Institut Polytechnique de Paris

UFR Sciences

Année Universitaire 2023/2024

TP2 Compte Rendu

Fusion par apprentissage supervisé

Module Fusion de données

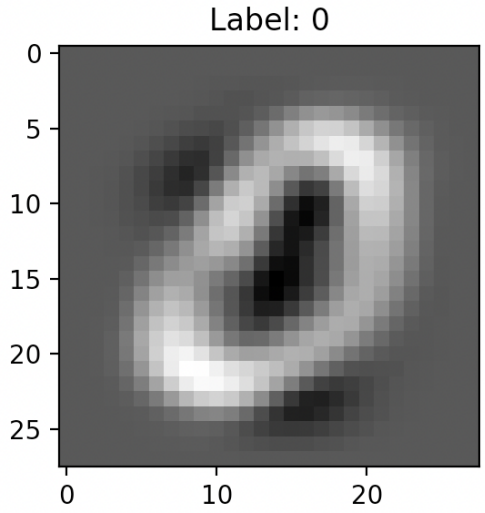
Enseignants :
Emanuel Aldea

Étudiants :
Ziyu LI

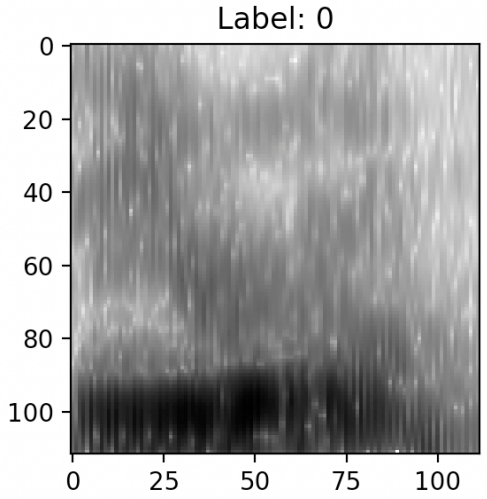
Table des matières

Introduction

Le but du TP consiste à réaliser plusieurs types de fusions entre deux modalités différentes, image et audio , pour un jeu de données relativement simple afin de permettre un entraînement raisonnable. Il s'agit des chiffres, la différence par rapport à MNIST étant qu'il y a également une information audio Fig.1(b) concernant la classe, en plus de l'information image Fig.1(a). Il s'agit d'images en resolution 28×28 et des signaux audio sous la forme des spectrogrammes de résolution 112×112.



(a) L'image de label 0.



(b) L'audio (spectrogrammes) de label 0.

FIGURE 1

Étant donné que l'objectif principal de ce TP est de comprendre les principes de fusion multimodale, nous avons choisi LeNet5 comme notre réseau neuronal. L'architecture principale de LeNet5 est illustrée dans la Fig.2, comprenant principalement deux couches de convolution et trois couches entièrement connectées nécessitant des mises à jour de paramètres. Après chaque couche de convolution, une opération de max-pooling est utilisée pour effectuer un sous-échantillonnage, réduisant ainsi davantage la taille des cartes de features. Cette conception vise à conserver les informations importantes des features tout en réduisant la complexité de calcul.

