〈論文〉

自律走行AIロボット用画像認識アルゴリズムについて
―ResNetの応用―

野　津　伸　治

Shinji NOTSU : An Implementation of Residual Network for Rapid Image Recognition Which an AI Car Can Run the Course by Itself

鳥取看護大学・鳥取短期大学研究紀要　第85号　抜刷

2022年 7月

自律走行AIロボット用画像認識アルゴリズムについて
―ResNetの応用―

野　津　伸　治[1]

Shinji Notsu : An Implementation of Residual Network for Rapid Image Recognition Which an AI Car Can Run the Course by Itself

任意のコースを搭載するカメラで取得した映像から自律走行する AI カーの画像認識を深層学習の畳み込みニューラルネットワークの一つ Residual Network の実装のための理論的なまとめを行った.

キーワード：深層学習　畳込層　ポーリング層　Residual Network

はじめに

人工知能（AI : Artificial Intelligence）は 2010 年以降第 3 のブームを迎えている．この度は単なるブームにとどまらずコンピュータの高性能化とインターネット上の Big Data を利用できる技術の進展から画期的なアルゴリズムが次々と提案され，自然言語認識，音声認識，画像認識などの分野で実装され目覚ましい成果が上がっている.

本研究ではカメラ映像を利用してコースを自動走行する AI ロボットのリアルタイム画像認識での深層学習アルゴリズムの実装に関する研究である．具体的には NVIDIA 社の GPU 搭載コンピュータ Jetson nano B01 4GB を搭載し，2 つの DC モータを前進・後退・左折・右折・停止の制御を行う．コースとロボット自身の位置関係を搭載カメラで画像処理を行う．この位置関係からそのまま進んでよい場合とよくない場合でそれぞれ 100 枚ずつ静止画像を撮影してあらかじめ学習させる．画像認識のための深層学習モデルで教師あり学習を行い，最適走行モデルをみつけて実際に走行させる．走行の過程で適切な動作ができない場合があればそこでの学習データを追加撮影して再度深層学習モデルのパラメータを調整する．これを走行精度や走行時間の改良を目標に繰り返し学習をさせていく.

最初に深層学習の中でも画像認識で定評のある畳み込み学習の概念を説明したのちに Residual Network について詳細の説明を行っていく.

1．深層学習

AI 学習の範疇において機械学習（Machine Learning）や深層学習（Deep Learning）や教師あり学習（Supervised Learning）・教師なし学習（Unsupervised Learning）・強化学習などの重要概念がある．画像認識に限定して教師あり深層学習に基本概念を説明する.

教師あり学習では入力データから出力パターンを識別したり予測したりする．連続値を予測する場合は回帰問題といい,離散値を予測する場合は分類問題という.

構造が単純なニューラルネットワーク（単純パーセプトロン）は線形分問題しか解くことはできない.

これを克服するために中間層（隠れ層）を多段にしていくことと誤差逆伝搬学習（Back Propagation）を取り入れた深層学習においてはより複雑な予測や

1　鳥取短期大学生活学科

分類が可能となる．深層学習では人間が入力データを識別する特徴量を与えることなく自ら獲得する学習も可能である．

（1） 画像入力データ

学習データ（Training Data）として {free,blocked} の 2 種類の JPEG 画像データ（Image Data）がある．これらの画像は，図 1 のような 224×224 ピクセルのカラーデータとする．

図 1　Captured image on the course

それぞれは 100 種類ずつある．またそれぞれのデータは教師データ（Supervised Data）としてその画像が {free, blocked} のいずれかであるかが対をなすデータセット（Paired Dataset）になっている．

（2） ニューラルネットワーク

ニューラルネットワーク（Neural Network）とは，図 2 のように入力データを受け取る入力層（Input Layer），出力パターンを表示する出力層（Output Layer），これらの中間に中間層（隠れ層：Hidden Layer）の 3 層モデルから成り立っている．

