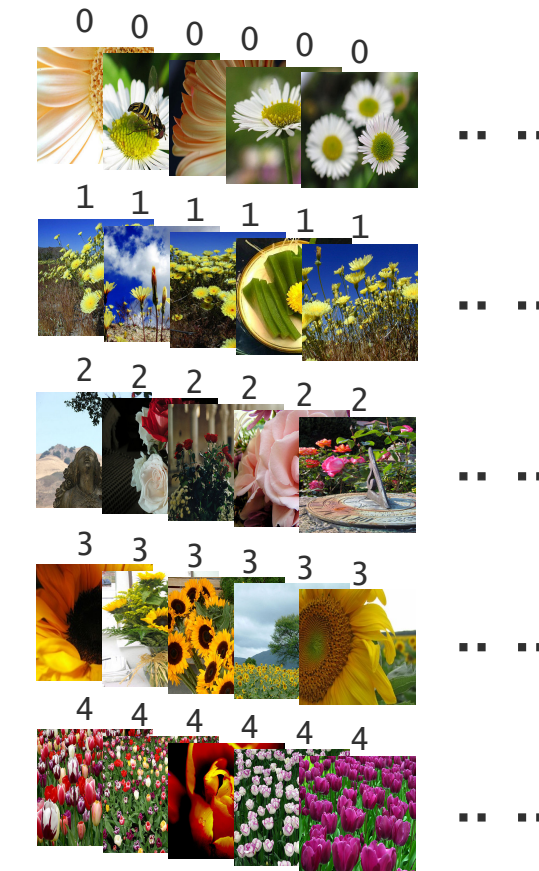
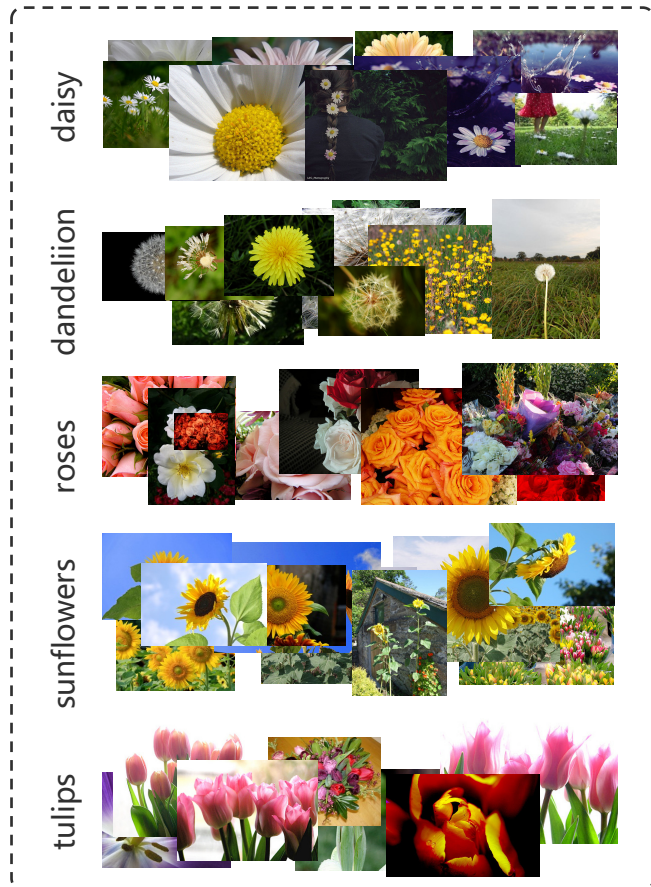
```
    def forward(self, x):
        x = torch.nn.functional.relu(self.conv1(x))
        x = self.pool1(x)
        x = torch.nn.functional.relu(self.conv2(x))
        x = self.pool2(x)
        x = x.view(-1, self.nn_inputs)
        x = torch.nn.functional.relu(self.fc1(x))
        x = torch.nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

畳み込み層が出力する
行列をベクトルに変換

畳み込み層ニューラルネットワーク



前処理



前処理



- 訓練画像を整理し、画像と教師ラベルの対応づけを行う、全訓練画像のリストを作成する。
- 画像に対する前処理を定義する。
 - 画像を行列データとして読み取る
 - サイズ変更、色調調整、アフィン変換
 - テンソルに変換
- リストの i 番目の画像がリクエストされたら、その画像を前処理して返す。



前処理



- 訓練画像を整理し、画像と教師ラベルの対応づけを行う、全訓練画像のリストを作成する。
- 画像に対する前処理を定義する。
 - 画像を行列データとして読み取る
 - サイズ変更、色調調整、アフィン変換
 - テンソルに変換
- リストの i 番目の画像がリクエストされたら、その画像を前処理して返す。