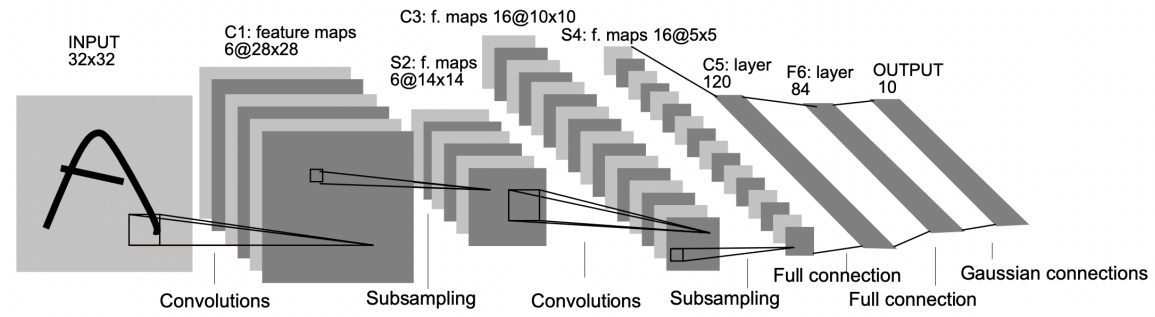


FIGURE 2 – Architecture de LeNet5

1 Performances des classifieurs unimodaux

1.1 Utilisation d'images comme entrée d'un réseau neuronal (Lenet)

Neural networks based on images as input - [parameters=61706, acc=64.10%]

```python
class LeNet5Image(nn.Module):
    def __init__(self):
        super(LeNet5Image, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5, stride=1)
        self.fc1 = nn.Linear(400, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # x = (batch_size, channels, height, width)
        x = F.relu(self.conv1(x.float()))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

1.2 Utilisation d'images comme entrée d'un réseau neuronal (Lenet)

Neural networks based on audio as input - [parameters=1311626, acc=40.98%]

```python
class LeNet5Audio(nn.Module):
    def __init__(self):
        super(LeNet5Audio, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5, stride=1)
        self.fc1 = nn.Linear(10816, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # x = (batch_size, channels, height, width)
        x = F.relu(self.conv1(x.float()))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

2 Performances des Classifieurs Multimodal

La fusion multimodale englobe une diversité de stratégies visant à intégrer des informations provenant de différentes modalités ou capteurs. Ces approches peuvent être classées en plusieurs catégories, chacune démontrant ses avantages dans des contextes spécifiques.

Fusion en amont (Early Fusion) : consiste à intégrer les informations de différentes modalités au niveau de l'entrée ou des couches basses du réseau, formant ainsi une entrée combinée. Ce type de fusion permet au modèle de considérer et de traiter simultanément des informations multimodales tout au long du processus. Il excelle généralement dans des tâches impliquant des features de bas niveau, car le modèle peut capturer les interactions entre les modalités dès les premières étapes.

Fusion en aval (Late Fusion) : intervient à des niveaux plus élevés du modèle, combinant les informations modales traitées individuellement. Cette méthode permet à chaque modalité de maintenir une certaine indépendance tout en fusionnant leurs features abstraites à des niveaux supérieurs. Elle peut être plus adaptée à des tâches impliquant des concepts sémantiques avancés.

Fusion intermédiaire (Feature Fusion) : implique la combinaison des features extraites de différentes modalités. Cela peut être réalisé par des opérations telles que la concaténation, l'addition, la moyenne, etc. Cette approche de fusion permet de préserver la spécificité des features

modales tout en permettant au modèle d'exploiter une richesse d'informations provenant de plusieurs sources.

Fusion de décisions (Decision Fusion) : consiste à combiner les prédictions ou décisions des différentes modalités à la sortie du modèle. Cette méthode est adaptée lorsque chaque modalité fournit des informations sur des aspects différents, permettant une décision globale plus complète.

De nombreuses raisons expliquent l'efficacité des fusions multimodales :

— **Complémentarité des informations :** Les différentes modalités offrent souvent des types d'informations distincts, permettant au modèle de comprendre de manière plus exhaustive les données.

— **Robustesse :** La fusion multimodale peut renforcer la robustesse du système face au bruit et aux variations, car les informations modales peuvent se corriger mutuellement.

— **Gestion de la rareté des données :** Dans certains cas, une modalité peut manquer de données, mais la fusion d'informations provenant d'autres modalités peut pallier cette rareté.

Dans les sections suivantes, nous nous concentrerons sur la mise en œuvre de fusion en amont et Fusion intermédiaire.

