Projet 7

Développer une preuve de concept

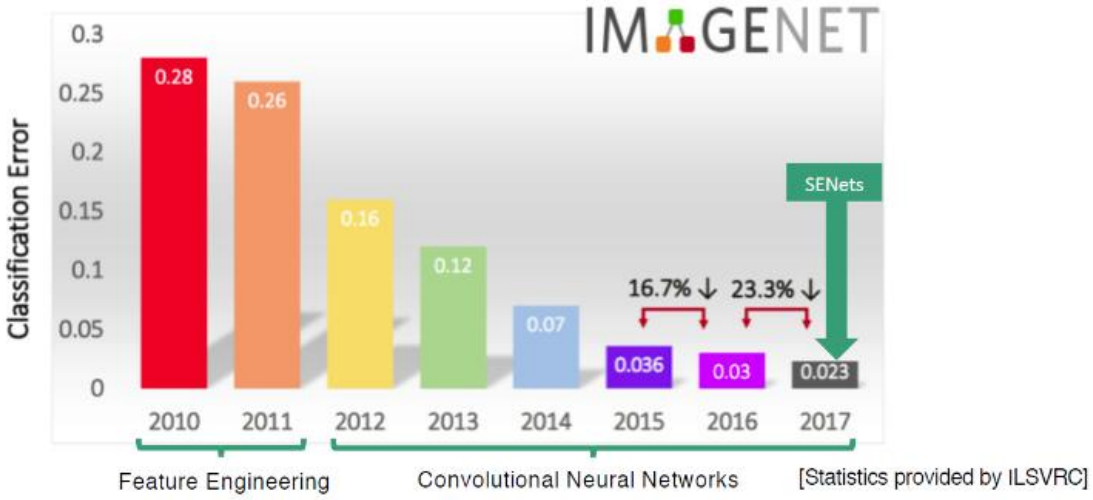
Plan de présentation

1. Introduction
2. Familles de modèles
3. Dataset et modèle de baseline
4. Implémentation de SE-ResNet50
5. Comparaison de méthodes et résultats
   - Keras vs. PyTorch
   - Xception vs. ResNet50 et SE-ResNet50
6. Conclusion
7. Bibliographie

1. INTRODUCTION

1. Introduction

- La vision par ordinateur est actuellement en expansion et son application s'étend dans des nouvelles domaines
- Ayant une solide expérience dans la recherche médicale, je suis particulièrement intéressée par les avancés de vision par ordinateur dans le milieu de la santé
- Les études récentes (2019) relatives à l'application de vision par ordinateur dans la recherche médicale :
  - Détection de cancer de sein [1]
  - Lecture de IRM de cerveau [2]
  - Détection de nodules pulmonaires avec tomodensitométrie (TDM) avec les modèles SE [3]
- Les modèles de vision par ordinateur, notamment les réseaux de neurones convolutifs ne cessent pas à s'améliorer. Depuis 2010, lors de la compétition ILSVRC, le taux d'erreur a diminué de 28 à 2.3 %
- C'est pour ces raisons que j'ai choisi d'approfondir mes connaissances dans ce domaine et revenir sur le projet 6 qui aborde utilisation de Deep Learning pour classifier automatiquement des images



Source : https://towardsdatascience.com/review-senet-squeeze-and-excitation-network-winner-of-ilsvrc-2017-image-classification-a887b98b2883

