1　畳み込みニューラルネットワーク

機械学習は通常、下図のようにニューロンと呼ぶ神経細胞を模倣したノードを大量に接続したニューラルネットワークを構築することで種々の問題を解くためのシステムを構築する

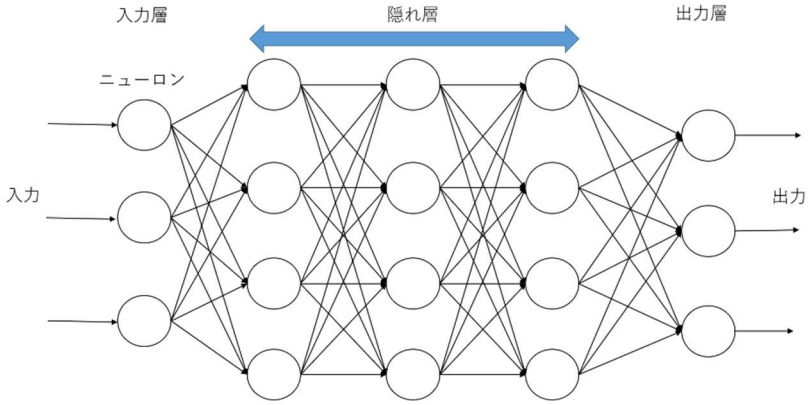

図　ニューラルネットワークの概要図

画像分類問題のような画像データに関連する問題に向けて、通常のニューラルネットワークではなく、畳み込みニューラルネットワーク（Convolutional Neural Network=CNN）と呼ばれる特殊なネットワークが提案されている

これは下図のように畳み込み層とプーリング層、全結合層という３つの層で構成され、フィルタ処理と縮小処理を繰り返して画像の本質的な特徴を抽出したあとに全結合層に送られ、通常のニューラルネットワークの場合と同様に画像分類される

