



Chat ou Chien?

Ikram El Morady

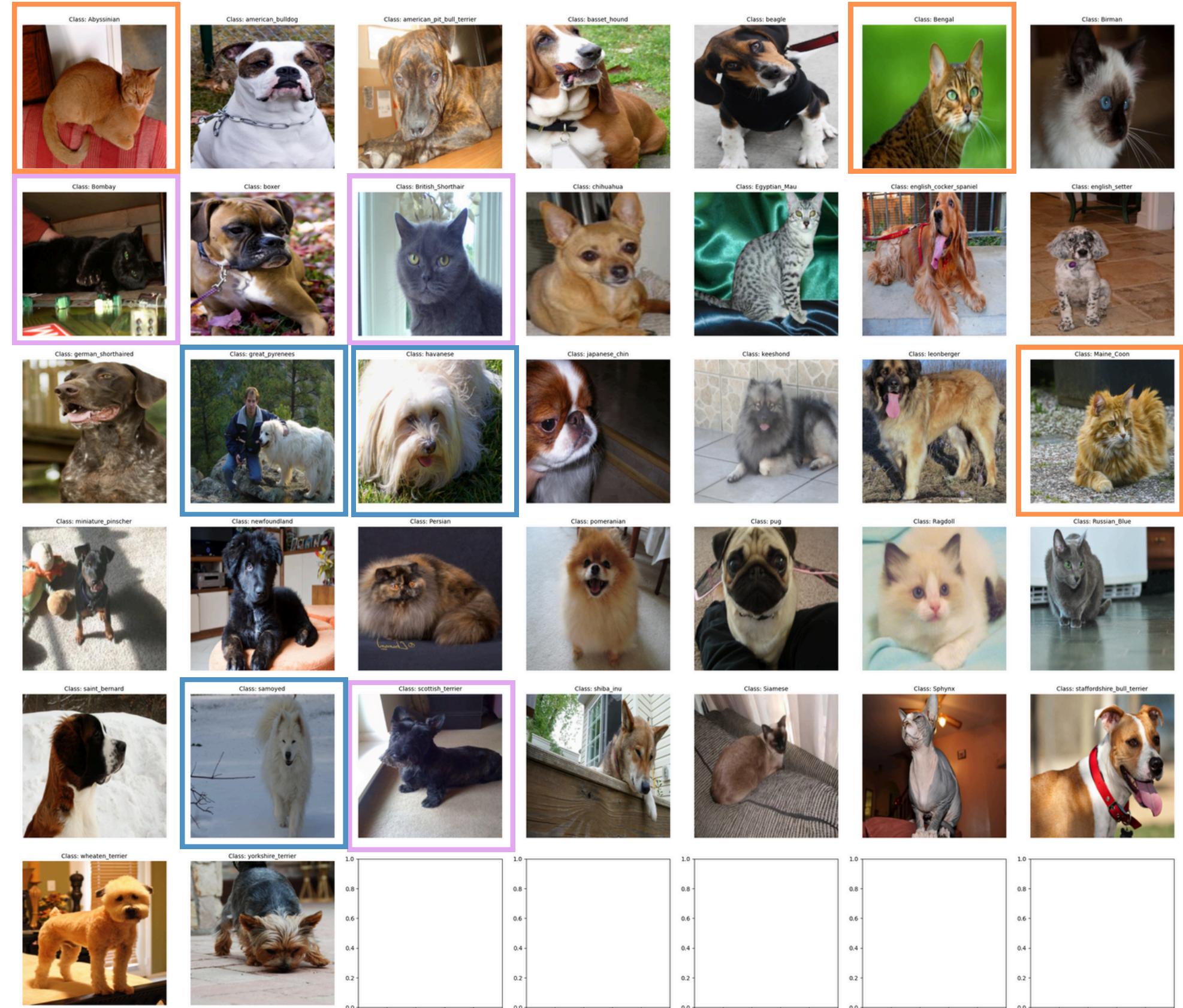
Sarah Procope

Yasmine Ennaceur

1. Exploration du dataset

Répartition par race

- 200 échantillons par race
- 37 classes



1. Exploration du dataset

Répartition par type

Classe 1 (Chat): 2371 échantillons

→ Déséquilibre des classes

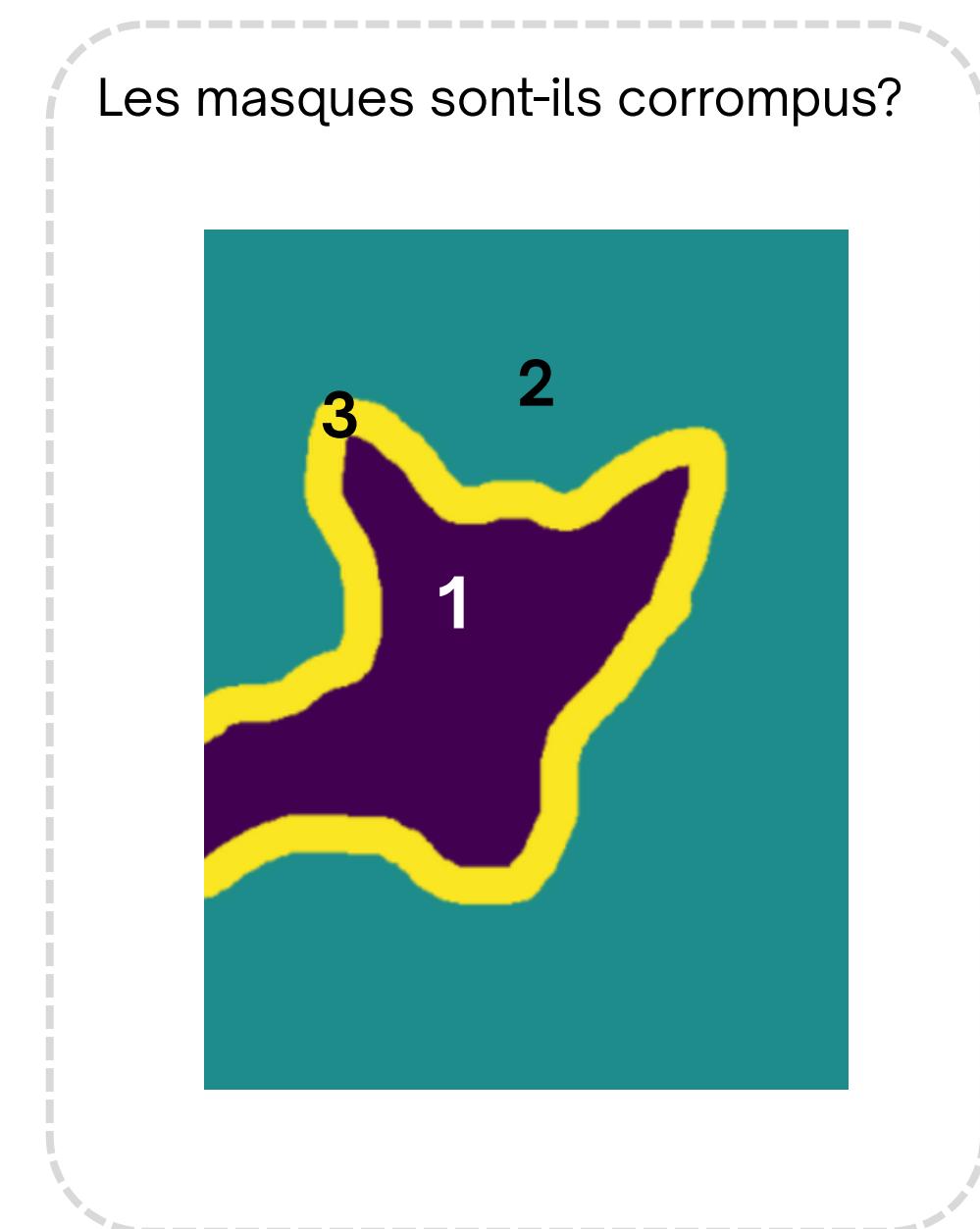
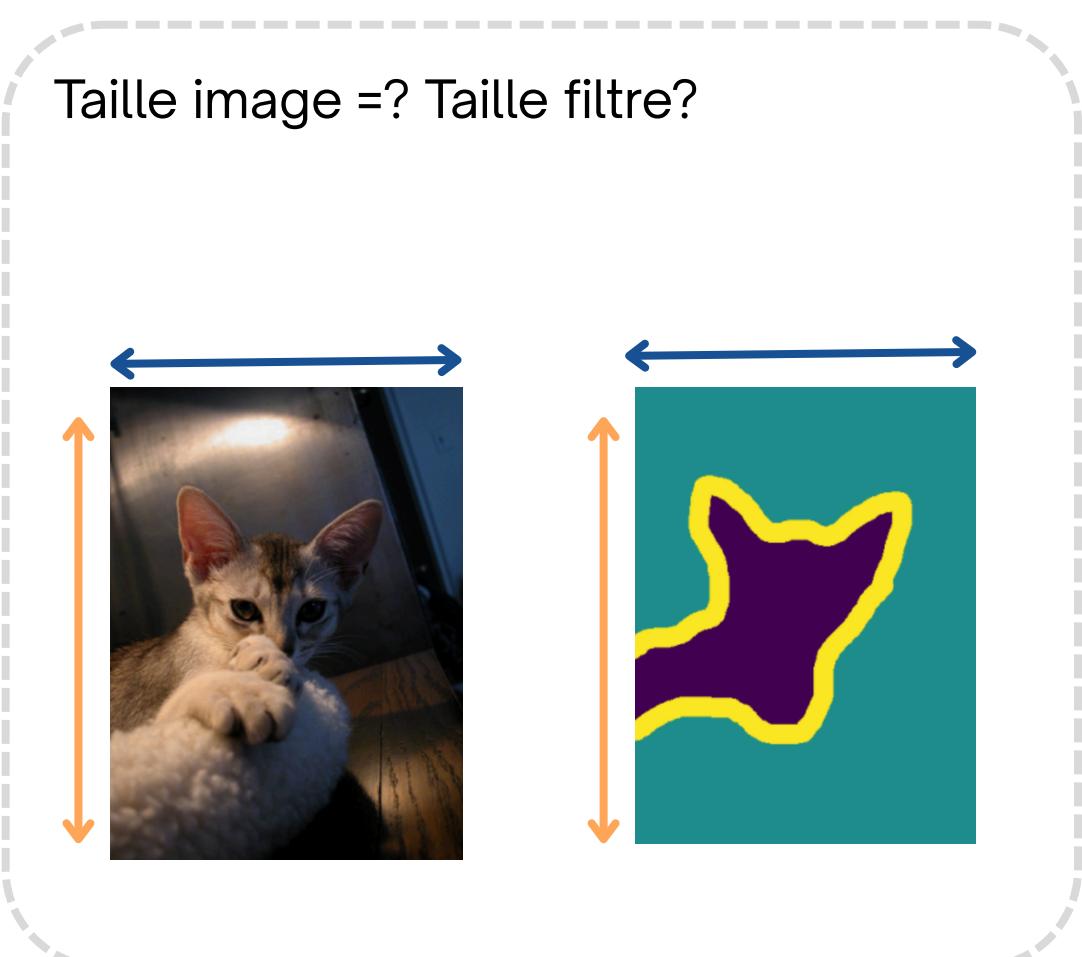
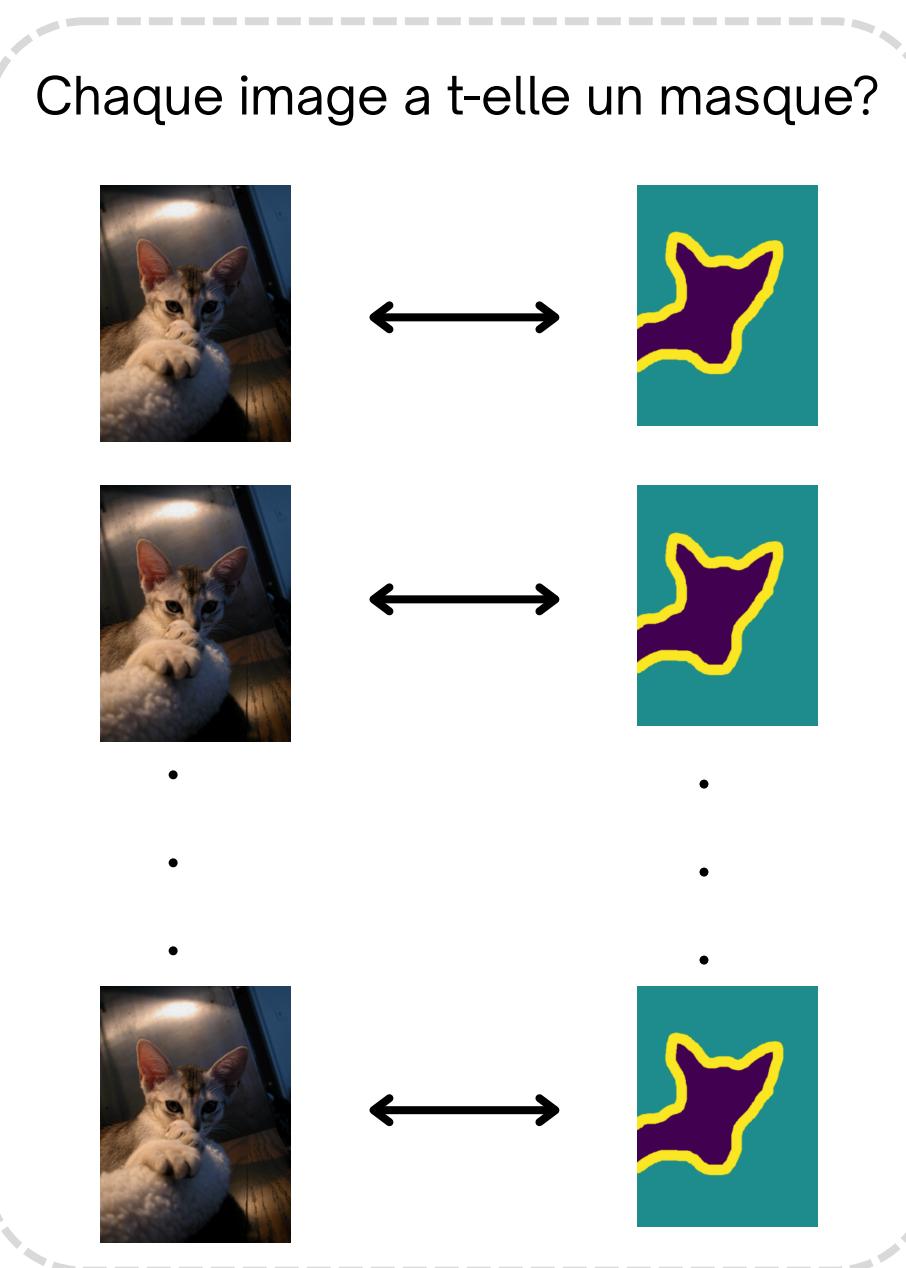
Classe 2 (Chien): 4978 échantillons



1. Exploration du dataset

Masques de segmentation

Les images sont-elles bien segmentées ??



1. Exploration du dataset

Masques de segmentation

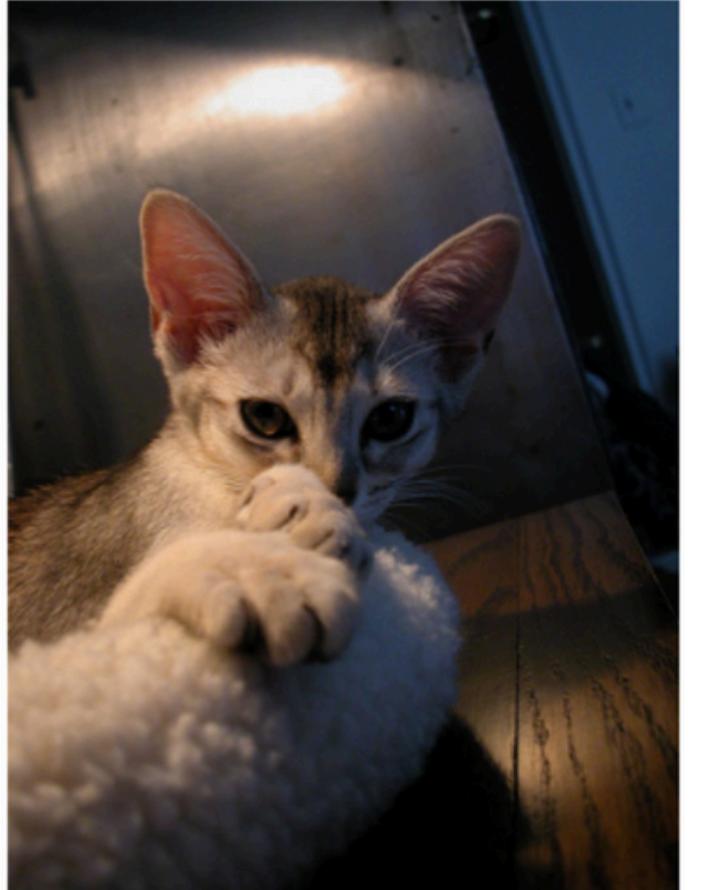
Les images sont-elles bien segmentées??

```
==== Début de la vérification des masques ===
```

```
Masques manquants : 0  
Bad sizes : 0  
Bad values : 0
```

```
==== Vérification terminée ===
```

