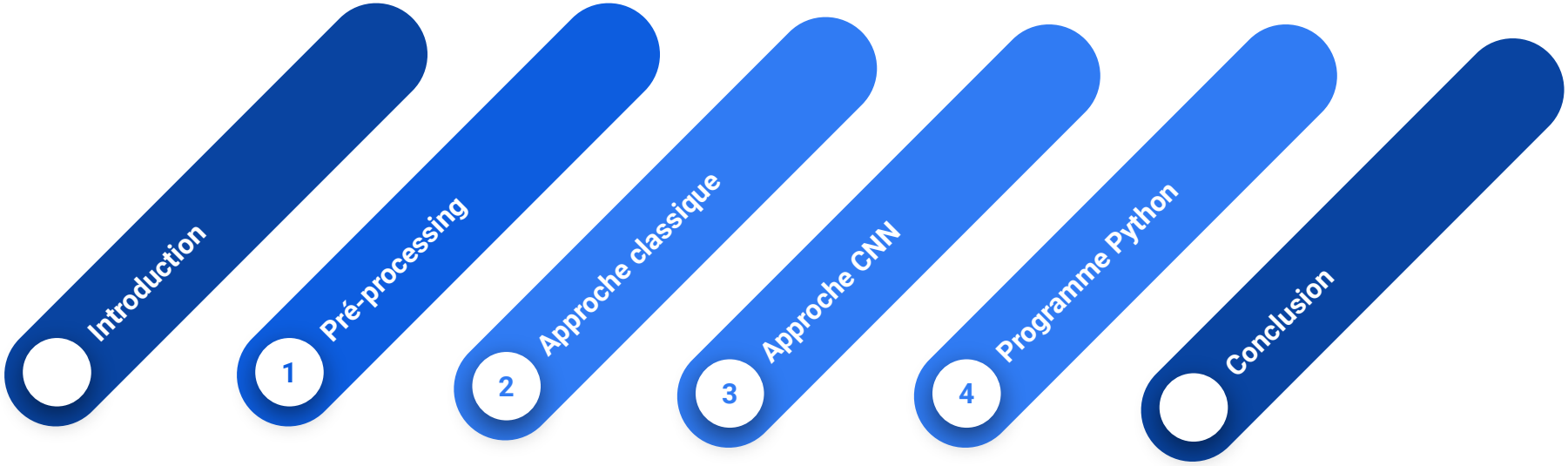
Projet 7:Réaliser des indexations automatiques des images

Réalisé par : Taher Haggui

Plan :


Introduction
1
Pré-processing
2
Approche classique
3
Approche CNN
4
Programme Python
Conclusion

Problématique :

Réaliser un algorithme de détection de la race du chien sur une photo, afin d'accélérer leur travail d'indexation.

Comment ?

1. Une approche classique de classification.

2. L’approche CNN .

3. Développer un programme python mettant en oeuvre les deux approches .

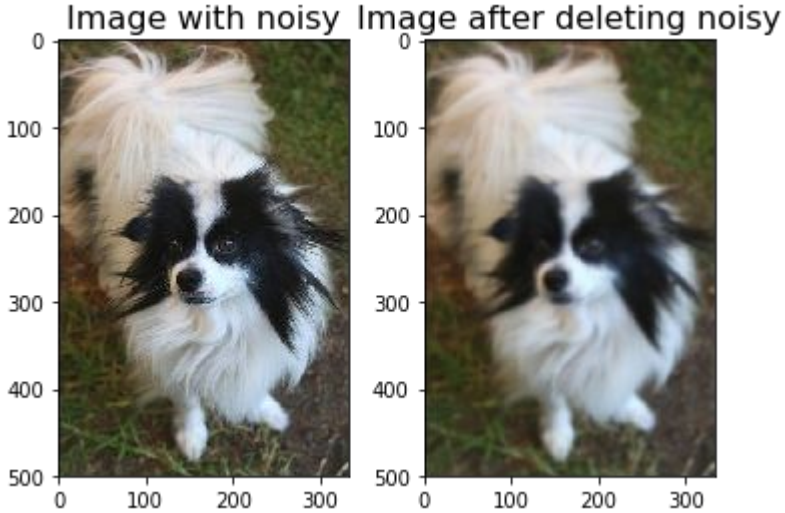
Résultat attendus:

Un programme python permettant de détecter la race de chien à partir de sa photo.