画像分類に用いられることが多いが、近年では画像生成や自然言語処理といった問題でも有効性が確認され、高い汎用性のもとに広く用いられている

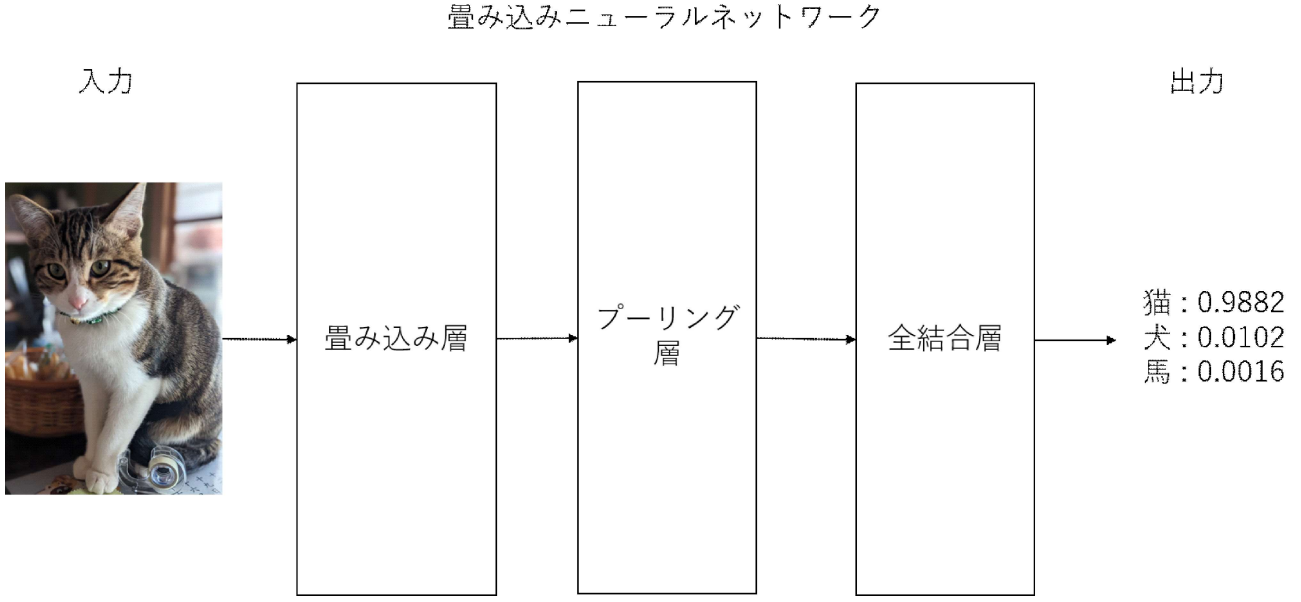

図　畳み込みニューラルネットワークの概要図

２　畳み込み層

畳み込み層では「畳み込み」というある種のフィルタ処理が実施される

これは図のような「カーネル」と呼ばれる格子状に数値が配置されたフィルタを用いて、画像の各層の値を格子状の値で定数倍したものの総和を取る形で画素値を演算することで輪郭などの特徴を抽出するものである

0	0	0
0	1	-1
0	0	0

図　カーネルの例

図　畳み込みのイメージ

例えば下のような画素値をもつ 4× 4 のサイズの画像に対して、2× 2 のカーネルを用いて畳み込みを行う例を考える

2	2	1	2
1	1	2	1
1	2	1	0
1	0	0	1

図　画像の例（画素値で表現）

1	0
0	1

図　カーネルの例

図のように画像に対して、左上からカーネルを重ね、カーネルの値とカーネルが重ねっている箇所の画素の値の積を計算して、その和を計算する

この図の場合、黄色でフォーカスしたセルの画素値と対応するカーネルの値の積和を計算する

よってその積和は　2× 1 ＋2× 0 ＋ 1 × 0 ＋1× 1 ＝ 3 となる

積和を計算した結果を画像の画素値として新しい画像の画素として埋める

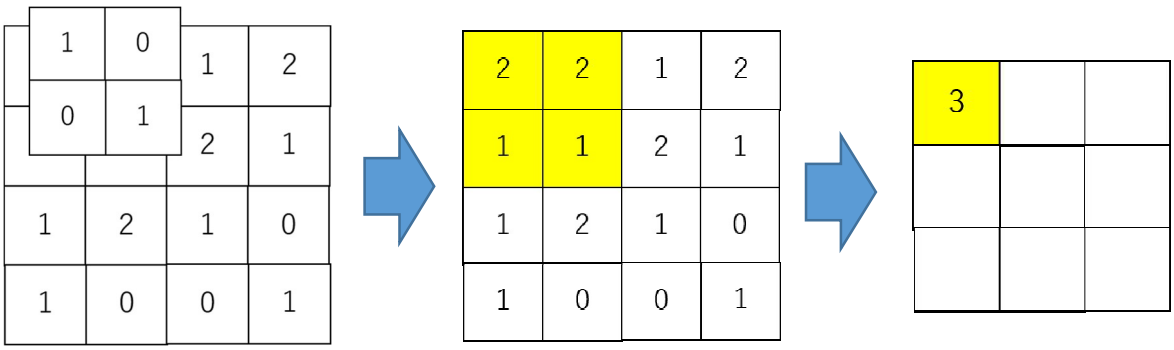

図　畳み込み計算の例 1

さらにカーネルを右に 1 ピクセルずらして同様に積和を計算する
この図の場合はオレンジ色でフォーカスしたセルの画素値と対応するカーネルの値の積和を計算する
よってその総和は 2× 1 +1× 0 +1× 0 +2× 1 = 4 となる
先の場合同様に求めた積和の値を画素値として埋める