```
class Dataset:  
    def __init__():  
        self.x = images  
        self.y = labels  
        self.transform = transforms  
  
    def __len__():  
        return len(self.x)  
  
    def __getitem__(i):  
        x = self.transform(self.x[i])  
        y = self.y[i]  
        return x, y
```

```
class Resize:  
    def __init__(x):  
        self.x = x  
    def __call__(x):  
        x = resize(x)  
        x
```

```
class Affine:  
    def __init__(x):  
        self.x = x  
    def __call__(x):  
        x = affine(x)  
        x
```

```
class Tensor:  
    def __init__(x):  
        self.x = x  
    def __call__(x):  
        x = tensor(x)  
        x
```

```
transforms = [  
    Resize(),  
    Affine(),  
    Tensor(),  
]
```

バッチ化



- 画像リストを取得し、必要に応じて順番を調整し、バッチ化の処理を行う。

```
train_dataset = Dataset(images, labels)
train_dataloader = DataLoader(
    train_dataset,
    batch_size=16, shuffle=True, num_workers=2)
```

```
class Dataset:
    def __init__(images, labels):
        self.x = images
        self.y = labels
        self.transform = transforms

    def __len__():
        return len(self.x)

    def __getitem__(i):
        x = self.transform(self.x[i])
        y = self.y[i]
        return x, y
```

```
class Resize:
    def __init__(x):
        self.x = x
    def __call__(x):
        x = resize(x)
        x
```

```
class Affine:
    def __init__(x):
        self.x = x
    def __call__(x):
        x = affine(x)
        x
```

```
class Tensor:
    def __init__(x):
        self.x = x
    def __call__(x):
        x = tensor(x)
        x
```

```
transforms = [
    Resize(),
    Affine(),
    Tensor(),
]
```

モデル訓練



- 小分けした画像データを少しずつモデルに代入し訓練を行う。

```
train_dataset = Dataset(images, labels)
train_dataloader = DataLoader(
    train_dataset,
    batch_size=16, shuffle=True, num_workers=2)
```

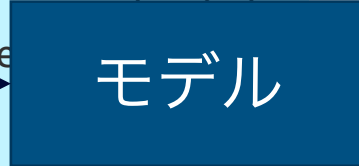
```
for inputs, labels in train_dataloader:
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

```
class Dataset:
    def __init__(self, images, labels):
        self.x = images
        self.y = labels
        self.transform = transforms

    def __len__():
        return len(self.x)

    def __getitem__(i):
        x = self.transform(self.x[i])
        y = self.y[i]
        return x, y
```

```
class Resize:
    def __init__(x):
        self.x = x
```



```
class Affine:
    def __init__(x):
        self.x = x
    def __call__(x):
        x = affine(x)
```

```
class Tensor:
    def __init__(x):
        self.x = x
    def __call__(x):
        x = tensor(x)
```

```
transforms = [
    Resize(),
    Affine(),
    Tensor(),
]
```