2.1 Fusion en Amont

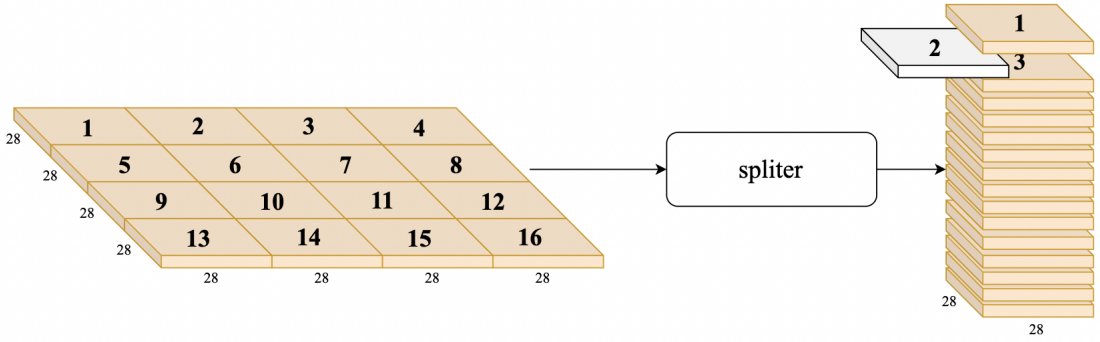


FIGURE 3 – Validation de l'algorithme de segmentation.

En tenant compte de la taille du spectre audio, qui est de 112x112, tandis que la taille de l'image est précisément de 28x28, nous pouvons découper l'audio en segments de la taille de l'image, puis les réassembler, comme illustré dans la Fig.3 L'algorithme spécifique et la mise en œuvre de la vérification sont décrits comme suit :

Tout d'abord, nous découpons le spectre audio en petites sections de 28x28, préservant ainsi les informations originales tout en adaptant la taille à celle de l'image. Ensuite, nous réassemblons ces sections dans l'ordre pour former une nouvelle représentation du spectre audio.

Pour vérifier ce processus, nous générons une matrice comme à celle illustrée dans la figure, puis procédons à la découpe et à la réassemblage. La méthode de vérification consiste à examiner si

les valeurs de chaque couche sont identiques et augmentent avec le nombre de couches. Cette vérification garantit que, lors de la découpe et de la réassemblage, aucune perte d'information ni de cohérence n'occurre, assurant ainsi une compréhension correcte des principes de fusion multimodale.

```python
matrix = torch.zeros((64, 1, 112, 112), dtype=torch.float32)

for i in range(4):
    for j in range(4):
        region_value = i * 4 + j + 1
        matrix[:, :, i*28:(i+1)*28, j*28:(j+1)*28] = region_value

print(f"Matrix Shape: {matrix.shape}")
print(matrix[0][0])

slices = torch.zeros(64, 16, 28, 28)

for j in range(64):
    for i in range(16):
        row_start = (i // 4) * 28
        col_start = (i % 4) * 28
        slices[j, i, :, :] = matrix[j, 0, row_start:row_start+28, col_start:col_start+28]

print(f"Matrix Reshaped: {slices.shape}")
print(slices[0][1])