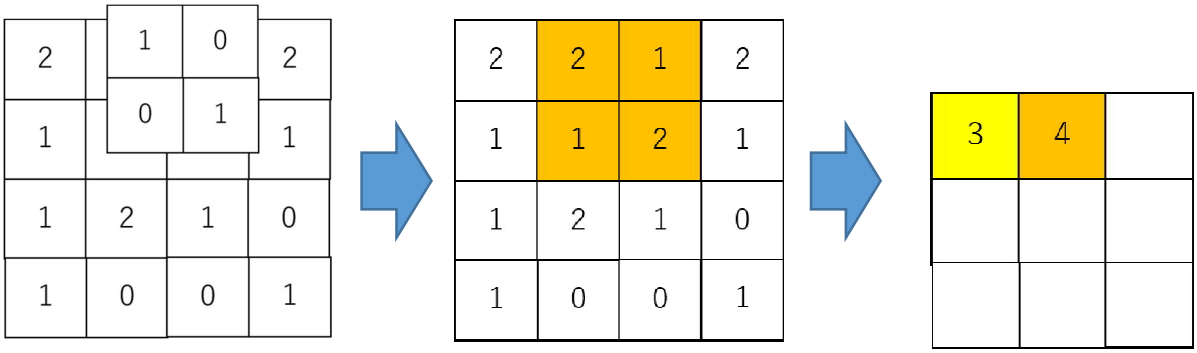

図　畳み込み計算の例 2

以上の積和計算をカーネルを移動させながら行うと図のような形で新たな画像を得ることができる
画像の隣り合うピクセルは互いに強い関連性を持ち、このような形で隣接する画素の値をまとめるとその画像に写っている物体の輪郭など特徴を捉えることができる
このような関連性のことを局所性とも呼び、カーネルのサイズやその値を変更することで様々な特徴を抽出することができる

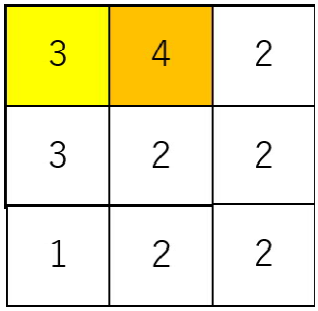

図　畳み込み結果の例

一般に画像は図のように RGB の三原色で構成されており、1 ピクセルにつき 3 つのデータが格納される
このデータ数をチャンネル数と呼び、RGB のカラー画像のチャンネル数は 3 ch となる

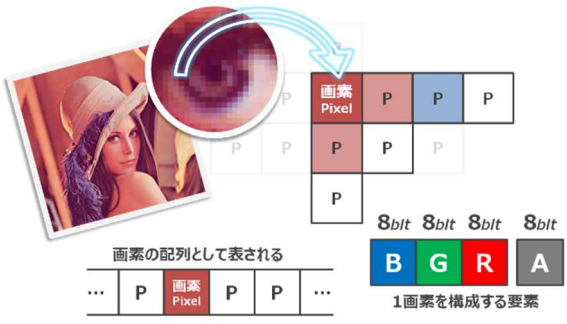

図　画像のデータ構成（RGB 表色系の例）

RGB の 3 チャンネル画像においては 3 原色となる RGB の割合を変更することで様々な色を作成しており、R の情報のみの画像と G の情報のみの画像、B の情報のみの画像を重ねることでカラー画像を構成している

図　RGB 三原色画像とカラー画像の生成

つまり、先の畳み込みにおいては R,G.B それぞれのチャンネルの画像に対してカーネルを用いてフィルタ処理しており、これを統合することで画像の局所的な特徴を抽出する
カーネルの種類もひとつのみではなく複数の種類を用いて統合することで様々な特徴の抽出を図る

