第二章 畳み込みニューラルネットワークと画像認識

ディープラーニングの応用として画像に対するアプリケーションでもっとも優れた成果を上げている分野は画像認識だと言えます。本章では画像認識に用いられる畳み込みニューラルネットワークの基本的な仕組みと代表的なアーキテクチャについて学んでいきます。

2.1 画像認識とは

画像認識は、パターン認識技術の１つで、カメラやビデオの画像に写っているモノや人の顔などのオブジェクトやそのオブジェクトの特徴である形、色、数などを認識する技術のことです。本節では画像認識の利用シーンをイメージできる情報を提供します。

画像認識のさまざまなアプリケーション

画像認識を利用したアプリケーションとしては、のちほど紹介するYOLOというアルゴリズムに代表される一般物体認識のほかに、顔認識、年齢推定などがあります。さらに応用的な話でいうと、ひびが入っているのかを見つけるような老朽箇所の検知であったり、一般物体認識のようにボックスで物体を認識するのではなく、画像のピクセルごとにクラスの分類を行うセグメンテーションという分野で用いられています。また文字認識という領域の基礎的な技術としても畳み込みニューラルネットワークが使われています。

| ç»åèªè­ã®ã¢ããªã±ã¼ã·ã§ã³
15
ä¸è¬ç©ä½èªè­ é¡èªè­ å¹´é½¢æ¨å®
èæ½ç®ææ¤ç¥ ã»ã°ã¡ã³ãã¼ã·ã§ã³ æå­èªè­
 

画像認識のアプリケーション

畳み込みニューラルネットワークとは

そもそも畳み込みとは、ある関数と別の関数を重ねて足し合わせる演算のことを指す数学用語です。ディープラーニングでは、画像認識に適したアルゴリズムが畳み込みニューラルネットワークで、その名の通り畳み込み処理を用いています。ここではニューラルネットワークにおける畳み込み処理の仕組みと動作について理解を深めることで畳み込みニューラルネットワークの基本を押さえましょう。

畳み込みニューラルネットワークの仕組み

下図を使って畳み込みの基本的な説明をします。図の真ん中の行列が左の人物の画像の耳の部分のピクセルの画素値になります。図の例ではカラーの画像になっていますが、以下の説明では説明を簡単にするために、グレースケールの画像と仮定して話を進めます。

畳み込みでは、畳み込みフィルターとかカーネルと呼ばれる、画像の一定の範囲の特徴を把握するための行列を用意します。図の真ん中のピクセルとその周りの関連性を知るために準備した畳み込みフィルターを使って畳み込みと呼ばれる行列計算を行います。

ç³ã¿è¾¼ã¿ (Convolution)
CURRENT_PIXEL_VALUE = 82
NEW_PIXEL_VALUE =
(-1 * 144) + (0 * 60) + (-2 * 19)
+ (0.5 * 188) + (4.5 * 82)...

畳み込みフィルターの大きさは任意に決めることができますが、この例では３×3の行列とします。図の真ん中のピクセルの値が82だとしたら、畳み込みを処理した後の値は、画像ピクセルの行列の一番左上の要素と畳み込みフィルターの一番左上の要素の掛け算と真ん中上同士の掛け算、一番右上同士の掛け算という具合に、それぞれの同じ位置にある要素ごとの掛け算となる行列の要素ごとの積を行ったのち、その計算結果の総和を取ったものとなります。この操作自体は非常に簡単です。ただしこれを教科書に出てくるような図の左にある式で表現すると、一見すると非常に複雑で慣れてない方には暗号のように添え字も非常に多く理解し辛いものになるかと思います。ただしさきほどのように実行している操作自体は非常にシンプルですのでその点を理解してください。

・畳み込みの計算方法

これから畳み込みフィルターを使った画像データの畳み込みの計算方法について具体的に説明します。下図は、バスケットボールの画像の拡大図です。畳み込みの計算は図のように左上からはじまって、赤い四角で囲まれたピクセルを含むその周りの9つのピクセルに対して畳み込み計算を行い、つぎにstrideで指定された畳み込みフィルターのずらす幅の分だけフィルターを横にずらして畳み込み計算を行うというような操作を延々と繰り返していきます。

畳み込みがどのようなことに有効なのかイメージいただけるよう解説します。下図の左のバスケットボールの画像に真ん中の畳み込みフィルターを掛け合わせてあげると、右図のように変換されます。

!32
1
2
1
0
0
0
-1
-2
-1
-1
-2
-1
0
0
0
1
2
1
-2
0
2
-1
0
-1
-1
0
1
https://www.bbkong.net/fs/alleyoop/molten_BGL7
 

図の真ん中の一番上の畳み込みフィルターは行列の真ん中の値が0で、左側が正の値、右側が負の値という形をしています。これを畳み込み計算すると、画像の左側が明るい部分のピクセル値の値が大きくなるため左側がより明るくなり、右側の部分は負の値なのでより暗くなるようにピクセル値が下がります。この畳み込みフィルターを使って畳み込み計算を行った結果が図の右上の画像になりますが、たしかに左のエッジがうまく検出できているのがわかります。

真ん中の畳み込みフィルターはその逆パターンで、右側が明るかったら大きな値を返すものですので、図の右側の真ん中の画像を見ればおわかりの通り、右側が明るいエッジを検出しています。