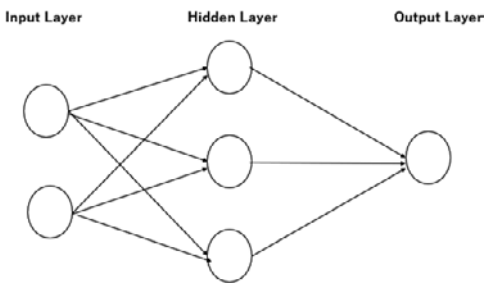

図 2　Neural Netowork

各層には複数のノードが存在し，異なる層の各ノードがアークでつながっている．中間層のあるノード（i）への入力（u_i）を考えると前層の複数のノードとつながっており出力（x_i）がある．その繋がりの強さを重み（W_i：Weight）で表す．ノードの入力は式（1）のように繋がっている前層のノードの出力と重みの和で表す．

$$u_i = \Sigma W_i x_i \quad \sim(1)$$

なおノードの出力は 0 以上 1 以下の値であるとする．ノードの次層への出力（z_i）は，活性化関数（Activation Function）と呼ばれ，式（2）で表すことができる．

$$Z_i = f(u_i) \quad \sim(2)$$

この活性化関数は次の二つの特徴を持っている：一つ目は閾値（threshold）以上で有効であるが，それ未満なら無効である．二つ目は正解との誤差から逆算して重み（W_i）を決定するために微分可能でなければならない．

（3） 活性化関数

活性化関数の持つべき 2 つの性質を備えた既知の関数として Sigmoid 関数，双曲線正接関数，および正規性線形関数がある．

Sigmoid 関数 f(u) は式（3）で表すことができる．

$$f(u) = 1/(1+e^{-u}) \quad \sim(3)$$

これは u=0 の時に f(u)=1/2 となり，u<0 の期は 0<f(u)<1/2 となり，u>0 の時に 1/2<f(u)<1 となる．

式（3）をグラフで描くと図 3 の様になる．

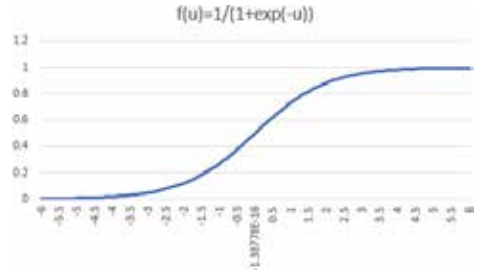

図 3　Sigmoid Function

双曲線正接関数（Hyperbolic Tangent Function）f(u) は式（4）で表すことができる．

$$f(u) = (e^{u}-e^{-u})/(e^{u}+e^{-u}) \quad \sim(4)$$

これは u=0 の時に f(u)=0 となり，u<0 の期は

$-1<f(u)<0$ となり，$u>0$ の時に $0<f(u)<1$ となる．

式（4）をグラフで描くと図 4 の様になる．

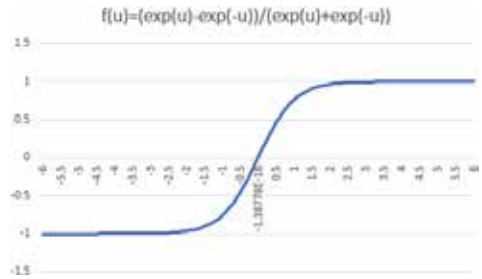

図 4　Hyperbolic Tangent Function

正規性線形関数（Rectified Linear Function）$f(u)$ は式（5）で表すことができる．

$f(u) = \max(0, u)$　～(5)

これは $u=0$ の時に $f(u)=0$ となり，$u<0$ の期は $f(u)=0$ となり，$u>0$ の時に $f(u)=u$ となる．

式（5）をグラフで描くと図 5 の様になる．

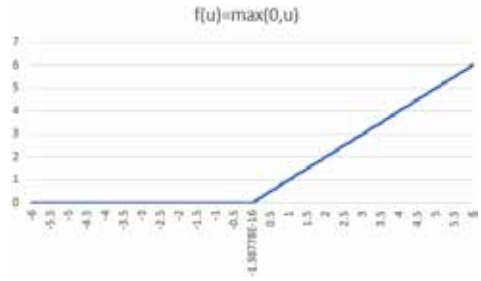

図 5　Rectified Linear Function

この関数のみは 1 番目の条件は満たすが，2 番目の条件は $u<0$ の時は微分係数＝0 で，$u>0$ の時は微分係数＝1 で，$u=0$ の時のみ満たさないが，$u=1$ とみなすように扱う．