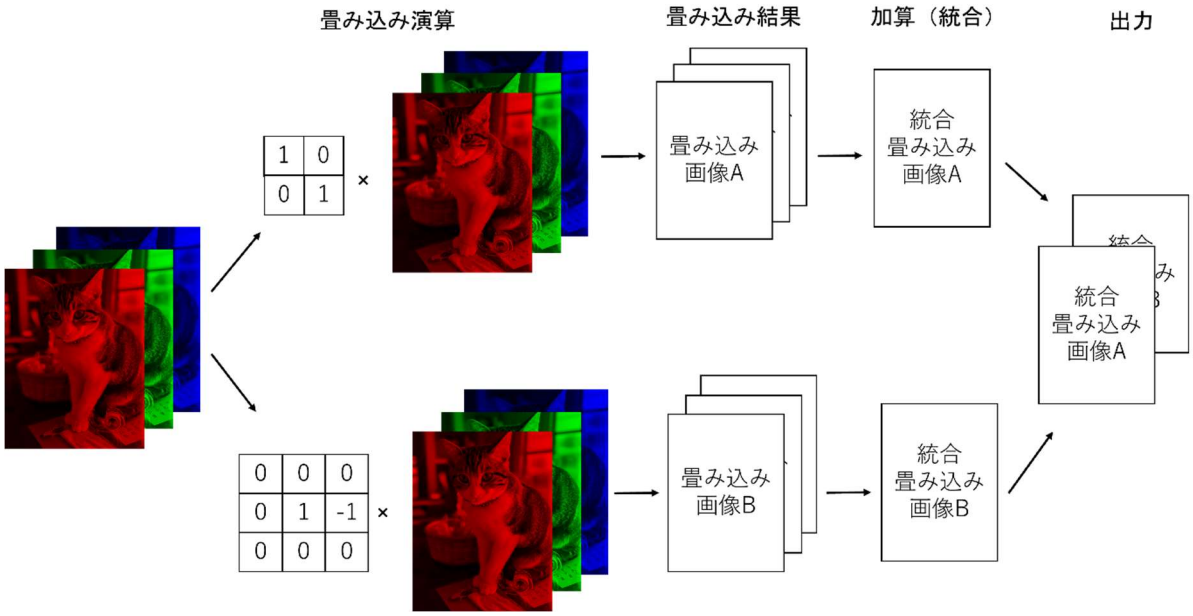

図　3 ch 画像に対する畳み込み処理の流れ（カーネル 2 個の場合）

Pytorch では畳み込み層は nn モジュールの機能を用いて実装する

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(3,6,5)

    def forward(self,x):
        x = self.conv(x)

        return x
```

nn.Conv2d()が畳み込み層を生成するための API であり、以下のパラメータを設定する

nn.Conv2d(画像のチャンネル数，カーネルの数，カーネルのサイズ)

上記の例は 3ch 画像で、6 個のカーネルを使用し、カーネルのサイズが 5×5 という意味である
その後、畳み込み演算のためのメソッド（この例では forward()）を作成し、その内部で
メンバ変数の形で呼び出すことで実装している

3　プーリング層

プーリングは図のように画像をあるサイズの領域に分割し、この分割された領域内の画素値のうち代表的な画素の値を取り出すことで新たな画像を生成する処理である
代表値の選び方としてその領域内の最大値を採用する方法を「MAX プーリング」と呼ぶ

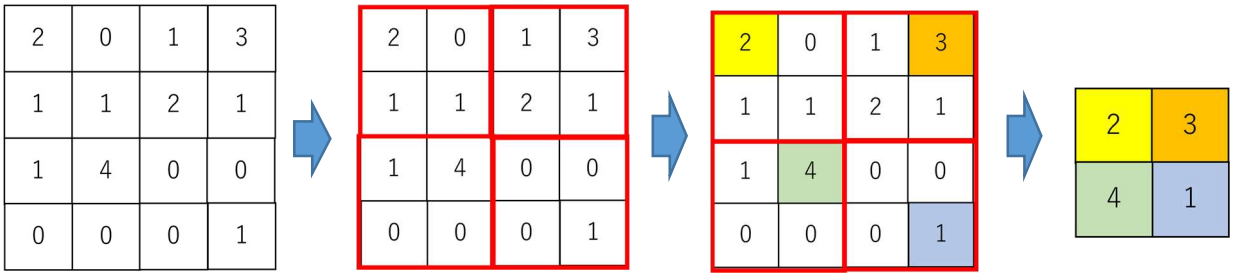

図　プーリング処理の流れ

隣接する複数のピクセルのうち代表的なもののみを取り出すことで画像全体をいわゆる「ぼかす」ことになり、ぼかすことで画像の局所的な細かい詳細な特徴ではなく画像全体のおおまかな特徴を捉えることが可能となる

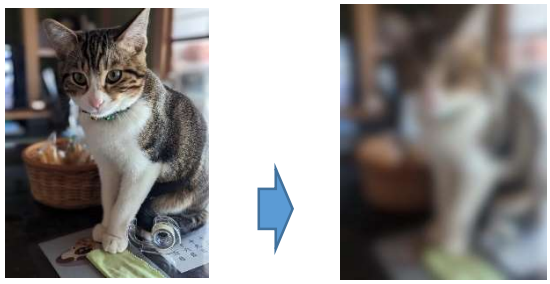

図　プーリング処理のイメージ

Pytorch ではプーリング層は畳み込み層同様に nn モジュールの機能を用いて実装する

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.pool = nn.MaxPool2d(2,2)

    def forward(self,x):
        x = self.pool(x)

        return x
```

nn.MaxPool2d()が畳み込み層を生成するための API であり、以下のパラメータを設定する

nn.MaxPool2d(領域サイズ，ストライド数)