Image : Abyssinian_10.jpg

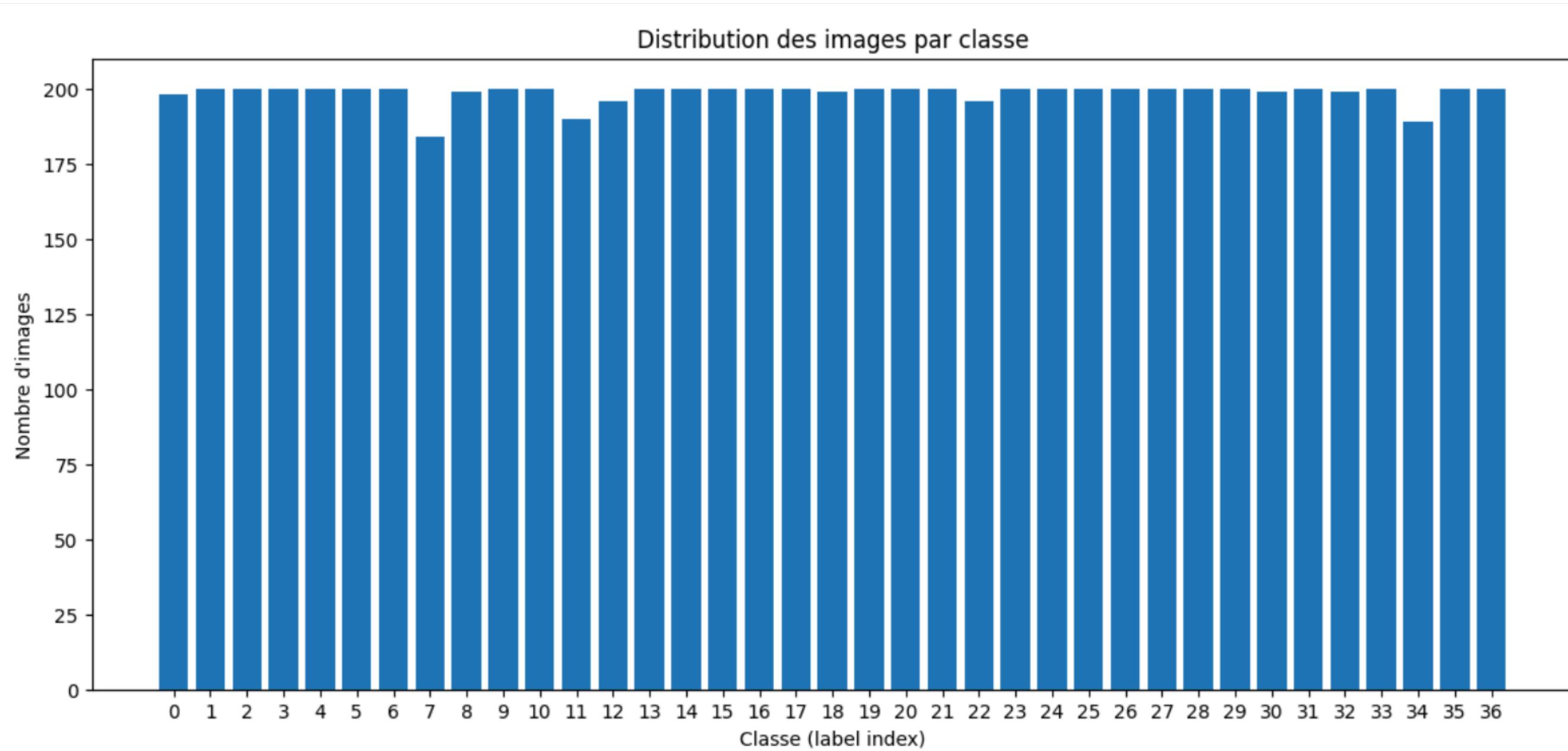


Mask : Abyssinian_10.png



1. Exploration du dataset

Déséquilibre de nombre de classe



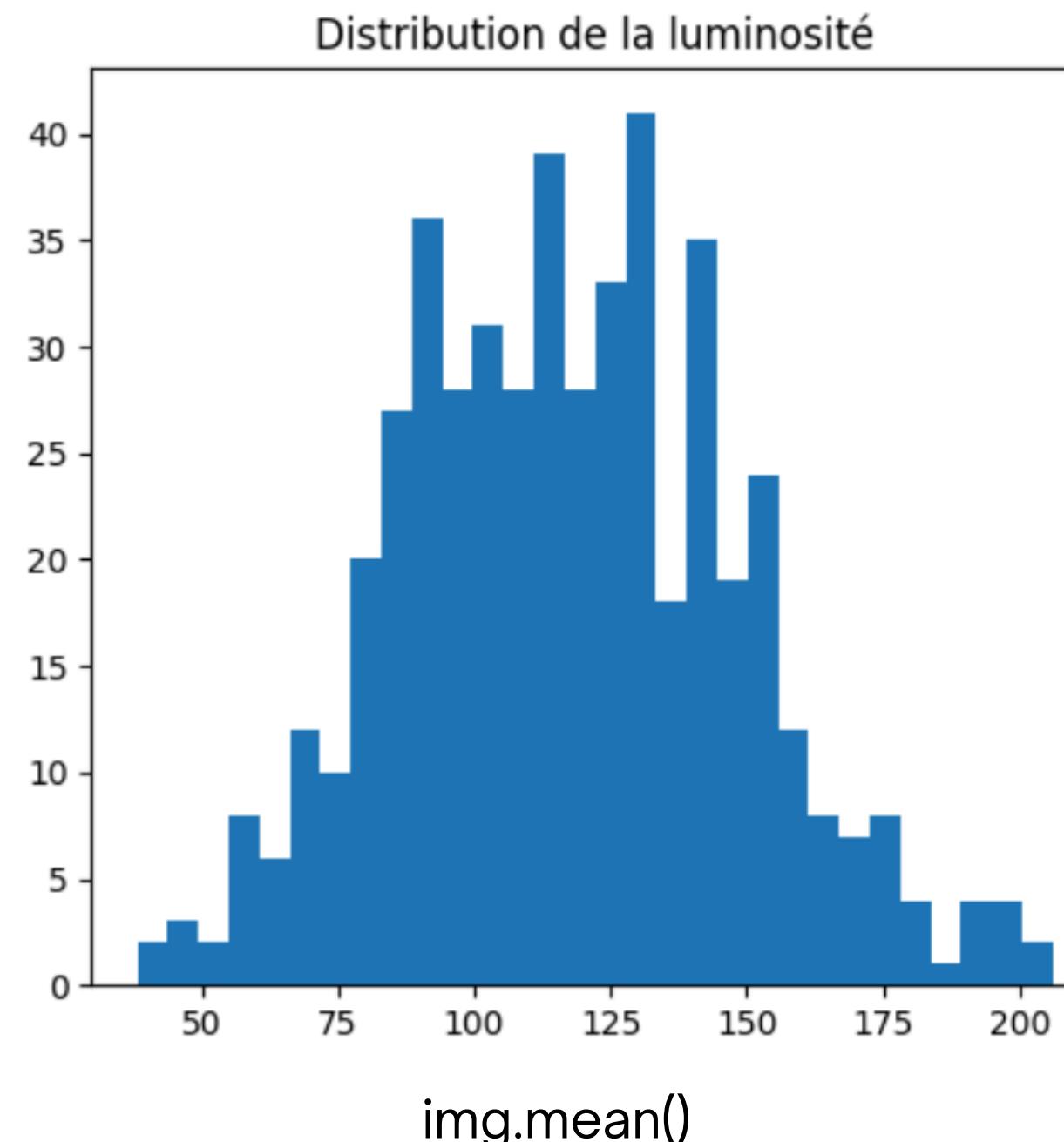
Un déséquilibre léger concerne:

- Chats : 7-Bombay, 11-Egyptian Mau, 34-Thai.
- Chiens : 12-English Cocker Spaniel, 22-Newfoundland.

1. Exploration du dataset

Étudier le biais visuel

- Luminosité



- Distribution (quasi) normale.
→ Dataset globalement cohérentes en termes d'exposition.
- Peu sont les images qui ont une luminosité extrême.
→ Des cas aberrant qui doivent exister pour tester la performance des modèles
- Un pic du nombre d'images ayant une moyenne de luminosité de 130.

2. Classification binaire

Objectif:

- Classification binaire avec labels

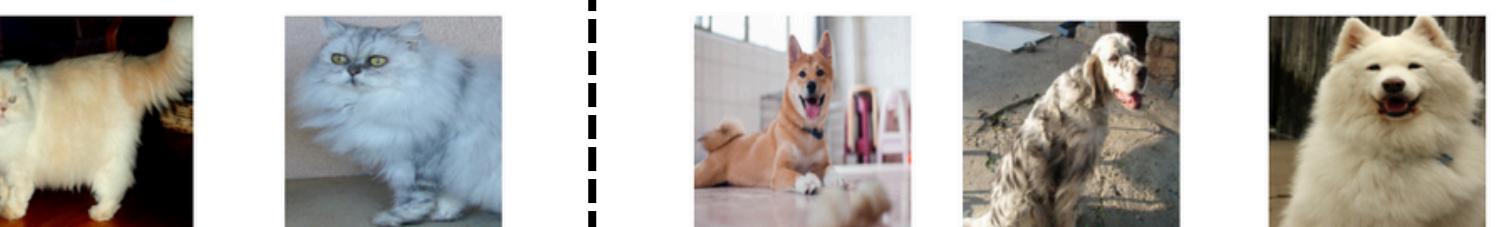
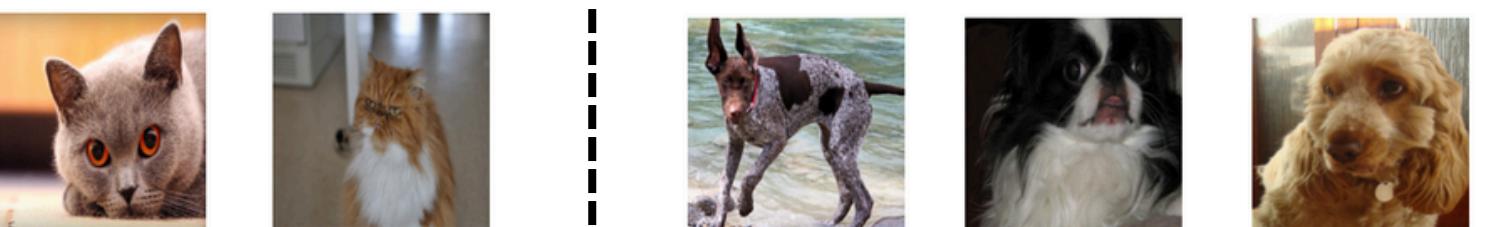


Challenge:

- Répartition du dataset : 2x plus de chiens
- Taille du dataset
- Qualité des images(obstruction, luminosité, position)

Motivation et plan:

- Commencer par une baseline codée from scratch.
- Tester la classification avec un modèle pré-entraîné.



Chat

Chien

2. Classification binaire

CNN simple

Image $(224^2 \times 3) \xrightarrow{\text{Conv+Pool}} (112^2 \times 32) \xrightarrow{\text{Conv+Pool}} (56^2 \times 64) \xrightarrow{\text{Conv+Pool}} (28^2 \times 128)$

$(28^2 \times 128) \xrightarrow{\text{Flatten}} (100352) \xrightarrow{\text{FC1 + Dropout}} (256) \xrightarrow{\text{FC2}} (1)$

Architecture du CNN

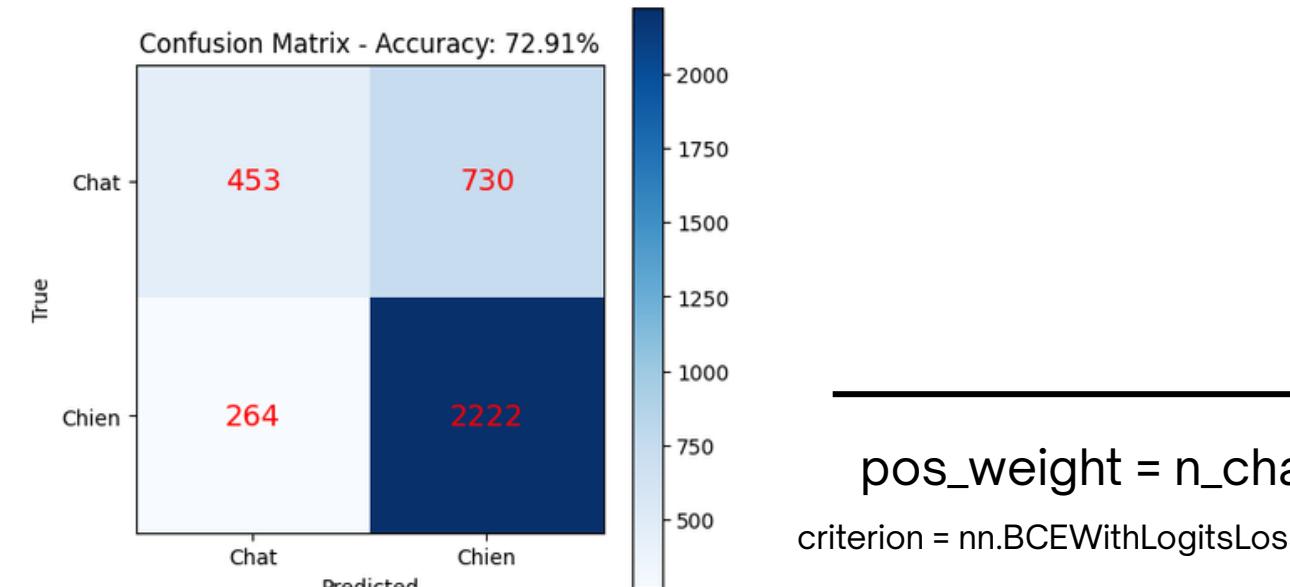
Fonction de perte - BCEWithLogitsLoss :

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top$$
$$l_n = -w_n [y_n \cdot \log(\sigma(x_n)) + (1 - y_n) \cdot \log(1 - \sigma(x_n))]$$

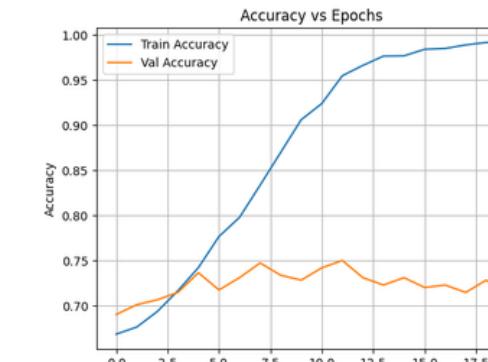
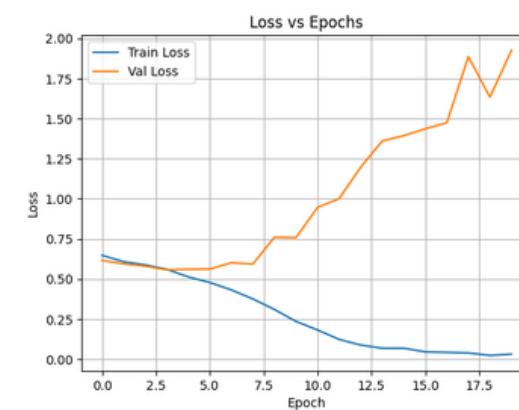
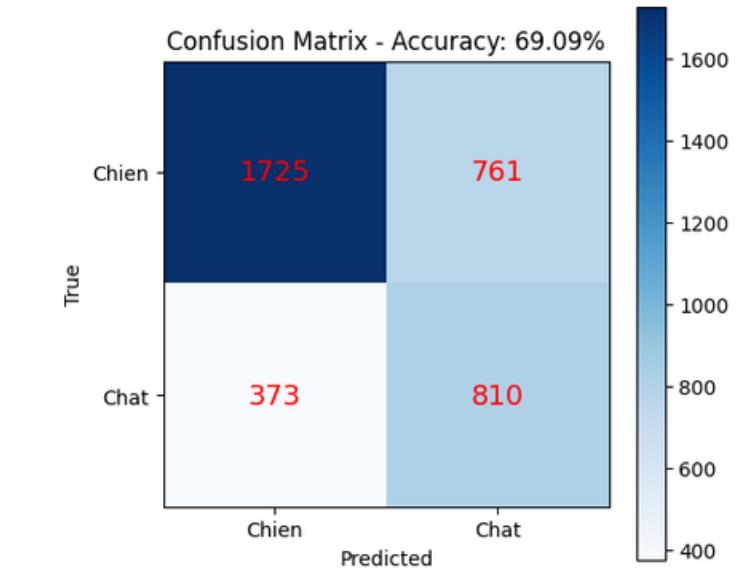

↑ ↑ ↑
poids labels réels fonction sigmoïde

2. Classification binaire

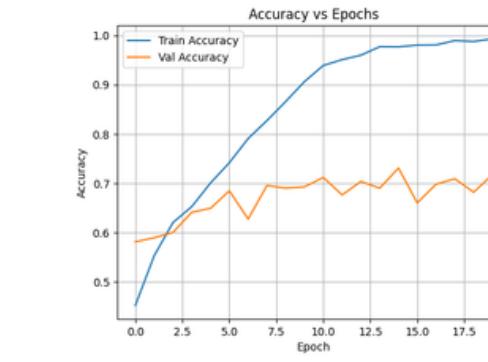
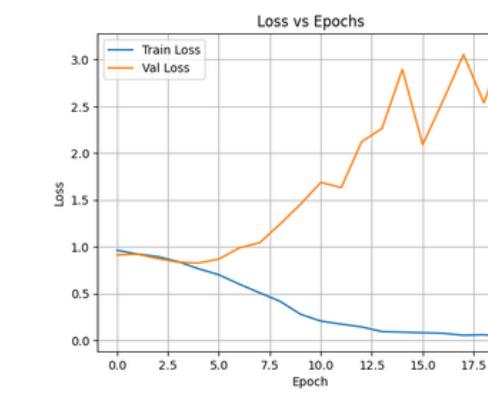
CNN simple



$\text{pos_weight} = \text{n_chats} / \text{n_chiens}$
criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)



