Segmentation Image

Théophile Hesters

January 2022

1 Le jeu de données

Pour le jeu de données mis à disposition nous allons utiliser le "Oxford Pets dataset" qui se compose d'image de chiens et de chats avec des annotations permettant de "segmenter" l'image et définir leur position

2 L'architecture U-Net

L'un des réseaux de neurones les plus utilisés pour la segmentation d'image est U-NET. Il s'agit d'un Modèle de Réseau de Neurones Entièrement Convolutif. L'architecture de U-NET est composée de deux " chemins ". Le premier est le chemin de contraction, aussi appelé encodeur. Il est utilisé pour capturer le contexte d'une image. Il s'agit en fait d'un assemblage de couches de convolution et de couches de "max pooling" permettant de créer une carte de caractéristiques d'une image et de réduire sa taille pour diminuer le nombre de paramètres du réseau. Le second chemin est celui de l'expansion symétrique, aussi appelé décodeur. Il permet aussi une localisation précise grâce à la convolution transposée.

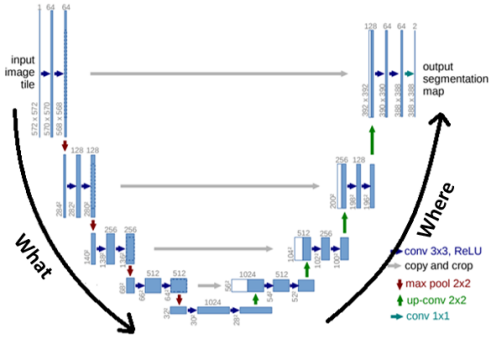

Figure 1: Architecture u-Net Encodeur Decodeur

3 Modèle de base avec Keras

Le modèle de base est un modèle entraîné from scratch à l'inverse du transfer-learning.La première partie du réseaux, l'Encodeur va servir à faire le sous-échantillonnage des entrées ou DownSampling (on détecte les features)

```
for filters in [64, 128, 256]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)
        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)
        # Project residual
        residual = layers.Conv2D(filters, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual
```

Dans la deuxième partie du réseaux nous allons faire de l'upsampling , du suréchantillonage (on localise les features)

```
for filters in [256, 128, 64, 32]:
        x = layers.Activation("relu")(x)
        x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation("relu")(x)
        x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)
        x = layers.UpSampling2D(2)(x)
        # Project residual
        residual = layers.UpSampling2D(2)(previous_block_activation)
        residual = layers.Conv2D(filters, 1, padding="same")(residual)
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual
```

Ensuite on va faire de la classification par pixel.

```
outputs = layers.Conv2D(num_classes, 3, activation="softmax", padding="same")(x)
```

Ce qui nous donne :

```
Total params: 2,937,027
Trainable params: 2,931,395
Non-trainable params: 5,632
```

On va utiliser *loss="sparse categorical crossentropy"* et *metrics="SparseCategoricalAccuracy"* pour l'accuracy du modèle.

Résultats

Optimizer	Train acc	Train Loss	Val acc	Val Loss	Epochs
Adam	0.9212	0.1910	0.8605	0.4363	15
rmsprop	0.9465	0.1263	0.8838	0.3972	15

4 En faisant du transfer learning

Pour le transfer learning nous allons utiliser les bases de mobilnetV2.
MobileNetV2 est une architecture de réseau neuronal convolutif qui cherche à être performante sur les appareils mobiles. Il est basé sur une structure résiduelle inversée où les connexions résiduelles se trouvent entre les couches d'étranglement. La couche d'expansion intermédiaire utilise des convolutions légères en profondeur pour filtrer les caractéristiques comme source de non-linéarité. Dans l'ensemble, l'architecture de MobileNetV2 contient la couche initiale à convolution complète avec 32 filtres, suivie de 19 couches résiduelles à goulot d'étranglement.

Dans ce modèle nous allons récupérer les activations des couches suivantes :

```
'block_1_expand_relu',      # 64x64
  'block_3_expand_relu',      # 32x32
  'block_6_expand_relu',      # 16x16
  'block_13_expand_relu',     # 8x8
  'block_16_project'          # 4x4
]
```

Afin de créer la partie Downsampling (partie extraction de features) On va ensuite préparer la partie upsampling avec des couches de Deconvolution

```
up_stack = [
  DeConv2D_block(filters=512, ksize=(3, 3)),   # 4x4 => 8x8
  DeConv2D_block(filters=256, ksize=(3, 3)),   # 8x8 => 16x16
  DeConv2D_block(filters=128, ksize=(3, 3)),   # 16x16 => 32x32
  DeConv2D_block(filters=64, ksize=(3, 3)),    # 32x32 => 64x64
]
```

On va ensuite créer le modèle UNet à partir des deux "morceaux" de modèles que nous avons créer

- DownStack (DownSampling)
- UpStack (UpSampling)

```
Cette fois ci on se retrouve avec :
Total params: 6,504,227
Trainable params: 6,471,395
Non-trainable params: 32,832
```

Comme pour le modèle précedent on va utiliser *loss="sparse categorical crossentropy"* et *metrics="SparseCategoricalAccuracy"* pour l'accuracy du modèle.

Résultats

Optimizer	Train acc	Train Loss	Val acc	Val Loss	Epochs
Adam	0.9368	0.1388	0.88922	0.3174	15
rmsprop	0.9433	0.1225	0.8833	0.4027	15

5 Comparaison finale des deux modèles

Les deux modèles ont des résultats assez similaires avec un avantage pour le transfer learning sur mobileNetV2 avec Adam en optimizer.

MobilNetV2

Optimizer	Train acc	Train Loss	Val acc	Val Loss	Epochs
Adam	0.9368	0.1388	0.88922	0.3174	15
rmsprop	0.9433	0.1225	0.8833	0.4027	15

Base Model

Optimizer	Train acc	Train Loss	Val acc	Val Loss	Epochs
Adam	0.9212	0.1910	0.8605	0.4363	15
rmsprop	0.9465	0.1263	0.8838	0.3972	15

Résultat pour le meilleur modèle

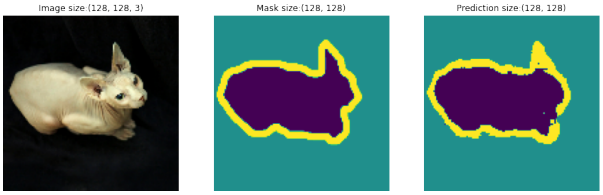

Figure 2: Architecture u-Net Encodeur Decodeur

6 Pour aller plus loin

Nous allons donc nous concentrer sur le modèle se basant sur mobileNetV2 et essayer d'améliorer ses performances en modifiants certains paramètres d'entraînement Dans un premier temps on va rajouter des epochs, changer le split pour la validation et voir si on peut déjà gagne quelques points de précisions.

Resultat sur le meilleur modèle avec 30 epochs au lieu de 15

Optimizer	Train acc	Train Loss	Val acc	Val Loss	Epochs
rmsprop	0.9726	0.0441	0.8850	0.5754	30

On peut apercevoir une meilleur précision sur le jeu d'entraînement mais pas une amélioration majeure par rapport aux autres modèles.

7 Conclusion

On pourrait continuer les tests en intégrant de la DataAugmentation, rajouter des epochs, modifier les différents hyperparamètres (learningRate...). Il serait intéressant aussi de faire des comparaisons plus avancées sur les temps d'apprentissages et d'inférences des deux modèles sur le même jeu de validation.