Correction du test du 20/12/2024

1) Qu'est-ce qu'un problème de Machine Learning ?
Quels sont les trois points auxquels on prête généralement attention quand on aborde un tel problème ? → Voir test précédent.

2) Qu'est-ce qu'une segmentation panoptique ?
C'est une tâche combinant segmentation sémantique et segmentation d'instance.

3) Qu'est-ce que la mIoU (mean Intersection over Union) ? C'est un score utilisé pour évaluer les performances sur une tâche de segmentation.
Illustrer un calcul de mIoU à partir d'une matrice de confusion pour une tâche de segmentation avec au moins trois classes (penser à préciser le rôle des lignes et des colonnes de votre matrice de confusion).
Exemple de calcul à partir d'une matrice de confusion pour un problème à trois classes :

Obs. / Pred	Classe 0	Classe 1	Classe 2
Classe 0	5	2	0
Classe 1	1	2	2
Classe 2	0	1	7

Pour la classe 0 : IoU = 5 / 8, pour la classe 1 : 2 / 8, pour la classe 2 : 7 / 10.
En moyenne : 0,525

4) On se donne l'architecture suivante (ResNet50) (voir énoncé)
4.**a)** Quel est la taille du champ réceptif associé à une composante d'une carte de caractéristique en sortie du troisième bloc (flèche rouge). Justifier.

En sortie de la première couche de convolution, le champ réceptif associé à un pixel d'une carte de caractéristique est de taille 7x7.
On rappelle que la stride est de 2 pour cette première couche.
Le maxpooling suivant agrège les composantes provenant de 9 carrés de taille 7x7 se chevauchant deux à deux sur une bande de 5 pixels de large*. En sortie du maxpooling, le champ réceptif est donc de 11x11.
Les convolutions suivantes de noyau de taille 1x1 ne changent pas la taille du champ réceptif. Par contre, après la première convolution 3x3, un pixel agrège les composantes provenant de 9 carrés de tailles 11x11 se chevauchant deux à deux sur des bandes de 9 pixels de large.
Le champ réceptif est donc de taille 15x15.
Pour les deux convolutions suivantes de noyau 3x3, les chevauchements sont larges de 14 pixels puis 16 pixels (stride=1). Ainsi, la taille du champ passe à 17 puis à 19.

*dans le recap 5, j'ai fait le calcul avec un maxpooling dont le noyau est de taille 2x2. Mais l'implémentation officielle comporte un maxpooling de noyau 3x3 avec une stride de 2. Si vous avez fait le calcul avec un maxpooling 2x2, tous les points seront accordés.

4.**b)** Ce réseau est-il adapté à une prédiction par pixel ? Pourquoi ?
Non : les dimensions spatiales des tailles de caractéristique sont réduites progressivement et la couche finale, une couche complétement connectée, implique la perte d'une des dimensions spatiales (applatissement).

5) On considère l'architecture donnée au verso.

5.**a)** Quel type de signal le réseau XNet prend-il en entrée (Images ? Séries-temporelles ? Donnée 3D ?)

Dans le .*forward* de **XNet**, self.inc(x) commence par l'application de `Double_conv(n_channels, size)`
qui implique d'abord une Conv1D. Le signal d'entrée doit donc être un vecteur, éventuellement une série temporelle.

5.**b)** Par quels mécanismes la résolution du signal d'entrée est-elle réduite ?

Par des stride =2 et des maxpooling1D.

Comment est-elle augmentée par la suite ?

Avec un `nn.ConvTranspose1d`

5.**c)** Par quels mécanismes le traitement multi-échelles est-il favorisé ?

Avec une « skip connection » (`torch.cat([x2, x1], dim=1)`) et un spatial pyramid pooling construit sur des convolution « atrous » (ASPP)

6) On se propose de pratiquer une régression quantile à partir de la fonction de coût suivante (z correspond est une composante de la sortie, y est une composante de la cible et t est un paramètre):

6.**a)** Quel est le nom de cette fonction de coût ?

La « Pinball Loss »

6.**b)** Pouvez-montrer que la quantité minimisant cette expression est effectivement un quantile ? (Partez du cas $t = 0.5$: c'est la même démonstration).

Faite en cours

```
class double_conv(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(double_conv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv1d(in_ch, out_ch, 3, padding=1), nn.ReLU(inplace=True),
            nn.Conv1d(out_ch, out_ch, 3, padding=1), nn.ReLU(inplace=True)
        )
    def forward(self, x):
        x = self.conv(x)
        return x

class Down(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(Down, self).__init__()
        self.mpconv = nn.Sequential(nn.MaxPool1d(2),
                                    double_conv(in_ch, out_ch))

    def forward(self, x):
        x = self.mpconv(x)
        return x

class Up(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(Up, self).__init__()
        self.up = nn.ConvTranspose1d(in_ch, in_ch, kernel_size=2, stride=2)
        self.conv = double_conv(2*in_ch, out_ch)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        diffX = x1.size()[2] - x2.size()[2]
        x2 = F.pad(x2, (diffX // 2, int(diffX / 2)))
        x = torch.cat([x2, x1], dim=1)
        x = self.conv(x)
        return x

class ASPP(nn.Module):
        # Atrous Spatial Pyramid Pooling
        ...

class XNet(nn.Module):
    def __init__(self, n_channels, n_classes, size=64,
```

$$\mathcal{L}_t(z, y) = t \times max(z - y, 0) + (t - 1) \times min(z - y, 0)$$

```
        super(XNet, self).__init__()
        self.inc = double_conv(n_channels, size)
        self.down1 = Down(size, 2*size, pooling=False)
        self.down2 = Down(2*size, 4*size, pooling=False)
        self.down3 = Down(4*size, 8*size, pooling_kernel_size=5,
                          pooling_stride=5)
        self.down4 = Down(8*size, 4*size, pooling=False,
                          dilation=2)
        self.atrous = ASPP(4*size, rates=atrous_rates)
        self.up = Up(4*size, n_classes, kernel_size=5, stride=5)
        self.n_classes=n_classes


    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x6 = self.atrous(x5)
        x = self.up(x6, x3)
        return x
```