Le modèle classe tous les individus comme des chiens



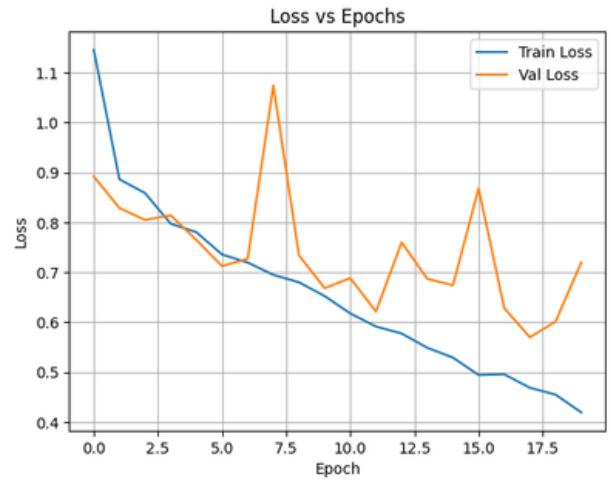
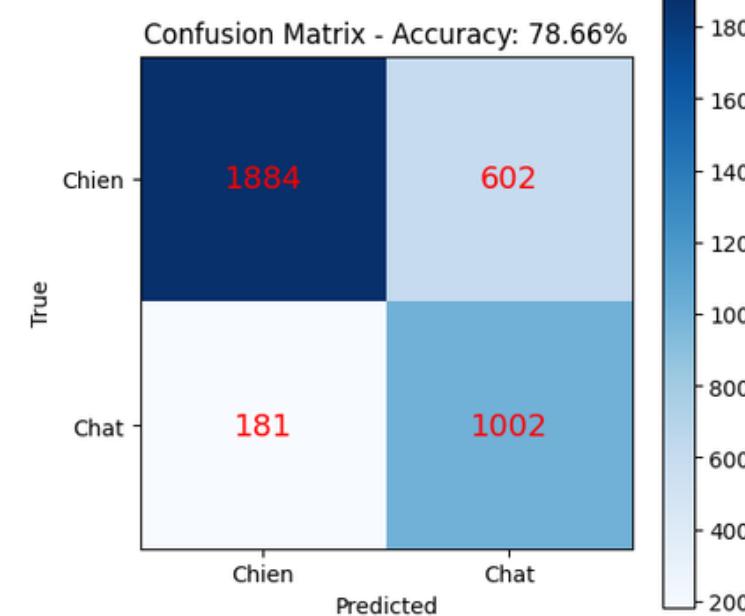
Le modèle classe un peu mieux les chats
mais l'accuracy n'est pas optimale

Conclusion : Modèle trop simple → ajout de couches de convolution supplémentaires

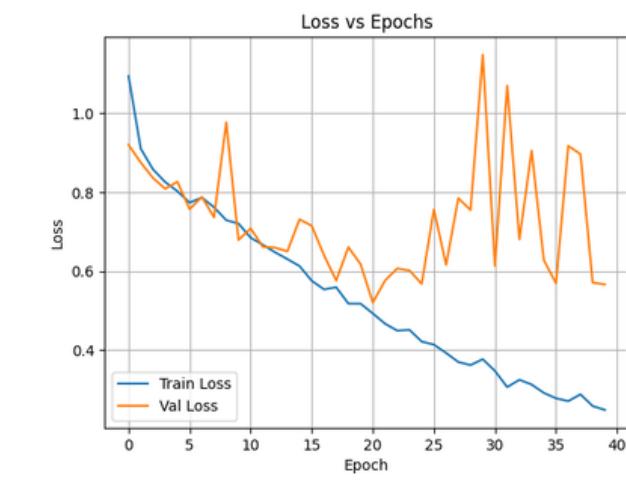
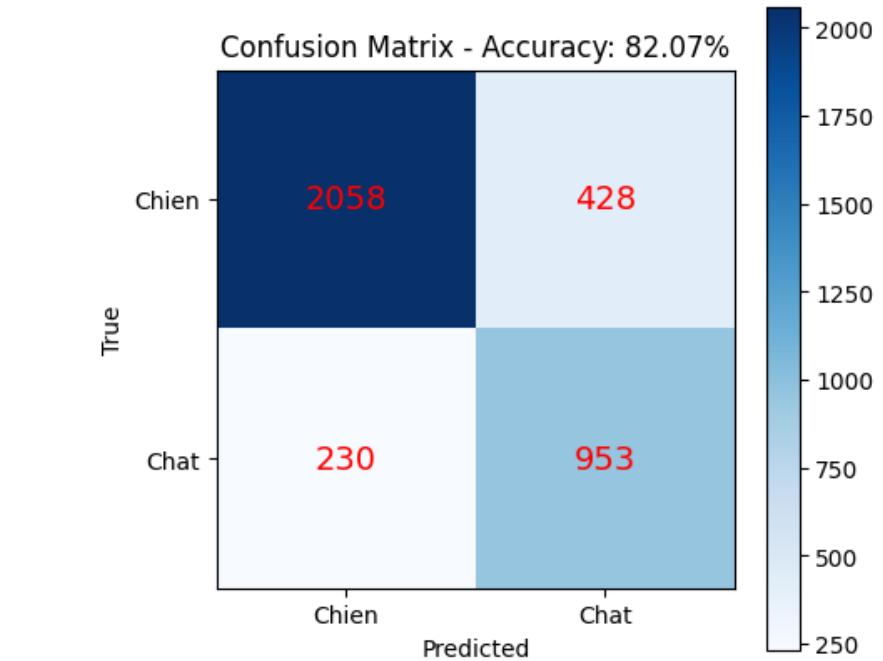
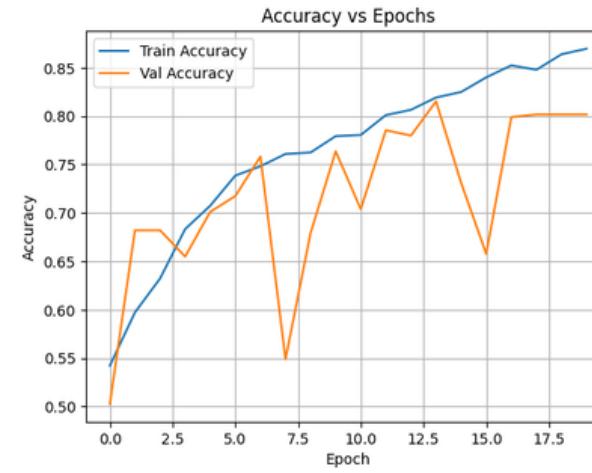
2. Classification binaire

CNN improved

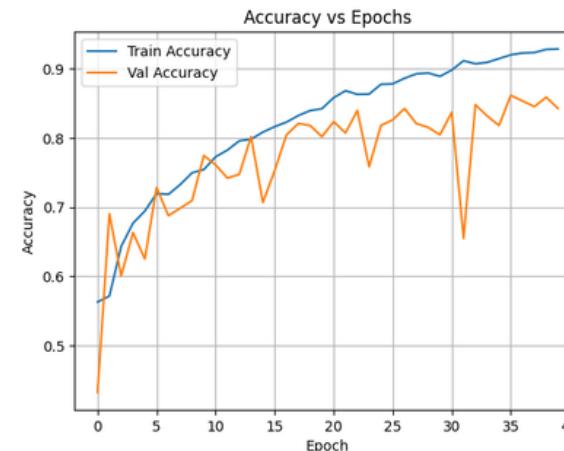
→ Ajout de Batch Normalization avant ReLu et plus de couches dans le modèle



20 epochs

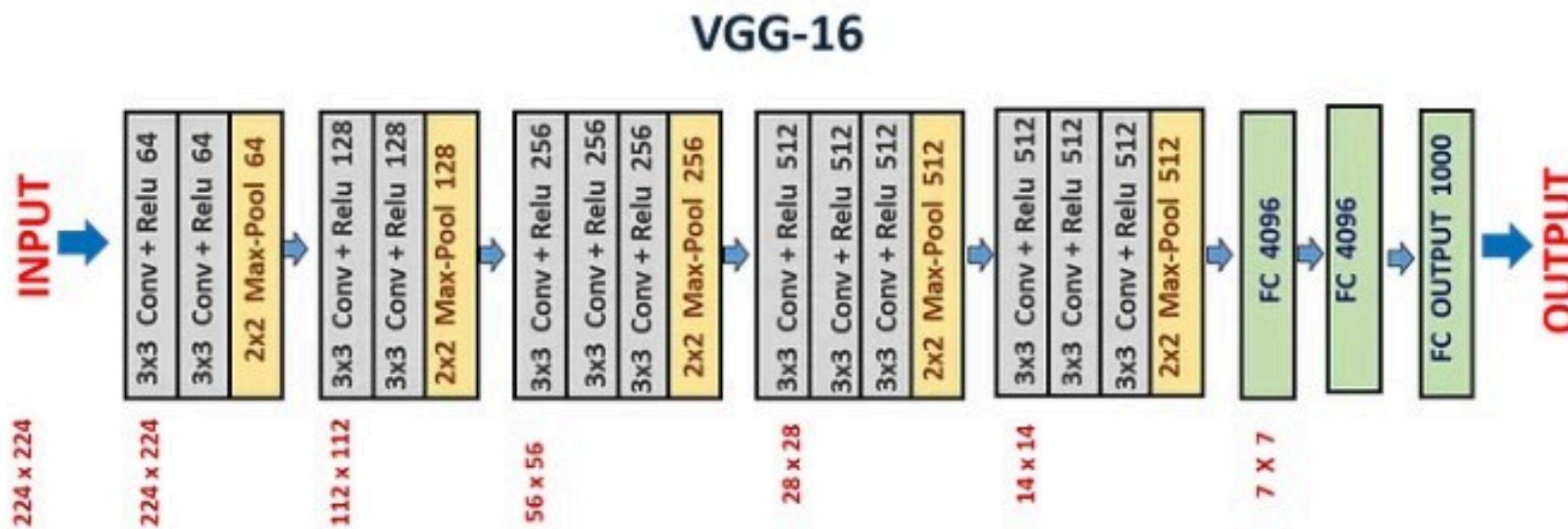


40 epochs

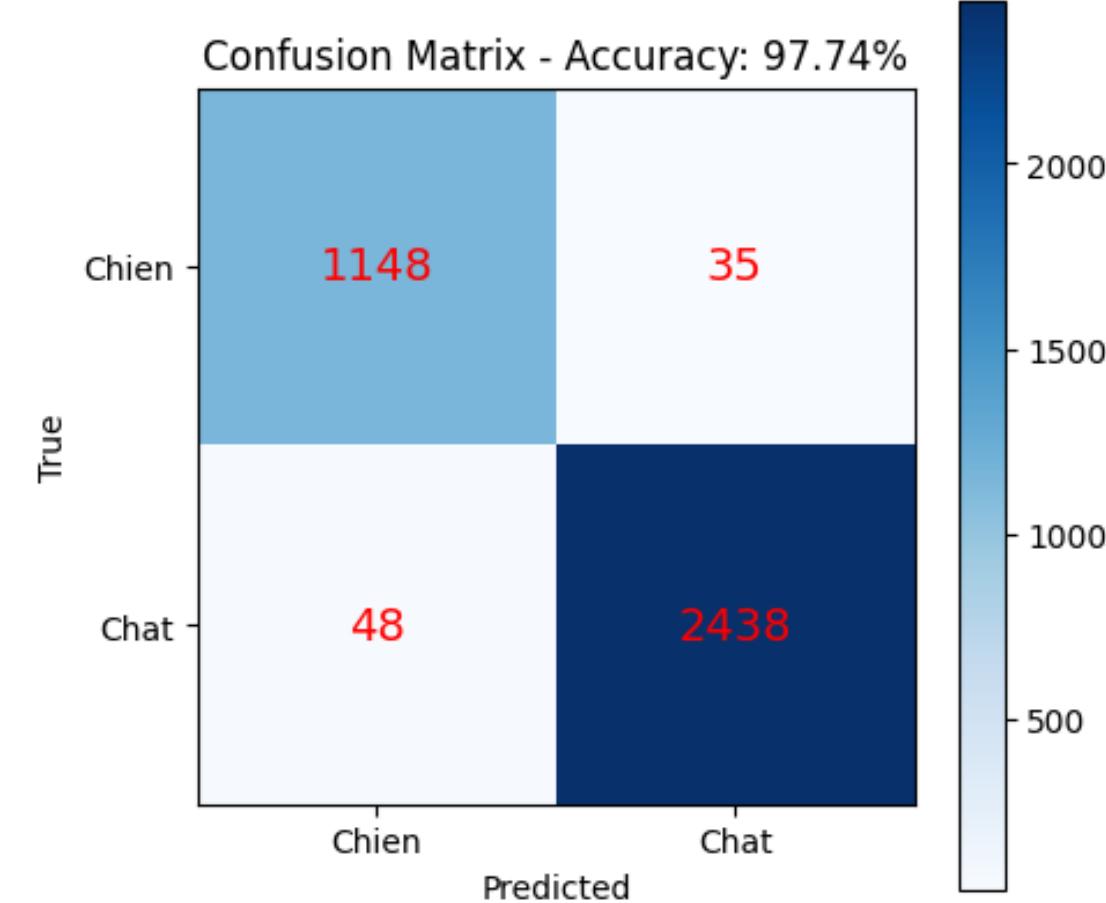


2. Classification binaire

Pretrained VGG-16 + fine-tuning 5 epochs



Dernière couche du classifier remplacée par couche linéaire de taille 1



2. Classification binaire pour aller plus loin

Objectif:

- Classification non supervisée

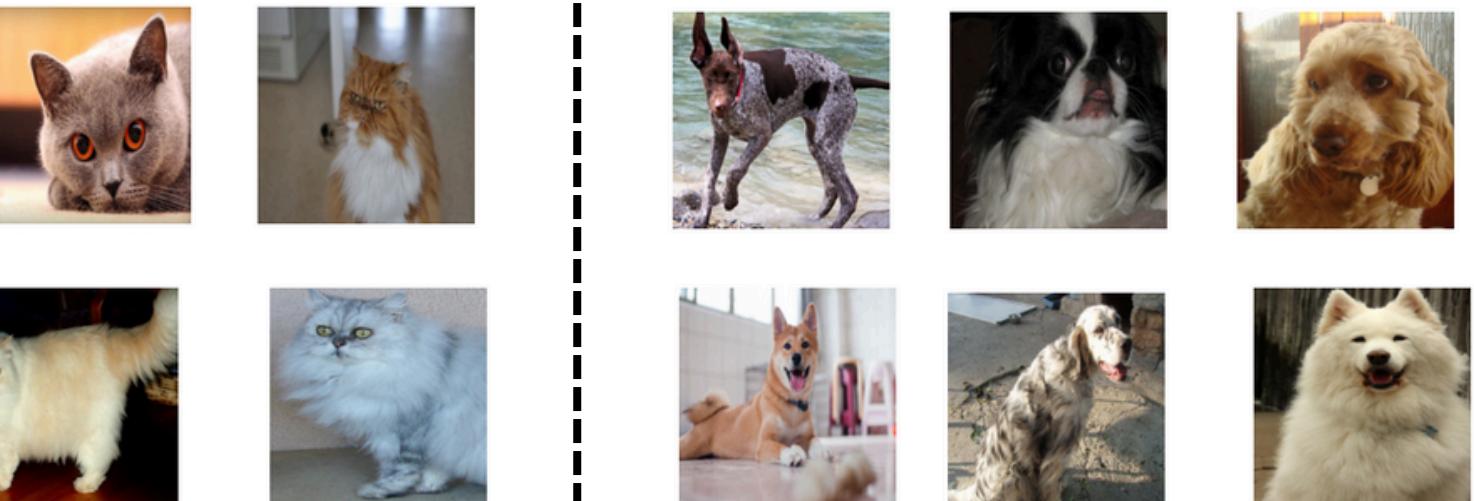


Challenge:

- Taille du dataset
- Taille de l'espace latent
- Clustering (K-Means, Agglomerative Clustering)
- Visualisation (T-SNE)

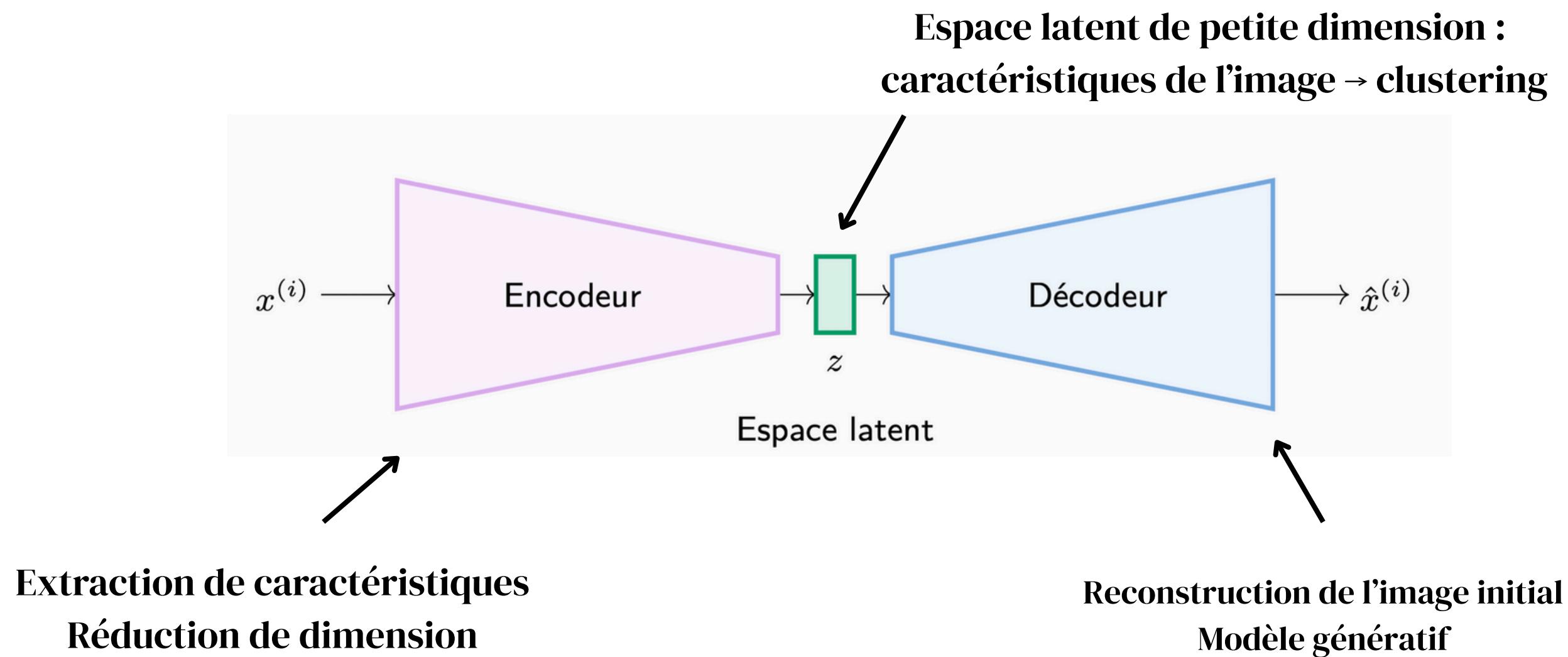
Motivation et plan

- Commencer par une baseline codée from scratch.
- Tester la détection avec un modèle pré-entraîné.