2．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（Convolutional Neural Network，以下畳み込み NN と表す）は畳み込み層（Convolutional Layer）とプーリング層（Pooling Layer）を持つニューラルネットワークである．畳み込み層は画像の特徴を抽出化する役割を担う．一方プーリング層はある部分的なところから 1 つの代表的な数値を抜き出す役割を担う．この二つの層は結合を省略化したり重みを共有化したりする特徴を持っている．畳み込み NN の全体像は図 6 のようにまず入力層あり，次に畳み込み層，プーリング層が交互に存在したのちに，結果の予測値の正答率を上げるために全結合層が 2 つあり，最後に出力層がある．

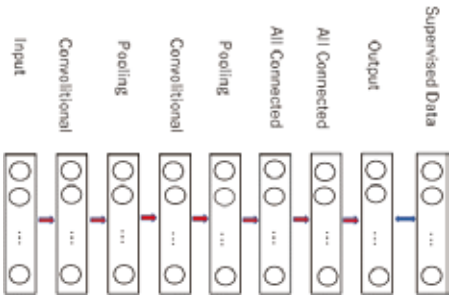

図 6　Convolutional Neural Network

（1） 畳みこみ層

畳み込み層での図 7 のような具体的な処理を説明する．画像データを $i=1, 2, 3$ かつ $j=1, 2, 3$ の要素 $\{0 \leqq x_{ij} \leqq 1\}$ に限定して，フィルター（＝重み）を $\{0 \leqq h_{ij} \leqq 1\}$ とすると，結果 $\{u_{ij}\}$ に対して式（6）の演算を定義する．

$u_{ij} = \Sigma x_{ij}\, h_{ij}$　～(6)

次に $i=2, 3, 4$ かつ $j=1, 2, 3$ の要素に対し同様の演算を行う．以降同様に対象をシフトしていく．

最後に $i=3, 4, 5$ かつ $j=2, 3, 4$ の要素に対し同様の演算を行う．

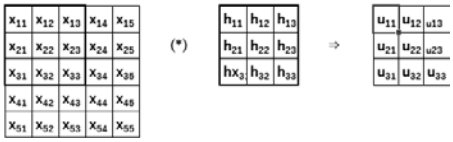

図 7　Convolutional Operations

例えば，$i=1, 2, 3$ かつ $j=1, 2, 3$ の要素で $\{x_{13}=1, x_{22}=\fallingdotseq 1, x_{31}\fallingdotseq 1, x_{ij}\fallingdotseq 0\}$ であり $\{h_{i=j}\fallingdotseq 1, h_{i\neq j}\fallingdotseq 0\}$ ならば $u_{11}\fallingdotseq x_{11}$ となる．このことはフィルターの要素の値の大小と似たパターンのものを画像の部分

データに掛け算すると結果が大きくなることを意味する．

すなわち画像の各部分に対してフィルター（＝重み）を共通のものを使うことで元の画像の特徴量を抽出することができるとともにパラメータの数を激減させることができる．

（2）プーリング層

プーリング層で図 8 のような具体的な処理を説明する．畳み込み操作が 1 回終わった i＝1, 2 かつ j＝1, 2 の画像データ $\{0 \leqq x_{ij} \leqq 1\}$ に対して結果 $\{u_{ij}\}$ に対して式（7）の演算を定義する．

$u_{11} = \max\{x_{11}, x_{12}, x_{21}, x_{22}\}$ ～(7)