下の畳み込みフィルターは、下側が明るかったら大きな値を返すもので、図の右側の下の画像をみると、下のエッジがうまく検出できています。

上のように別々の畳み込みフィルターを使用することで、異なるエッジを抽出することができます。

それでは、さきほどのバスケットボールの画像に対してKerasを用いて具体的な畳み込みニューラルネットワークのサンプルプログラムを動かしながら動作を見ていきます。

最初にすでにおなじみの以下のような必要となるモジュールをインポートします。

|  |
| --- |
| import numpy as np  import tensorflow as tf  import matplotlib.pyplot as plt  *# jupyter notebook上で画像を表示するためのおまじない*  %matplotlib inline  from keras.preprocessing import image |

つぎにバスケットボール画像を以下のコードを使って読み込みます。Kerasで用いられるload\_img関数は扱える画像データ形式がPIL（Python Image Library）になっています。PIL形式のデータをディープラーニングで使えるようにするために配列形式に変換します。その際、配列の各要素のデータ形式を16ビッド符号付の整数（int16）に変換する、astype((‘int16)という型変換も同時に行っています。またその次の行のexpand\_dims関数では、第二引数のaxisで指定された次元の前に１つ次元を追加しています。そのため、画像の形についてprint関数を使って出力すると、(1, 244, 244, 3)のように最初に1次元追加されています。ここで1次元増やしている理由は、Kerasで用いられるConv2D等の畳み込みの関数では、関数の最初の引数が畳み込みフィルターの数を指定する形式になっているためです。

|  |
| --- |
| # 画像の読み込み  img = image.load\_img('./basketball.png', target\_size=(224, 224))  # PIL形式からnumpy形式に変換  x = image.img\_to\_array(img).astype('int16')  # (height, width, channel) -> (minibatch, height, width, channel)  x = np.expand\_dims(x, axis=0)  print(x.shape) |

上記のサンプルコードをもとに実際に読み込んだ画像は以下のimgshow関数を使って表示されます。



|  |
| --- |
| # convolution layerを用意  model = tf.keras.models.Sequential([  tf.keras.layers.Conv2D(filters=1, kernel\_size=(3, 3), input\_shape=(224, 224, 3))  ]) |

つぎに畳み込みのフィルターを作成します。

さきほどの畳み込みフィルターを再現するために、Conv2Dと呼ばれる関数を使います。Conv2Dとは2次元の畳み込みを意味します。Convというのが畳み込みの英語にあたるConvolutionの略で、2Dは2次元を意味します。Conv2Dを使って今回使用する引数としては、第一引数からそれぞれ、畳み込みフィルターの数が1つ、フィルターのサイズにあたるカーネルサイズが３×３、画像の入力の形が244×244でカラーなのでRGBの3チャネルであることを指定しています。

このように畳み込み層を用意してあげると、以下のように畳み込みフィルターのそれぞれの値はランダムに適当なものが入ります。

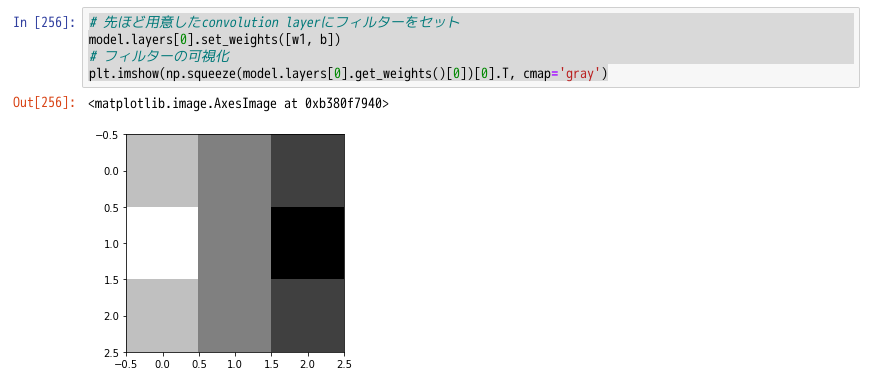


今回は特別に、さきほどバスケットボールの画像を用いた畳み込みフィルターの効果を見ていただくために、この畳み込みフィルターを強制的に設定するところをお見せします。普段の畳み込み処理では今から説明するget\_weightsからset\_weightsを用いた処理を行う部分は行いませんのでその点はご注意ください。

get\_weightsという関数を用いて手動で畳み込みフィルターの配列を取りこみます。

|  |
| --- |
| # 左側が明るくて右側が暗いエッジを検出するフィルター  w1 = np.array([[[1], [2], [1]],  [[0], [0], [0]],  [[-1], [-2], [-1]]],  dtype='float32')  # 簡単のためRGBチャンネル全て同じフィルターとする  w1 = np.array([w1, w1, w1]) |

つぎにここではさきほどお見せした左側が明るかったら大きい値を返すような左のエッジを検出するための畳み込みフィルターの値をnp.array関数を使って設定します。最後にset\_weights関数を使ってその畳み込みフィルターの配列を設定し、その畳み込みフィルターについて再度imshow関数を使って可視化しています。



つぎにpredict関数を用いてモデルの予測を行ったあと、畳み込みフィルターで変換された画像を見やすくするために、フィルター化された画像の画素をそれぞれフィルター化された画素の最大値で割った値に変換してimshow関数で可視化を行うという処理を行っています。

|  |
| --- |
| filtered = model.predict(x)  filtered /= np.max(filtered)  # plt.imshow(filtered[0], cmap='gray', vmin=0, vmax=255)  plt.imshow(np.squeeze(filtered), cmap='gray', vmin=0, vmax=1) |