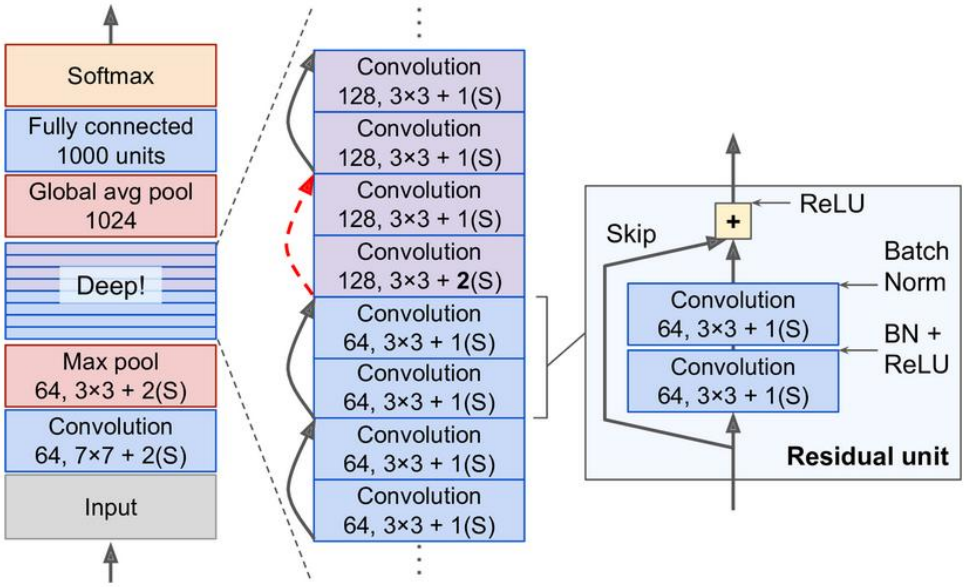
2. FAMILLES DE MODÈLES

2. Familles de modèles

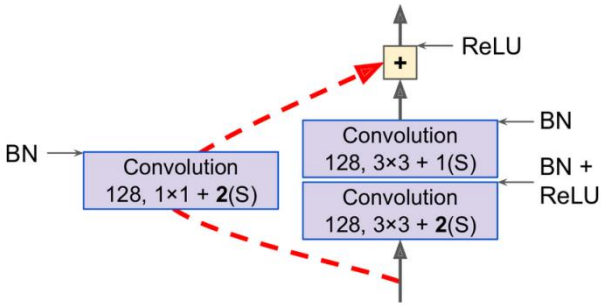
ResNet (Residual Network)

- Kaiming He et al. [4] a introduit le modèle qui a emporté la compétition ILSVCR en 2015
- Le modèle développe l'idée d'utiliser un grand nombre de couches avec un peu de paramètres
- Introduction « d'unité résiduelle » : Le signal qui rentre dans la couche est additionné à l'output de la couche située plus haut
- Quand la taille de output ne corresponde pas à la taille de input, le signal passe par « skip connexion »
- Variantes : ResNet34, ResNet50, ResNet152
- Le modèle ResNet50, appliqué dans le Projet 6, n'a pas été retenu à cause de tendance à surapprendre. Ce modèle nous servira comme une base de réseau « squeeze and excitation »

Architecture de ResNet



Skip connexion



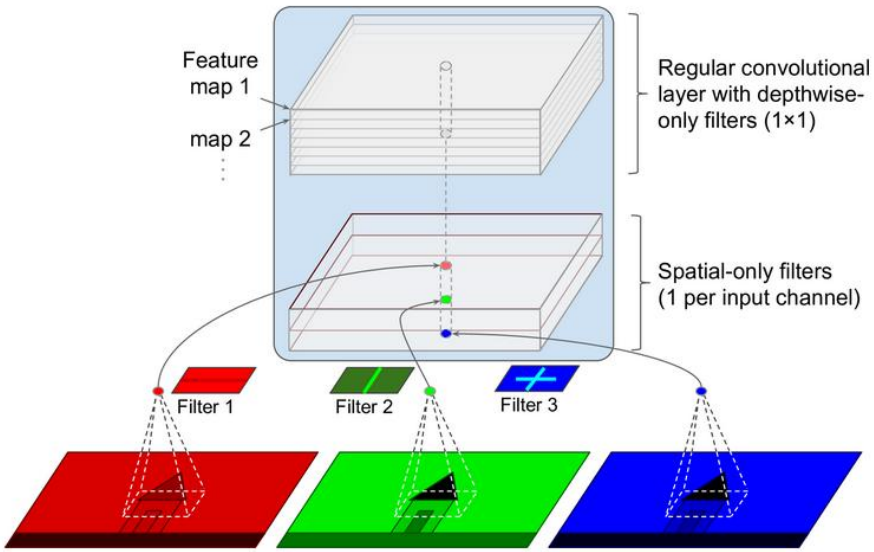
Source : Aurélien Géron : Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow

2. Familles de modèles

Xception (Extreme Inception)

- Variante d'Inception proposée en 2016 par François Chollet (auteur de Keras) [5]
- Il s'agit d'une combinaison de principes de GoogLeNet et ResNet. A la place de module « Inception », le modèle utilise des couches spécifiques appelées « Depthwise separable convolutional layers » : couche convolutionnelles séparables en profondeur
- Couches séparables : basé sur l'hypothèse que nous pouvons modéliser les structures spatiales (ex. oval) et structures multicanal (par exemple bouche + yeux + nez = visage) séparément
- Ce modèle a atteint la meilleure performance dans le Projet 6. Il nous servira donc comme le modèle de baseline

Depthwise separable convolutional layer



Source : Aurélien Géron : Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow

2. Familles de modèles

SENet (Squeeze and Excitation Networks)

- Introduit par Jie Hu et col. [6], gagnant de compétition ILSVRC 2017

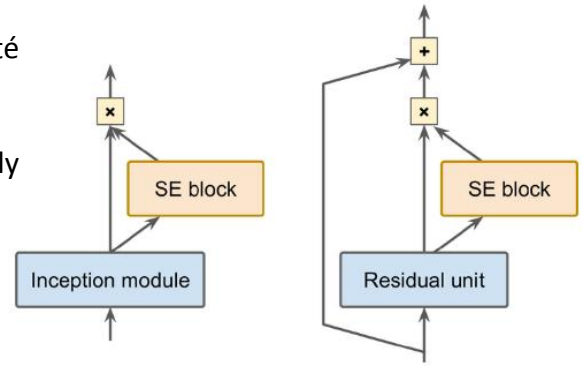
- Ajoute le module « squeeze and excitation » après chaque module Inception ou Unité résiduelle (ResNet)

- Module est composé par une couche Global average pooling et deux couches fully connected

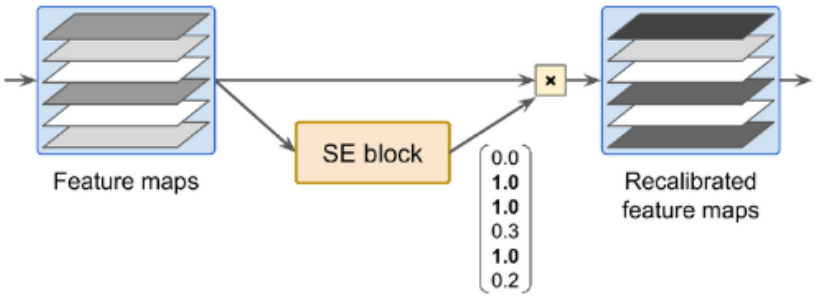
- **But** : re-calibrer les poids de feature maps

- **Pourquoi on souhaite re-calibrer les poids** :
  - Le modèle apprend des liens entre des éléments dans les images.
  - Exemple : Les yeux, le nez et la bouche sont souvent sur la même photo
  - Si nous avons une forte activation des maps qui représentent les éléments nez + yeux, mais une faible activation de map qui corresponde à la bouche, le modèle va booster la feature map correspondante à la bouche

