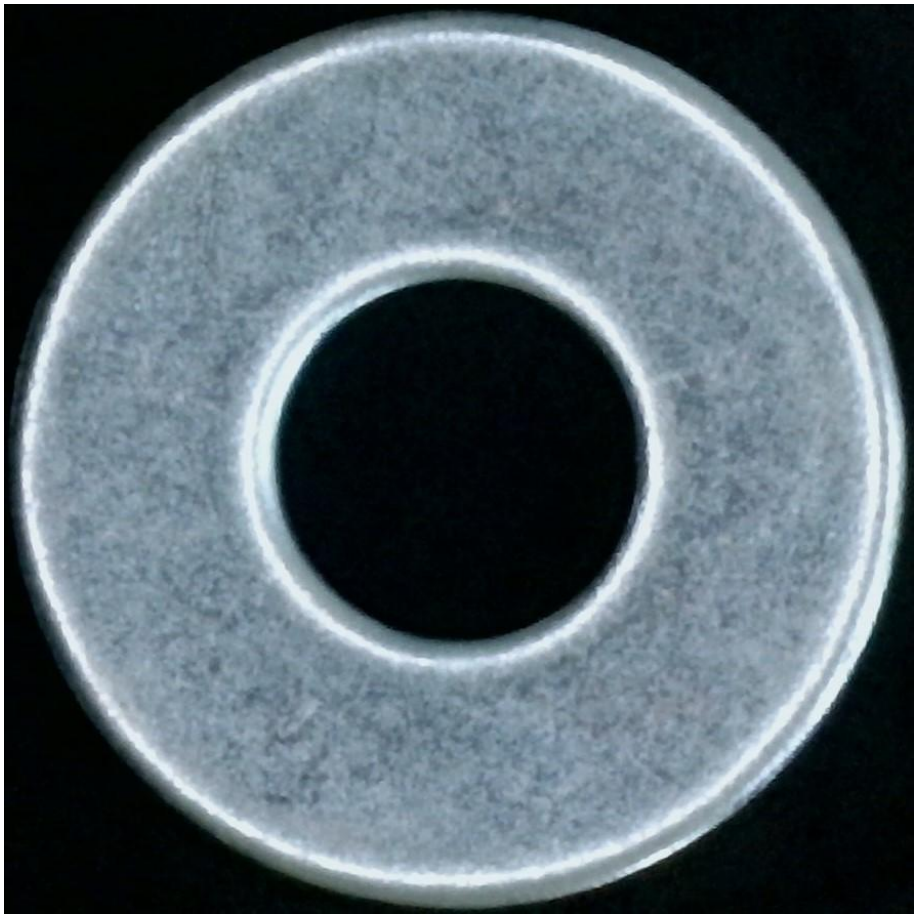


プロジェクト **: Washer**データセット上の異常検知。

提出者: Rahman Sagidur.

問題提起:

優れた学習アルゴリズム（例：Variational AutoEncoder [VAE]）を用いて、「Washer」データセットの異常を検出する。”washer_ok"データセットでモデルを学習し"washer_ng　"データセットでモデルをテストします。

ヒートマップを生成することで、異常を予測検出します。

概要:

異常検知は、コンピュータビジョンの課題です。コンピュータビジョンでは、異常を検出する方法がいくつかあります。"画像分割」は、異常を検出する方法の1つです。

画像分割のアルゴリズムは数多くあります。今回のプロジェクトでは、4つのアルゴリズムを適用しました。データセットを分析した後、私は調査を行い、これらのアルゴリズムを選びました。これらのアルゴリズムをデータセットに適用し、異常を検出しました。

データセット: データセットはRGB画像です。

- ☐ washer_ok :30枚の画像です。
- ☐ washer_ng : 92枚の画像です。

学習データ : washer_ok (30枚の画像です。)

検証データ : washer_ng_kizu (10 0枚の画像です。)

テストデータ : washer_ng_sabi (4 0枚の画像です。)

アルゴリズム: 今回のプロジェクトでは、以下のアルゴリズムを採用しました。

1. Variational AutoEncoder (VAE).
2. U-Net.
3. U-Net Xception Style.
4. Multi-Res-U-Net.

開発環境:

OS: Ubuntu 20.04

CPU: AMD Ryzen7 3700U

Development IDE: Anaconda (version: 4.9.2) , Jupyter Notebook(6.0.3)

Programming Language: python (version: 3.7.9)

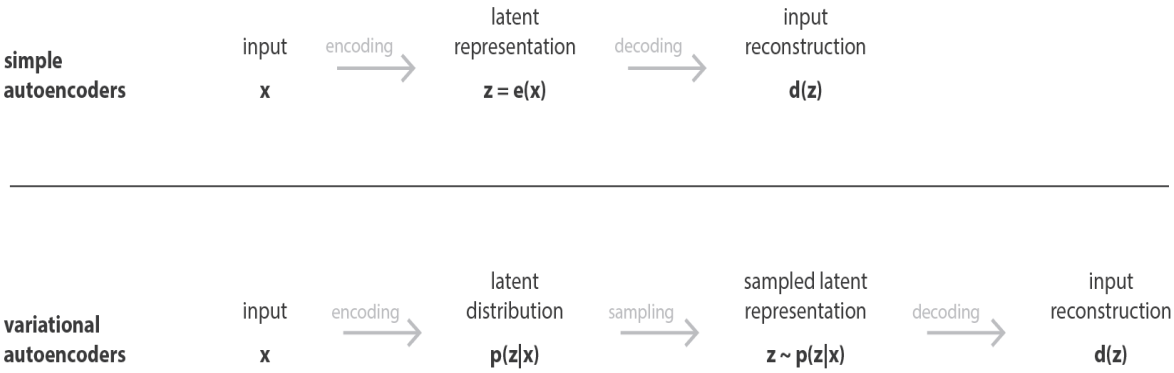
Frameworks: Tensorflow (version: 2.3.0); keras (version: 2.4.3)

