



Qiita Advent Calendar 2020 終了！ 今年のカレンダーはいかがでしたか？

[> ランキングを見る](#)

@shoji9x9 2019年05月11日に更新



TensorFlow2.0を使ってFashion-MNISTをResNet-50で学習する

[TensorFlow](#) [ResNet](#) [Fashion-MNIST](#) [GoogleColaboratory](#) [TensorFlow2.0](#)

⚠ この記事は最終更新日から1年以上が経過しています。

はじめに

今回はCNNの中でも比較的新しいResNetに取り組んでみたいと思います。と言っても2015年に発表されたようなので、もう4年前ですね・・・。

ResNetとは？

正確に説明する力はないのですが、Skip Connection (Shortcut) を利用することで従来よりも深い層を持つことを実現したネットワークと理解しています。日本語ですと、

- [Residual Network\(ResNet\)の理解とチューニングのベストプラクティス](#)
- [畳み込みニューラルネットワークの最新研究動向 \(～2017\)](#)

あたりがわかりやすいかと思います。あとは元の論文ですね。英語ですが12ページだけなので、その気になれば読めると思います（自分はところどころしか読んでいないです

が・・・)。

今回のテーマ

以前取り組んだFashion-MNISTの分類をResNet-50で実現しようと思います。今回は制約はなしにしました（ResNetの学習には時間がかかりそうだったので）。

環境

- Google Colaboratory
- TensorFlow 2.0 Alpha

コード

こちらです。

なぜかGitHub上ではうまく開けませんでした。GitHubのURLは[こちら](#)です。

コード解説

ResidualBlock

```
from tensorflow.keras.layers import Conv2D, Dense, BatchNormalization, Activation, MaxPool
from tensorflow.keras import Model
```

```
class ResidualBlock(Model):
    def __init__(self, channel_in = 64, channel_out = 256):
        super().__init__()
```

```
        channel = channel_out // 4
```

```
channel = channel_out // 2
```

```
self.conv1 = Conv2D(channel, kernel_size = (1, 1), padding = "same")
self.bn1 = BatchNormalization()
self.av1 = Activation(tf.nn.relu)
self.conv2 = Conv2D(channel, kernel_size = (3, 3), padding = "same")
self.bn2 = BatchNormalization()
self.av2 = Activation(tf.nn.relu)
self.conv3 = Conv2D(channel_out, kernel_size = (1, 1), padding = "same")
self.bn3 = BatchNormalization()
self.shortcut = self._shortcut(channel_in, channel_out)
self.add = Add()
self.av3 = Activation(tf.nn.relu)
```

```
def call(self, x):
    h = self.conv1(x)
    h = self.bn1(h)
    h = self.av1(h)
    h = self.conv2(h)
    h = self.bn2(h)
    h = self.av2(h)
    h = self.conv3(h)
    h = self.bn3(h)
    shortcut = self.shortcut(x)
    h = self.add([h, shortcut])
    y = self.av3(h)
    return y

def _shortcut(self, channel_in, channel_out):
    if channel_in == channel_out:
        return lambda x : x
    else:
        return self._projection(channel_out)

def _projection(self, channel_out):
    return Conv2D(channel_out, kernel_size = (1, 1), padding = "same")
```

ResNetではこのブロックを積み重ねていきますので、それをクラスにします。今回はResNet-50ですので、Bottleneck Architectureを利用し一旦次元削減してから復元する処理になっています。余談ですが、Bottleneck Architectureではない通常のアーキテクチャーで実装するとResNet-34になります。

Skip Connectionは `self.add` の部分になります。このブロック内で計算してきた `h` とこのブロックの入力である `x` を足し合わせています（その前の `self.shortcut` で `x` の次元を合わせています）。このようにすることで逆伝播の際に勾配消失しづらくなるそうです。

ResNet50

```
class ResNet50(Model):
    def __init__(self, input_shape, output_dim):
        super().__init__()

        self._layers = [
            Conv2D(64, input_shape = input_shape, kernel_size = (7, 7), strides=(2, 2), padding='valid'),
            BatchNormalization(),
            Activation(tf.nn.relu),
            MaxPool2D(pool_size = (3, 3), strides = (2, 2), padding = "same"),
            ResidualBlock(64, 256),
            [
                ResidualBlock(256, 256) for _ in range(2)
            ],
            Conv2D(512, kernel_size = (1, 1), strides=(2, 2)),
            [
                ResidualBlock(512, 512) for _ in range(4)
            ],
            Conv2D(1024, kernel_size = (1, 1), strides=(2, 2)),
            [
                ResidualBlock(1024, 1024) for _ in range(6)
            ],
            Conv2D(2048, kernel_size = (1, 1), strides=(2, 2)),
            [
                ResidualBlock(2048, 2048) for _ in range(3)
            ],
            GlobalAveragePooling2D(),
            Dense(1000, activation = tf.nn.relu),
            Dense(output_dim, activation = tf.nn.softmax)
        ]

    def call(self, x):
        for layer in self._layers:
            if isinstance(layer, list):
                for l in layer:
```

```
def layer(x):
    x = l(x)
else:
    x = layer(x)
return x
```