上記の例は 2× 2 のサイズの領域に分割し、ストライド数（後述）を 2 として隣の領域にシフトさせる形でプーリングを実施するという意味である

その後、クラス内で畳み込み演算のためのメソッド（この例では forward()）を作成し、その内部でメンバ変数の形で呼び出すことで実装している

4　パディング

CNN において畳み込みやプーリングを行うと出力されるデータは元の画像データよりも縮小された形となってしまう

CNN では畳み込みとプーリングを繰り返し実行して特徴を抽出するため、最終的にデータの縦横サイズは 1× 1 となってしまう問題が発生する

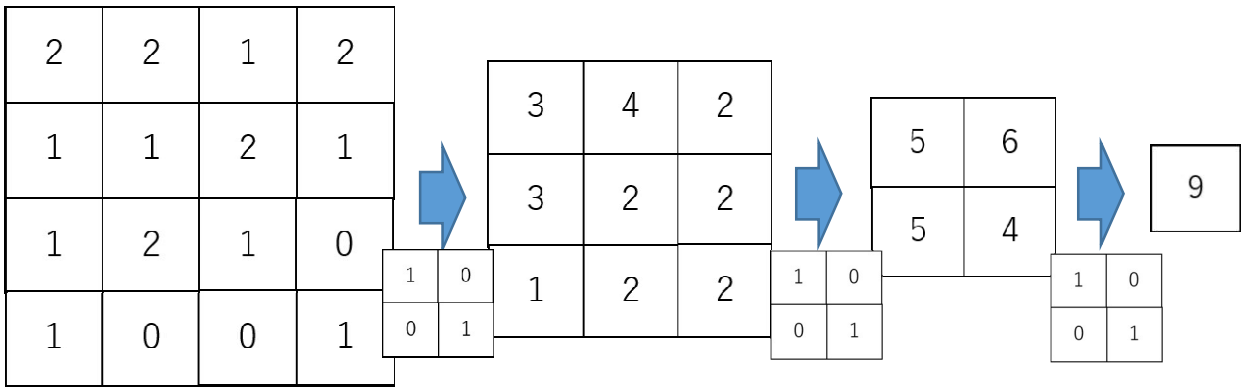

図　畳み込みによるデータの縮小

このため、CNN においては画像データに対して周囲を何らかの値をもつピクセルで埋めるパディングという手法でデータのサイズを変更する操作が行われる

パディングにおいて、0 の画素値で埋める手法を特にゼロパディングと呼ぶ

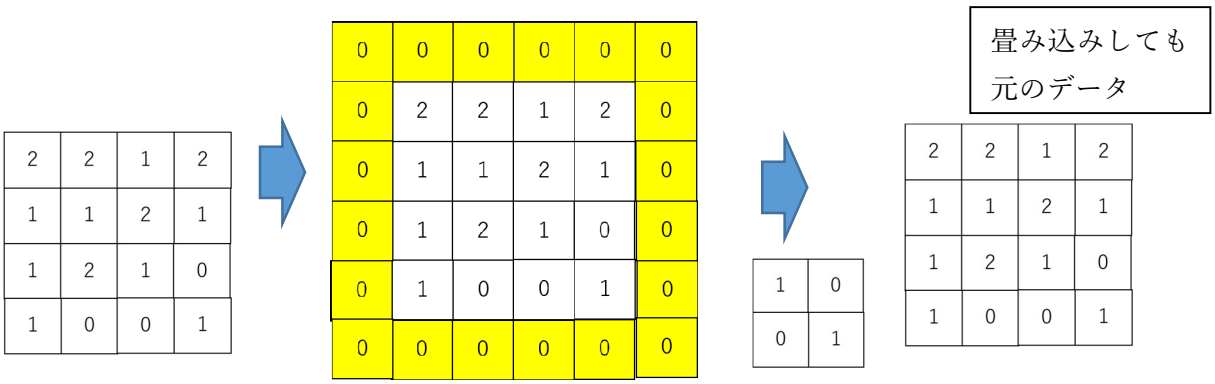

図　データのゼロパディング

5　ストライド

畳み込みにおいてカーネルのピクセル移動量をストライドと呼ぶ

通常はストライドは 1 ピクセル移動するストライド 1 で計算されることが多いが、2 ピクセル移動させるストライド 2 で計算する場合もある

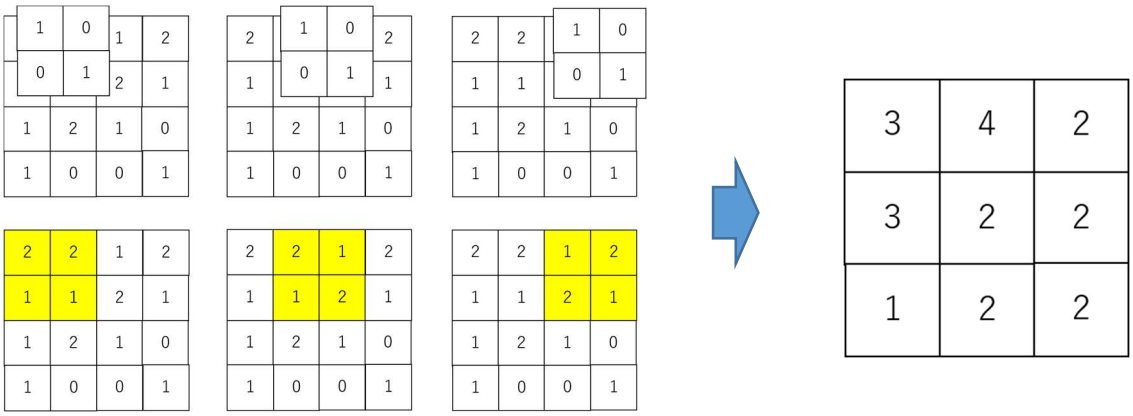