```

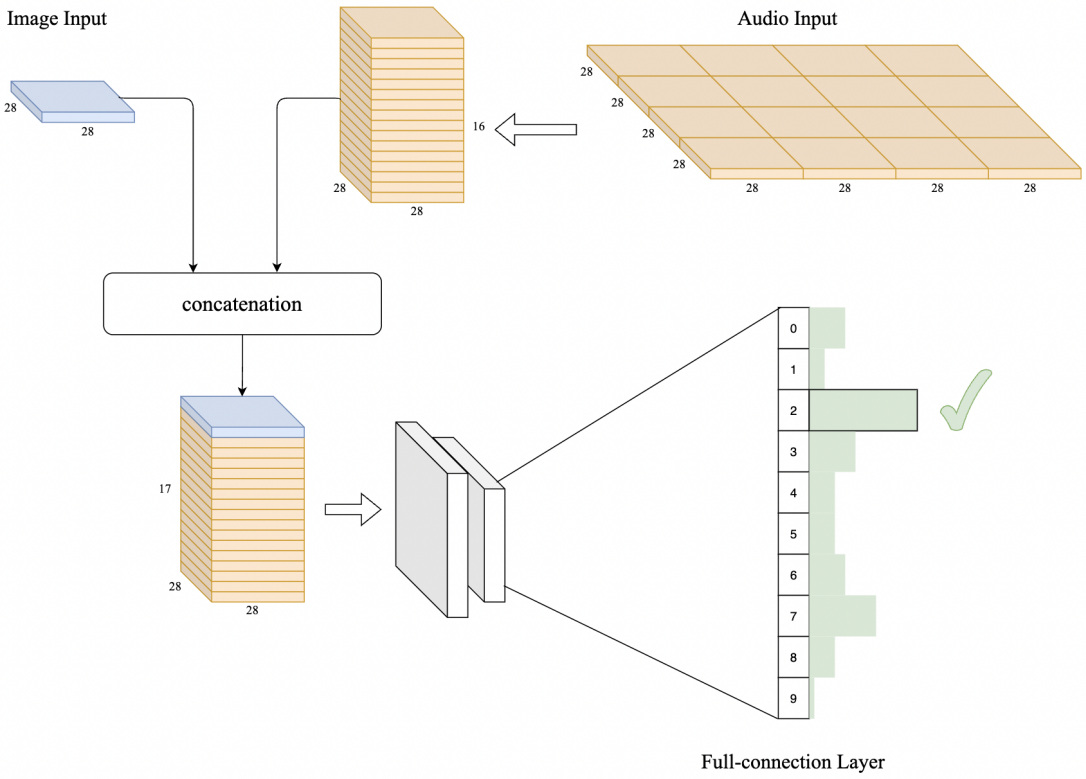


FIGURE 4 – Architecture pour la fusion du côté des données d'entrée (early fusion).

Neural networks based on early data fusion - [parameters=155374, acc=68.55%]

```python
class JointModel_early(nn.Module):
    def __init__(self):
        super(JointModel_early, self).__init__()

        # Common layers after merging
        self.conv1 = nn.Conv2d(17, 32, kernel_size=5, stride=1)
        self.conv2 = nn.Conv2d(32, 48, kernel_size=5, stride=1)

        self.fc1 = nn.Linear(768, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

```

```
    def forward(self, input_data):
        img, audio = input_data

        # Image branch
        x_img = img.float()

        # Audio branch
        audio_concatenated = torch.zeros(audio.size(0), 16, 28, 28) # Split audio into 16 28x28 chunks

        for j in range(audio.size(0)):
            for i in range(16):
                row_start = (i // 4) * 28
                col_start = (i % 4) * 28
                audio_concatenated[j, i, :, :] = audio[j, 0, row_start:row_start+28, col_start:col_start+28]

        x = torch.cat((x_img, audio_concatenated), dim=1)

        # Common layers for fusion
        x = F.relu(self.conv1(x.float()))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = F.relu(self.conv2(x.float()))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        return x
```

2.2 Fusion Intermédiaire

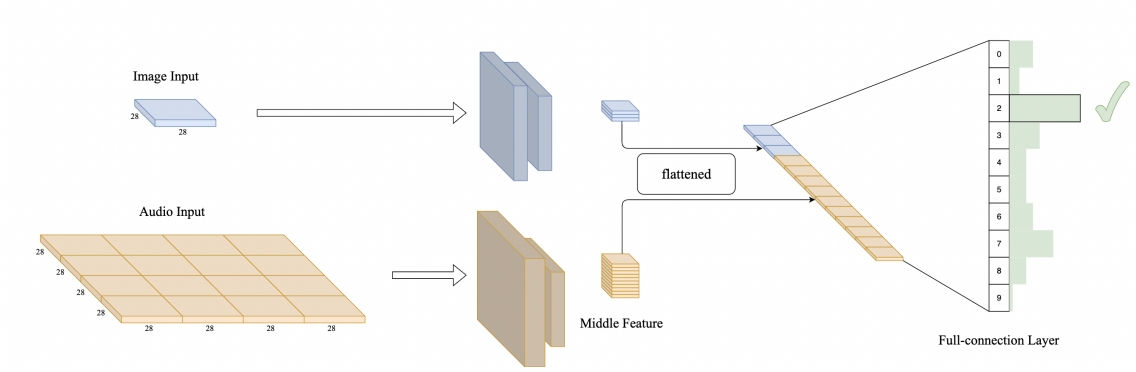


FIGURE 5 – Architecture pour la fusion du côté des features des entrées (feature fusion).