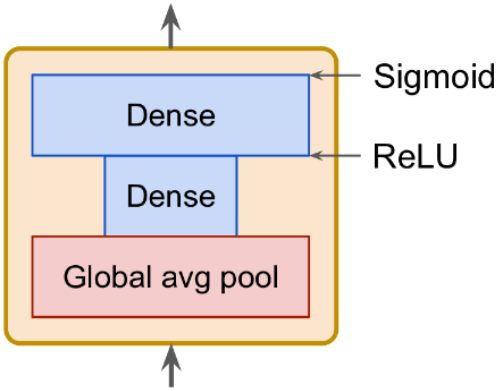
Module SE-Inception et SE-ResNet



Le module SE re-calibre feature maps à la sortie



Architecture de module SE



Source : Aurélien Géron : Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow

3. DATASET ET MODÈLE DE BASELINE

3. Dataset et modèle de baseline

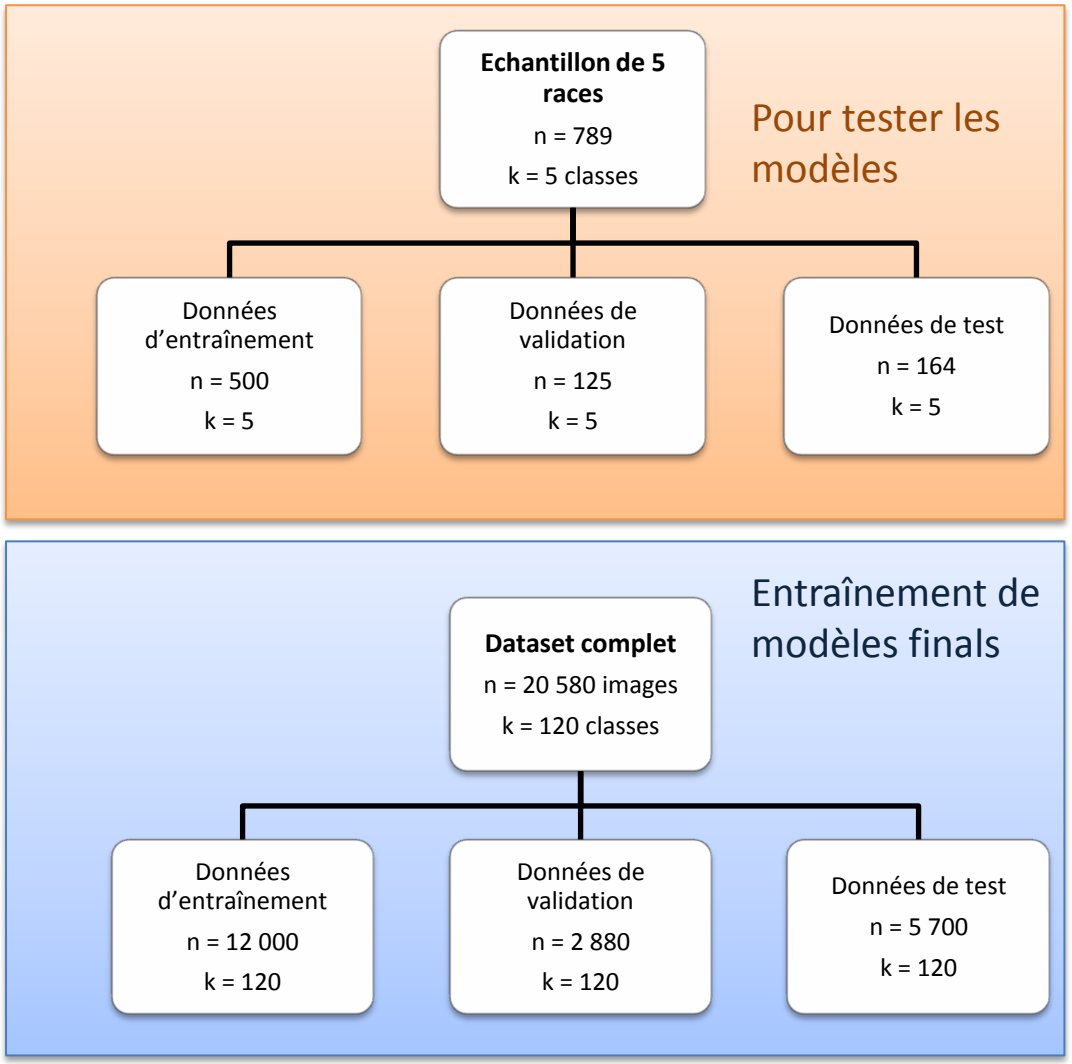
- Nous disposons d'une photothèque de chiens, Stanford Dogs Dataset qui contient :
  - 120 catégories (races de chien)
  - 20 580 images