図　ストライド 1 の場合の畳み込み

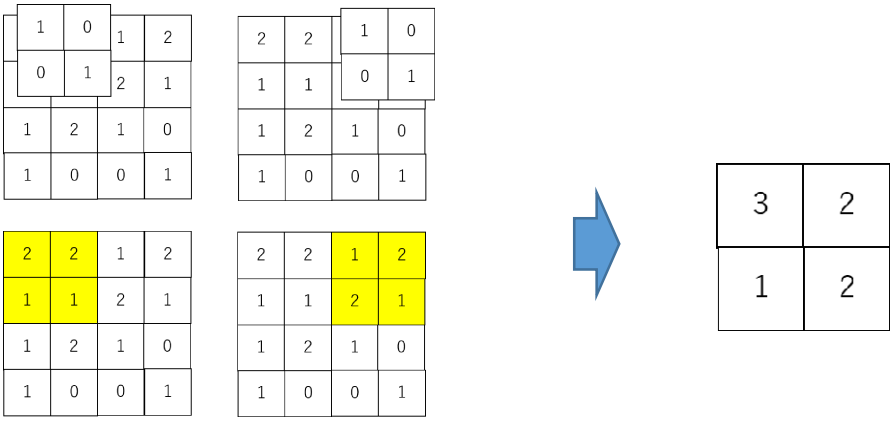

図　ストライド 2 の場合の畳み込み

図をみれば明らかであるが、ストライド量が大きい畳み込みほど出力される畳み込み画像のサイズは小さくなることに注意

入力画像の縦横サイズをそれぞれ H_I、W_I
カーネルの縦横サイズをそれぞれ H_K 、W_K,
パディングの幅をD
ストライド量をS
とすると出力される畳み込み画像の縦横サイズ H_C、W_C は

$$H_C = \frac{H_I - H_K + 2D}{S} + 1$$

$$W_C = \frac{W_I - W_K + 2D}{S} + 1$$

となり、ＣＮＮのモデル構築の際に利用される

６　ドロップアウト

CNN やニューラルネットワークを用いた学習においては繰り返し逆誤差伝播によって勾配を求めて最適なパラメータを求めるという計算を行うが、繰り返し数が多くなると特定のニューロンのみが強く反応するようになってしまい（いわゆる癖のようなもの)、ネットワークの性能が大きく低下する事態を引き起こすこと過学習が発生してしまう
従って、機械学習においては適切な学習回数を設定する必要があるが、問題によって適切な学習回数は異なるため一概にこの回数学習すればよいという指標を示すことは難しい
（個々人の経験、ノウハウや損失関数の値の振る舞いで調整されているのが実際のところ）