Pré-processing des images:Débruitage

Filtre utilisé: Filtre Gaussien

Illustration:



Pré-processing: égalisation d'histogramme d'images

Objectif: **Améliorer le contraste des images**

Démarche: **harmoniser la distribution des niveaux de couleur de l'image, de sorte que chaque niveau de l'histogramme contient idéalement le même nombre de pixels.**

Égalisation d'histogramme d'images: Illustration

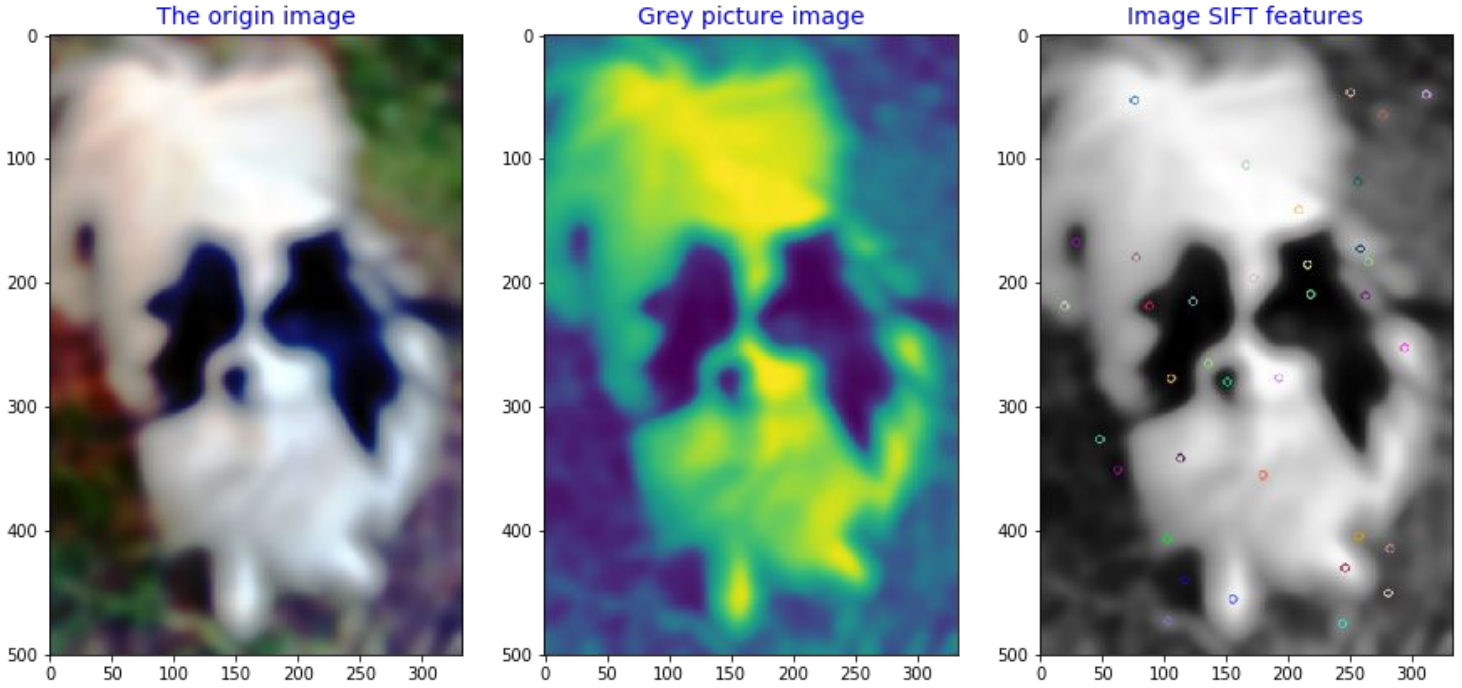
Classification: Approche classique

Keypoints et leur descripteurs:

- **Algorithme de SIFT**
- **Méthode permettant de détecter les keypoints de chaque image et leurs descripteurs.**

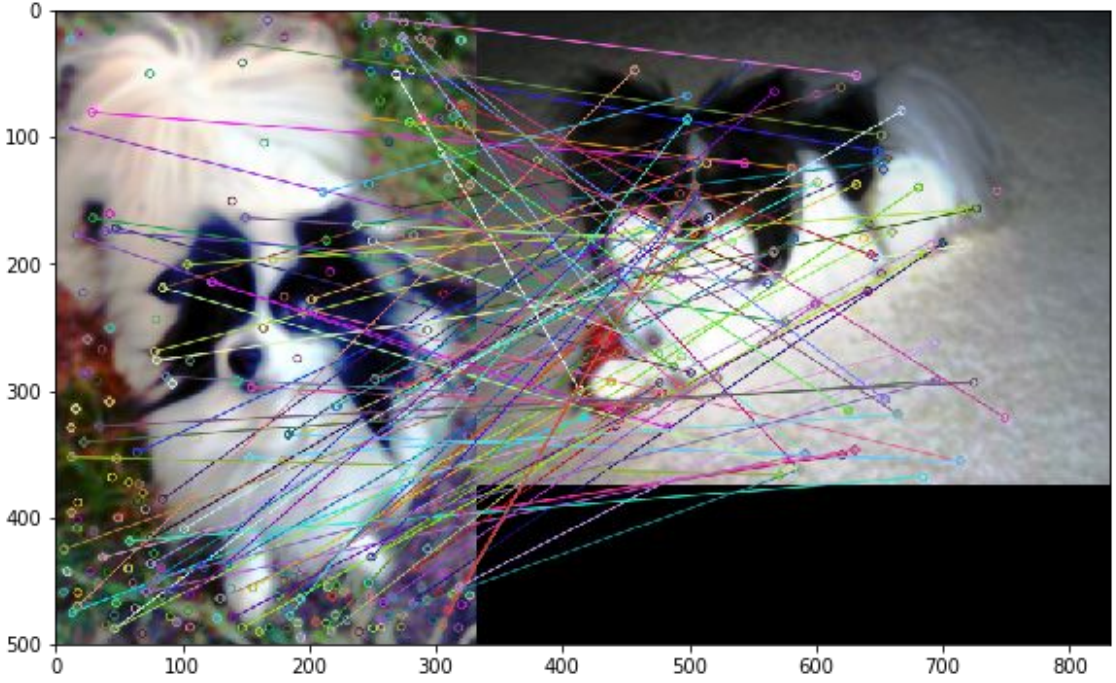
Classification: Approche classique

Keypoints et leur descripteurs (Illustration):



Classification: Approche classique

Images matches:



Classification: Approche classique

Descripteurs des images:

- ➔ Appliquer la fonction qui permet de détecter les keypoints et les descripteurs de chaque image.
- ➔ Stocker dans une liste les descripteurs de chaque image.
- ➔ Stocker tous les descripteurs verticalement dans un seul array à l'aide de méthode vstack de numpy.

Classification: Approche classique

Clustering :

Méthode de clustering: KMean

- ➔ Nous n'avons pas pu déterminer le meilleur nombre de cluster pour des contraintes de mémoire. Ainsi nous avons choisi un nombre de 500 clusters à identifier.

- ➔ Nous n'avons pas pu déterminer le coefficient de silhouette de cette clustering pour des raisons de limite de mémoire.