モデル訓練

```
# dataset
train_dataset = Dataset(images, labels)
train_data_loader = DataLoader(train_dataset, batch_size=4)

# model
model = SimpleCNN()
model.train()

# parameters
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
n_epochs = 50

# training
for epoch in range(n_epochs):
    running_loss = 0.0

    # mini batch
    for inputs, labels in train_data_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print('Epoch: {}; Loss: {}'.format(epoch, running_loss))
```

- モデルを訓練モードに切り替え、計算時に生じた勾配情報を保存する。

モデル訓練

```
# dataset
train_dataset = Dataset(images, labels)
train_data_loader = DataLoader(train_dataset, batch_size=4)

# model
model = SimpleCNN()
model.train()

# parameters
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.001)
n_epochs = 50

# training
for epoch in range(n_epochs):
    running_loss = 0.0

    # mini batch
    for inputs, labels in train_data_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print('Epoch: {}; Loss: {}'.format(epoch, running_loss))
```

- 損失関数
 - 多クラス分類問題では一般的にクロスエントロピー関数を用いる。
 - 回帰分析では MSELoss などを使用する。
 - 独自で定義した関数を使用することもできる。その際、outputs と labels を受け取り、1 つの値を返す関数を定義すればよい。
- 最適化アルゴリズム
 - SGD や Adam などがよく使われる。
- エポック
 - 過学習を起こさない程度のエポックを設定。
 - 検証データを使用して early stopping を組み込む。

モデル訓練

```
# dataset
train_dataset = Dataset(images, labels)
train_data_loader = DataLoader(train_dataset, batch_size=4)

# model
model = SimpleCNN()
model.train()

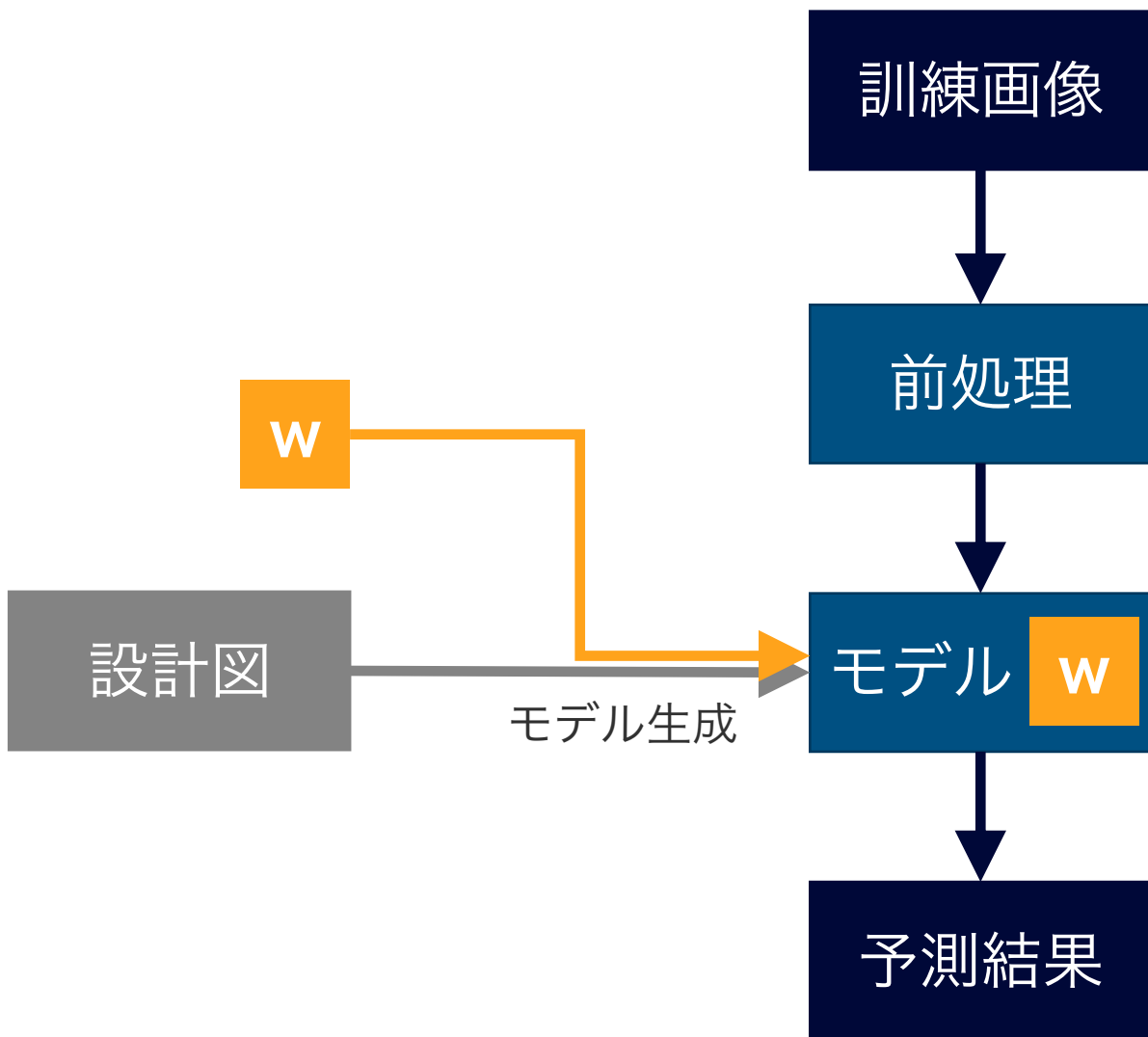
# parameters
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
n_epochs = 50

# training
for epoch in range(n_epochs):
    running_loss = 0.0

    # mini batch
    for inputs, labels in train_data_loader:
        optimizer.zero_grad()           # 1
        outputs = model(inputs)          # 2
        loss = criterion(outputs, labels) # 3
        loss.backward()                  # 4
        optimizer.step()                 # 5
        running_loss += loss.item()

    print('Epoch: {}; Loss: {}'.format(epoch, running_loss))
```