Libraries: numpy(version: 1.20.2); matplotlib(version: 3.3.4); pillow(version: 8.2.0)

学習モデル:

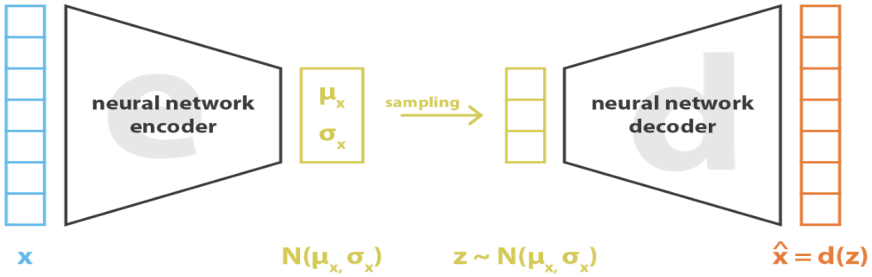
アルゴリズム 1: Variational AutoEncoder (VAE).

VAE: Variational AutoEncoder（VAE）は、オーバーフィッティングを回避し、潜在空間が生成プロセスを可能にする優れた特性を持つように学習を正則化したAutoencoderであると定義できます。

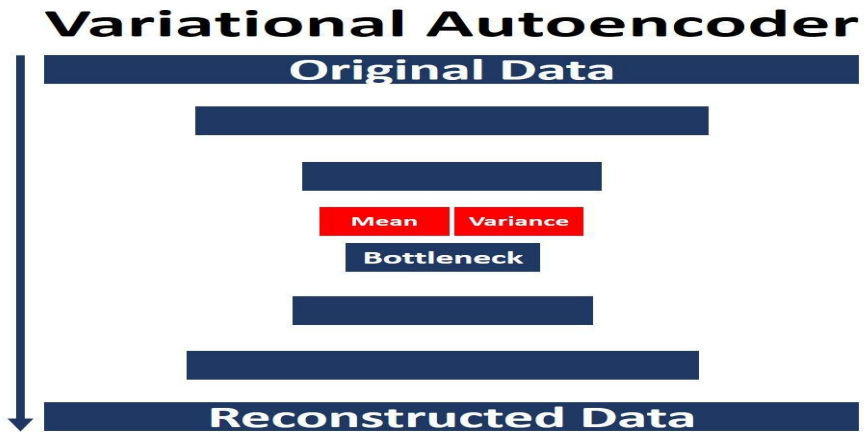


1. 入力は、潜在空間上の分布としてエンコードされます。
2. 潜在的な空間からの点は、その分布からサンプリングされます。
3. サンプリングされたポイントがデコードされ、再構成誤差が計算できます。
4. 最後に、再構築エラーがネットワークを通じて逆伝播されます。

VAE アーキテクチャ



$$\text{loss} = \| x - \hat{x} \|^2 + KL[\, N(\mu_x, \sigma_x), N(0, I) \,] = \| x - d(z) \|^2 + KL[\, N(\mu_x, \sigma_x), N(0, I) \,]$$



VAEモデルのトレーニング、パフォーマンスと結果

トレーニングパラメーター:

img_size = (128, 128)　　# (64,64)

batch_size = 1 #2,3

number_classes = 2 #3,4

number_channels = 3

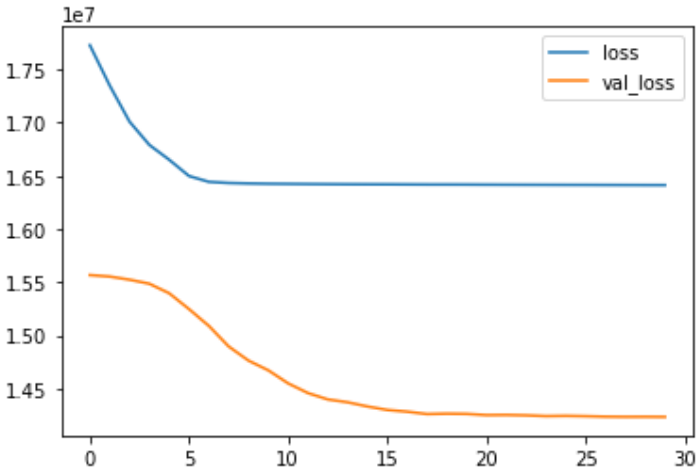
latent_space_dim = 4 #2, 20

optimizer = ADAM (lr = 0.0001)　# SGD

epochs = 30 # 50, 100

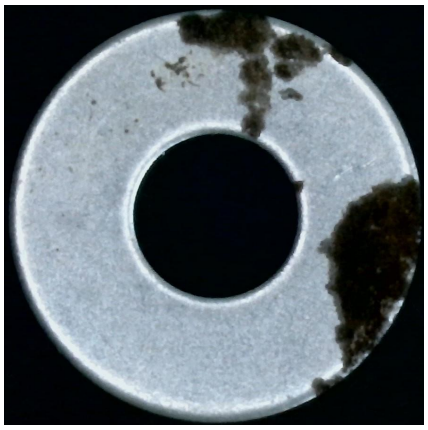
パフォーマンス: img_size、batch_size、number_classes、optimizer、latent_space_dim、 epochs を変えて適用しました。**ここでは、最適化された結果を添付しました。**