Classification: Approche classique

Bag of visual word :

- ➔ Nous avons crée un array de dimension nbreimage x 500.
- ➔ Dans chaque ligne, correspond à chaque image. Nous avons calculé à l'aide d'une boucle le nombre d'occurence de chaque cluster pour chaque image.

Classification: Approche classique

Classification :

- ➔ Nous avons séparé les données en données test et données d'entraînement . 60% des données sont des données d'entraînement.
- ➔ Nous avons choisi, la méthode SVM pour faire notre classification des données.

Classification: Approche classique

Classification : Performance de modèle

- ➔ Nous avons utilisé la méthode GridsearchCV pour déterminer le meilleur coefficient de régularisation de cette méthode par rapport à nos données . Nous avons trouvés un meilleur coefficient correspond à 1,5..

- ➔ La performance de cette méthode sur les données test révèle un coefficient de performance de **23%** .

Classification: Approche CNN

Choix du modèle :

Nous avons choisi d'utiliser :

- ➔ Le transfert learning.
- ➔ Le réseau VGG16

Approche CNN: Implémentation

Implémentation du modèle :

1. Utiliser les couches de convolution de modèle pré entraîné VGG16.
2. Ajouter une couche dense avec 210 sorties et ayant comme activation la fonction Relu.
3. Ajouter une deuxième couche dense avec 200 sorties et ayant comme activation la fonction Relu.
4. Ajouter une dernière couche dense avec un nombre de sortie correspond au nombre de classe et la fonction softmax comme fonction d'activation.
5. Réentraîner les couches supérieur à la cinquième couche sur nos données .

Approche CNN: Préparation des données

- ➔ Split automatique des images en deux dossiers d'entraînement et de validation.

- ➔ Data augmentation

Approche CNN: Compilation

Fonction d'erreur: Entropie croisée (cross entropy).

Optimiseur : SGD (taux d'apprentissage: 0.001).

Métrique : La fonction de précision accuracy.

Approche CNN: Entraînement sur toutes les classes

Fonction d'apprentissage : **fit_generator .**

époque : **600 époques.**

Pas par époque: **16 .**

Verbose : 2.

Approche CNN: Entraînement sur toutes les classes

Fonction d'apprentissage : **fit_generator .**

époque : **600 époques.**

Pas par époque: **16 .**

Verbose : 2.

Entraînement sur toutes les classes: illustration

Les premières époques:

```
Epoch 1/600
 - 373s - loss: 4.8920 - acc: 0.0078 - val_loss: 4.8445 - val_acc: 0.0234
Epoch 2/600
 - 395s - loss: 4.8641 - acc: 0.0000e+00 - val_loss: 4.8442 - val_acc: 0.0000e+00
Epoch 3/600
 - 395s - loss: 4.8386 - acc: 0.0000e+00 - val_loss: 4.8066 - val_acc: 0.0117
Epoch 4/600
 - 425s - loss: 4.7966 - acc: 0.0039 - val_loss: 4.8003 - val_acc: 0.0117
Epoch 5/600
 - 397s - loss: 4.7997 - acc: 0.0078 - val_loss: 4.7979 - val_acc: 0.0117
Epoch 6/600
 - 423s - loss: 4.7917 - acc: 0.0117 - val_loss: 4.7897 - val_acc: 0.0195
Epoch 7/600
 - 426s - loss: 4.7911 - acc: 0.0000e+00 - val_loss: 4.7886 - val_acc: 0.0040
Epoch 8/600
 - 442s - loss: 4.7815 - acc: 0.0039 - val_loss: 4.7885 - val_acc: 0.0195
Epoch 9/600
 - 409s - loss: 4.7932 - acc: 0.0156 - val_loss: 4.7840 - val_acc: 0.0156
Epoch 10/600
```