この問題を解決する一つの手段として、学習回数を調整するのではなくネットワークを構成しているニューロンを消去することで特定のニューロンにのみ過剰にデータが伝播されるのを抑止する手法が提案されている
この手法をドロップアウトと呼び、ランダムに出力層以外のニューロンを消去することで実現される

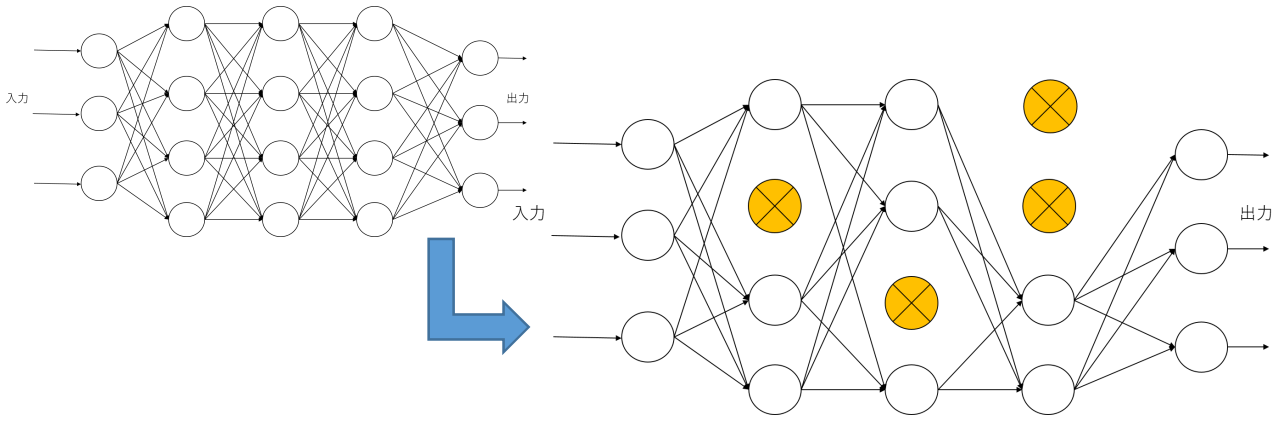

図　ドロップアウトのイメージ

Pytorch ではドロップアウは nn モジュールの機能を用いて実装される

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.dropout = nn.Dropout(p=0.5)

    def forward(self,x):
        x = self.dropout(x)

        return x
```

nn.Dropout ()がドロップアウトを実装するための API であり、以下のパラメータを設定する

nn.Dropout(ドロップアウト率)

上記の例は p=0.5 として層の 5 割を消去するという意味である

その後、クラス内で畳み込み演算のためのメソッド（この例では forward()）を作成し、その内部でメンバ変数の形で呼び出すことで実装している

7　全結合層

全結合層とは畳み込み層とプーリング層で出力されたデータをその名前とおり統合する層であり、Pytorch では nn モジュールの Linear()を用いることで実装できる

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = nn.Linear(256,10)

    def forward(self,x):
        x = self.fc(x)
```

nn.Dropout ()が全結合を実装するための API であり、以下のパラメータを設定する

nn.Linear(入力データの次元, 出力データの次元)

上記の例は入力データの次元数が 256 次元で出力が 10 次元という意味である

その後、クラス内で畳み込み演算のためのメソッド（この例では forward()）を作成し、その内部でメンバ変数の形で呼び出すことで実装している

８　CNN の実装

以上を踏まえて CNN のサンプをクラスとして実装してみる
ここでは一例として RGB の 3ch で 32×32 ピクセルの画像が入力され、これが 10 種類の物体に分類される例を考える
クラスとして Net のクラスを nn.Module を継承して定義し、メンバ変数としてコンストラクタに
畳み込み層２つ、プーリング層１つ、全結合層２つ、活性化関数として ReLU 関数、ドロップアウトを定義する

８－１　各層の定義

畳み込み第１層として入力画像が 3ch であるので、チャンネル数３、カーネルの数８つ（適当)、サイズ５（適当）を設定する
畳み込み第２層は畳み込み層の第１層のカーネル数が８であるため、出力されるデータ数は８となるため、チャンネル数は８、カーネルの数 16（適当)、カーネルサイズ５（適当）を設定する
プーリング層としては領域を２× ２のサイズで分割してストライド数は２（適当）とする
ドロップアウトは割合を 0.5（適当）として定義しておく
活性化関数としては ReLU 関数を設定する

