SY32 – TD Appr. Profond 02 : Conception et entraînement de réseaux de neurones profonds pour de la classification binaire

Exercice 1 : Classification de visages

On souhaite apprendre un réseau de neurones convolutifs pour classer les images de visages du TD-A03. On propose d'étudier tout d'abord le réseau de neurones convolutif Net0 illustré sur la figure 1.

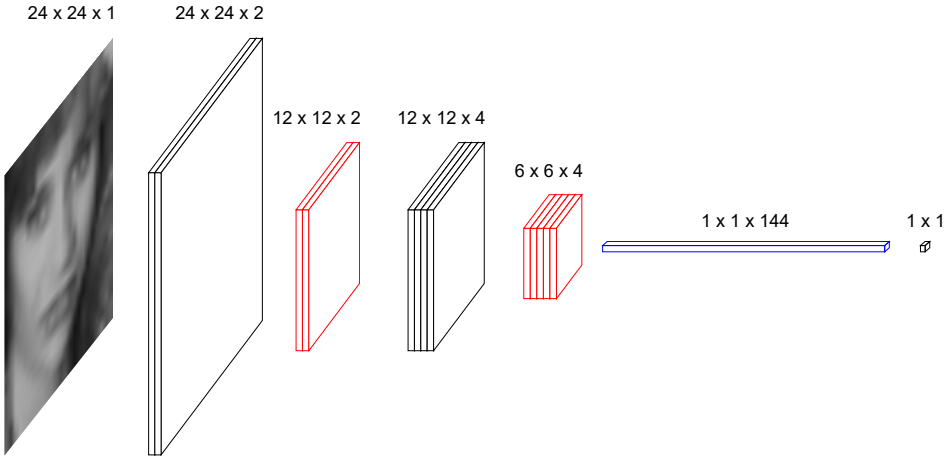

FIGURE 1 – Net0 : réseau de neurones convolutifs à trois couches.

Les différentes étapes constituant le réseau sont les suivantes :

- Entrée : une image en noir et blanc de taille 24×24.
- Une première convolution avec 2 noyaux de tailles 3×3 avec un *padding* de 1 pixel, suivie d'une activation ReLU (Rectified Linear Unit).
- Une réduction de dimension par *max pooling* sur des fenêtres 2×2.
- Une deuxième convolution avec 4 noyaux de tailles 3×3 avec un *padding* de 1 pixel, suivie d'une activation ReLU (Rectified Linear Unit).
- Une réduction de dimension par *max pooling* sur des fenêtres 2×2.
- Un aplatissement du tenseur en un vecteur de longueur 144.
- Une couche entièrement connectée (*fully connected*) avec une seule sortie.

1 . Calculer le nombre de paramètres du réseau Net0 illustré sur la figure 1.

2 . La classe `FaceNet` du fichier `faceClassifier.py` implémente le réseau Net0 avec PyTorch. Utiliser la commande ci-dessous pour vérifier le nombre de paramètres calculé à la question précédente :

```
from faceClassifier import FaceNet
net0 = FaceNet()
print(sum(p.numel() for p in net0.parameters()))
```

3 . Charger les données d'apprentissage et de test à l'aide de la commande ci-dessous :

```
import numpy as np
from skimage.io import imread

X_train = np.zeros((24, 24, 15000), dtype=np.uint8)
for i in range(3000):
    X_train[:, :, i] = imread('imageface/train/pos/%05d.png' % (i + 1))
for i in range(12000):
    X_train[:, :, i + 3000] = imread('imageface/train/neg/%05d.png' % (i + 1))

y_train = np.concatenate((np.ones(3000), -np.ones(12000)))

X_test = np.zeros((24, 24, 6256), dtype=np.uint8)
for i in range(1000):
    X_test[:, :, i] = imread('imageface/test/pos/%05d.png' % (i + 1))
```

```
for i in range(5256):
    X_test[:, :, i + 1000] = imread('imageface/test/neg/%05d.png' % (i + 1))

y_test = np.concatenate((np.ones(1000), -np.ones(5256)))
```

4 . Pour l'apprentissage du réseau de neurones, on utilise une descende de gradient stochastique avec un sous-ensemble de taille `batch_size` des données d'apprentissage. On doit également définir combien d'itérations `n_epoch` doivent être faites sur l'ensemble des données. L'apprentissage des paramètres du réseau se fait à l'aide de la fonction `fit` :

```
clf = FaceClassifier()
clf.fit(X_train, y_train, n_epoch=5, batch_size=50)
```

Calculer à l'aide de la fonction `predict` le taux d'erreur sur l'ensemble de test pour un réseau entraîné avec `n_epoch`=5, 10, 15, 20, ...