このような処理をすると最終的にさきほどお見せした以下の画像が得られます。



畳み込みのフィルターについてこのような内容に触れる機会も少ないのではないかと思います。ほかの畳み込みフィルターについてもサンプルコードに含まれておりますので、興味がある方はぜひご自身で動かしてみてください。

・ストライドとパディングについて

取り込んだ画像に対して畳み込みフィルターをずらしながら畳み込み処理を行っていく必要があるので、ストライドとパディングというパラメータを設定します。具体的な畳み込みフィルターのをずらしていくための動作を理解しておくことで、実践でのパラメータ設定の参考にしてください。

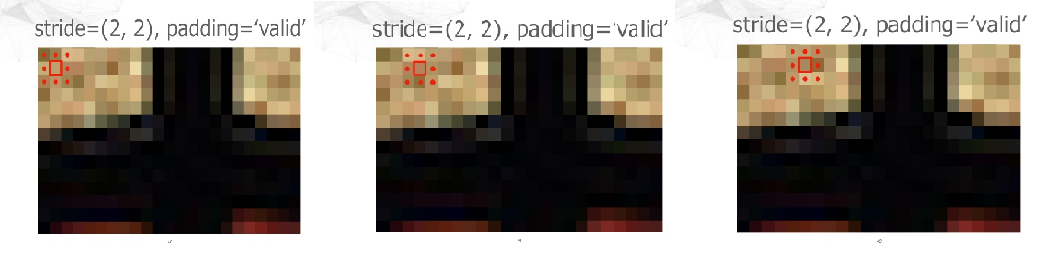
ストライドとパディングの動き

ストライドはどのくらいのステップで畳み込みフィルターを動かすかを決める値です。図のstride=(1,1)では、最初一番左上から横方向に１ピクセルづつ畳み込みフィルターを動かしていって、右上の端までいったら次１個下のピクセルに進んで再度左端から同じ処理を繰り返していきます。

stride(2,2)の場合は、1個飛ばしで進んでいくということを指定しています。

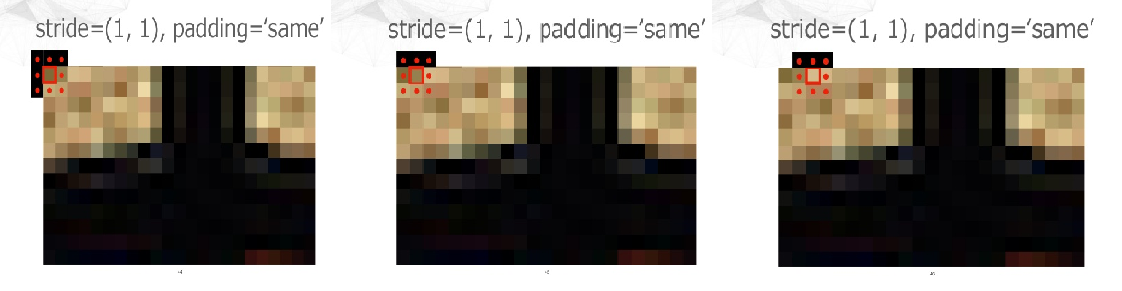


ストライド（1、１）の場合の畳み込みフィルターの動き



ストライド（1、１）の場合の畳み込みフィルターの動き

勘のいい読者の方は気づかれているかと思いますが、画像の端のピクセルに対しては畳み込みフィルターを使って計算することができません。なぜかというと端のピクセルに畳み込みフィルターを当てはめようとすると画像からはみ出てしまうからです。それをあえて以下の図のように画像の周りに０の値を埋めて（パディングして）あげて端から畳み込みの計算を行うということも非常によく行われています。このようにすることで端のピクセルに対しても畳み込みの計算ができるようになります。



パディングを用いた畳み込み動作

ディープラーニングフレームワークを使って畳み込みニューラルネットワークを触っていくときに、いきなりストライドやパディングが出てくると理解が難しいと思いますが、今の説明の通りやっていることは非常に単純ですので、怖がる必要はまったくありません。

Max Poolingとは

最大値プーリングという言い方もされますが、図のように４×4の入力画像があったとします。その中の２×２の領域を取ってきてそれの最大値を出力するというそれだけの機能です。たとえば図の赤枠の２×２の行列であれば、各要素の中で右下の192という値が最大値なので192が出力されます。同様に右上の２×２の行列では最大値は144、それ以降も同様に図の右の２×２の行列のような数値がMax Poolingを行うことによって得られます。これは入力の行列の中から一番特徴的な値としてこの場合は最大値だけを取ってくる仕組みです。Max Poolingはノイズを除去したり余計なものを見なくしたりするなどで用いられる計算です。

Max Pooling
!47
https://www.coursera.org/learn/introduction-tensorï¬ow
 

Max Poolingの説明

ここまでの部分を理解いただけるとディープラーニングの入り口にたどり着くことができます。

早速、具体的なモデルの例としてAlexNetについて説明します。

AlexNetは2012年のImageNetという大規模な画像のデータセットを使って正解率を競うコンペティションで優勝したモデルです。名前の由来はアレックスさんが筆頭著者となっている論文のモデルが優勝したからです。ImageNetのデータセットは140万枚以上で2万カテゴリ以上の画像がありますが、図で使用しているものは、2万枚を超える画像データで、犬、猫、車など1000個以上のカテゴリのラベルがついていて、それぞれの画像の形は224×224×3です。図のAlexNetはそれらの画像を左から取り込んで右に処理が進んでいくように描かれているためその順番で説明します。