1. 古い損失（勾配情報）を消去する。
2. 画像データをモデルに代入し予測する。
3. 予測値と教師ラベルから損失（誤差）を計算する。
4. 損失をネットワーク全体に逆伝播する。
5. 逆伝播された損失を利用してパラメータを更新する。



```
# model
model = SimpleCNN()
model.load_state_dict(torch.load('weight.pth'))
model.eval()
```

```
# preprocess an image
img_fpath = '/path/to/image.jpg'
im = PIL.Image.open(img_fpath)
im = ImageOps.exif_transpose(im)
im = inference_transform(im)
im = im.unsqueeze(0)
```

```
# inference
output = model(input)
# [[0.0935, -5.4933, -1.1156, 6.0684, 0.0409]]
```

```
output_softmax = torch.softmax(output, dim=1)
# [[0.0025, 0.0000, 0.0001, 0.9950, 0.0023]]
```

```
output_sigmoid = torch.softmax(output)
# [[0.5233, 0.0041, 0.2468, 0.9977, 0.5102]]
```

モデル構造

```
from torch.nn as nn
from torchvision import models
```

```
model = models.vgg16(pretrained=True)
```

```
print(model)
```

classifier[5] の出力数 4096 を受け取る。



```
model.classifier[6] = nn.Linear(4096, 5)
```



出力数を 5 に設定する。

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```


モデル構造

```
from torch.nn as nn
from torchvision import models

model = models.resnet18(pretrained=True)
```

```
print(model)
```

avgpool の出力数 512 を受け取る。



```
model.fc = nn.Linear(512, 5)
```



出力数を 5 に設定する。

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

モデル構造

```
from torch.nn as nn
from torchvision import models
```

```
model = models.vgg16(pretrained=True)
model.classifier[6] = nn.Linear(4096, 5)
```

```
for param in model.features.parameters():
    param.requires_grad = False
```

```
vgg(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

モデル構造