Note : il est possible d'appeler plusieurs fois la fonction `fit` afin de poursuivre l'apprentissage avec des itérations supplémentaires.

On souhaite maintenant doubler le nombre de noyaux des deux couches de convolution afin d'obtenir le réseau Net1 illustré sur la figure 2.

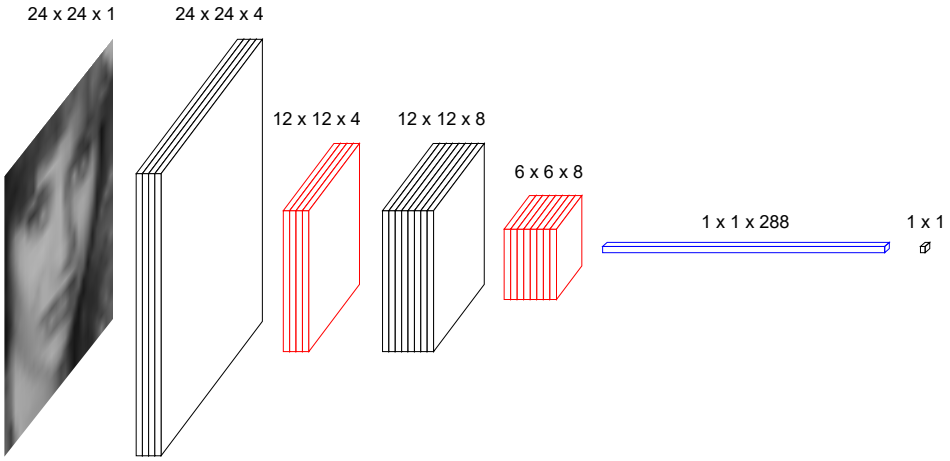

FIGURE 2 – Net1 : réseau de neurones convolutifs à trois couches.

5 . Dans la classe `FaceNet`, modifier les couches `conv1`, `conv2` et `fc1` de la méthode `__init__` afin de coder l'architecture du réseau Net1, puis reprendre l'étude précédente avec ce nouveau réseau.

`Conv2d` : `https://pytorch.org/docs/stable/nn.html#torch.nn.Conv2d`

`Linear` : `https://pytorch.org/docs/stable/nn.html#torch.nn.Linear`

Maintenant, au lieu d'augmenter le nombre de noyaux sur les deux premières couches de convolution, on souhaite plutôt ajouter une troisième couche de convolution et de réduction de dimension afin d'obtenir le réseau Net2 illustré sur la figure 3.

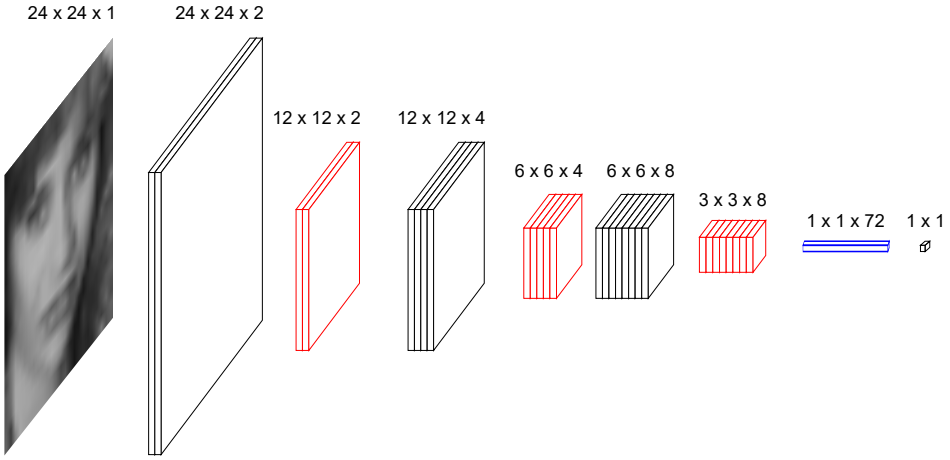

FIGURE 3 – Net2 : réseau de neurones convolutifs à quatre couches.

6 . Dans la classe `FaceNet`, modifier les méthodes `__init__` et `forward` afin de coder l'architecture du réseau Net2, puis reprendre l'étude avec ce nouveau réseau.

`relu` : `https://pytorch.org/docs/stable/nn.functional.html#torch.nn.functional.relu`

`max_pool2d` : `https://pytorch.org/docs/stable/nn.functional.html#torch.nn.functional.max_pool2d`