モデルロスのグラフ： 損失は1.77le7から始まり、1.64le7で終了しました。

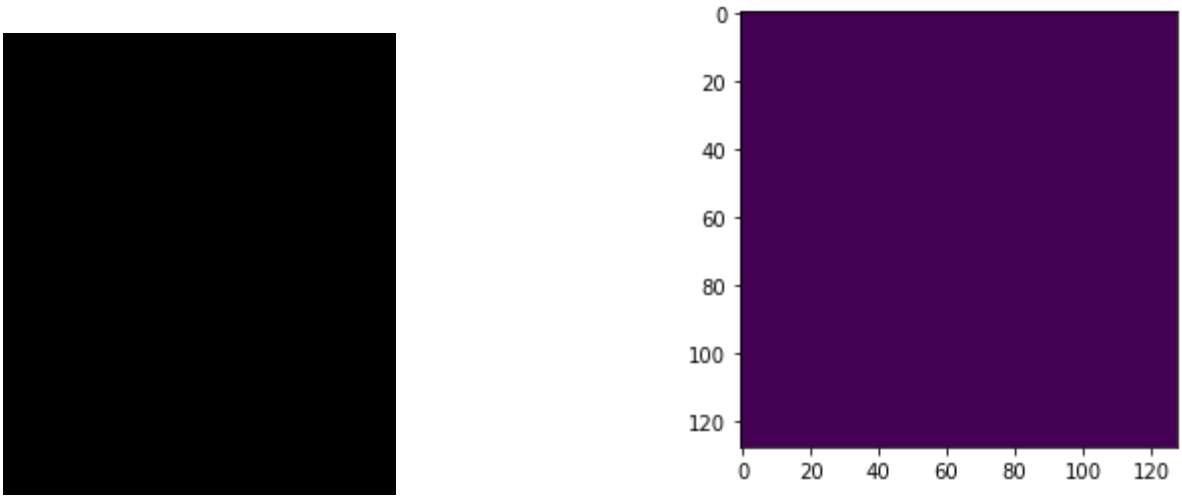


VAEのロスとバリデーション ロスグラフ

ヒートマップ上のモデル出力:



入力画像



出力画像

制限事項 **:**まず第一に、データセットは優れたモデルの最も重要なポイントです。

ここでは、モデルはデータ制限の問題に直面しました。モデルは少ないデータからは学習できませんでした。

おすすめポイント: データ数を増やせば、学習期間中にモデルの学習効果を高めることができます。このモデルは、バッチサイズ、オプティマイザ、学習率、データ増強、潜在的なもの（VAEの場合）などのパラメータの微調整が必要です。

参考:

1. https://ermongroup.github.io/cs228-notes/extras/vae/
2. https://www.jeremyjordan.me/variational-autoencoders/
3. https://towardsdatascience.com/variational-autoencoders-vaes-for-dummies-step-by-step-tutorial-69e6d1c9d8e9

アルゴリズム 2: U-Net.

U-Net: U字型のアーキテクチャは、特定のエンコーダ・デコーダ方式で構成されています。最初のパスは収縮パス（エンコーダーとも呼ばれる）で、画像内のコンテキストをキャプチャするために使用されます。

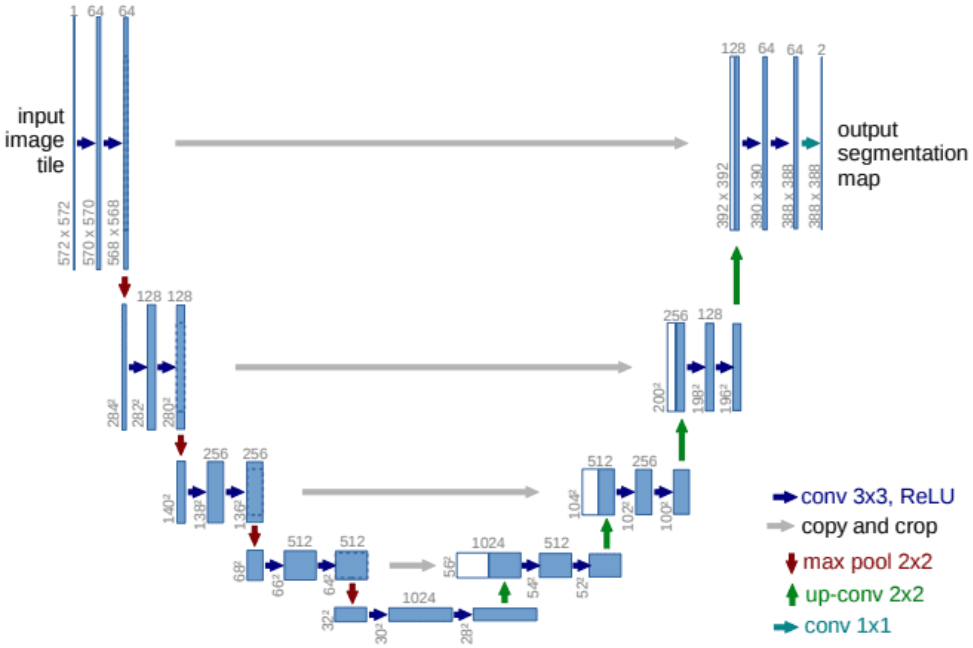
エンコーダーは、コンボリューション層とマックスプーリング層の伝統的なスタックにすぎない。