- La photothèque est relativement petite pour que nous puissions entraîner notre propre modèle à partir de scratch

⇒ Méthode retenue dans le projet 6 : Transfer Learning

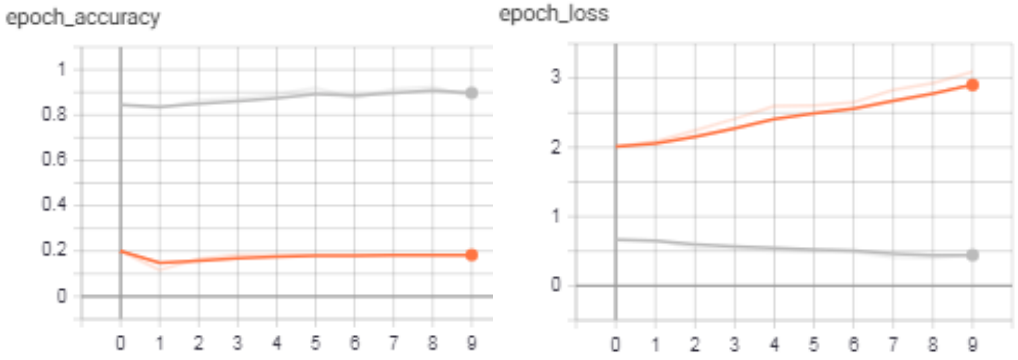
⇒ Utilisation de modèles entraînés sur le dataset ImageNet (1.2 millions de photos)

Data Flow :



3. Dataset et modèle de baseline
Tests sur un échantillon de 5 races

ResNet50
Batch size = 32



- Précision : 26.83 %

- **Conclusion :**
Surapprentissage

Name	Smoothed	Value	Step	Time	Relative
resnet50_2_run_2019_12_30-11_34_36/train	0.8972	0.88	9	Mon Dec 30, 13:00:18	22m 53s
resnet50_2_run_2019_12_30-11_34_36/validation	0.1828	0.1833	9	Mon Dec 30, 13:00:18	22m 53s

Xception
Batch size = 32

- Précision : 98.17%

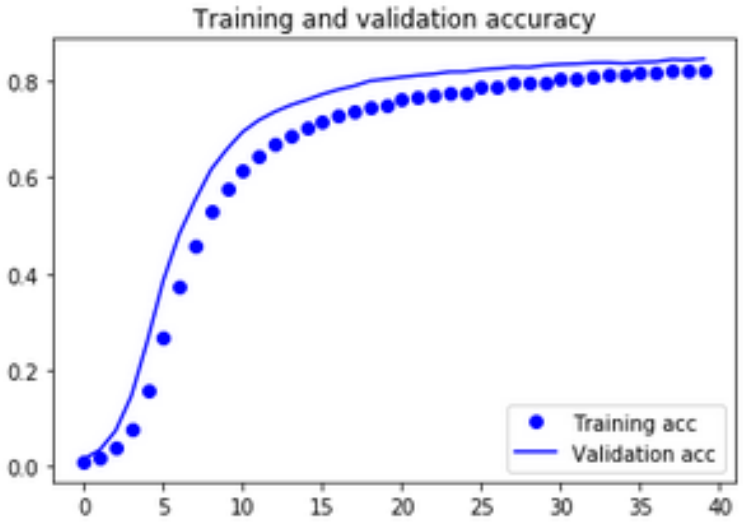
Conclusion :
Le modèle a moins de tendances à surapprendre et une meilleure précision même avec seulement 10 epochs