```
from torch.nn as nn
from torchvision import models
```

```
model = models.vgg16(pretrained=True)
model.classifier[6] = nn.Linear(4096, 5)
```

```
for param in model.features.parameters():
    param.requires_grad = False
```

```
for param in model.avgpool.parameters():
    param.requires_grad = False
```

VGG(

```
(features): Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace=True)
  (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace=True)
  (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (25): ReLU(inplace=True)
  (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): ReLU(inplace=True)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace=True)
  (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
(classifier): Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace=True)
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace=True)
  (5): Dropout(p=0.5, inplace=False)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
)
)
```

モデル構造

```
from torch.nn as nn
from torchvision import models

model = models.vgg16(pretrained=True)
model.classifier[6] = nn.Linear(4096, 5)

for i, param in \
    enumerate(model.features.parameters()):

    param.requires_grad = False

    if i > 15:
        break
```

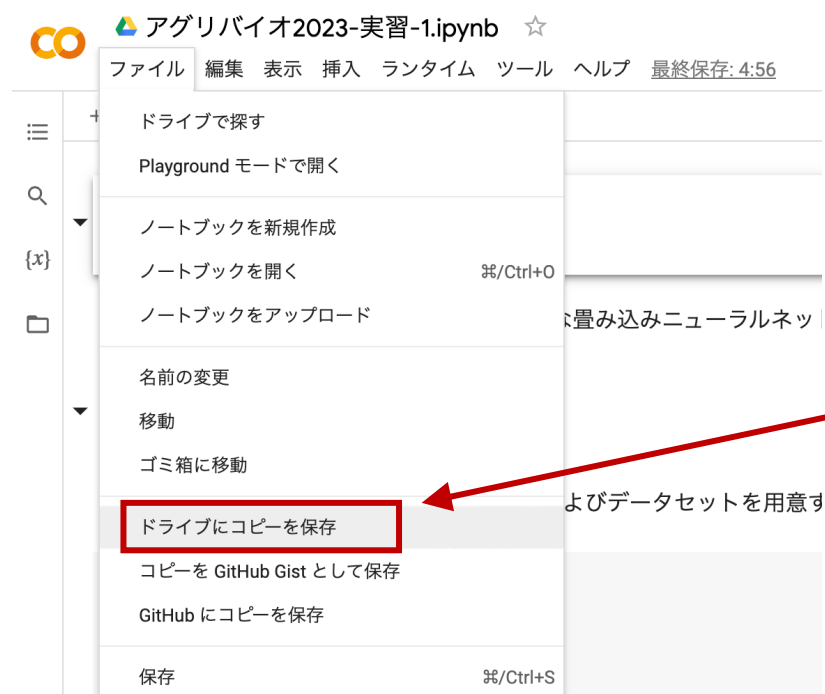
VGG(

```
(features): Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace=True)
  (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace=True)
  (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (25): ReLU(inplace=True)
  (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): ReLU(inplace=True)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace=True)
  (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
(classifier): Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace=True)
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace=True)
  (5): Dropout(p=0.5, inplace=False)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
)
)
```

ノートブック

実習 1 <https://colab.research.google.com/drive/1aLUcbYPVMg8Ga1KJc6IvwqqkGcPD9P0s>

実習 2 https://colab.research.google.com/drive/18P9C1HpZm6_47UqYMRyPIIdG-dq3y9ZH9



URL にアクセスした直後のデータは孫が所有者となっているため、他の人が編集できない。一度「ファイル」より「ドライブにコピーを保存」をクリックして自分のドライブにコピーしてください。クリック後、コピーされたファイルが自動的に表示されます。表示されない場合は、グーグルドライブを確認して、該当ファイルをクリックして開いてください。