2. Classification binaire non supervisée

2.1 Auto Encoder



2. Classification binaire non supervisée

2.1 Auto Encoder Convolutionnel

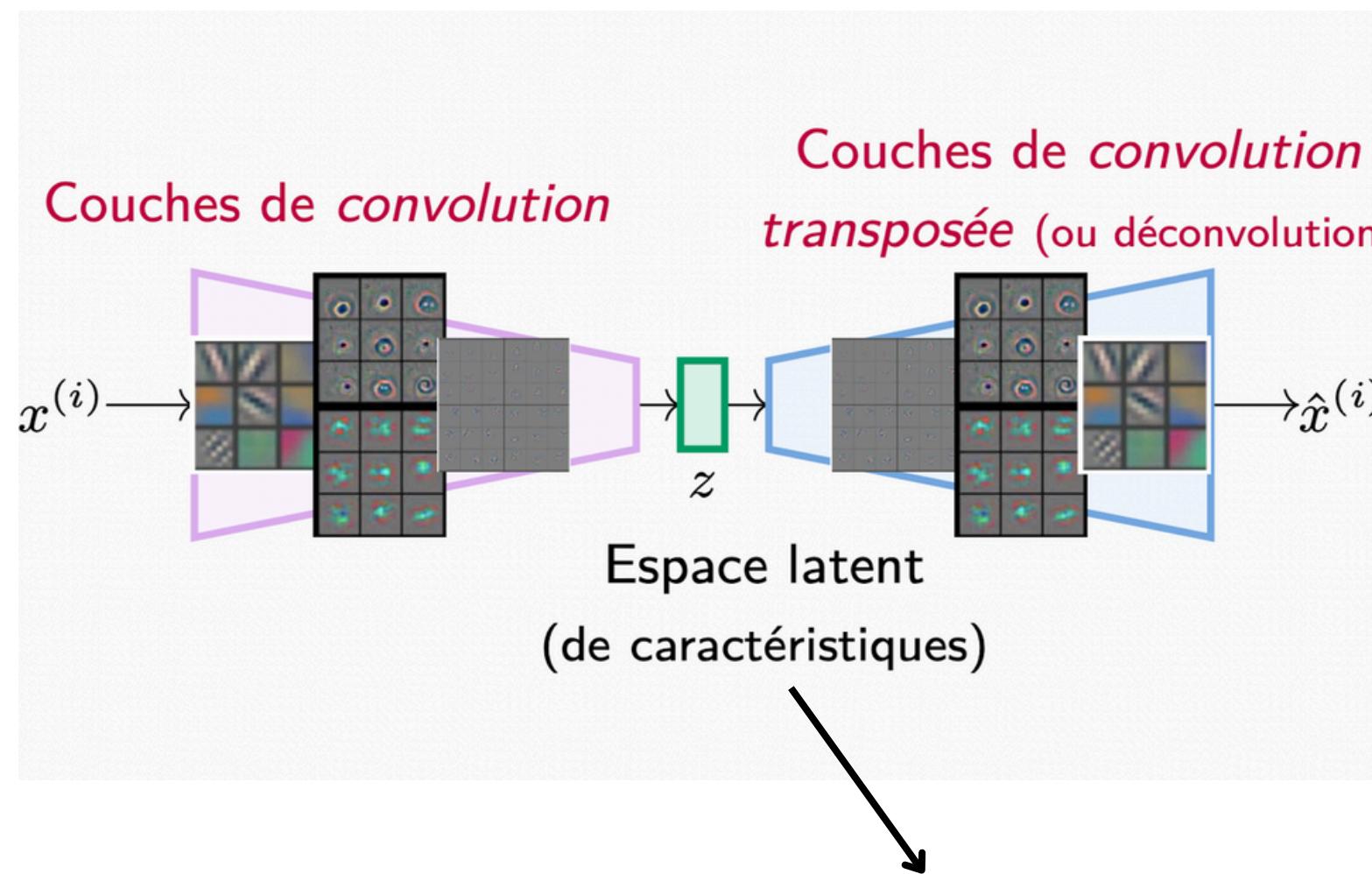
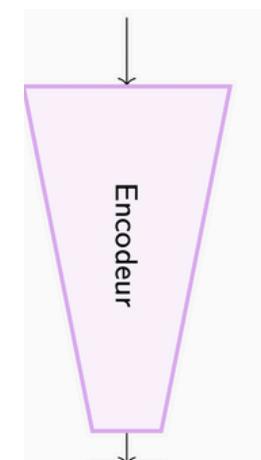
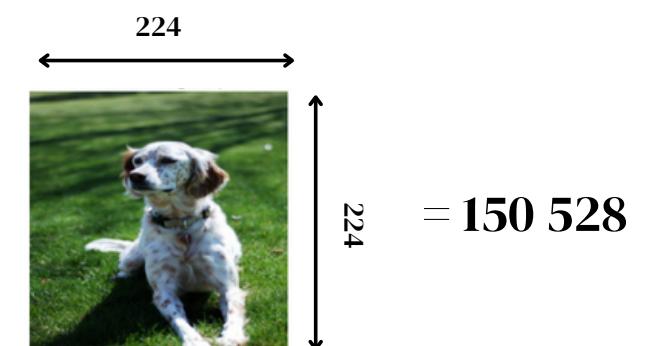


Image de taille : 3 canaux X

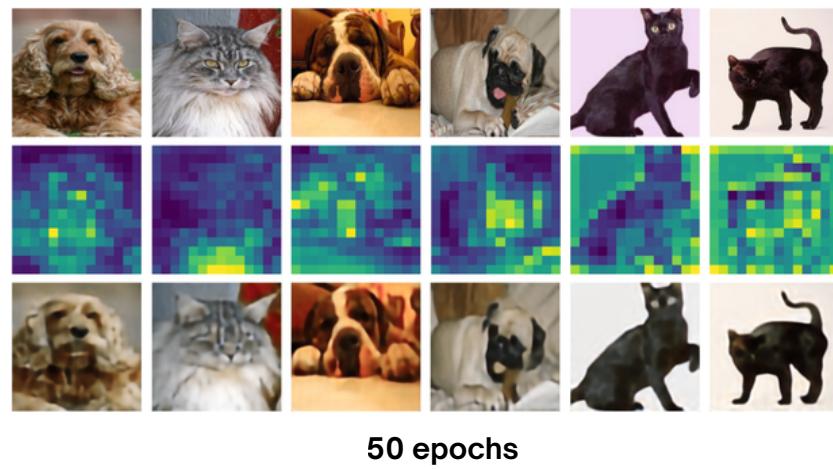


128

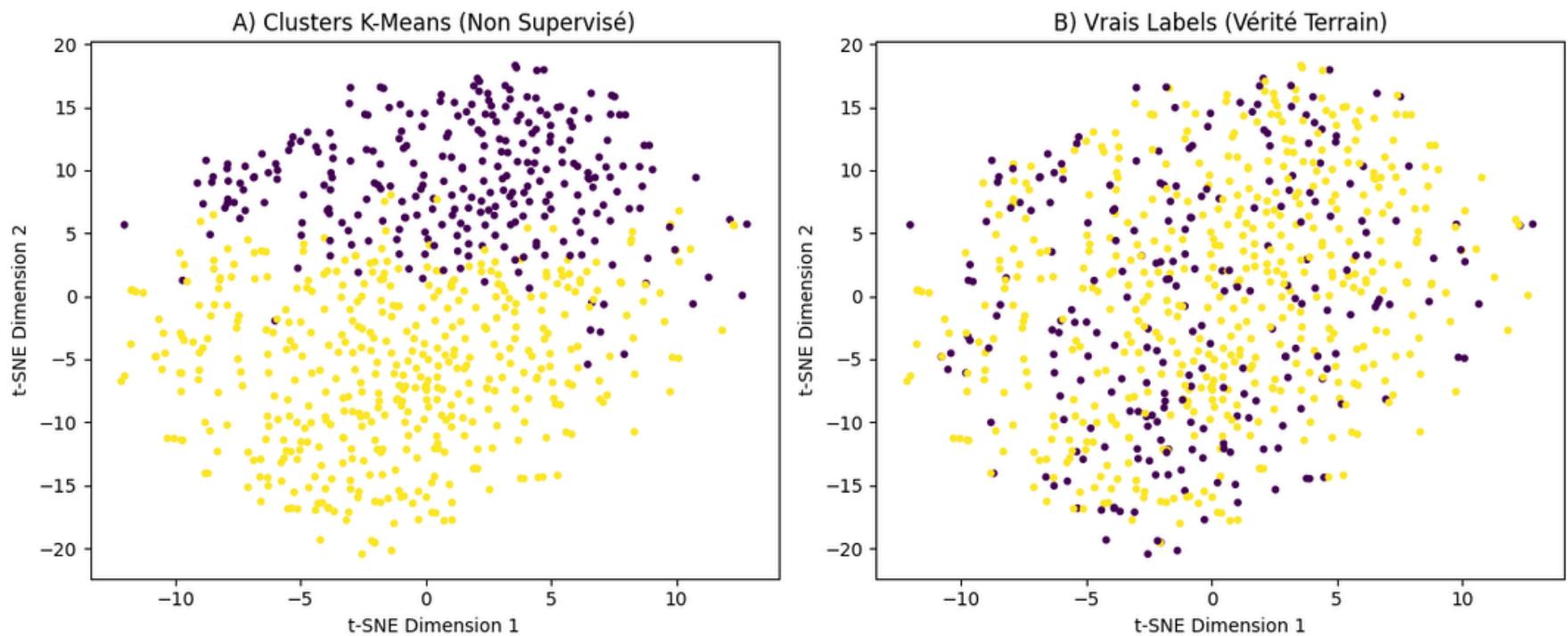
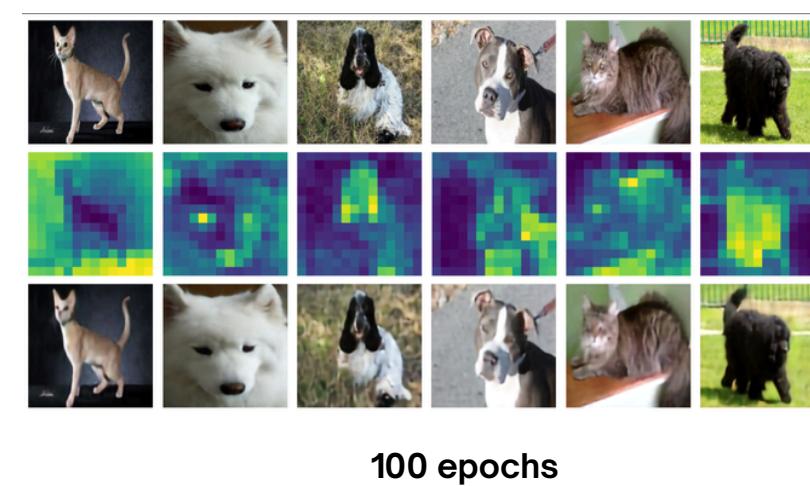
Trouver la valeur qui permet de bien extraire les caractéristiques de l image pour la classification

2. Classification binaire

2.1 Auto Encoder Convolutionnel



→ Clustering :



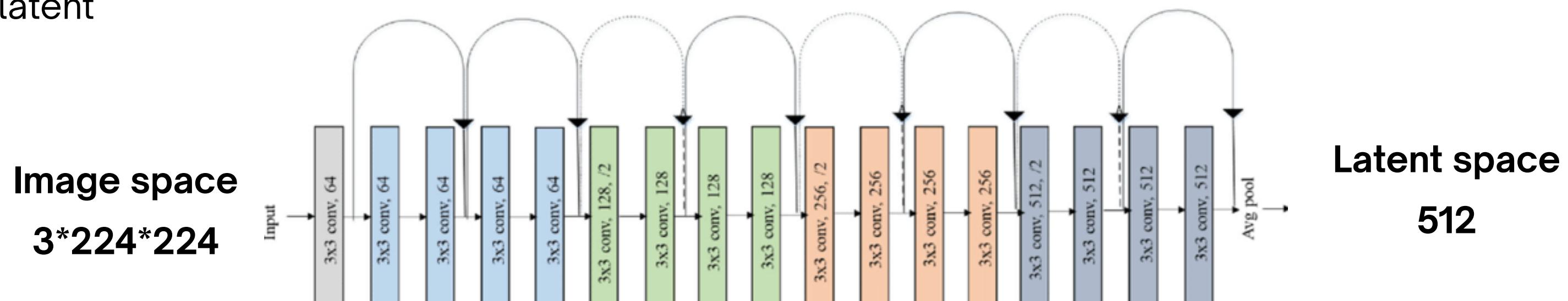
→ L'espace latent ne représente pas bien les caractéristiques chien/chat : le modèle n'arrive pas à apprendre les caractéristiques des deux types d'animaux

2. Classification binaire

2.2 ResNet pré-entraîné comme Auto-Encodeur

Généralisation : Le modèle a déjà appris des caractéristiques très générales valables pour presque toutes les images.

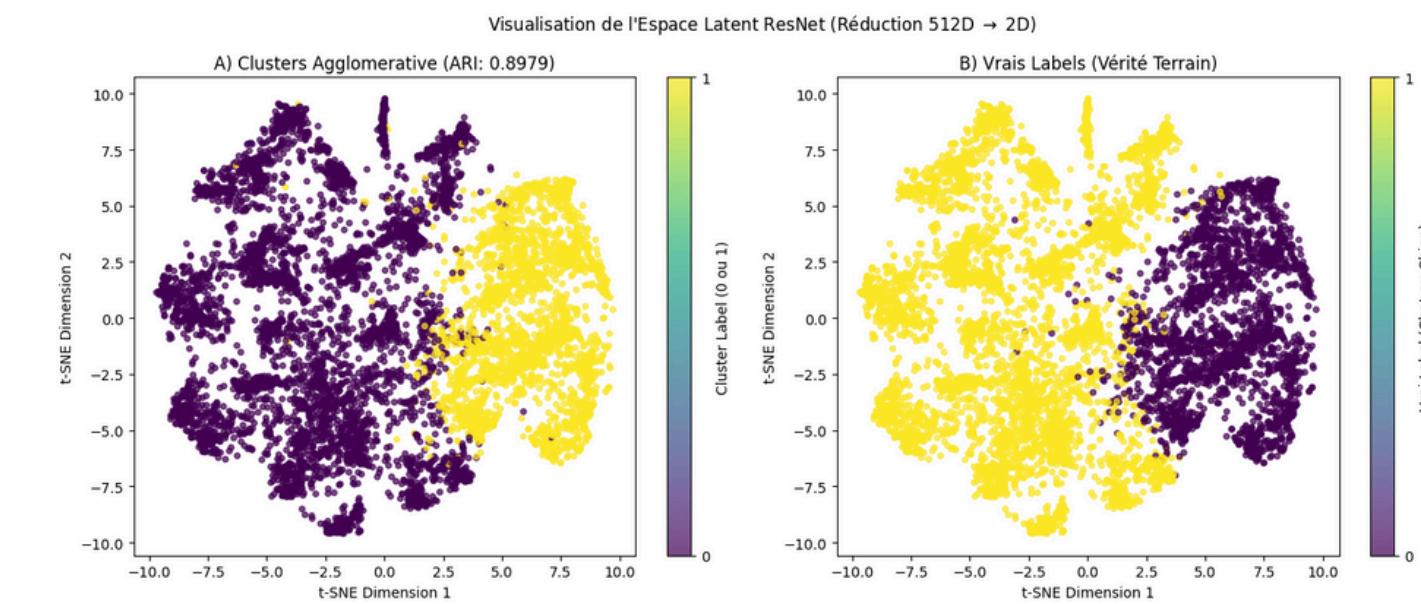
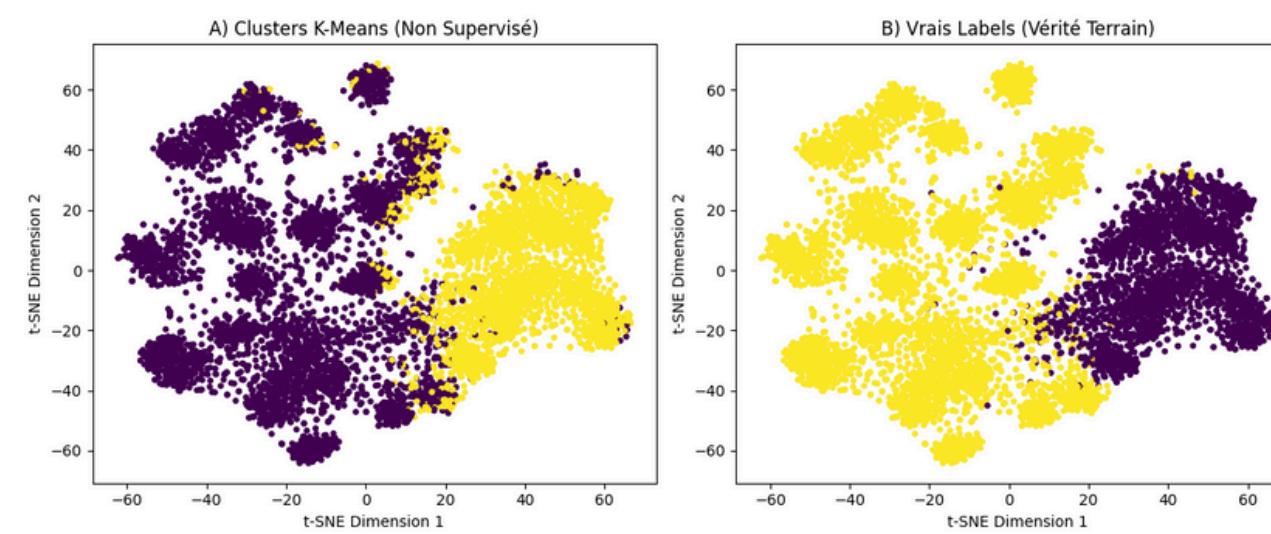
On enlève la dernière couche de classification : la sortie de la couche pooling est notre espace latent