2つ目のパスは、対称的な拡大パス（デコーダとも呼ばれる）で、転置された畳み込みを使用して正確なローカライズを可能にするために使用されます。

このように、このネットワークはエンド・ツー・エンドの完全な畳み込みネットワーク（FCN）です。

畳み込み層だけで構成されており、高密度層は含まれていないため、あらゆるサイズの画像を受け入れることができます。

U-Netのアーキテクチャ



U-Netモデルのトレーニング、パフォーマンスと結果

トレーニングパラメーター:

img_size = (256, 256)　　# (572,572) (512,512)

batch_size = 1 #2,3

number_classes = 2 #3,4

number_channels = 3

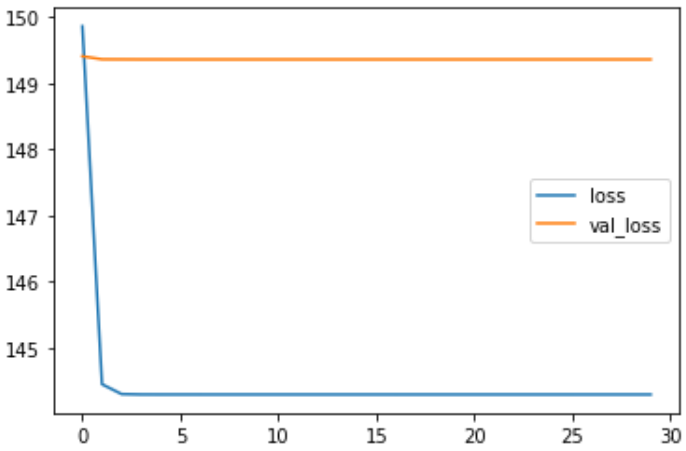
loss= categorical_crossentropy

optimizer = ADAM　# SGD

epochs = 30 # 50, 100

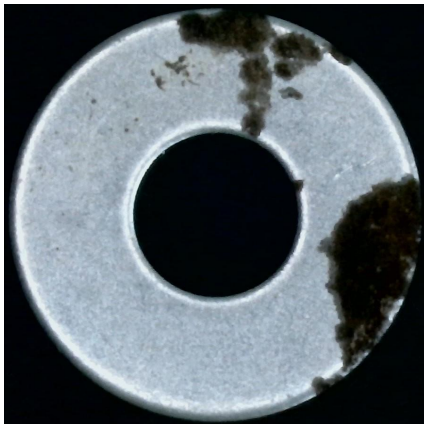
パフォーマンス： img_size、batch_size、number_classes、optimizer、epochsを変えて適用しました。**ここでは、最適化された結果を添付しました。**

モデルロスグラフ： 損失は145.3586から始まり、30エポックの後、144.2891で終了しました。

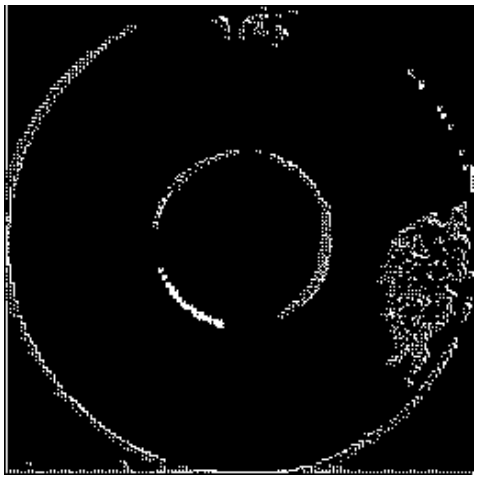


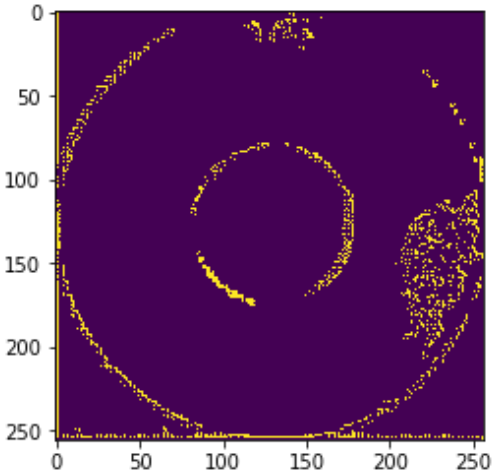
U-Netモデルのロスとバリデーション ロスグラフ

ヒートマップ上のモデル出力:



入力画像





出力画像

制限事項 **:** まず第一に、データは良いモデルの最も重要なポイントです。ここでは、モデルは非常に少ないデータで学習されました。

そのため、U-Netモデルは少ないデータからあまり学習できませんでした。

おすすめポイント : データ数を増やせば、学習期間中にモデルの学習効果を高めることができます。このモデルでは、バッチサイズ、オプティマイザ、学習率、データの増強などのパラメータを微調整する必要があります。そうすれば、U-Netモデルの性能が向上する可能性があります。

参考 :

1. http://www-o.ntust.edu.tw/~cweiwang/ISBI2015/challenge2/isbi2015_Ronneberger.pdf
2. https://arxiv.org/pdf/1505.04597.pdf

アルゴリズム 3: U-Net Xception Style.