一番左の板状の形がその画像の形を示しています。次の箱が48個の畳み込みフィルターを用いた畳み込み処理の部分を示しています。その出力結果をMax Poolingの計算を行って、さらに128個の畳み込みフィルターを用いて畳み込み計算を行いMax Poolingを行うというような処理を進めていって、最終的に分類を行うために全結合ニューラルネットワークのDense Layerにつなげてあげて1000個の分類を行うというようなネットワークです。

AlexNet
!48
 

AlexNetのアーキテクチャ構造

図のように畳み込み層、Max Poolingを繰り返し最後に全結合ニューラルネットワークで分類するというような形は広く利用されているもので、いわゆる畳み込みネットワークというとこういう構造をしていることが多いです。そのためこれは畳み込みニューラルネットワークの基礎的な構造をしています。

層を深くすると何がうれしいか

ディープラーニングはその名の通りニューラルネットワークの層を深くしたものです。ここでは層を深くする意味について、各層の学習結果を見ていくことで具体的にイメージができるようにしましょう。

それぞれの層で学習される特徴の違い

たとえば図の左上のようにニコちゃんマークのような丸に口がある画像があるとします。さきほどの畳み込みフィルターの例で見ていただいたように、最初に畳み込み処理を行うと、図の1段目の例のようにエッジという画像のあいまいな形というか画像の特徴となる部分を抽出できます。次の層では、それをまた入力として畳み込み処理を行うとさらに細かい特徴を抽出できます。図の上から2段目の層をみていただくとわかる通り、真ん中が丸に近いとか、上が半月に近いとか、左が半月に近いとかドーナッツ型というように一段目より細部にわたった特徴量を抽出することができます。畳み込みを重ねていくことで、だんだんと画像の表現をつかむための特徴量を、内部で細かく抽出していくことができます。最終的に多段の畳み込みで抽出した細かい特徴を入力として、全結合ニューラルネットワークを使って分類を行うというのが、さきほどのAlexNetに代表される畳み込みニューラルネットのアーキテクチャになります。

!49slide from Keynote Speech by Laurence Moroney at Deep Learning Day 2019
 

それぞれの層の特徴量の学習イメージ

次にどのような特徴量が得られているかについて、もう少し複雑なデータセットに対して見ていきます。

下図のサンプルコードでは、データセットとして馬と人間を含んだ多量の画像を使用します。このデータセットを入力として馬か人かを当てるようなニューラルネットワークを作ってみます。

最初に馬と人のデータセットを取り込んで、最初の10個の馬の名前を出力しています。

|  |
| --- |
| import os  import zipfile  d = ‘/home/ubuntu/work/dl4-practitioners/figures/horse-or-human’  # Directory with our training horse pictures  train\_horse\_dir = os.path.join(d, 'horses')  # Directory with our training human pictures  train\_human\_dir = os.path.join(d, 'humans')  train\_horse\_names = os.listdir(train\_horse\_dir)  print(train\_horse\_names[:10]) |

以下がディープニューラルネットワークのモデルの構成になります。

|  |
| --- |
| model = tf.keras.models.Sequential([  # Note the input shape is the desired size of the image 300x300 with 3 bytes color  # This is the first convolution  tf.keras.layers.Conv2D(16, (3,3), activation='relu', input\_shape=(300, 300, 3)),  tf.keras.layers.MaxPooling2D(2, 2),  # The second convolution  tf.keras.layers.Conv2D(32, (3,3), activation='relu'),  tf.keras.layers.MaxPooling2D(2,2),  # The third convolution  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),  tf.keras.layers.MaxPooling2D(2,2),  # The fourth convolution  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),  tf.keras.layers.MaxPooling2D(2,2),  # The fifth convolution  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),  tf.keras.layers.MaxPooling2D(2,2),  # Flatten the results to feed into a DNN  tf.keras.layers.Flatten(),  # 512 neuron hidden layer  tf.keras.layers.Dense(512, activation='relu'),  # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('horses') and 1 for the other ('humans')  tf.keras.layers.Dense(1, activation='sigmoid')  ]) |

このサンプルプログラムの例では、AllexNetと同様に、畳み込み層とMaxPoolingを5層重ねています。畳み込み層の出力は複数次元の行列になるため、それを1次元のベクトルに変換するためにFlatten関数を使い、最終的に馬か人間かの分類をするために全結合ニューラルネットワークでユニット数を１個に集約して人か馬かの0か１かの分類にしています。

以上の設定で作ったモデルのサマリを視覚化すると以下の図になります。畳み込み処理を行うネットワークがOutput Shapeに示される行列の形で表示されています。最初のconv2d\_6では、298×298の画像サイズで最後の16という数字は、Cond2Dの最初の引数で設定した畳み込みフィルターの数となります。