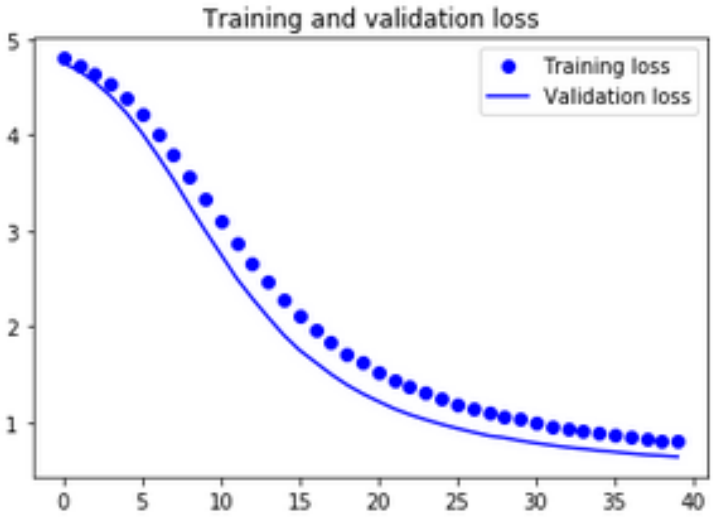
Name	Smoothed	Value	Step	Time	Relative
xception2_run_2019_12_30-14_27_37/train	0.8869	0.896	9	Mon Dec 30, 16:37:46	1h 2m 53s
xception2_run_2019_12_30-14_27_37/validation	0.9166	0.92	9	Mon Dec 30, 16:37:46	1h 2m 53s

3. Dataset et modèle de baseline
Modèle final

- Le modèle final était exécuté à l'aide de AWS Sagemaker (CPU avec 61Go + GPU avec 16Go de RAM)

- Résultats de fine-tuning partiel (nous avons conservés les poids de 7 premières couches – block 1 du modèle)





Performance finale :
- Temps d'exécution : 2h 0min
- Précision : 85.75 %

4. IMPLÉMENTATION DE SE-RESNET50

4. Implémentation de SE-ResNet50

- Le modèle original entraîné sur le dataset ImageNet est mise à disposition par les auteurs au format .caffe
  - La bibliothèque caffe, écrite en C++ est paramétré à la base pour le système d'exploitation Linux
  - Des conversion pour Windows 10 existent, mais l'implémentation n'et pas simple

- Lien vers les modèles au format PyTorch [7] :
  - ResNet50 et SE-ResNet50 entraînés sur ImageNet à l'aide de 8 GPUs
  - Précision test (top1) :
    - ResNet50 : 76.15 %
    - SE-ResNet : 77.06 %

- Problématique :
  - Implémenter le modèle à l'aide d'une nouvelle bibliothèque dont je ne suis pas familière – le projet 6 étant fait à l'aide de Tensorflow et Keras

5. COMPARAISON DE MÉTHODES ET RÉSULTATS

5. Comparaison de méthodes et résultats
Keras vs. PyTorch

Keras :

- Bibliothèque open source écrit initialement par François Chollet
- High-level API, qui permet de développer et faire tourner des modèles de Deep Learning avec une syntaxe simple
- Plus une « boîte noire », mais facile pour commencer
- Utilisé par les entreprises, facile à déployer

PyTorch :

- Bibliothèque open source développée par Facebook
- Low-level API, la programmation est basée sur les opérations avec des tensors (matrices multidimensionnelles)
- Flexibilité, possibilité de personnaliser les fonctions et modèles facilement
- Utilisé en milieu académique, les nouveaux algorithmes sont souvent entraînés d'abord avec PyTorch