畳み込み層第１層で 32×32 のサイズの画像が処理されると、画像サイズは 28×28 となる
さらにこれが２× ２の領域分割数、ストライド数２でプーリングされると 14×14 となる
その後、畳み込み層第２層で 14×14 のサイズの画像が処理されると、画像サイズは 10×10 となり、
これが上記と同様にプーリングされると画像サイズは５× ５となる

畳み込み層第２層のカーネル数 16 個であり、畳み込み層第２層のプーリング結果が５×５のサイズの画像であるため、全結合層としては合計として 16×５× ５を入力とし、256 次元（適当）を出力する層を全結合層第１層として定義する
全結合層第 １層のネットワークにドロップアウトを適用して最終的な出力を行う全結合層第２層を定義するため、入力を線結合層第１層の出力 256 次元に合わせて 256 次元、分類する画像の種類が 10 種類であるため出力を 10 次元とする

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        #畳み込み層1の定義（3ch,カーネル数8,カーネルサイズ5）
        self.conv1 = nn.Conv2d(3,8,5)
        #畳み込み層2の定義（6ch,カーネル数16,カーネルサイズ5）
        self.conv2 = nn.Conv2d(8,16,5)
        #プーリング層の定義(領域サイズ2,ストライド2)
        self.pool = nn.MaxPool2d(2,2)
        #ドロップアウトの設定
        self.dropout = nn.Dropout(p=0.5)
        #活性化関数の設定
        self.relu = nn.ReLU()
        #全結合層の設定
        self.fc1 = nn.Linear(16*5*5,256)
        self.fc2 = nn.Linear(256,10)
```

８－２　各層の連結（実装）

以上で定義したものを用いてＣＮＮを構築する

畳み込み層第１層にデータが渡され、ReLU 関数にて次の層に伝播し、プーリングされたデータが多端込み第 2 層に渡され、同じく ReLU 関数にて次の層に伝播、またプーリングされ全結合層第１層で統合され、ドロップアウトを適用したうえで ReLU 関数にて全結合層第 2 層に伝播して、最終出力を得るという流れで実装する

以上はクラス内の forward というメソッドを作成し、その内部で記述する形で実装することとする

```
def forward(self,x):
    x = self.relu(self.conv1(x))
    x = self.pool(x)
    x = self.relu(self.conv2(x))
    x = self.pool(x)
    x = self.relu(self.fc(x))
    x = self.dropout(x)
    x = self.fc2(x)

    return x
```

８－３　インスタンスとネットワークの確認

以上クラスを定義したらこれを適当な変数を用いてインスタンスする

今回は net という変数でインスタンスすることとし、print()にてその構造を表示することでネットワークの確認を行う

結果、図のようにＣＮＮが構築できていることが確認できる

```
>>>
    ========================= RESTART: C:/Users/owner/Desktop/aaa.py =========================
    Net(
      (conv1): Conv2d(3, 8, kernel_size=(5, 5), stride=(1, 1))
      (conv2): Conv2d(8, 16, kernel_size=(5, 5), stride=(1, 1))
      (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (dropout): Dropout(p=0.5, inplace=False)
      (relu): ReLU()
      (fc1): Linear(in_features=400, out_features=256, bias=True)
      (fc2): Linear(in_features=256, out_features=10, bias=True)
    )
>>>
```

以上、全体のコードは以下のような形

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        #畳み込み層1の定義 (3ch,カーネル数8,カーネルサイズ5)
        self.conv1 = nn.Conv2d(3,8,5)
        #畳み込み層2の定義 (6ch,カーネル数16,カーネルサイズ5)
        self.conv2 = nn.Conv2d(8,16,5)
        #プーリング層の定義(領域サイズ2,ストライド2)
        self.pool = nn.MaxPool2d(2,2)
        #ドロップアウトの設定
        self.dropout = nn.Dropout(p=0.5)
        #活性化関数の設定
        self.relu = nn.ReLU()
        #全結合層の設定
        self.fc1 = nn.Linear(16*5*5,256)
        self.fc2 = nn.Linear(256,10)

    def forward(self,x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = self.relu(self.fc(x))
        x = self.dropout(x)
        x = self.fc2(x)

        return x



net = Net()
print(net)
```