先ほど作成したResidualBlockや畳み込み層などを組み合わせています。論文の表が参考になるかと思います。今回は論文に忠実に実装したつもりですが、Fashion-MNISTを取り扱う場合、層の数や次元のチューニングは必要かもしれません。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10 ⁹	3.6×10 ⁹	3.8×10 ⁹	7.6×10 ⁹	11.3×10 ⁹

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

モデル作成

```
model = ResNet50((28, 28, 1), 10)
model.build(input_shape = (None, 28, 28, 1))
model.summary()
```

```
""" 結果
Model: "res_net50"
```

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	multiple	3200

batch_normalization_v2 (Batch Normalization)	multiple	256

activation (Activation)	multiple	0

max_pooling2d (MaxPooling2D)	multiple	0

residual_block (ResidualBlock)	multiple	75904
residual_block_1 (ResidualBlock)	multiple	71552
residual_block_2 (ResidualBlock)	multiple	71552
conv2d_11 (Conv2D)	multiple	131584
residual_block_3 (ResidualBlock)	multiple	282368
residual_block_4 (ResidualBlock)	multiple	282368
residual_block_5 (ResidualBlock)	multiple	282368
residual_block_6 (ResidualBlock)	multiple	282368
conv2d_24 (Conv2D)	multiple	525312
residual_block_7 (ResidualBlock)	multiple	1121792
residual_block_8 (ResidualBlock)	multiple	1121792
residual_block_9 (ResidualBlock)	multiple	1121792
residual_block_10 (ResidualBlock)	multiple	1121792
residual_block_11 (ResidualBlock)	multiple	1121792
residual_block_12 (ResidualBlock)	multiple	1121792
conv2d_43 (Conv2D)	multiple	2099200
residual_block_13 (ResidualBlock)	multiple	4471808
residual_block_14 (ResidualBlock)	multiple	4471808
residual_block_15 (ResidualBlock)	multiple	4471808
global_average_pooling2d (GlobalAveragePooling2D)	multiple	0
dense (Dense)	multiple	2049000
dense_1 (Dense)	multiple	10010
=====		

```
Total params: 26,313,218
Trainable params: 26,267,778
Non-trainable params: 45,440
```

""

この `model.build` の意味は正直よくわかっていないのですが、[公式](#)によると、どんな input が来るかわからないサブクラスのために必要とのことで、今回は確かにこれまで書いたコードとは違い input shape を引数として与えているので、そのせいかなあと考えています。

その他

バッチサイズ

バッチサイズは128にしました。論文には256と書かれていましたが、メモリエラー寸前だったので・・・。

最適化手法

Adamにしました。論文にはSGD + Momentumと書かれており、少し試してみたのですが学習の進みが遅いように見受けられたので・・・。余裕があれば、SGD + Momentumでも試してみます。

結果

400エポック訓練した結果、Test Accuracyが91.3%と[前回](#)を下回ってしまいました。チューニングすればもう少し精度が上がるかもしれません。

参考にさせて頂いたコード

- <https://github.com/yusugomori/deeplearning-tf2>
- <https://github.com/kefth/fashion-mnist>

追記 : SGD + Momentumの結果

400エポック訓練した結果、Test Accuracyが91.4%でした。Adamとそう変わらなかったですね。

[編集リクエスト](#)[📄 ストック](#)[LGTM](#)

5



@shoji9x9

2020年1月よりMaaS関係に従事。プライベートでは機械学習、Kaggleに取り組んでいます。

[フォロー](#)

ユーザー登録して、Qiitaをもっと便利に使ってみませんか。

[登録する](#)[ログインする](#)

💬 コメント

この記事にコメントはありません。

あなたもコメントしてみませんか :)

[ユーザー登録](#)

すでにアカウントを持っている方は[ログイン](#)

How developers code is here.



Qiita

[About](#) [利用規約](#) [プライバシー](#) [ガイドライン](#) [リリース](#) [API](#) [ご意見](#) [ヘルプ](#) [広告掲載](#)

Increments

[About](#) [採用情報](#) [ブログ](#) [Qiita Team](#) [Qiita Jobs](#) [Qiita Zine](#)

© 2011-2021 Increments Inc.