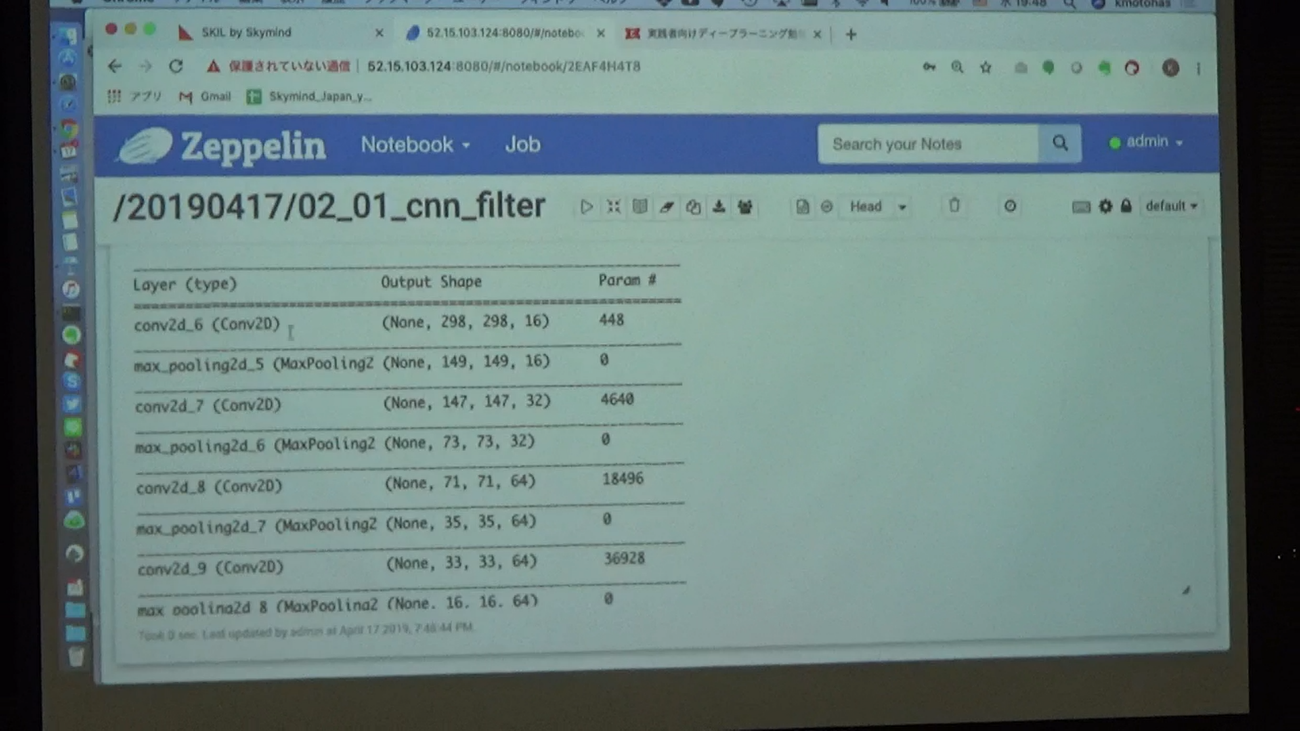
その出力に対してMax Poolingを行うと今回は2×2の領域から1つの最大値を取るので、画像サイズは半分の値になります。

つぎにその出力に再度畳み込み処理を行うと、両端のピクセルは畳み込みができないため２つだけ小さくなった147×147の画像サイズになっています。つぎのMax Poolingではさらにその画像サイズが半分になるというような処理を繰り返していき、畳み込みが終わると７×７×６４の行列に対してフラットを使って1次元のベクトルにします。１次元のベクトルは７×７×６４＝３１３６個の値を取ることになります。ここまでくると最初のDense Layerでは512個に絞りこんで、最後のDense Layerで馬か人かを0か1の数値を使って判定を行うニューラルネットワークができあがります。

実際の構築したモデルの概要を知りたい場合には、以下のコードを入力してください。

|  |
| --- |
| model.summary() |

実際の出力結果は以下のようになります。



出来上がったモデルに対して、compile関数を使って、ここでは特に深い意味はありませんが、RSMpropというAdamとは異なる最適化アルゴリズムと評価関数としては正解率（accuracy）を定義して分類問題を解いてあげます。あとはfit関数を使って訓練を行うことでモデルが出来上がります。

|  |
| --- |
| from tensorflow.keras.opmitimziers import RMSprop  model.compile(loss=’binary\_crossentropy”,  optimizer=RSMprop(lr=0.0001),  metrics([‘acc’]) |

それぞれの畳み込みの層とMax Poolingを行ったあとに、どんなものに変換されているかを可視化してみます。この変換後の出力は特徴マップと呼ばれます。上図に訓練されているネットワークでは、畳み込みフィルターのそれぞれ別々の役割を担ってくれています。

たとえば人の画像を入力すると最初の層では人の体の輪郭や色の情報が出ているとか明るいところがさらに明るくなっているなど、さまざまなものが別々の画像として出力されます。それに対して層がどんどん深くなっていくにつれて、もっと抽象的な特徴が得られていきます。ここまでいくと人が理解できるような特徴でなくなっていったりしますが、このようなさまざまな特徴をニューラルネットワークは使って馬や人を認識できるようになります。

!50
https://medium.com/@lmoroney_40129/codelabs-from-googlemlsummit-f9d53cac8d24
 

人の特徴量マップ

馬の場合も同様に馬の輪郭が見えているものや色の勾配を抽出しようとしているような層があってだんだんと抽象的な情報に落ち着いていって最終的に分類が行われるというのがこちらの図で表しているところです。