2. Classification binaire

2.2 Modèles préentraînés

- Clustering avec K= 2 classes sur les images encodées avec ResNet
- Visualisation de l'espace latent avec T-SNE



KMeans
ARI 0.75

Agglomerative Clustering
ARI 0.89

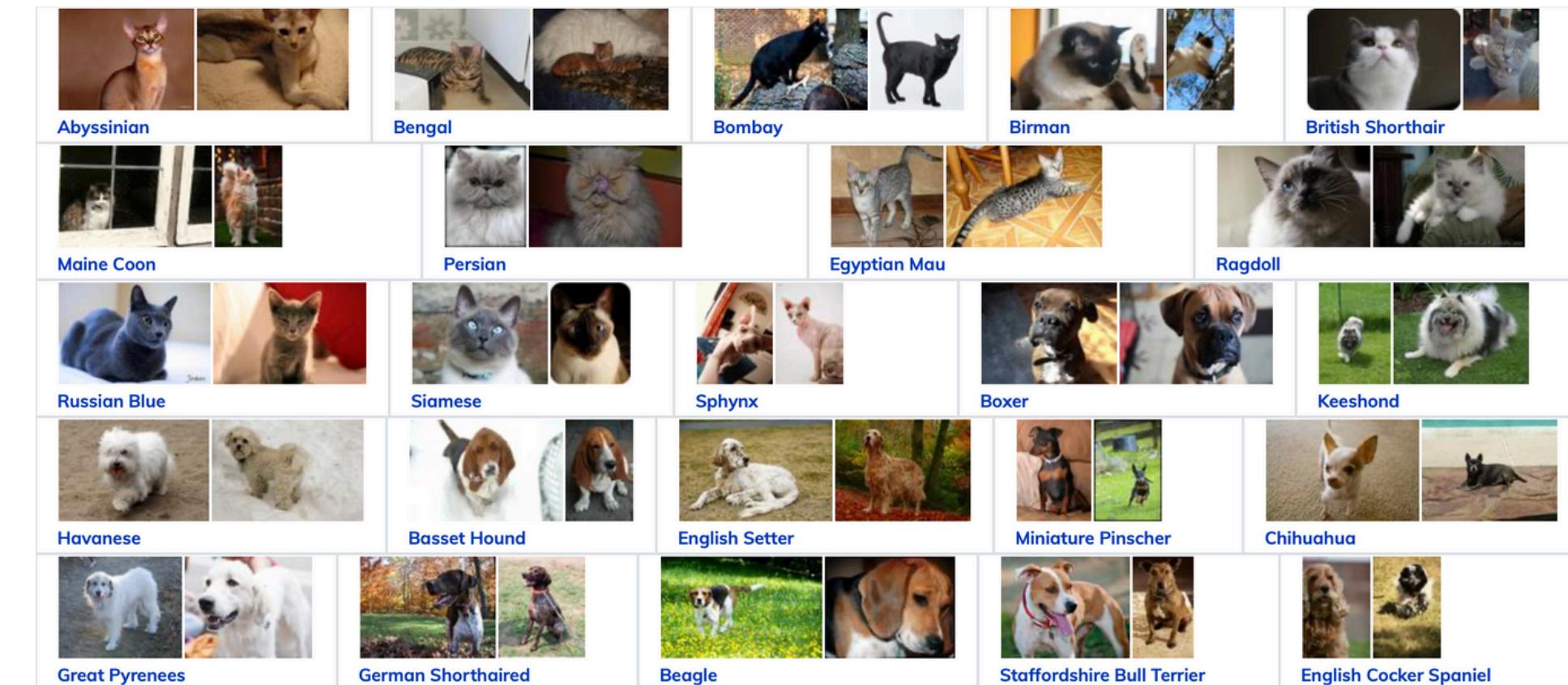
3. Classification fine

Objectif:

- Distinguer des sous-catégories visuellement très proches vs. la classification simple (Chat vs. Chien).

Le Défi de la classification :

- Taille des données, variations d'angle, d'éclairage, nécessité d'un modèle capable d'extraire des détails subtils.



Motivation et plan

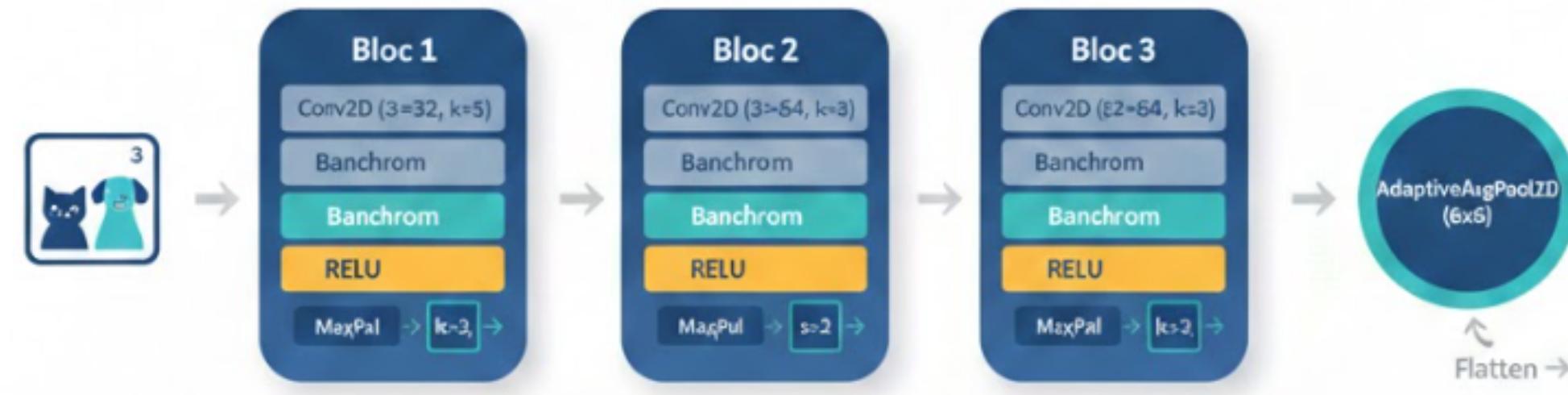
- Commencer par une baseline codée from scratch.
- Tester la détection avec un modèle pré-entraîné.

3. Classification fine

3.1 Baseline

A. SimpleCNN baseline

Architecture de type LeNet/AlexNet minimaliste afin d'établir une référence initiale pour quantifier la difficulté de la tâche.



Choix des hyperparamètres

```
learning_rate = 0.0001
num_epochs = 15
num_classes = 37

criterion = nn.CrossEntropyLoss()

model_baseline = SimpleCNN(num_classes=num_classes)

optimizer = optim.Adam(model_baseline.parameters(), lr=learning_rate)
scheduler = ReduceLROnPlateau(optimizer, mode='max', factor=0.1, patience=5)
```

3. Classification fine

3.1 Baseline

A. SimpleCNN baseline

Résultats

Epoch 9/15 | Duration: 40.63s
Train Loss: 2.3375 | Val Loss: 2.6180 | Val Acc: 0.2717

Epoch 10/15 | Duration: 40.59s
Train Loss: 2.2804 | Val Loss: 2.5118 | Val Acc: 0.3098
New best validation accuracy: 0.3098. Model saved.

Epoch 11/15 | Duration: 40.71s
Train Loss: 2.2062 | Val Loss: 2.5492 | Val Acc: 0.3179
New best validation accuracy: 0.3179. Model saved.

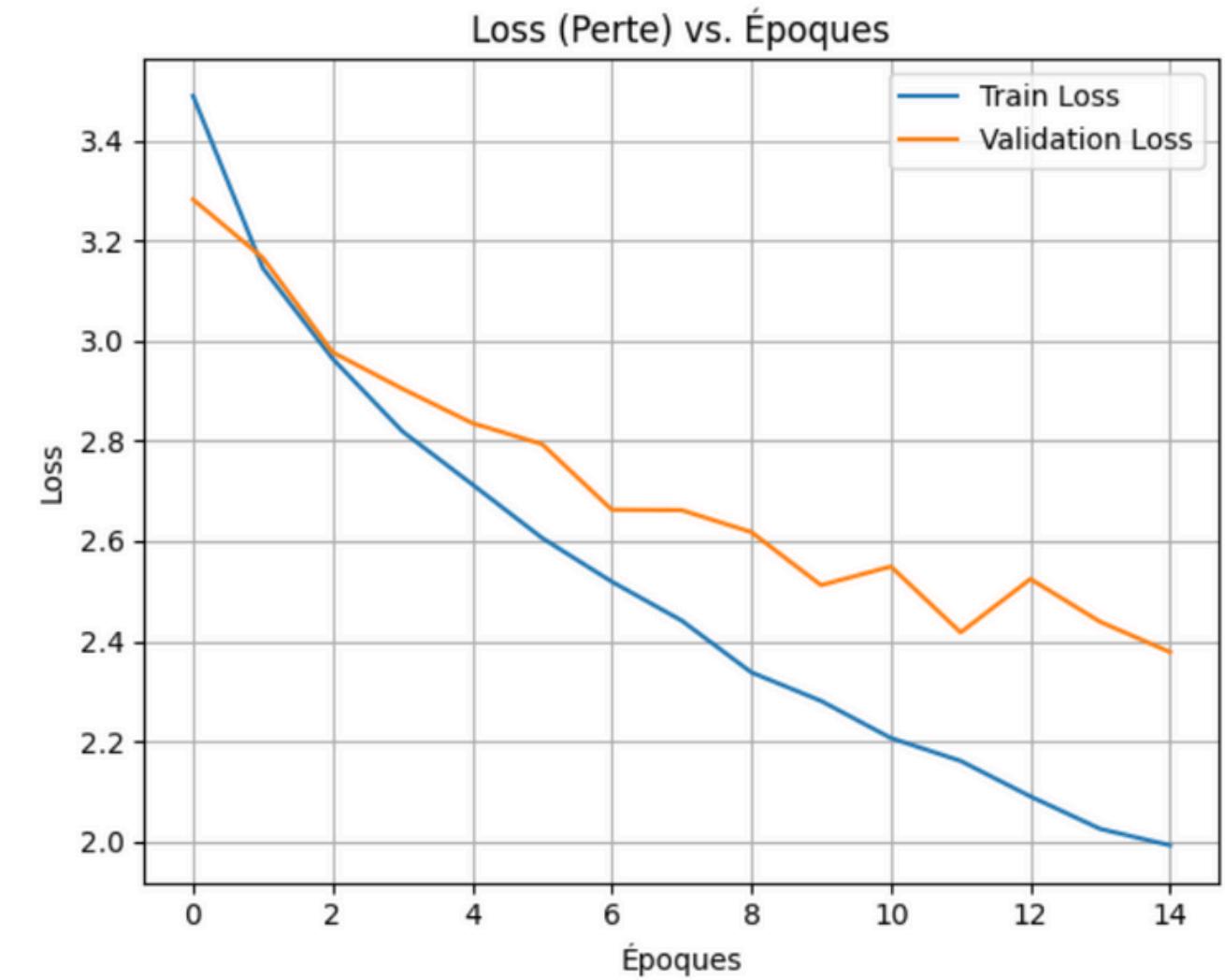
Epoch 12/15 | Duration: 40.60s
Train Loss: 2.1604 | Val Loss: 2.4175 | Val Acc: 0.3505
New best validation accuracy: 0.3505. Model saved.

Epoch 13/15 | Duration: 40.58s
Train Loss: 2.0895 | Val Loss: 2.5242 | Val Acc: 0.3098

Epoch 14/15 | Duration: 40.69s
Train Loss: 2.0245 | Val Loss: 2.4386 | Val Acc: 0.3179

Epoch 15/15 | Duration: 40.57s
Train Loss: 1.9918 | Val Loss: 2.3782 | Val Acc: 0.3397

Baseline training complete.

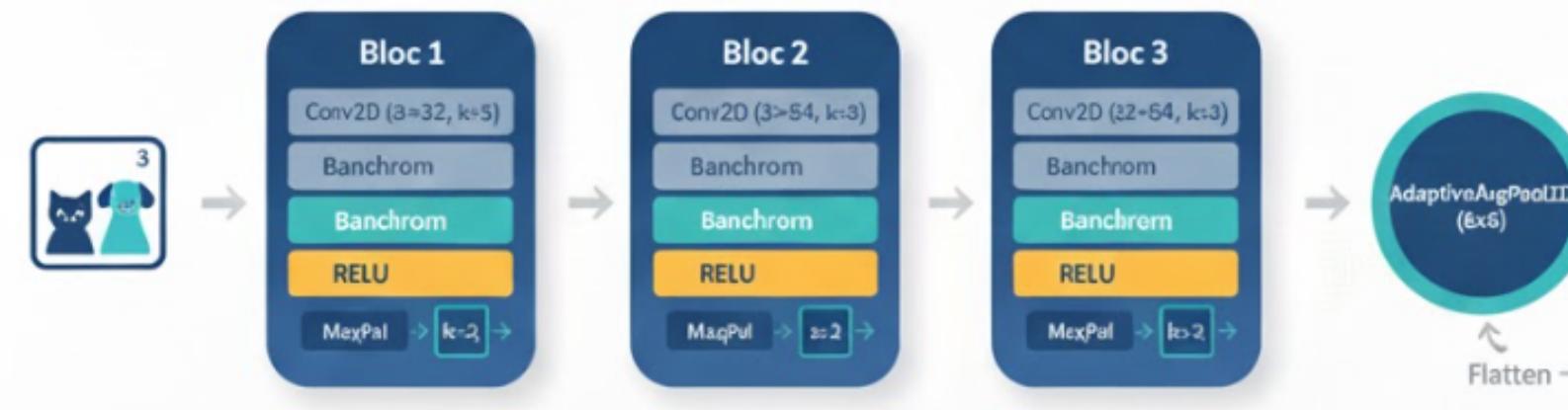


3. Classification fine

3.1 Baseline

B. DeeperCNN pour une baseline

Augmenter la capacité du réseau en ajoutant plus de couches de convolution (approche **VGG-like**). Le modèle est passé à 6 couches Conv et plus de filtres.



Résultats

Epoch 13/15 | Duration: 69.98s

Train Loss: 3.6707

Val Loss: 3.6584 | Val Acc: 0.0136

Epoch 14/15 | Duration: 69.24s

Train Loss: 3.6687

Val Loss: 3.6596 | Val Acc: 0.0136

Epoch 15/15 | Duration: 69.25s

Train Loss: 3.6739

Val Loss: 3.6584 | Val Acc: 0.0136

Training complete.

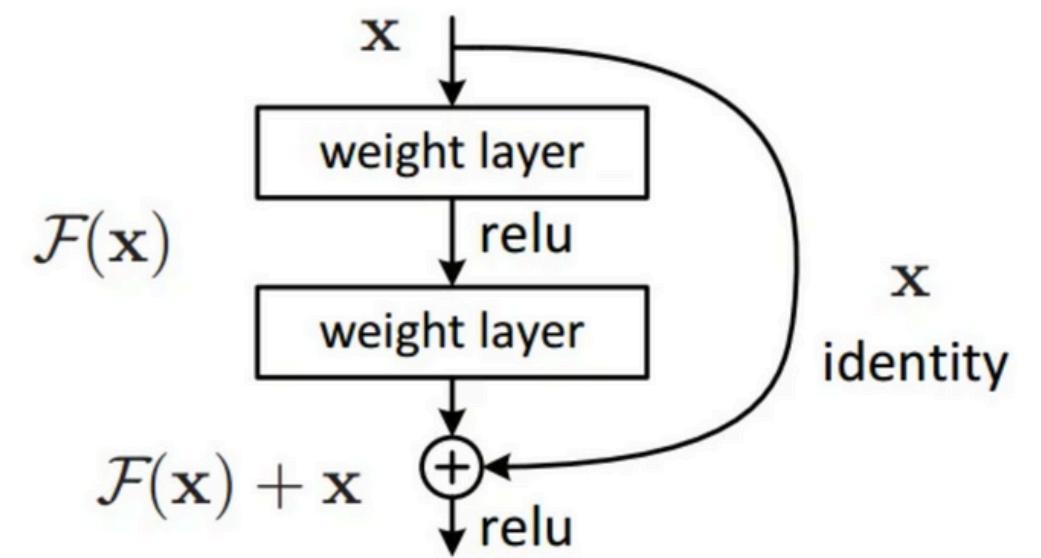
Plus on ajoute des couches, plus le problème devient impossible à optimiser!!!

3. Classification fine

3.1 Baseline

C. Residual blocks

Plutôt que d'apprendre directement sur une transformation $H(x)$, le réseau apprend sur fonction résiduelle $H(x)=F(x)+x$ qui est beaucoup plus simple à optimiser !