Neural networks based on feature fusion - [parameters=128328, acc=72.50%]

```
class JointModel_feature(nn.Module):
    def __init__(self):
        super(JointModel_feature, self).__init__()

        # Image branch
        self.conv1_img = nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=2)
        self.conv2_img = nn.Conv2d(6, 16, kernel_size=5, stride=1)

        # Audio branch
        self.conv1_audio = nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=2)
        self.conv2_audio = nn.Conv2d(6, 16, kernel_size=5, stride=1)

        # Common layers after merging
        self.fc1 = nn.Linear(11216, 10)


    def forward(self, input_data):
        img, audio = input_data

        # Image branch
        x_img = F.relu(self.conv1_img(img.float()))
        x_img = F.max_pool2d(x_img, kernel_size=2, stride=2)
        x_img = F.relu(self.conv2_img(x_img))
        x_img = F.max_pool2d(x_img, kernel_size=2, stride=2)
        x_img = x_img.view(x_img.size(0), -1)

        # Audio branch
        x_audio = F.relu(self.conv1_audio(audio.float()))
        x_audio = F.max_pool2d(x_audio, kernel_size=2, stride=2)
        x_audio = F.relu(self.conv2_audio(x_audio))
        x_audio = F.max_pool2d(x_audio, kernel_size=2, stride=2)
        x_audio = x_audio.view(x_audio.size(0), -1)

        # Concatenate both branches
        x = torch.cat((x_img, x_audio), dim=1)

        # Common layers
        x = self.fc1(x)

```

```
        return x
```

3 Discussion

3.1 Amélioration des Performances par la Fusion Multimodale

TABLE 1 – Résultats Expérimentaux Multimodaux

Type de Fusion Multimodale	Paramètres (k)	Précision (%)
Image	**61.706**	64.10
Spectre de Audio	1311.626	40.98
Fusion en amont	155.374	68.55
Fusion intermédiaire	128.328	**72.5**

Dans le Tab. 1, nous avons comparé les performances de différentes méthodes de fusion multimodale lors d'expériences. Ces méthodes comprennent les features d'image, les features de spectre audio, ainsi que la fusion en amont et la fusion intermédiaire.

Tout d'abord, à partir du tableau, nous pouvons observer que le modèle formé uniquement avec les features d'image affiche mieux précision que formé uniquement avec les features d'audio, atteignant 64,10%. Et, cette méthode monomodale présente un nombre de paramètres relativement faible (61,706k). Mais, le modèle pourrait atteindre un certain plafond de performance en utilisant uniquement les features d'image, rencontrant des limitations pour une amélioration ultérieure.

En revanche, le modèle utilisant uniquement les features de spectre audio affiche une précision plus faible, soit 40,98%, mais avec un nombre de paramètres relativement élevé (1311,626k). Ceci indique que l'ajout de paramètres ne conduit pas nécessairement à une amélioration des performances, en particulier dans un contexte monomodal.

Lors de la fusion multimodale, deux approches ont été considérées : la fusion en amont et la fusion intermédiaire. La fusion en amont intervient après l'extraction des features respectives de chaque modalité, tandis que la fusion intermédiaire se déroule au niveau intermédiaire du modèle. Nous observons que la méthode de fusion intermédiaire affiche de meilleurs résultats en termes de précision, atteignant 72,5%. Bien que cette méthode de fusion augmente le nombre de paramètres (128,328k), cette augmentation conduit à une amélioration significative des performances par rapport à l'approche monomodale.

En résumé, à travers la comparaison expérimentale des méthodes de fusion multimodale, nous constatons que la stratégie de fusion intermédiaire est la plus performante dans cette tâche. En introduisant des informations multimodales, la fusion intermédiaire augmente le nombre de paramètres du modèle, comblant les lacunes des features d'image et améliorant davantage les performances du modèle. Cela souligne que, lors du traitement de données multimodales, une fusion en profondeur des informations modales peut mieux exploiter les relations entre chaque modalité, résolvant les éventuels problèmes de limitation des performances des modèles monomodaux.

3.1.1 Réduction des Paramètres de Fusion intermédiaire en Réduisant le Nombre de Noyaux Convolutifs

[parameters=54152, acc=70.24%]

```
self.conv1 = nn.Conv2d(1, 3, kernel_size=5, stride=1, padding=2) # <= self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
self.conv2 = nn.Conv2d(3, 6, kernel_size=5, stride=1) # <= self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
```

En réduisant le nombre de filtres de chaque couche de convolution, nous avons réussi à diminuer efficacement le nombre de paramètres du réseau neuronal. Au cours de nos expériences, cette stratégie s'est avérée très efficace, réduisant le nombre de paramètres de **128,328** à **54,152**. Il est important de noter que ce nombre de paramètres est même inférieur à celui du modèle utilisant uniquement des images en entrée (**61,706** paramètres). Ce qui est particulièrement encourageant, c'est que malgré la diminution du nombre de paramètres, la précision du modèle a significativement augmenté, passant de **64.10%** (image unique) à **70.24%** (fusion intermédiaire) comme Tab. 2.