このU-Net Xception Styleモデルは、U-Netアーキテクチャを採用しています。つまり、このモデルも理想的なEncoderとDecoderの方法に従っています。

このモデルは、U-NetスタイルのConvNetを使って、160x160xRGBのPETの画像から、画像の各ピクセルの同じ160x160次元のアノテーションマップにマッピングする方法を示しました。

これには、ペットをその境界線と背景から分離することが含まれます。

U-Net Xception Style モデルのトレーニング、パフォーマンスと結果

トレーニングパラメーター:

img_size = (160, 160)　　# (256,256) (224,224)

batch_size = 1 #2,3,4

number_classes = 2 #3,4

number_channels = 3

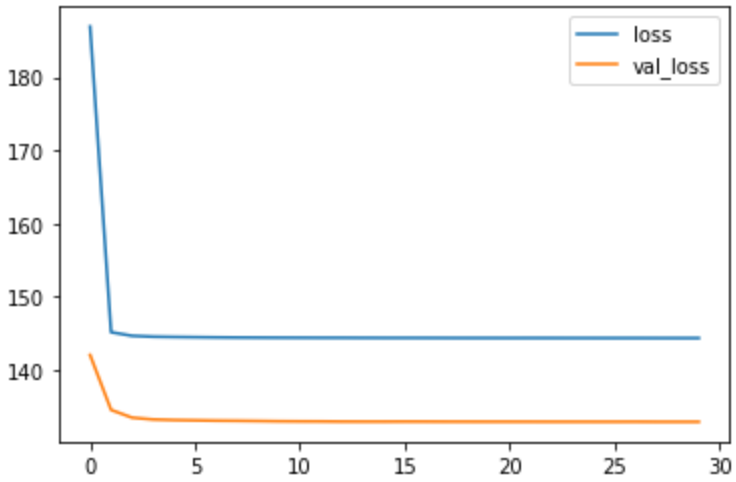
loss= categorical_crossentropy

optimizer = ADAM　# RMSPROP

epochs = 30 # 50, 100

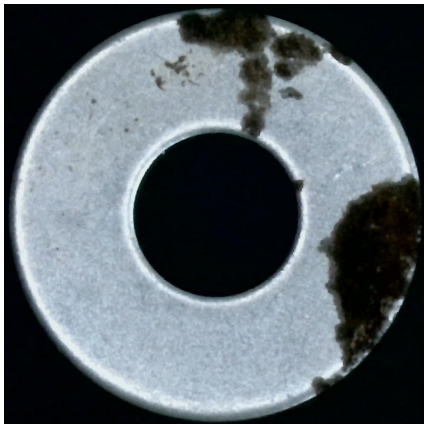
パフォーマンス： img_size、batch_size、number_classes、optimizer、epochsを変えて適用しました。**ここでは、最適化された結果を添付しました。**

モデルロスグラフ**：** 損失は、30エポック後に186.9564から始まり、144.2846で終了しました。

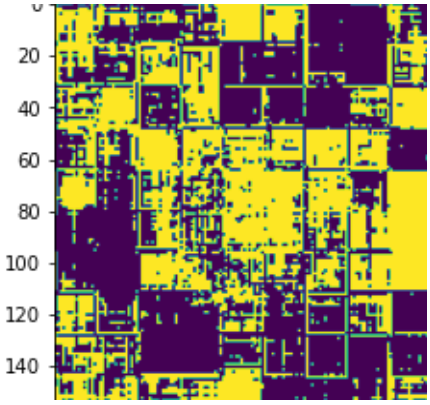


U-Net Xception Styleモデル**のロスとバリデーション ロスグラフ**

ヒートマップ上のモデル出力:



入力画像



出力画像

制限事項 **:** まず第一に、データは良いモデルの最も重要なポイントです。ここでは、モデルは非常に少ないデータで学習されました。

そのため、U-Net Xception Styleのモデルは、少ないデータからは学習できませんでした。

おすすめポイント : データ数が増えれば、学習期間中にモデルがよりよく学習できるようになります。

このモデルでは、バッチサイズ、オプティマイザ、学習率、データ増強などのパラメータを微調整する必要があります。

そうすれば、U-Net Xception Styleモデルの性能が向上する可能性があります。

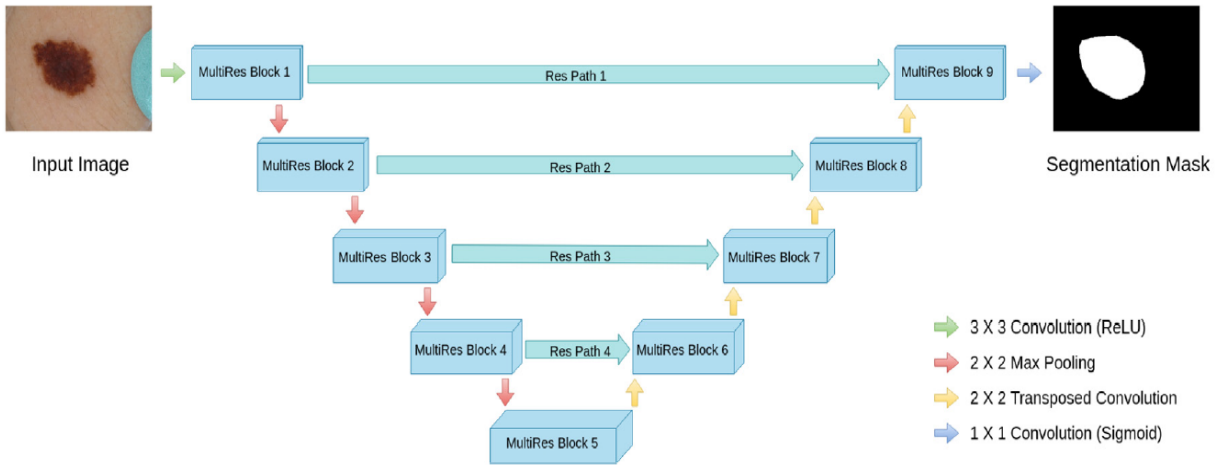
参考:

1. https://keras.io/examples/vision/oxford_pets_image_segmentation/

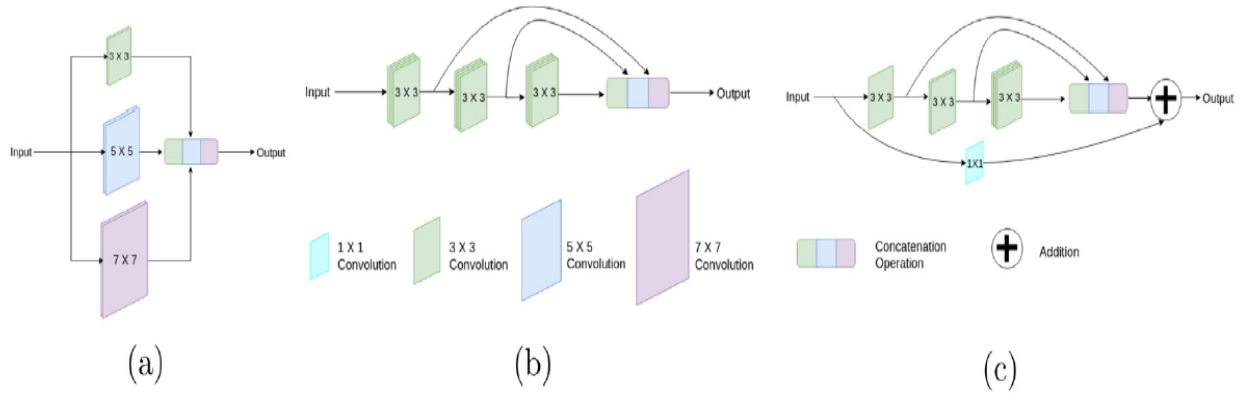
アルゴリズム 4: Multi-Res-U-Net.

Multi-Res-U-Net: このプロジェクトでは、著者らは驚異的なU-Netアーキテクチャから動機を得て、U-Netモデルを修正しました。

Multi-Res-U-Net: U-Netでの変更点



From 2 Conv Layers in U-Net to Multi-Res Block in Multi-Res-U-Net



Developing the proposed MultiRes block from (a) to ©, © MultiRes Block

- オリジナルのU-Netの各レベルにおける2つの畳み込み層のシーケンスについては、提案されたMultiResブロックに置き換えられています。

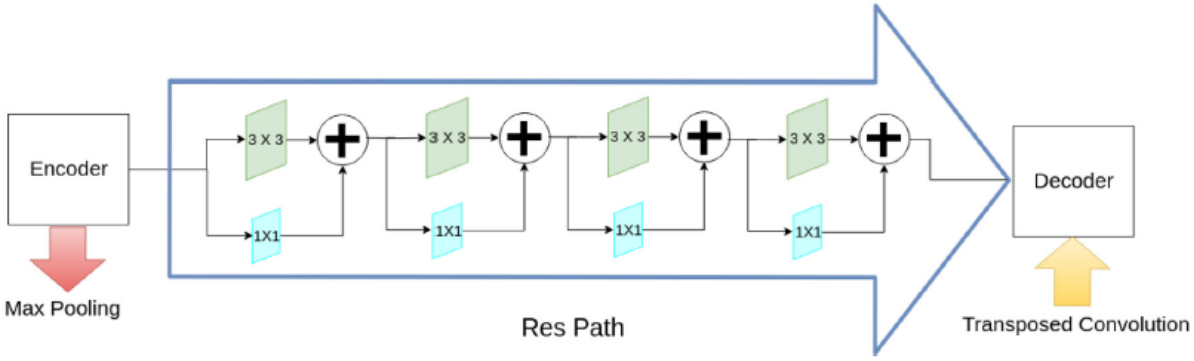
(a):まず、3×3、5×5、7×7の畳み込みフィルターを並列に使用して、異なるコンテキストサイズからの空間的特徴を調整することで、シンプルなInceptionのようなブロックから始めます。