Résultats

```
Epoch 14/15 | Duration: 41.76s
  Train Loss: 1.1304
  Val Loss: 2.2597 | Val Acc: 0.3995
Epoch 15/15 | Duration: 41.40s
  Train Loss: 1.0699
  Val Loss: 2.0982 | Val Acc: 0.4049
  New best validation accuracy achieved: 0.4049. Model saved to best_resnetlike_model.pth
ResNetLike model training complete.
```

Performance finale sur l'ensemble de TEST :

Test Loss: 2.4615
Test Accuracy: 0.3579

3. Classification fine

3.1 Transfer learning

ResNet-18

- Télécharger le modèle avec les poids d'images Net.
- Geler les poids de toutes les couches à part la dernière.
- Changer la sortie de 1000 à 37 classes.
- Le leaning rate est de 0.00005 (très petit mais appliqué sur une seule couche)

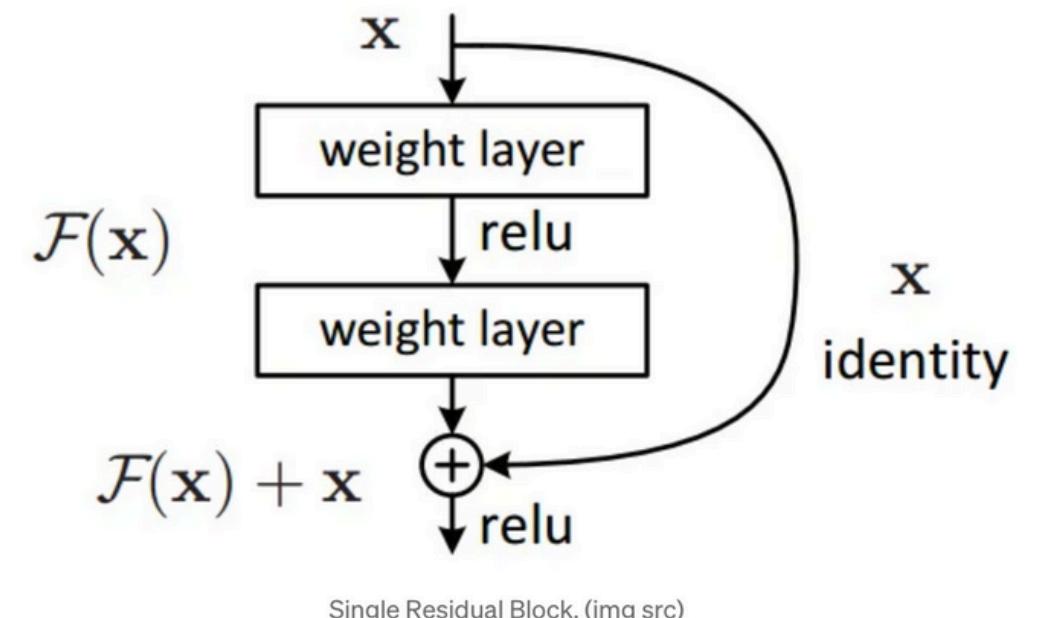
Résultats

```
Epoch 10/10 | Duration: 36.52s
Train Loss: 1.6480
Val Loss: 1.5364 | Val Acc: 0.7935
Current LR: 0.000050
New best validation accuracy: 0.7935. Model saved.
```

Transfer Learning training complete.

```
--- Évaluation Finale du Modèle Pré-entraîné sur l'Ensemble de TEST ---
Best ResNet18 TL model loaded from best_resnet18_transfer_learning_model.pth

ResNet18 TL Model Test Loss: 1.5647
ResNet18 TL Model Test Accuracy: 0.7691
```



3. Classification fine

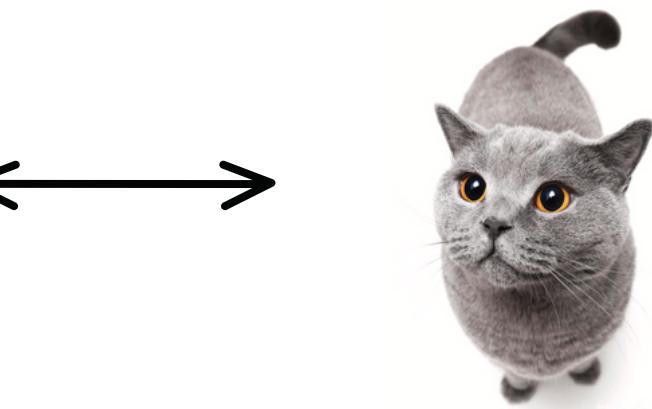
3.1 Transfer learning

C. ResNet-18

Les classes les plus confondues:



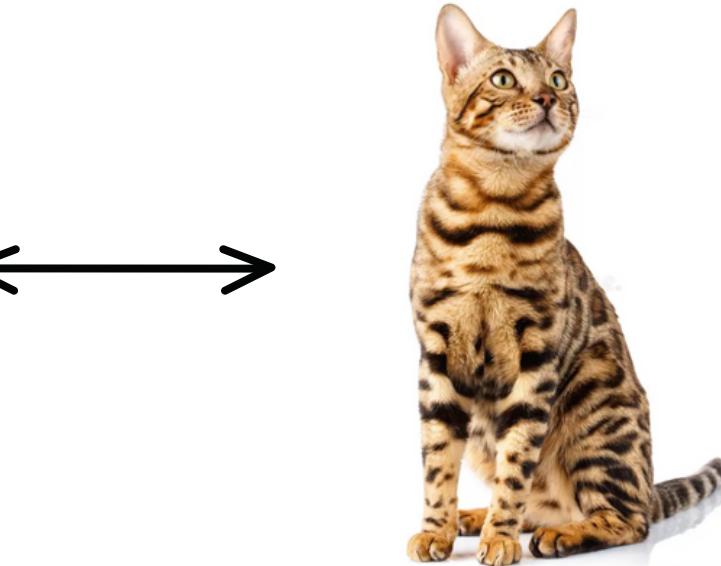
28- Russian blue



10- British short hair



1- Abyssinian



6- Bengal

3. Classification fine

3.1 Fine Tuning

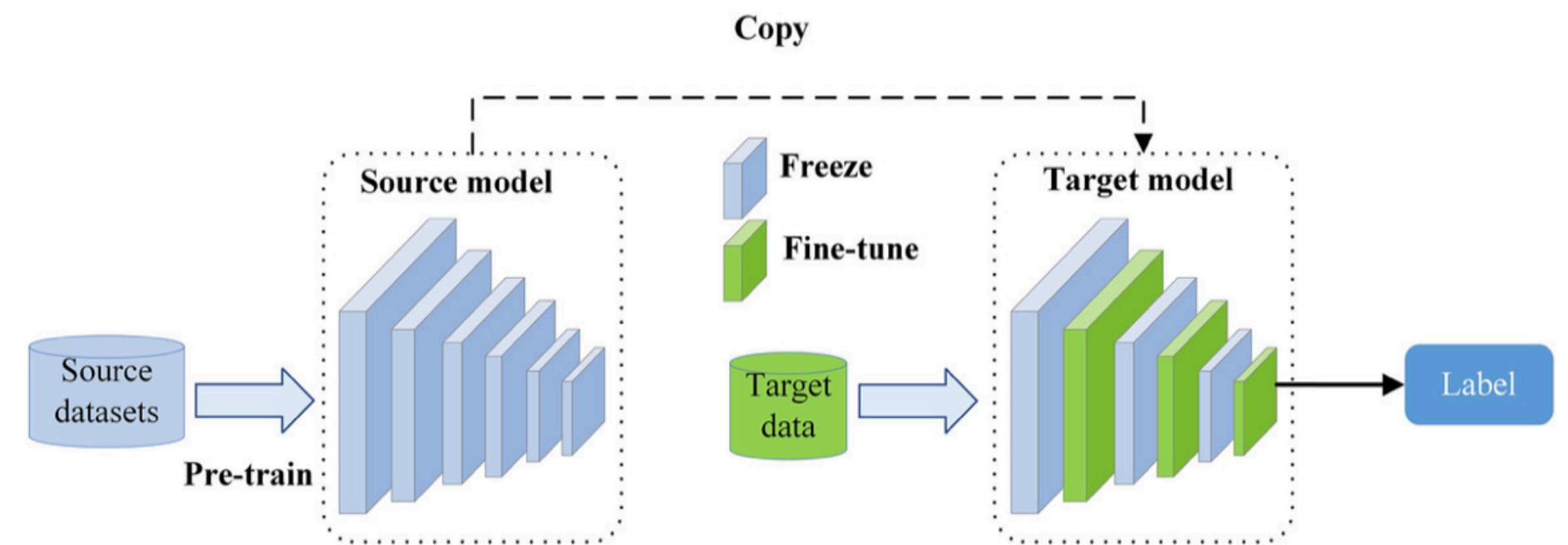
ResNet-18

- Dégel des couches intermédiaires (Layer 3 et 4).
- Learning rate ($5e-6$) encore plus petit que celui du Transfer learning.
- Changer la sortie de 1000 à 37 classes.

Résultats

--- Évaluation Finale du Modèle de Fine-Tuning sur l'Ensemble de TEST ---
Meilleur modèle de Fine-Tuning chargé depuis `best_resnet18_fine_tuning_model.pth`

ResNet18 Fine-Tuning Model Test Loss: 0.6048
ResNet18 Fine-Tuning Model Test Accuracy: 0.8774



3. Classification fine

3.1 Fine tuning

ResNet-18



34-stafffordshire bull terrier



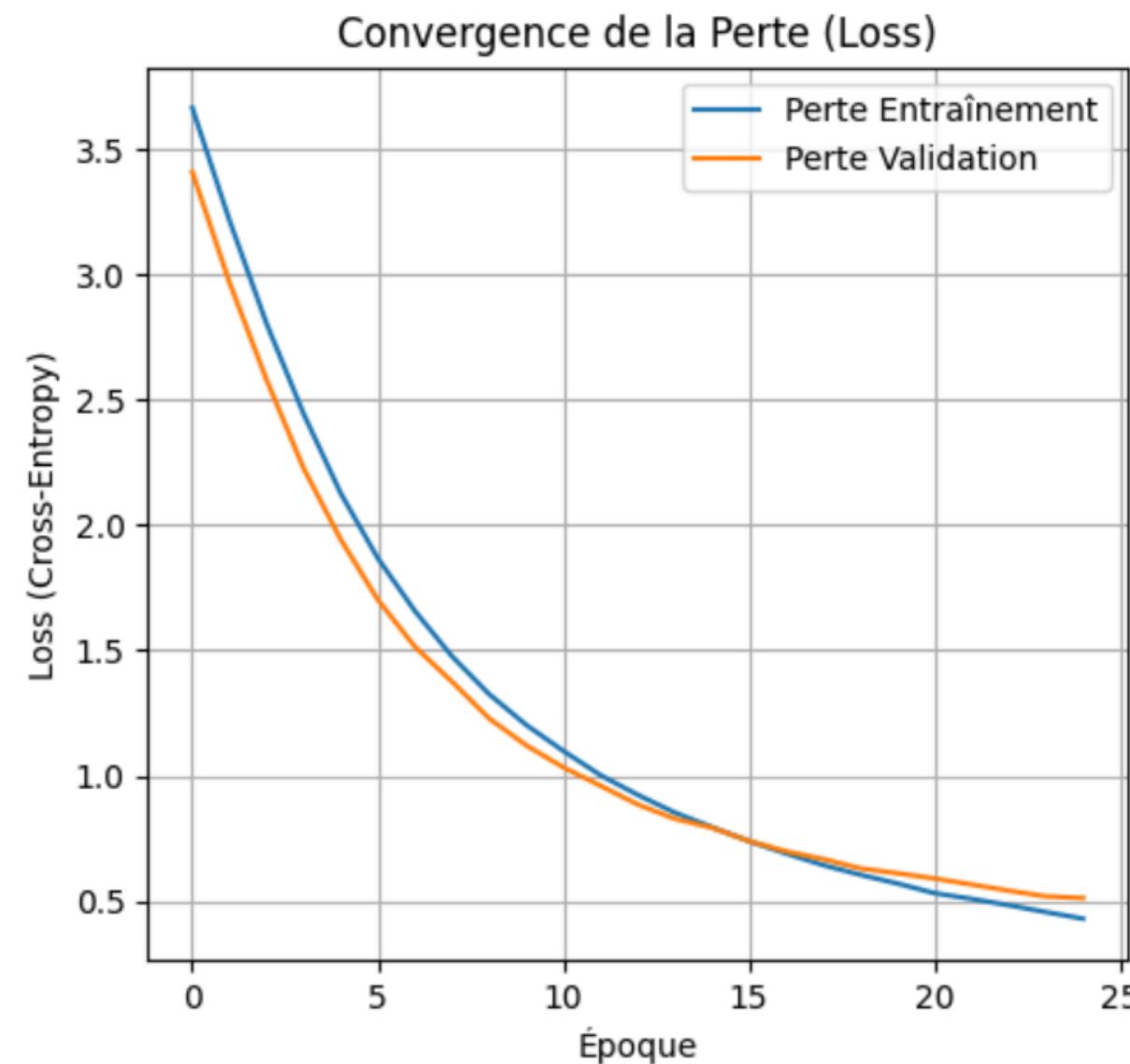
3- american pit bull terrier

Les classes les plus confondues sont toujours les mêmes

3. Classification fine

3.1 Fine tuning

ResNet-18



4. Segmentation de l'animal:

Objectif:

- Le but est de classifier chaque pixel de l'image : appartient-il à l'animal, à la bordure ou au fond ?
- On cherche donc à produire un masque de 3 classes précis qui détourne parfaitement l'animal.



Le Défi de la Segmentation :

- L'information de "haut niveau" (fin du réseau) est riche en contexte mais pauvre en détails géométriques.
- Reconstruire un masque à partir de ces seules données produit des contours grossiers.

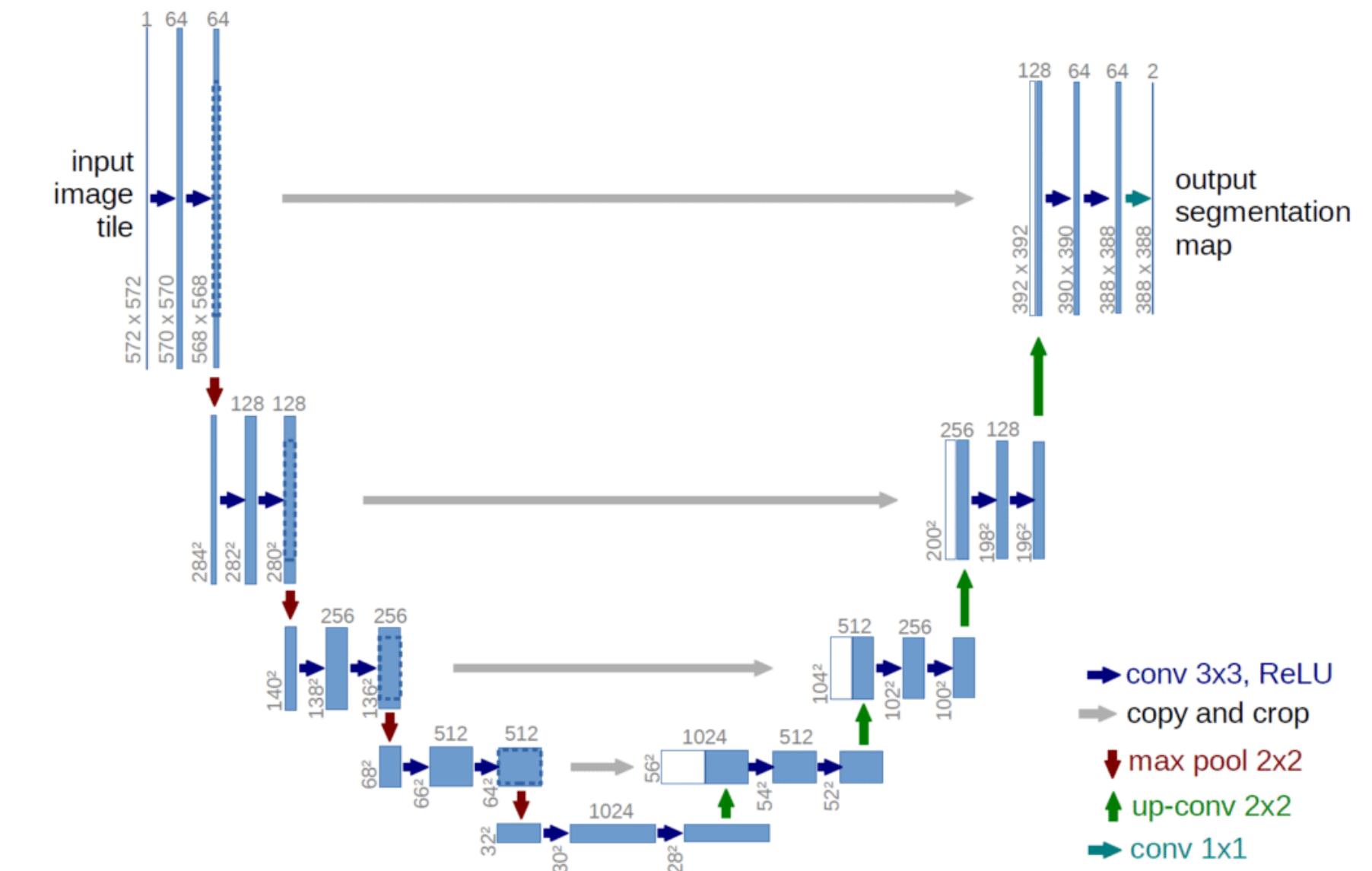
4. Segmentation de l'animal:

4.1 Réseau U-Net

Idée du réseau:

U-Net repose sur une architecture Encoder–Decoder symétrique:

- L'**encodeur** extrait le contexte global
- Le **décodeur** reconstruit la localisation précise des objets.
- Les **skip connections** permettent de conserver les détails spatiaux et d'améliorer la qualité des contours.