!51
https://medium.com/@lmoroney_40129/codelabs-from-googlemlsummit-f9d53cac8d24
 

馬の特徴量マップ

代表的な畳み込みニューラルネットワークのアーキテクチャ

ディープラーニングでは、性能を競うさまざまなコンペティションがあり有名なコンペティションで優勝したモデルは広く利用されています。ここでは、画像認識の領域で有名な畳み込みニューラルネットワークのアーキテクチャを紹介することで、アーキテクチャの基本的な構造と特徴を説明します。この説明を通じてその他のアーキテクチャについても読み解く際の三共になる情報を提供したいと思います。

畳み込みニューラルネットワークの発展の足跡

畳み込みニューラルネットワークはImageNetコンペティション（ILSVRC）とともに成長してきた歴史があり、2011年まではディープラーニング以外の技術を用いたアーキテクチャが優勝していましたが、2012年にさきほど説明したAlexNetという畳み込みニューラルネットワークを使ったモデルがいきなり登場し、これまでの分類の誤差の値が急激に下がり精度が上がったことで画像分析分野でブレークスルーが起きたと言われています。

それ以降も畳み込みニューラルネットワークを使用したさまざまモデルが登場し、以下の図のように2015年くらいまで線形的に精度が向上していきました。

ImageNet ã³ã³ãã®åªåã¢ãã«
!53
https://www.slideshare.net/ren4yu/ss-84282514
 

ImageNetコンペティションの優勝モデル

さきほどはAlexNetを紹介したので、ここではResNetを紹介します。

そのあとこのリストには載っていませんが、ビデオのようなリアルタイムの画像処理も可能なYOLOという有名なアーキテクチャがあるのでそれについて説明します。

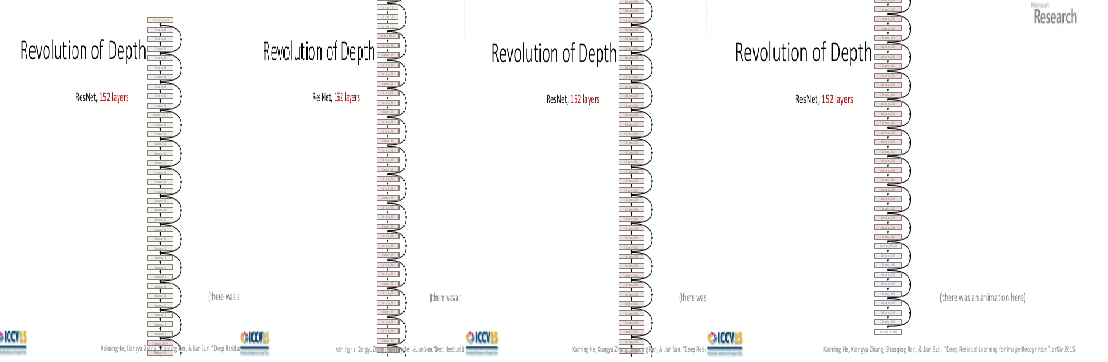
・ResNet

RenNetは2015年のImageNetコンペティションの優勝モデルです。何か画期的だったかというと、のちほど詳しく紹介しますが、Residual（残差）モジュールと呼ばれるショートカット機構を導入したところです。Residualモジュールを導入することによってそれまでは不可能であった層の数を各段に増やすことに成功しました。ResNetが登場する前のディープラーニングの層の深さは、AlexNetでは8層で、2014年では22層が最高で、その当時はこれでも非常に深い層と言われていました。ところがResNetでは、152層というすさまじい深さのものができるようになりました。深さが深くなるほど複雑な特徴を獲得することができるので、精度が高いものができあがるようになったと言えます。

ResNet
â¢ 2015å¹´ã®ImageNetã³ã³ã (ILSVRC) åªåã¢ãã«
â¢ Residualã¢ã¸ã¥ã¼ã«ï¼ã·ã§ã¼ãã«ããæ©æ§ï¼ã®å°â¼
!54
http://image-net.org/challenges/talks/ilsvrc20...

ResNetとそれまでのディープラーニングモデルの層の深さの違い

具体的なResNetのアーキテクチャの以下の図になります。こちらの図はMicrosoft Researchの作ったモデルになりますが、このブロックの１つ１つが畳み込み層です。そのブロックが合計で152層並んでいます。AlexNetの例と異なる点はただ単にブロックを積み上げただけでなく、横に回り道ができているところです。この横道のことをResidual（残差）とかスキップコネクションと呼びます。



下図の左の形が通常の畳み込みニューラルネットワークで１つのブロックが畳み込みニューラルネットワークです。それに対して右がResidual Netで、入力をそのまま足すというもう一つの別のパスを用意してあげることです。

ResNet
!59
Deep Residual Learning
â¢ Plaint net
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. âDeep Residual Learnin...

この機構を用いることで何かうれしいかというと、パラメータの更新行う場合、誤差逆伝播と呼ばれるアルゴリズムを使いますが、その誤差を何もせずに伝播させていくと、層が深くなると誤差が伝搬できなくなるという問題が生じてしまっていましたが、この横道を用いることで、もとの誤差が減衰することなく伝えてれくるようになるため、層が深くなっても訓練が進むようになることにあります。

さらにこの機能はResNetの前のモデルにあたるGoogleNetから実装されている内容になりますが、図の右のようなアーキテクチャを１つの塊のモジュールとして扱い、モジュールを積み重ねていくことで畳み込みニューラルネットワークを構築するということにあります。このようなモジュールを積み上げていく方式はここ数年のトレンドになっていて、SeNetなどでも用いられています。