Cette observation souligne l'efficacité de la fusion multimodale au niveau des features. Elle indique que l'introduction de données multimodales joue un rôle crucial dans les performances globales du modèle. Ces résultats suggèrent que la combinaison de données multimodales pendant la phase d'extraction des features contribue de manière significative à l'amélioration des performances.

TABLE 2 – Résultats Expérimentaux Multimodaux

Type de Fusion Multimodale	**Paramètres (k)**	**Précision (%)**
Image	61.706	64.10
Spectre de Audio	1311.626	40.98
Fusion en amont	155.374	68.55
Fusion intermédiaire (modifié)	**54.152**	**70.24**

3.2 L'Effet de la Confusion de l'Ordre de Concaténation des Entrées Coupées sur l'Efficacité de la Fusion (la fusion en amont)

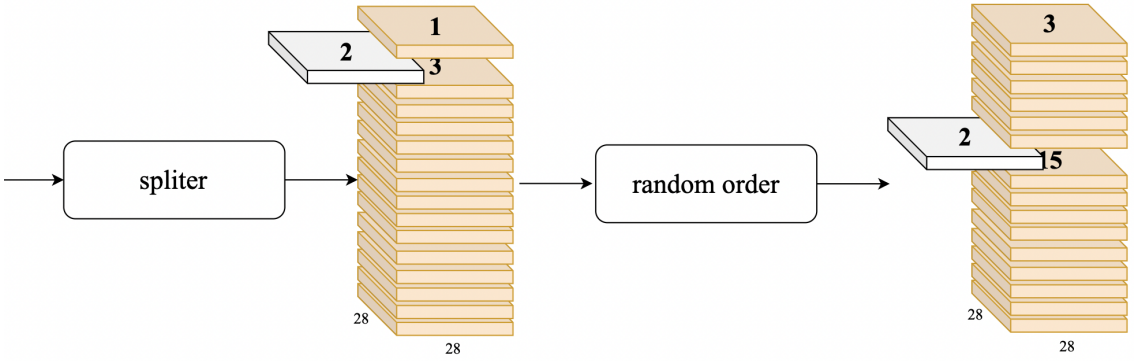


FIGURE 6 – Perturber l'ordre des features.

Dans le Sec.2.1, nous avons découpé le spectre audio en segments, divisant une image de 112×112 en 16 matrices de taille identique, puis les avons concaténées le long de la direction des canaux. Cependant, l'ordre de concaténation peut influencer les performances de fusion des

données ? Afin d'étudier la relation entre l'ordre de concaténation des entrées coupées et l'effet de fusion, nous avons conçu l'expérience illustrée dans la Fig. 6. Voici le code pertinent pour l'expérience :

```
audio_concatenated = torch.zeros(audio.size(0), 16, 28, 28) # Split audio into 16 28x28 chunks
for j in range(audio.size(0)):
    for i in range(16):
        row_start = (i // 4) * 28
        col_start = (i % 4) * 28
        audio_concatenated[j, i, :, :] = audio[j, 0, row_start:row_start+28, col_start:col_start+28]
perm_indices = torch.tensor([ 9,  0,  4,  5, 13, 11,  1,  8,  2, 15,  7, 14,  3, 10, 12,  6]) # can be alternative
shuffled_audio = audio_concatenated[:, perm_indices, :, :]
```

TABLE 3 – Effet de l'Ordre de Concaténation des Entrées Coupées sur l'Effet de Fusion

Séquence d'échange de dimensions	**Précision (%)**
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	68.55
9, 0, 4, 5, 13, 11, 1, 8, 2, 15, 7, 14, 3, 10, 12, 6	69.87
12, 7, 4, 10, 0, 15, 13, 11, 2, 14, 3, 5, 8, 9, 6, 1	67.60
7, 4, 12, 5, 11, 1, 13, 9, 15, 2, 6, 3, 0, 10, 8, 14	69.44

Suite à nos expérimentations approfondies (comme présenté dans le tab. 3), nous avons remarqué que l'ordre de concaténation des données découpées n'avait pas d'impact significatif sur les performances finales du modèle. Cette observation peut être attribuée à l'invariance par translation des réseaux neuronaux : ces derniers sont capables de maintenir leur performance même lorsque l'ordre de concaténation des données varie. Ceci s'explique par le fait que la position spatiale des features n'est pas cruciale pour le réseau en raison de son invariance vis-à-vis des translations.