Les dernières époques:

```
Epoch 592/600
 - 372s - loss: 0.9497 - acc: 0.7188 - val_loss: 1.9597 - val_acc: 0.4883
Epoch 593/600
 - 370s - loss: 1.1929 - acc: 0.6328 - val_loss: 1.9798 - val_acc: 0.5312
Epoch 594/600
 - 373s - loss: 1.2628 - acc: 0.6250 - val_loss: 1.9476 - val_acc: 0.4922
Epoch 595/600
 - 372s - loss: 1.3570 - acc: 0.6055 - val_loss: 1.7975 - val_acc: 0.5352
Epoch 596/600
 - 370s - loss: 1.2560 - acc: 0.6562 - val_loss: 1.9010 - val_acc: 0.4922
Epoch 597/600
 - 370s - loss: 1.1187 - acc: 0.6758 - val_loss: 2.1196 - val_acc: 0.4531
Epoch 598/600
 - 369s - loss: 1.1618 - acc: 0.6406 - val_loss: 2.1356 - val_acc: 0.4492
Epoch 599/600
 - 369s - loss: 1.0207 - acc: 0.6836 - val_loss: 2.1774 - val_acc: 0.4453
Epoch 600/600
 - 369s - loss: 1.2990 - acc: 0.5742 - val_loss: 1.8888 - val_acc: 0.4453

<keras.callbacks.History at 0x7f6dca9405c0>
```

Entraînement sur toutes les classes: Principaux résultats

Résultat initial (époque 1)
- Loss de validation=4.84
- Accuracy de validation =0.02

Résultat final (après 600 époques)
- Loss de validation=1.89
- Accuracy de validation=0.44

Meilleur résultat (époque 575)
- Loss de validation=1.54
- Accuracy de validation =0.58

Approche CNN : 10 classes

Modèle: même architecture du modèle choisie.

Compilation: méme paramètres de compilation.

Apprentissage:

- époques = 50
- verbose=2
-pas par époque = 12
-callbacks: checkpoints et un early stopping (min_delta=0.05 , patience=20,monitor=accuracy de validation)

Approche CNN : 10 classes

Modèle: même architecture du modèle choisie.

Compilation: méme paramètres de compilation.

Apprentissage:

- époques = 50
- verbose=2
-pas par époque = 12
-callbacks: checkpoints et un early stopping (min_delta=0.05 , patience=20,monitor=accuracy de validation)

Entraînement sur 10 classes : Illustration

Les premières époques

```
Epoch 1/50
 - 389s - loss: 0.6495 - acc: 0.7852 - val_loss: 0.9185 - val_acc: 0.6911
Epoch 2/50
 - 374s - loss: 0.9011 - acc: 0.6726 - val_loss: 1.0560 - val_acc: 0.6289
Epoch 3/50
 - 385s - loss: 0.6782 - acc: 0.7969 - val_loss: 0.8623 - val_acc: 0.7073
Epoch 4/50
 - 367s - loss: 0.7132 - acc: 0.7656 - val_loss: 0.7288 - val_acc: 0.7422
Epoch 5/50
 - 361s - loss: 0.6192 - acc: 0.7656 - val_loss: 0.7208 - val_acc: 0.7439
Epoch 6/50
 - 365s - loss: 0.8104 - acc: 0.6953 - val_loss: 0.9435 - val_acc: 0.6953
Epoch 7/50
 - 356s - loss: 0.6291 - acc: 0.7730 - val_loss: 0.7847 - val_acc: 0.7109
Epoch 8/50
 - 361s - loss: 0.5921 - acc: 0.7734 - val_loss: 1.0894 - val_acc: 0.6789
Epoch 9/50
 - 365s - loss: 0.5243 - acc: 0.8398 - val_loss: 0.5284 - val_acc: 0.7969
Epoch 10/50
 - 360s - loss: 0.6065 - acc: 0.7812 - val_loss: 1.0012 - val_acc: 0.6789
Epoch 11/50
 - 365s - loss: 0.5088 - acc: 0.8320 - val_loss: 0.6518 - val_acc: 0.7930
Epoch 12/50
 - 352s - loss: 0.4957 - acc: 0.8356 - val_loss: 0.5397 - val_acc: 0.8089
Epoch 13/50
 - 365s - loss: 0.3515 - acc: 0.8867 - val_loss: 0.8654 - val_acc: 0.7305
Epoch 14/50
 - 368s - loss: 0.6135 - acc: 0.7930 - val_loss: 0.8290 - val_acc: 0.7480
Epoch 15/50
 - 367s - loss: 0.3876 - acc: 0.8672 - val_loss: 0.5237 - val_acc: 0.8438
Epoch 16/50
 - 364s - loss: 0.4535 - acc: 0.8633 - val_loss: 0.7747 - val_acc: 0.7266
```