下図はCIFER-10というカラー画像のデータセットを用いた訓練の進み具合を示したもので、さきほどの単純な畳み込みニューラルネットワークを用いた場合とResNetを用いた場合の違いを表しています。図の表は横軸が訓練経過時間で縦軸が誤差率つまり間違った割合を示しています。

図の黄色のものが層の数が20個だったときの結果で、青が32層、緑が44層、赤が56層の結果ですが、左側の表の単純な畳み込みニューラルネットワークの結果では、層を重ねるごとに逆に誤差率が上がってしまっています。それに対して右側の表はResNetのもので、スキップコネクションを追加したことにより層が増えるにつれて誤差率が下がっているという傾向が読み取れます。

!60
CIFAR-10 experiments
0 1 2 3 4 5 6
0
5
10
20
iter. (1e4)
error(%)
plain-20
plain-32
plain-44
plain-56
20-layer
32-laye...

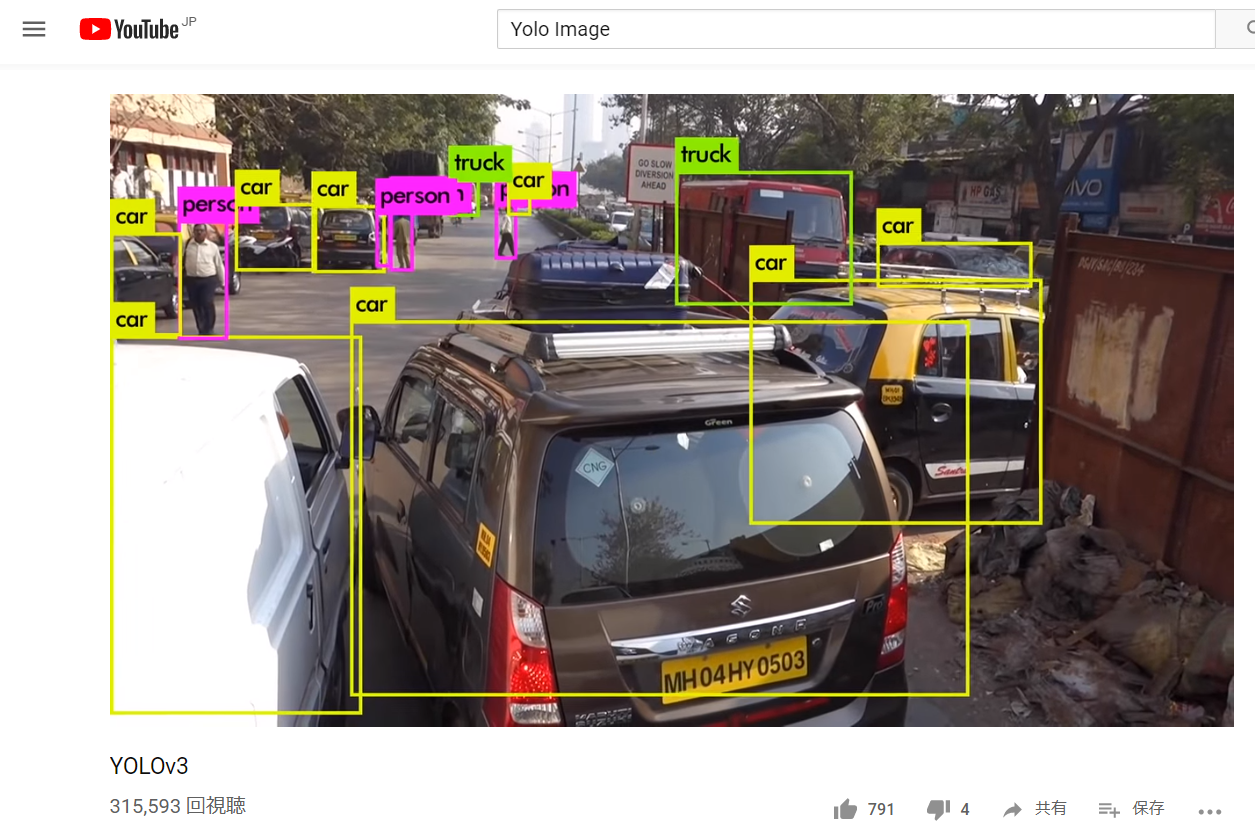
スキップコネクションがあることによる学習の進み方の違い

ResNetにはスキップコネクション以外にもバッチ正則化などいくつかの工夫がなされています。その内容についてさらに興味があるかたは、2年前の情報になりますが、DeNAの内田さんによる“畳み込みニューラルネットワークの研究動向”という素晴らしいスライドがありますのでそちらを参照してみてください。

<https://www.slideshare.net/ren4yu/ss-84282514>

YOLOについて

YOLOは、動画の中の各フレームに写っている人や物体の検知を非常に高速に行ってくれるモデルです。



YOLOを使った動画分析の例

上の図のように１フレームにこれだけ多くの人や車、バイクなどがあってもそれぞれを的確に認識できています。さらにYOLOが人気なのはGPUを使えば高速に実行できるため動画のようなリアルタイムの画像に対しても利用できるところです。