次に対象を i＝3, 4 かつ j＝1, 2 の画像データに同様の処理を行い u_{12} の計算する．

順次対象をシフトしていき i＝3, 4 かつ j＝3, 4 でま同様の処理を行い u_{22} の計算する．

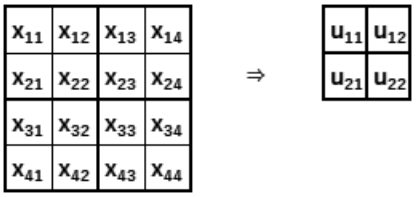

図 8　Pooling Operations

このプーリング層での処理は，画像のデータを表す数値の並びが多少ずれても同じ結果を返すこと，すなわち位置がある程度変化しても出力の値が変化しないという頑健性（Robustness）の獲得をしたことになる．

（3） ソフトマックス関数

出力層の活性化関数として式（8）のようにソフトマックス関数（Softmax Function）を用いることで結果を一つに絞り込みやすくする．

$y_i = e^{ui} / \Sigma\, e^{ui}$ ～(8)

但し，$0 \leqq e^{ui} \leqq 1$ でかつ $\Sigma\, e^{ui} = 1$

分母は正規化の役割を果たしており，式全体としては確率に対応している．またべき乗にすることによって大小をより強調している．

（4） 誤差関数

出力層の結果である予測値 y_i と教師データ d_i との差を比較する誤差関数として交差エントロピー（Cross Entropy）E を式（9）のように定義する．なお，誤差関数として交差エントロピーを使う理由はソフトマックス関数との相性が良いからである．

$E = -\Sigma\, d_i \log_e y_i$ ～(9)

誤差関数の性質として誤差が大きいときには大きな値を，小さなときには小さい値を返す．数値例で確認する（d_1, d_2, d_3）＝（0, 1, 0）であるときに，（y_1, y_2, y_3）＝（0.05, 0.8, 0.15）ならば，$E = -1x \log_e 0.8 = 0.09$ となる．一方（y_1, y_2, y_3）＝（0.1, 0.2, 0.7）ならば，$E = -1x \log_e 0.2 = 1.61$ となる．

学習である重みの調整，すなわち畳み込み層のフィルターや全結合層での重みを，交差エントロピーが最小になるように誤差伝播法で調整していく．

3．誤差伝搬法

畳み込み NN で予測を行わせ，教師データを使ってその精度を向上させるには，各層での重みと閾値のパラメータを調整していかなくてはならない．出力層での結果 $\{y_1, y_2, y_3\}$ と教師データ $\{d_1, d_2, d_3\}$ の誤差を比較して最小化できるようにパラメータを決定する．

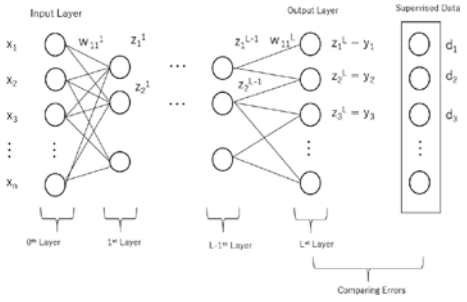

図 9　Deep Learning

図 9 で深層学習の用語の説明を行う．出力層の結果 z_1^L は L 層目の 1 番目の出力を表すものとし簡便

のため y_1 と置き換える．また出力層をひとつ前の層，すなわちL−1層目の1番目の出力を z_1^{L-1} と表記する．

L−1層の1番目のノードからL層の1番目のノードの重みは w_{11}^{L} と表すものとする．同様の定義はL層目から0層目まで行うものとする．

ここでl層でのj番目のノードの出力 u_j^l は式(10)のように重み付き線形和として定式化できる．

$u_j^l = w_{0j}^l + w_{1j}^l z_1^{l-1} + w_{2j}^l z_1^{l-1} + \cdots$ ～(10)

但し，w_{0j}^l はバイアス（bias）とする．

したがって活性化関数 z_j^l は式（11）のように定式化できる．一般的には，出力以外の隠れ層で共通であるが，l番目の層が変わってもよい．

$z_j^l = f^l(u_j^l)$ ～(11)

z層の出力（＝予測値y）と教師データ（d）の誤差で定義する誤差関数を最小化するように重みを最適化する問題に帰着できた．

ここで更新量について考えてみる．非線形の誤差関数Eのある点 w_i の接線の傾き（勾配 : Gradient）が0より大きいなら w_i を減らし，勾配が0未満なら w_i を増やしていく形で最適解を探索する．そこでこの更新量を式（12）で定義する．