Entraînement du modèle:

Fonction de perte:

On a un problème de segmentation multi-classes, donc on utilise l'entropie croisée

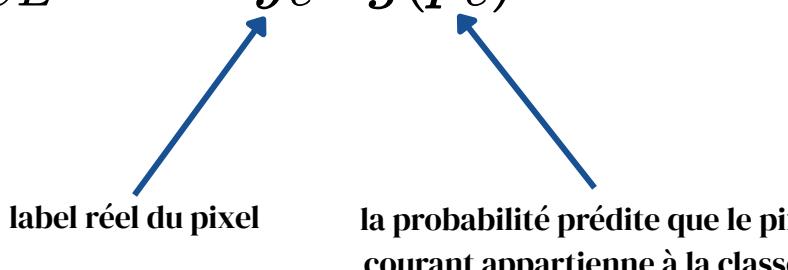
Idée:

- Chaque pixel est traité comme une classification indépendante

$$L_{CE} = -\sum y_c \log(p_c)$$

avec

$$p_c = \frac{e^{z_c}}{\sum e^{z_k}}$$



Optimisation et paramètres d'entraînement:

- Optimiseur : Adam
- Learning rate = 1e-4
- Nombre d'époques : 10
- Taille de batch = 16

Métriques:

1- Coefficient de Dice:

Cette métrique mesure le recouvrement entre le masque prédit et le masque réel:

$$Dice = \frac{2|P \cap G|}{|P| + |G|}$$

avec:

- P : pixels prédits comme appartenant à une classe.
- G : pixels réels de cette classe.

2- Intersection over Union (IoU):

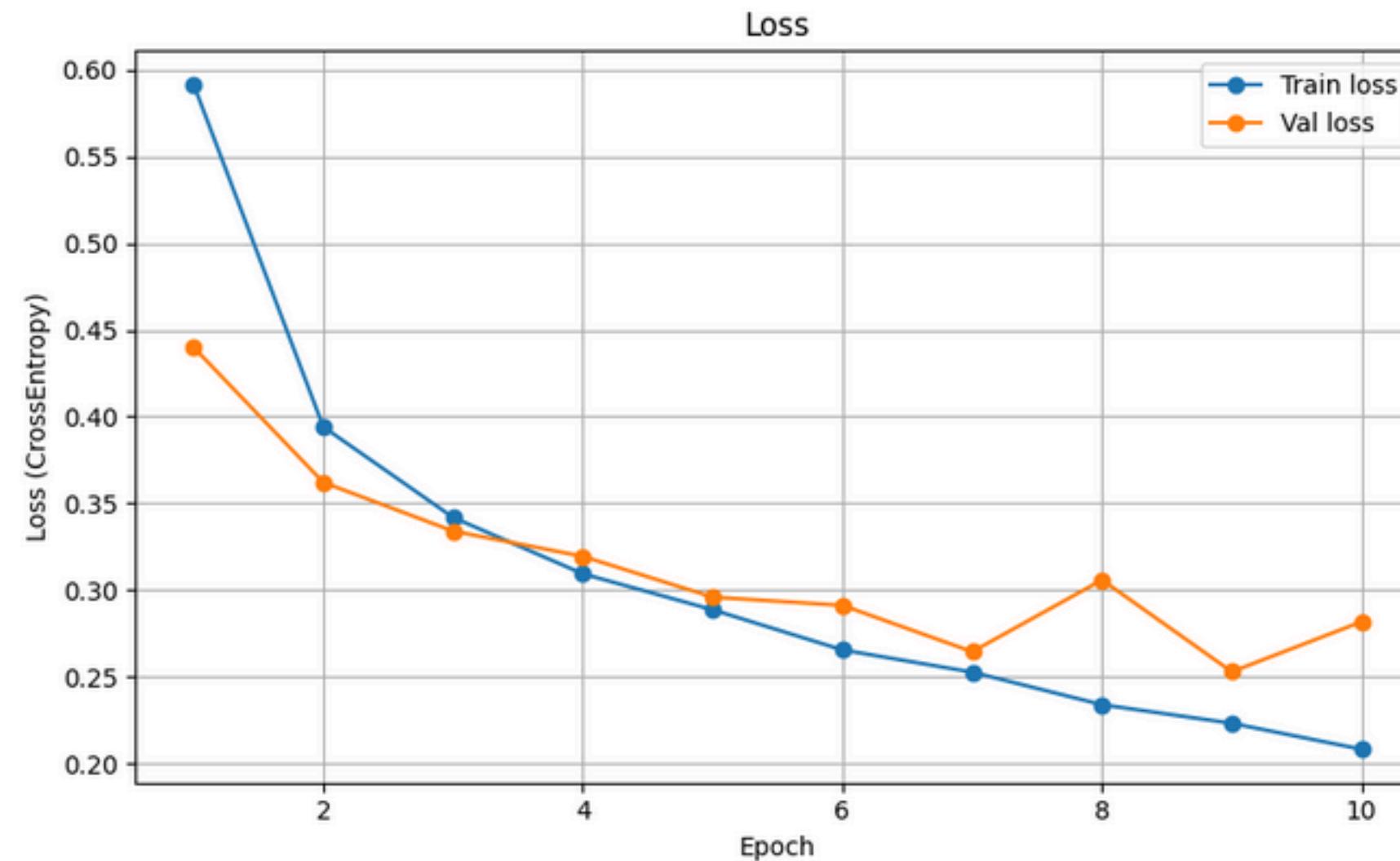
Cette métrique mesure le rapport entre l'intersection et l'union des régions prédites et réelles.

$$IoU = \frac{|P \cap G|}{|P \cup G|}$$

Entraînement du modèle:

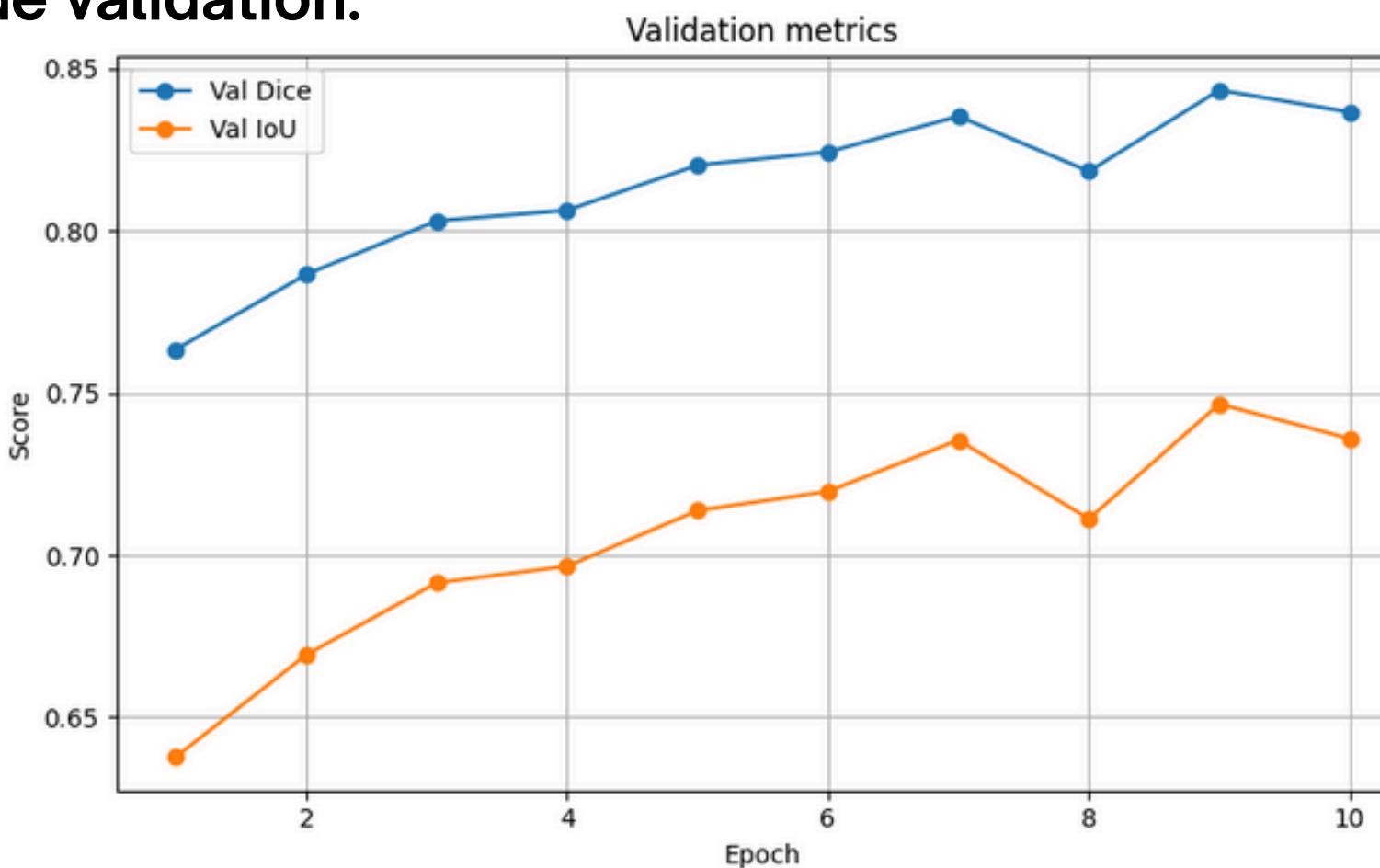
Courbes de perte:

- La perte d'entraînement décroît de manière régulière.
- La perte de validation suit une tendance similaire.



Sélection du meilleur modèle:

Courbes des métriques sur le jeu de validation:



```
Epoch 01/10 | Train loss: 0.5917 | Val loss: 0.4400 | Val Dice: 0.7635 | Val IoU: 0.6377
Epoch 02/10 | Train loss: 0.3941 | Val loss: 0.3620 | Val Dice: 0.7866 | Val IoU: 0.6692
Epoch 03/10 | Train loss: 0.3418 | Val loss: 0.3339 | Val Dice: 0.8031 | Val IoU: 0.6915
Epoch 04/10 | Train loss: 0.3093 | Val loss: 0.3193 | Val Dice: 0.8064 | Val IoU: 0.6966
Epoch 05/10 | Train loss: 0.2886 | Val loss: 0.2958 | Val Dice: 0.8203 | Val IoU: 0.7137
Epoch 06/10 | Train loss: 0.2654 | Val loss: 0.2911 | Val Dice: 0.8243 | Val IoU: 0.7196
Epoch 07/10 | Train loss: 0.2526 | Val loss: 0.2644 | Val Dice: 0.8353 | Val IoU: 0.7355
Epoch 08/10 | Train loss: 0.2338 | Val loss: 0.3058 | Val Dice: 0.8184 | Val IoU: 0.7112
Epoch 09/10 | Train loss: 0.2231 | Val loss: 0.2528 | Val Dice: 0.8434 | Val IoU: 0.7466
Epoch 10/10 | Train loss: 0.2081 | Val loss: 0.2817 | Val Dice: 0.8366 | Val IoU: 0.7359
```

Meilleur modèle sauvegardé avec Val Dice = 0.8434 -> /home/el-morady/Téléchargements/HDDL_fv/best_unet_segmentation.pth

Résultats:

IoU métrique:

--- Résultats d'Évaluation sur le Jeu de Test ---

IoU par classe: [0.8629198465658271, 0.926282743267391, 0.5642097168642541]

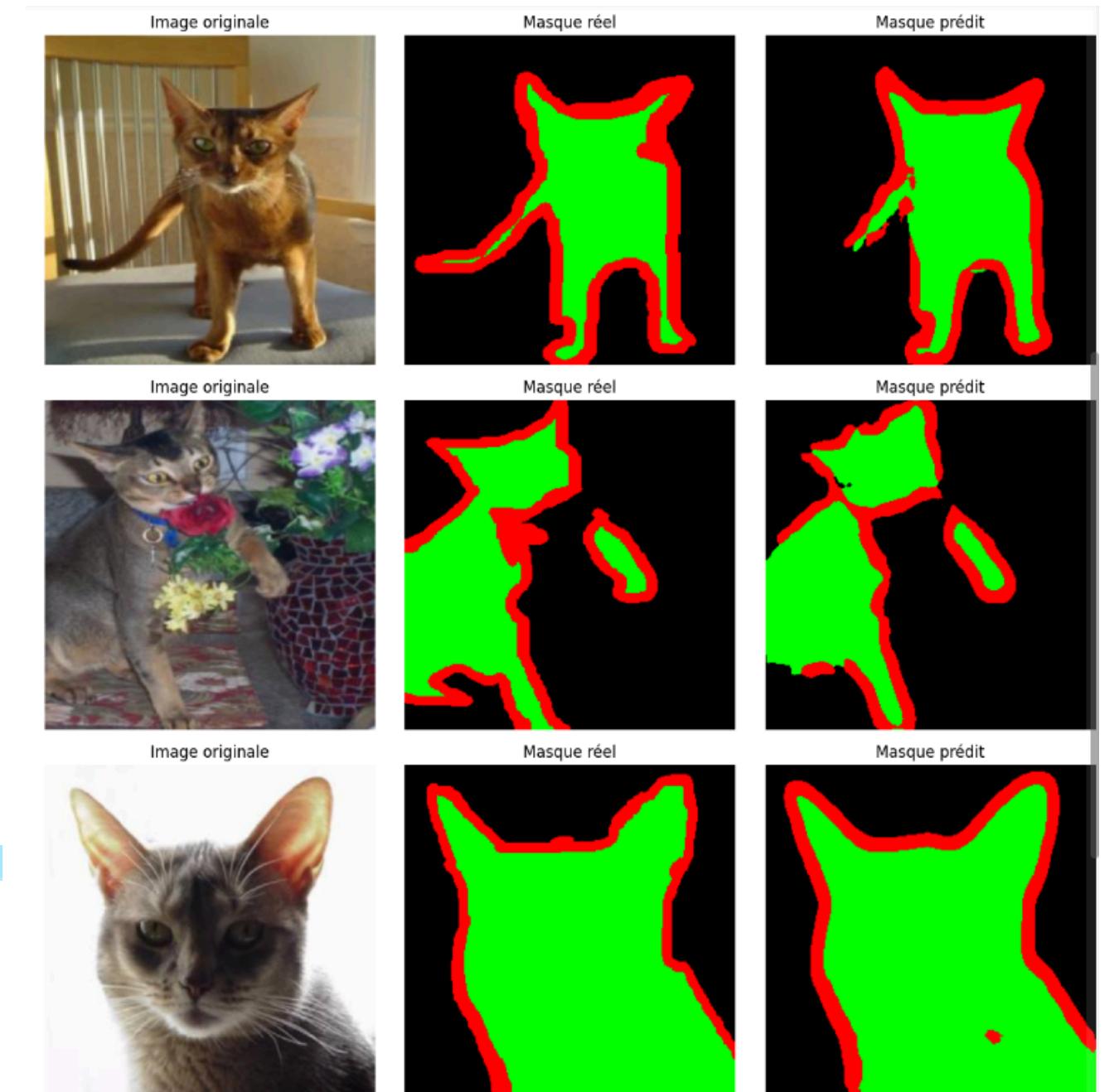
IoU moyen: 0.7845

Dice métrique:

--- Résultats d'Évaluation sur le Jeu de Test ---

Dice par classe: [0.9258973689182944, 0.9615324857442276, 0.7205007794110672]

Dice moyen: 0.8693



Performances de segmentation:

2- Chat et chien:

Dice moyen - Chats : 0.8626113853509159
Dice moyen - Chiens: 0.8610232195857341
IoU moyen - Chats : 0.7799841417638037
IoU moyen - Chiens: 0.7769301191117389

1- Selon les races:

	Race	Dice	IoU
0	miniature_pinscher	0.832434	0.738174
1	german_shorthaired	0.833594	0.738137
2	american_bulldog	0.839323	0.743295
3	basset_hound	0.843075	0.751028
4	japanese_chin	0.845696	0.760824
5	Persian	0.846993	0.766752
6	beagle	0.847535	0.755865
7	Sphynx	0.849554	0.759233
8	boxer	0.850444	0.758014
9	Siamese	0.850751	0.763011
10	Birman	0.854178	0.768034
11	yorkshire_terrier	0.854571	0.766448
12	keeshond	0.856118	0.779002
13	english_cocker_spaniel	0.856985	0.771327
14	american_pit_bull_terrier	0.857582	0.769305

5. Classification et segmentation conjointes:

Objectif:

- Concevoir un modèle capable de réaliser simultanément :
 - la segmentation de l'animal (au niveau des pixels).
 - la classification de l'image (au niveau global).
- Étendre l'architecture U-Net afin de prendre en compte ces deux tâches dans un cadre d'apprentissage multi-tâche.