ここではYOLOのアーキテクチャについて簡単に説明します。

YOLOã®ã¢ã¼ã­ãã¯ãã£
!63
 

YOLOのアーキテクチャ図

このアーキテクチャもさきほど説明したAlexNetと一緒で、畳み込み層とMax Pooling層を何層にも重ね合わせています。層の中のユニットの数や畳み込みのフィルター数の違いなどに加え、畳み込み層の最後では、高さ７ピクセル、横７ピクセルという特徴量マップまで絞り込んで、各ピクセルの中の境界ボックス（Bounding Box）を書いてあげたり、その境界ボックスがどのクラスに属するかを見てあげたりということをやっています。図の真ん中の下の四角をみると、各ピクセルのうち、青の部分は犬、黄色の部分は自転車、赤は車のクラスになっているということを認識してあげて、さらに境界ボックスにするために四辺形として欠けている部分について、適切に閾値を切って残してあげるよう調整にして最終的に出力として、図の右のような犬を囲むボックス、自転車を囲むボックス、車を囲むボックスを生成することで検知ができています。

YOLOの詳しいアーキテクチャとしては、損失関数の定義とかたくさん出てくるボックスの候補の中から不要なものを差っ引くような機能などが実装されていますが、畳み込みニューラルネットワークの基本的な要素ではないので、本書での説明では割愛させていただきます。

コラム

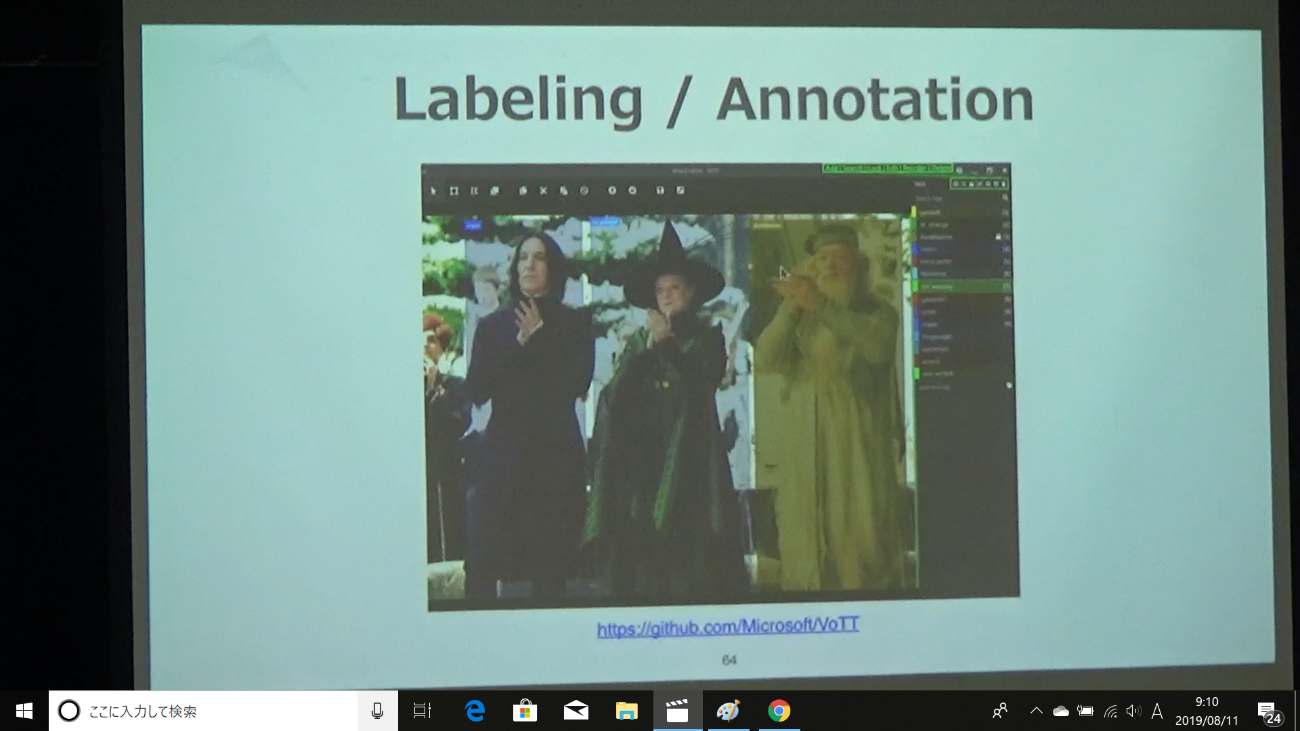
機械学習用の画像準備について

機械学習は入力データと答えをモデルに入れることでルールを見つけ出すような構造になっていると説明しました。今回の例のように領域とクラスを出力としたい場合に、どういう風に答えを用意するかについて説明します。

最近はこのようなことを支援するツールとして以下のリンクに示されるように非常に便利なものが出てきています。１枚1枚答えを準備しなければならないので時間はかかりますが、このようなツールのおかげで以前よりも準備にかかる時間が短縮されるようになりました。こちらは無料公開されていてさらにNode.jsで動くためプラットフォームの依存性もないため、ご興味がある方は試してみていただけるといいかと思います。

<https://github.com/Microsoft/VoTT>

こちらのツールは、動画とか画像があった場合に、たとえば人の画像があったらその人の画像部分をマウス等でドラッグするとその領域を記憶することができて、かつその領域はスネイプ先生ですというような情報を右のタグのところでラベルとしてまとめてクラスの関連付けを行うことができます。



アノテーションのツール例

このような作業はアノテーションと呼ばれますが、近年アノテーションが機械学習の実装作業の中で時間と労力を要するため、個人や企業等でさまざまな支援ツールが出てきています。

畳み込みニューラルネットワークの事前訓練