(b): そして、この大きなフィルターは、3×3のフィルターの連続に因数分解されます。

(c): 最後に、MultiResブロックは、連続する3つの層のフィルターの数を徐々に増やし、寸法を節約するために1×1のフィルターとともに、残留接続を追加することによって確立されます。

- これは、DenseNetのDenseBlockと同様に、ResNetを起点とした残余経路です。

ResPath in Multi-Res-U-Net



ResPath with 1×1 and 3×3 filters

- ResPathには、上図のように3×3と1×1のフィルターがあります。3×3と1×1のフィルターの数は、ネットワーク内のレベルに応じて、以下の表のようになります。
- 著者は、エンコーダとデコーダの特徴マップ間のセマンティックギャップの強度が減少する可能性があるという仮説を立てた。
- このような非線形演算を追加することで、エンコーダとデコーダの特徴の間のセマンティックギャップを減らすことができると期待されています。

フィーチャーマップの数

U-Netと公平に比較するためには、2つのモデル間で同程度のパラメータ数を維持する必要があります。

$$W = \alpha \times U$$

- ここで、UとWはそれぞれU-NetとMultiResUNetの1つの畳み込み層に含まれるフィルターの数です。α=1.67が使われています。したがって、以下のようなフィルター数には、すでにαがかけられています。

Multi-Res-U-Netアーキテクチャの概要

以下に、Multi-Res-U-Netのアーキテクチャの詳細を示します。

MultiResUNet					
Block	Layer (filter size)	#filters	Path	Layer (filter size)	#filters
MultiRes Block 1	Conv2D(3,3)	8	Res Path 1	Conv2D(3,3)	32
	Conv2D(3,3)	17		Conv2D(1,1)	32
MultiRes Block 9	Conv2D(3,3)	26		Conv2D(3,3)	32
	Conv2D(1,1)	51		Conv2D(1,1)	32
MultiRes Block 2	Conv2D(3,3)	17		Conv2D(3,3)	32
	Conv2D(3,3)	35		Conv2D(1,1)	32
MultiRes Block 8	Conv2D(3,3)	53		Conv2D(3,3)	32
	Conv2D(1,1)	105		Conv2D(1,1)	32
MultiRes Block 3	Conv2D(3,3)	35	Res Path 2	Conv2D(3,3)	64
	Conv2D(3,3)	71		Conv2D(1,1)	64
MultiRes Block 7	Conv2D(3,3)	106		Conv2D(3,3)	64
	Conv2D(1,1)	212		Conv2D(1,1)	64
MultiRes Block 4	Conv2D(3,3)	71		Conv2D(3,3)	64
	Conv2D(3,3)	142		Conv2D(1,1)	64
MultiRes Block 6	Conv2D(3,3)	213	ResPath 3	Conv2D(3,3)	128
	Conv2D(1,1)	426		Conv2D(1,1)	128
MultiRes Block 5	Conv2D(3,3)	142		Conv2D(3,3)	128
	Conv2D(3,3)	284		Conv2D(1,1)	128
	Conv2D(3,3)	427	Res Path 4	Conv2D(3,3)	256
	Conv2D(1,1)	853		Conv2D(1,1)	256

Multi-Res-U-Netモデルのトレーニング、性能と結果

トレーニングパラメーター:

img_size = (160, 160)　　# (256,192)

batch_size = 1 #2,3,4

number_classes = 2 #3,4

number_channels = 3

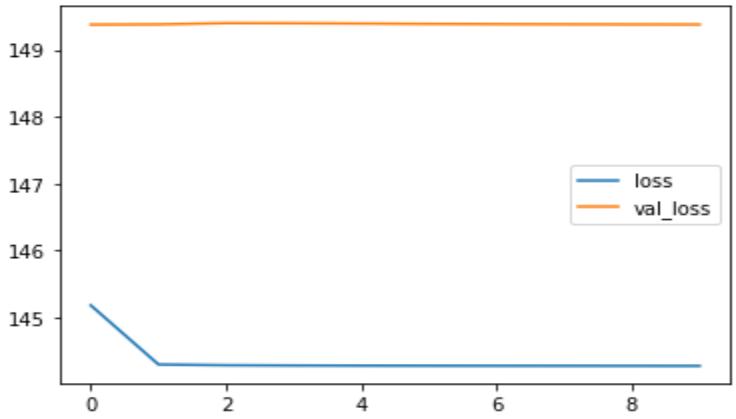
loss= categorical_crossentropy

optimizer = ADAM   # RMSPROP

epochs = 10 # 30

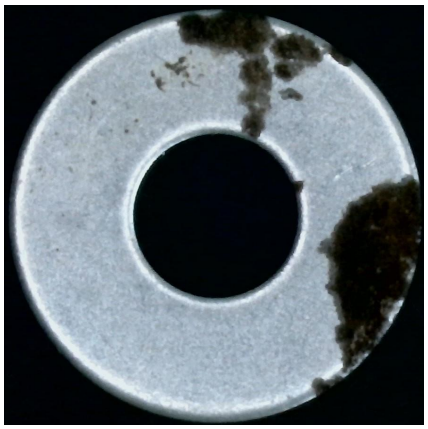
パフォーマンス： img_size、batch_size、number_classes、optimizer、epochsを変えて適用しました。最適化された結果をここに添付します。4つのモデルの中では、Multi-Res-U-Netが最も良い結果となりました。