$\Delta w_{ij}^l = \eta \partial E / \partial w_{ij}^l$ ～(12)

但し，η は学習係数である．

実際にこの値を隠れ層の低い層から順に計算すると高い層への影響の計算が解析学的に膨大になるため，高い層から低い層に逆に計算をしていく．数学的には式（13）のようにn変数の連鎖律（Chain Rule）を使って偏微分の値を計算していく．

$\partial z / \partial w_k = \Sigma \partial z / \partial u_i \cdot \partial u_i / \partial w_k$ ～(13)

回帰問題に従い重みの更新計算の前提の1つ目に出力層の結果である予測値 y_i と教師データ d_i の誤差を偏微分の計算がしやすいように式（14）のような二乗誤差関数（Squared Error Function）を利用することにする．

$E = 1/2 \Sigma (y_i - d_i)^2$ ～(14)

また重みの更新計算の前提の2つ目に出力層の活性化関数 $f^L(u)$ を式（15）のような恒等関数として定義する．

$f^L(u) = u$ ～(15)

まず出力層の重みの計算を式（16）のように定義して，式（17）のように $\delta^L_j = \partial E / \partial u^L_j$ の書き換えと偏微分の計算を行う．なお，ネットワークのつながり方を理解して計算を来なうことがポイントとなる．

$\partial E / \partial w^L_{ij} = \partial E / \partial u^L_j \cdot \partial u^L_j / \partial w^L_{ij}$ ～(16)

$\partial E / \partial w^L_{ij} = \delta^L_j z^{L-1}_i$ ～(17)

式（18）の計算を行い式（19）の結果を得る．

$\delta^L_j = \partial E / \partial u^L_j = \partial E / \partial z^L_j \cdot \partial z^L_j / \partial u^L_{ij}$ ～(18)

$\partial E / \partial u^L_j = (y_i - d_i) z^{L-1}_i$ ～(19)

次にl番目の中間層の重みの更新 $\partial E / \partial w^l_{ij}$ について式(20)のように考えて式(21)のように解ける．

$\partial E / \partial w^l_{ij} = \partial E / \partial u^l_j \cdot \partial u^l_j / \partial w^l_{ij}$ ～(20)

$\partial E / \partial w^l_{ij} = \delta^l_j z^{l-1}_i$ ～(21)

ここで δ^l_j について式（22）から式（23）のように整理する．

$\delta^l_j = \partial E / \partial u^l_j = \Sigma \partial E / \partial u^{l+1}_k \cdot \partial u^{l+1}_k / \partial u^l_j$ ～(22)

$= \Sigma \delta^{l+1}_k \partial u^{l+1}_k / \partial z^l_j \cdot \partial z^l_j / \partial u^l_j$ ～(23)

ここで δ^l_j の値は δ^{l+1}_k の値から求められることがわかる．$\delta^L (= d_j - y_j) \rightarrow \delta^{L-1} \rightarrow \delta^{L-2} \rightarrow \cdots \rightarrow \delta^1$，つまり出力層から入力層までの重みが逆方向に求めることができる．

4．Residual Network

Residual Network は，畳み込みNNを発展させて画像認識精度を競うImageNetコンペティション(2015年）で優勝したMicrosoft Research（後にFacebook AI Researchへ移籍）のKaiming Heが考案したアルゴリズムで注目を集め研究が進展している．従来の畳み込みNNと比較して大きく2つの特徴がある．まず隠れ層が非常に深い（2012年には8層で2014年には22層までだったものが，2015年には152層になった）こととShortcut Connectionが導入されたことである．

層の深さと表現力の高さは比例するが，単純に深すぎる層は計算が莫大になるだけでなく，利用する活

性化関数によって導関数が 0.25 から 1.0 未満となりそれが層の数だけ掛け合わされて勾配が非常に小さな値に収束する勾配消失（Gradient Disappearance）と呼ばれる多層ネットワーク特有の現象が発生して学習の停止や速度の問題が出てくる．