Idée:

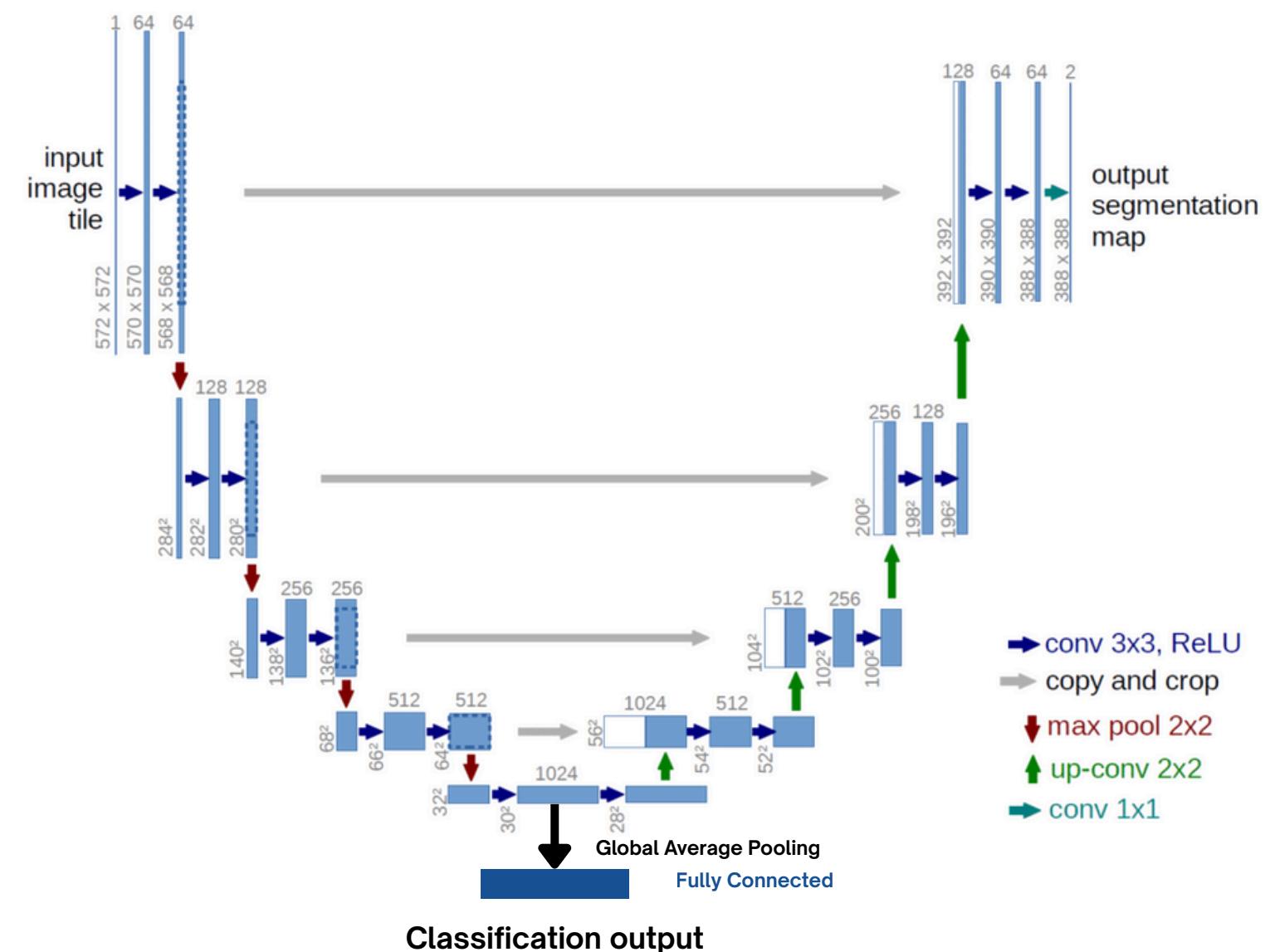
- Ajouter une head qui permet de faire de la classification (binaire/fine)

5. Classification et segmentation conjointes:

5.1 Réseau U-Net avec classification head

Le principe général du réseau:

- Un encodeur U-Net partagé extrait des caractéristiques communes à partir de l'image.
- Le décodeur est utilisé pour la segmentation de l'animal.
- Une tête de classification est ajoutée au niveau du bottleneck pour prédire la classe de l'image.



5. Classification et segmentation conjointes:

5.2 Fonction de perte:

La fonction de perte totale est une combinaison pondérée des pertes associées à chaque tâche :

$$L_{tot} = L_{seg} + \lambda L_{cls}$$

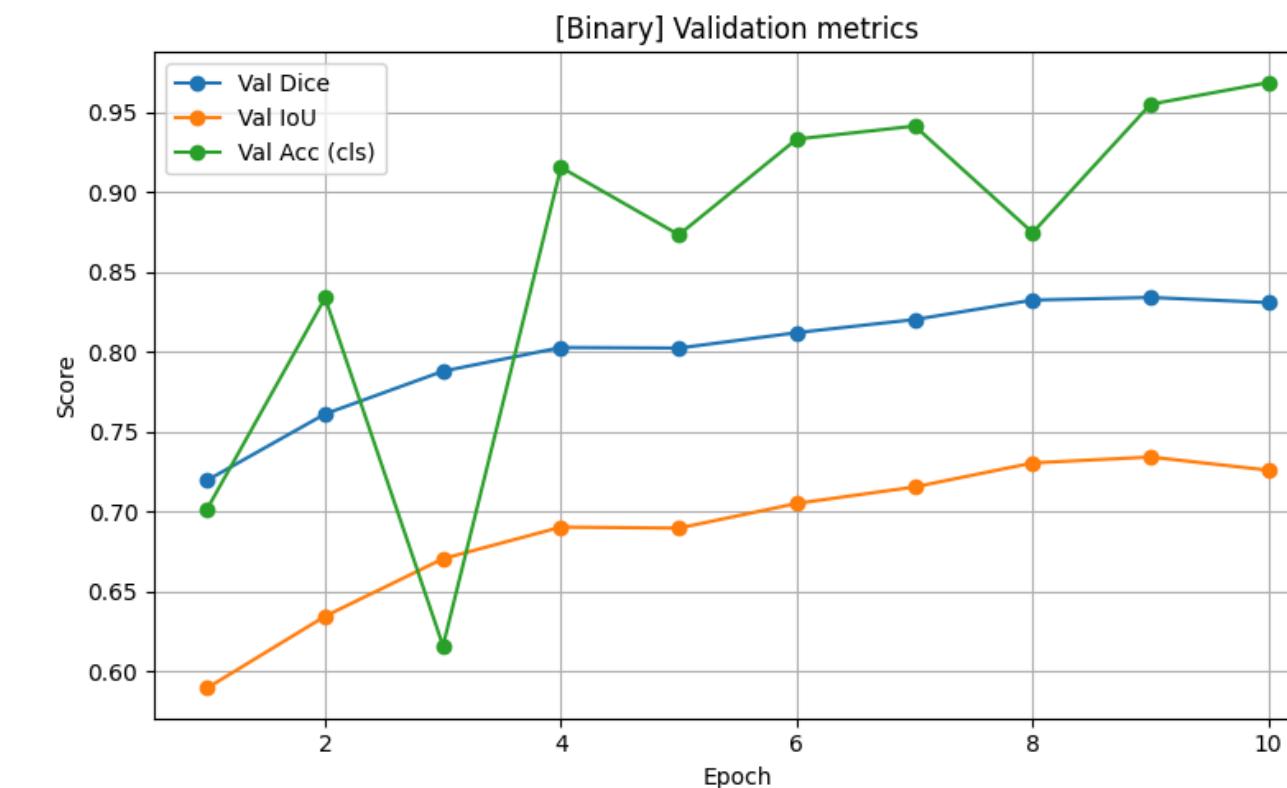
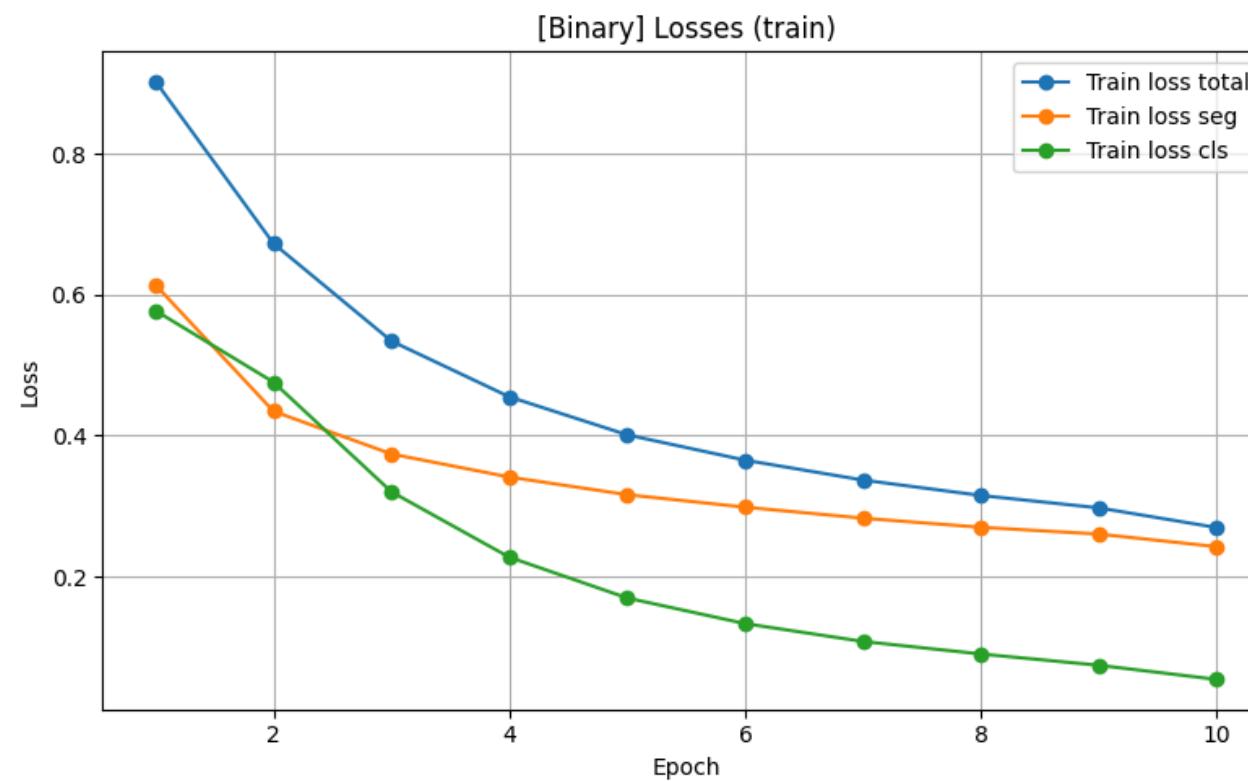
Entropie croisée multi-classe coefficient de pondération Entropie croisée

Choix dans notre projet: $\lambda = 0.5$

5. Classification et segmentation conjointes:

5.1 Réseau U-Net avec classification head

Classification binaire



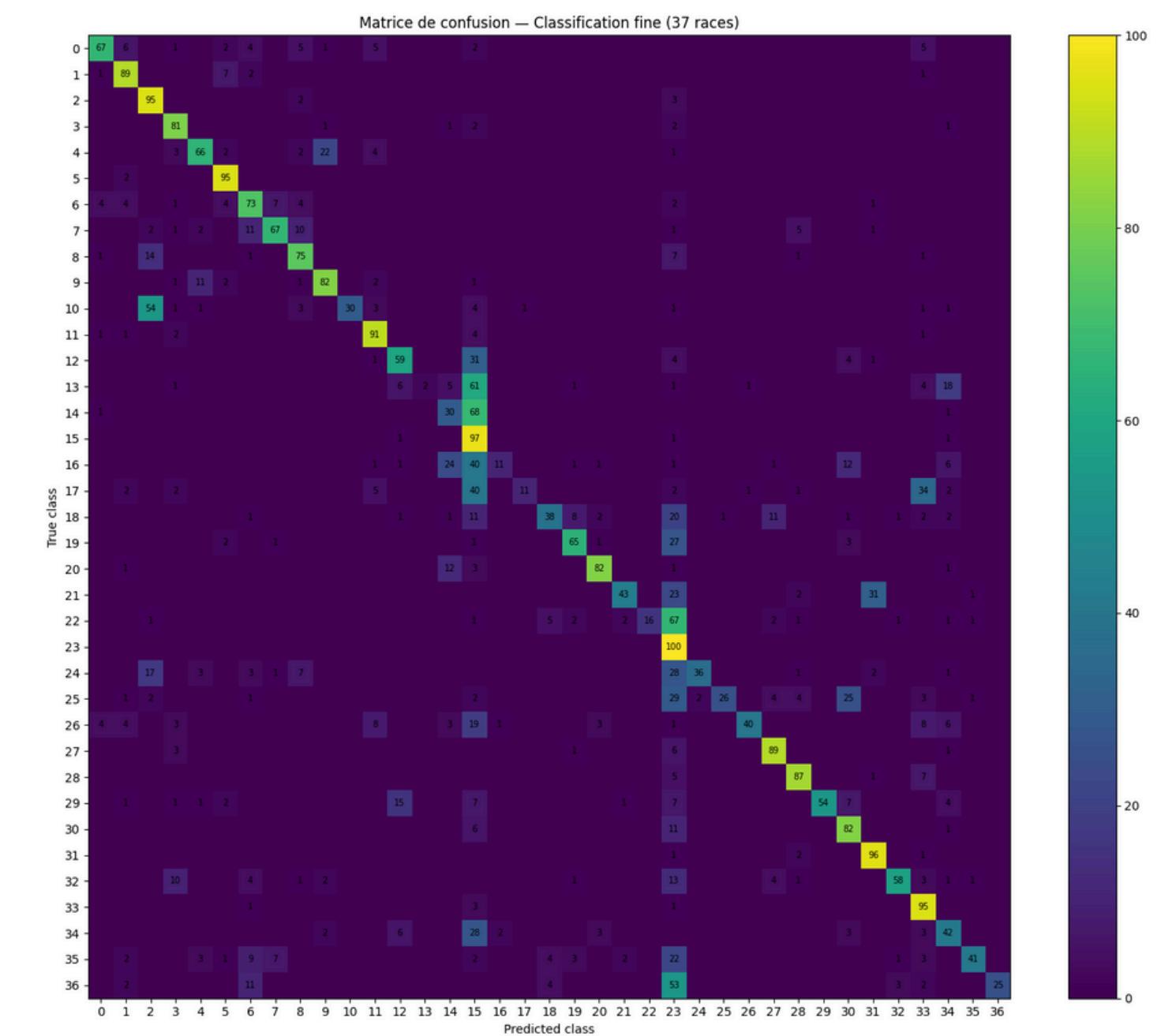
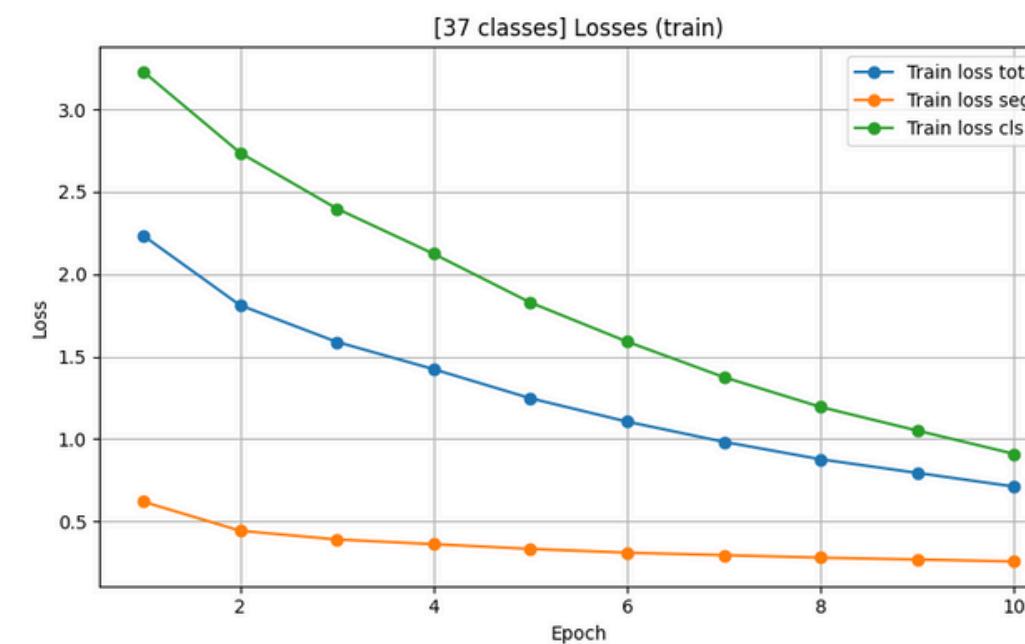
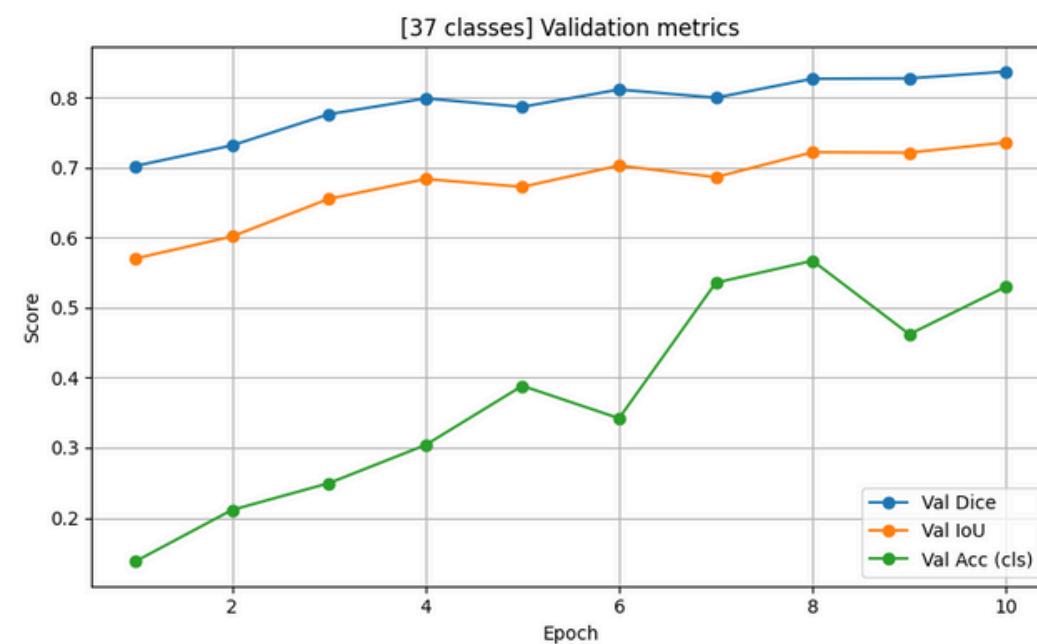
Résultat sur données de test:

[Test - Binary] Dice=0.8498 | IoU=0.7563 | Acc_cls=0.9899
Confusion 2x2 (rows=true, cols=pred):
[[1153 30]
 [7 2479]]

5. Classification et segmentation conjointes:

5.1 Réseau U-Net avec classification head

Classification des races



Résultat sur données de test:

[Test - 37 classes] Dice=0.8497 | IoU=0.7541 | Acc cls=0.6094