モデルロスグラフ**：** 損失は10エポック後の145.1759から始まり、144.2701で終了しました。

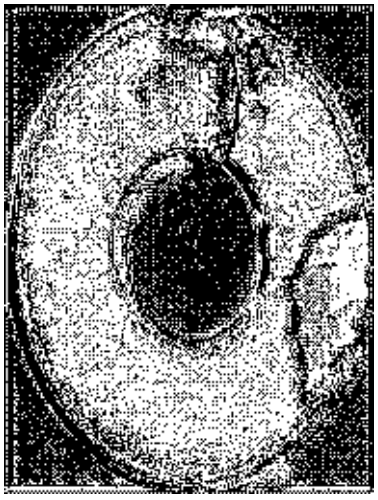


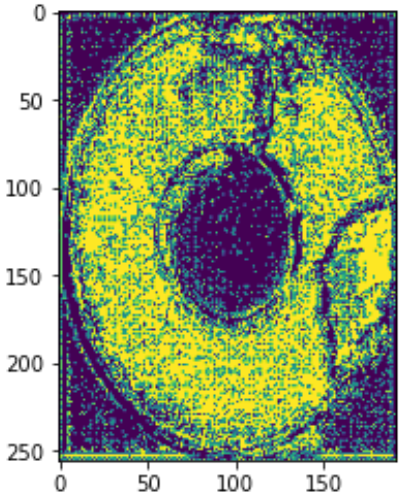
Multi-Res-U-Netの損失と検証損失グラフです。

ヒートマップ上のモデル出力: ここでは、入力画像サイズが(256,192)の場合のモデル出力画像を添付しました。この画像サイズは研究論文で使用されています。



入力画像

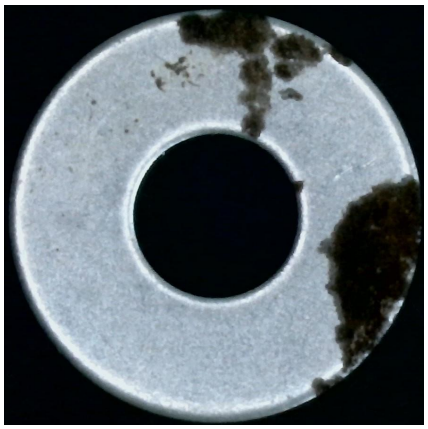




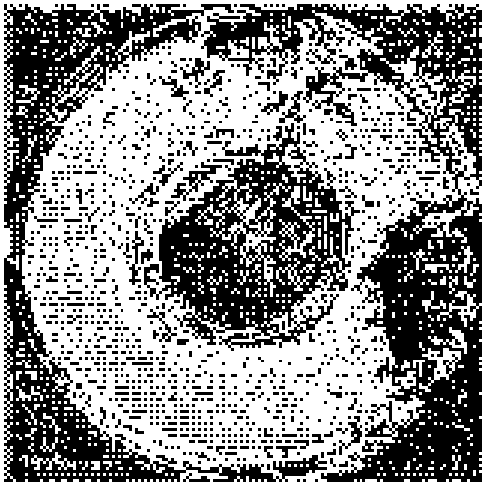
出力画像

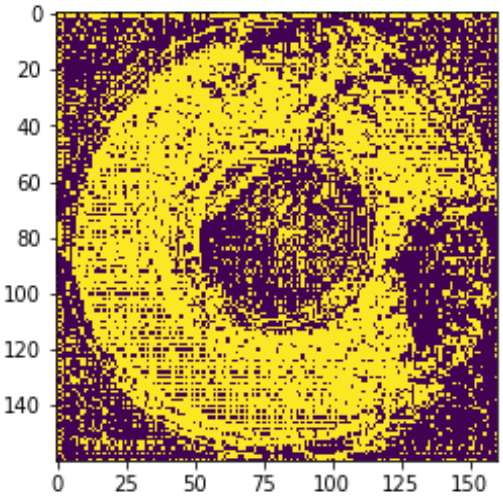
ヒートマップ上のモデル出力: ここでは、入力画像のサイズが(160,160)の場合のモデル出力画像を添付しています。

(256,192)の画像サイズと(160,160)の画像サイズでは、どちらも優れた性能を発揮しました。



入力画像





出力画像

制限事項：まず第一に、データは良いモデルの最も重要なポイントです。ここでは、非常に少ないデータでモデルを学習しました。Multi-Res-U-Netモデルは、少ないデータから少ない特徴を学習することができました。

このモデルは、ほとんどの欠陥のある画像に対して良い結果を与えます。

おすすめポイント：データ数が増えれば、学習期間中にモデルがより多くのことを学べるようになります。このモデルは、バッチサイズ、オプティマイザ、学習率、データ増強などのパラメータを微調整する必要があります。そうすれば、Multi-Res-U-Netモデルの性能が向上する可能性が高くなります。

参考 : https://arxiv.org/pdf/1902.04049.pdf

モデル比較

まず、VAEアルゴリズムを研究し、次にEncoderとDecoderの考え方に沿った他のセグメンテーションアルゴリズムを研究しました。

その結果、Washerデータセットに適した4つのアルゴリズムをリストアップしました。

この4つのモデルを導入し、トレーニングを行った結果 その中でも2つのモデルが最も良い結果をもたらしました。

モデル:

1. MultiResUNet
2. U-Net.