Les dernières époques

```
Epoch 28/50
 - 361s - loss: 0.1693 - acc: 0.9375 - val_loss: 0.8589 - val_acc: 0.8049
Epoch 29/50
 - 365s - loss: 0.1447 - acc: 0.9531 - val_loss: 0.6208 - val_acc: 0.8320
Epoch 30/50
 - 361s - loss: 0.2074 - acc: 0.9258 - val_loss: 0.5663 - val_acc: 0.8415
Epoch 31/50
 - 365s - loss: 0.1677 - acc: 0.9453 - val_loss: 0.6417 - val_acc: 0.8242
Epoch 32/50
 - 361s - loss: 0.1365 - acc: 0.9491 - val_loss: 1.3858 - val_acc: 0.7195
Epoch 33/50
 - 368s - loss: 0.3802 - acc: 0.8633 - val_loss: 0.6817 - val_acc: 0.8086
Epoch 34/50
 - 365s - loss: 0.1974 - acc: 0.9492 - val_loss: 0.5916 - val_acc: 0.8477
Epoch 35/50
 - 362s - loss: 0.2192 - acc: 0.9453 - val_loss: 0.5479 - val_acc: 0.8496
Epoch 36/50
 - 366s - loss: 0.1960 - acc: 0.9453 - val_loss: 0.6992 - val_acc: 0.7773
Epoch 37/50
 - 352s - loss: 0.2038 - acc: 0.9178 - val_loss: 0.8030 - val_acc: 0.7764
Epoch 38/50
 - 365s - loss: 0.1216 - acc: 0.9609 - val_loss: 0.7018 - val_acc: 0.8164
Epoch 39/50
 - 361s - loss: 0.2121 - acc: 0.9375 - val_loss: 0.6654 - val_acc: 0.8293
Epoch 40/50
 - 366s - loss: 0.1329 - acc: 0.9609 - val_loss: 0.6297 - val_acc: 0.8320
Epoch 41/50
 - 362s - loss: 0.0843 - acc: 0.9727 - val_loss: 0.8634 - val_acc: 0.7683
Epoch 42/50
 - 356s - loss: 0.1678 - acc: 0.9209 - val_loss: 0.7204 - val_acc: 0.8281
Epoch 43/50
 - 366s - loss: 0.1444 - acc: 0.9570 - val_loss: 0.9738 - val_acc: 0.7461
```

Entraînement sur 10 classes : Principaux résultats

- ★ Entrainement s'arrête à 43 ème époque .

- ★ Meilleur performance obtenue à l’époque 23 , avec une valeur de loss égal 0.47 et une accuracy de 85% .

Programme Python

Programme Python:

Nous avons développé un programme python permettant de déterminer la race du chien en se basant sur les deux approches:

- ➔ L'approche Classique .
- ➔ L'approche CNN.

Conclusion :

- ➔ Dans ce projet , nous avons pu mettre en oeuvre l'avantage majeure de l'approche CNN par rapport à l'approche classique pour le cas de classification des images.

- ➔ La performance de notre algorithme pour l'approche CNN , reste insatisfaisant pour toutes les classes. Ainsi l'amélioration de cette algorithme est fortement recommandée pour améliorer les performances de notre modèle de classification .