Je suppose que c'est la raison pour laquelle nous devons renforcer l'effet de l'ordre d'entrée sur le modèle à l'aide d'une série de méthodes telles que l'intégration de position (**Position Embedding**).

3.2.1 Fusion de Features Basée sur la Cross Attention

Nous mettons en œuvre le mécanisme de cross attention à l'aide de nn.MultiheadAttention. Ce mécanisme vise à capturer les relations et les dépendances entre les différentes parties de la séquence d'entrée.

Dans la méthode directe, les features de l'image (x_img) et de l'audio (x_audio) sont transmises au module d'attention par l'intermédiaire de self.attention(x_img, x_audio, x_audio). Cela signifie que les poids d'attention sont calculés sur la base de l'interaction entre les features de l'image et de l'audio pour obtenir la feature (x_attended) à laquelle on prête attention. x_attended, _ = self.attention(x_img, x_audio, x_audio) Le _ dans self.attention(x_img, x_audio, x_audio) représente les poids d'attention, mais ne sera pas utilisé dans le calcul ultérieur.

Cependant, les résultats ne sont pas très satisfaisants et la précision de l'ensemble de test final est proche de celle du modèle unimodal avec l'image seule en entrée.

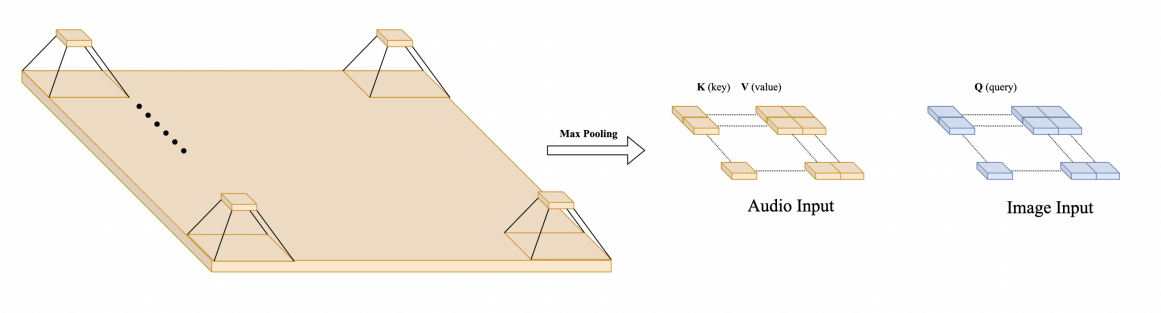


FIGURE 7 – Réduction du spectre audio à la taille d'une image par maxpooling.

4 Conclusion

Dans ce TP, nous avons approfondi l'étude des méthodes visant à optimiser les performances des modèles dans des tâches multimodales en ajustant la structure et les paramètres des réseaux neuronaux.

Comparaison des stratégies de fusion multimodale :

Nous avons d'abord comparé systématiquement différentes stratégies de fusion multimodale, notamment la fusion en amont et la fusion intermédiaire. Les résultats expérimentaux ont clairement démontré que la stratégie de fusion intermédiaire était la plus performante, atteignant une précision de 72,5%. Cela souligne l'importance de la fusion multimodale au sein de structures de modèles profonds.

Efficacité des données multimodales :

Nous avons observé que malgré la réduction du nombre de paramètres, l'introduction de données multimodales a considérablement amélioré la précision du modèle. Cette constatation souligne l'efficacité de l'apprentissage multimodal pour améliorer les performances du modèle et met en lumière l'importance d'exploiter pleinement les informations provenant de différentes modalités dans les tâches d'apprentissage profond.