この速度低下を回避する従来の提案のブレークスルーとなったものが Shortcut Connection と残差ブロックの導入である．

（1） 残差ブロック

残差ブロックは，畳み込み層と Skip Connection のそれぞれの要素を足し合わせて構成されている．

（2） Shortcut Connection

Shortcut Connection は，深層化して勾配の積が 0 になる前に重みを 0 にするために計算をスキップする働きをする．これにより層の深化の限界を押し上げて精度を向上させることができる．

5．PyTorch での実装

PyTorch は，Facebook AI Research が 2016 年から公開している python 言語用の機械学習ライブラリである画像認識の深層学習において PyTorch を利用するのが定石である．PyTorch の基本的なデータ構造に Tensor 型と ndarray 型がある．前者は GPU での演算が可能であり，CPU での演算より高速化が可能である．AI ロボットの自律走行において獲得した推論モデルの実行に NVIDIA の Jetson nano（CUDA コアが 128 個ある）を利用した．また推論モデルの高速学習には同じく NVIDIA の GeForce GTX1650（CUDA コアが 896 個ある）を用いた．ここではエッジ AI でのリアルタイム処理という観点から畳み込み層が 18 層の ResNet-18 の実装[1]について考える．衝突回避用の二つの画像クラス {free, blocked} を学習に用いる．GPU でモデルの実行を 30 エポック行う．best_model_resnet18.pth を作成する．

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
dataset = datasets.ImageFolder(
  'dataset',
  transforms.Compose([
    transforms.ColorJitter(0.1, 0.1, 0.1, 0.1),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
     [0.229, 0.224, 0.225])
  ])
)
train_dataset, test_dataset = torch.utils.data.
random_split(dataset, [len(dataset) －50, 50])
train_loader = torch.utils.data.DataLoader(
  train_dataset,
  batch_size = 8,
  shuffle = True,
  num_workers = 0,
)

test_loader = torch.utils.data.DataLoader(
  test_dataset,
  batch_size = 8,
  shuffle = True,
  num_workers = 0,
)
model = models.resnet18（pretrained = True）
model.fc = torch.nn.Linear(512, 2)
device = torch.device('cuda')
model = model.to(device)
NUM_EPOCHS = 30
BEST_MODEL_PATH = 'best_model_resnet 18.
```

```
pth'
best_accuracy = 0.0

optimizer = optim.SGD(model.parameters(), lr =
0.001, momentum = 0.9)

for epoch in range(NUM_EPOCHS):

  for images, labels in iter(train_loader):
      images = images.to(device)
      labels = labels.to(device)
      optimizer.zero_grad()
      outputs = model(images)
      loss = F.cross_entropy(outputs, labels)
      loss.backward()
      optimizer.step()

  test_error_count = 0.0
  for images, labels in iter(test_loader):
      images = images.to(device)
      labels = labels.to(device)
      outputs = model(images)
      test_error_count + = float(torch.sum
      (torch.abs(labels − outputs.argmax(1))))

  test_accuracy = 1.0 − float(test_error_count)/
  float(len(test_dataset))
  print('%d: %f' % (epoch, test_accuracy))
  if test_accuracy > best_accuracy:
    torch.save(model.state_dict(), BEST_
    MODEL_PATH)
    best_accuracy = test_accuracy
```

おわりに

GPU 搭載の AI ロボットが，任意のコースを自律走行をさせるために，搭載するカメラからリアルタイムに取得する画像データを画像認識させた．認識に用いた畳み込みニューラルネットワークの中間層を 18 層にした Residual Network のアルゴリズムの実装版を用いた．学習モデルの高速獲得にエッジ AI とは別のより高速なコンピュータ上で GPU を用いる実験をした．従来型のセンサーを情報を用いて人間が判断ロジックを実装する場合と比較して，ロジック自身の獲得の確認ができた．

参考文献

1）JetBot https://github.com/NVIDIA-AI-IOT/jetbot/blob/master/notebooks/collision_avoidance/train_model_resnet18.ipynb（2022 年 4 月 5 日）.