Exemple d'utilisation de méthode fit_generator de Keras pour entraîner le modèle :

```
# Measure the time of execution
start_time = time.time()

# Set epochs
epochs = 40

# Train the model
history1 = my_CNN.fit_generator(
    train_data_gen,
    validation_data=valid_data_gen,
    steps_per_epoch=np.ceil(train_size/batch_size),
    validation_steps=np.ceil(valid_size/batch_size),
    epochs=epochs,
    callbacks=[checkpoint_cb, early_stopping_cb],
    verbose=1)

# Print the time of execution
print("--- " + str(int((time.time() - start_time)//3600)) + " hours "
      + str(int(((time.time() - start_time)%3600)//60)) + " mins ---")
```

5. Comparaison de méthodes et résultats
Keras vs. PyTorch

Exemple de fonction PyTorch pour entraîner le modèle [8] :

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

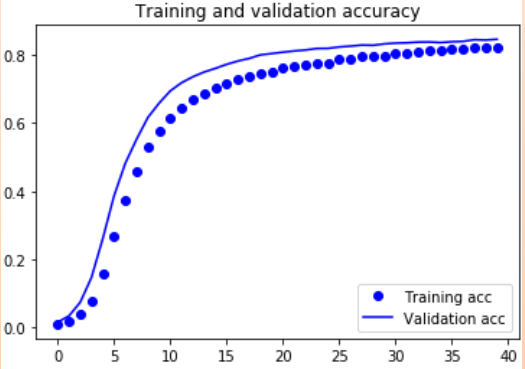
```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    # Add empty lists to stock the values
    train_loss = []
    train_acc = []
    valid_loss = []
    valid_acc = []

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'valid']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            # Upgrade the loss and accuracy values:
            if phase == 'train':
                train_loss.append(epoch_loss)
                train_acc.append(epoch_acc)
            else:
                valid_loss.append(epoch_loss)
                valid_acc.append(epoch_acc)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'valid' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model, train_loss, train_acc, valid_loss, valid_acc
```

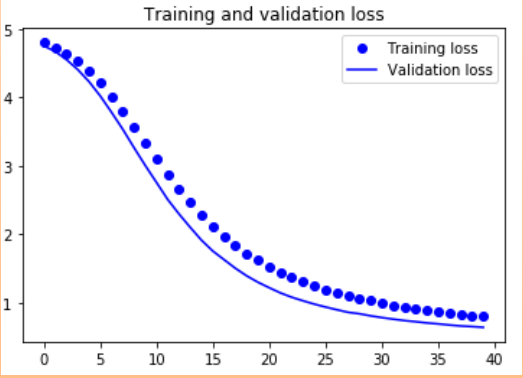
5. Comparaison de méthodes et résultats
Xception vs. ResNet50 et SE-ResNet50

Résultats d'exécution avec AWS Sagemaker, 40 epochs

Xception (Keras)
- *Batch size = 32*
- *Fine tuning partiel*
- *Callback Early stopping*

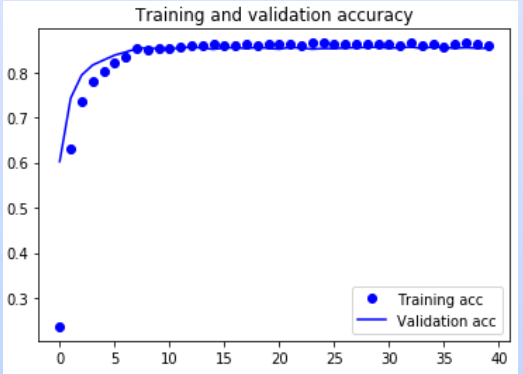
• Temps d'exécution : 2h 0min
• Précision : 86 %

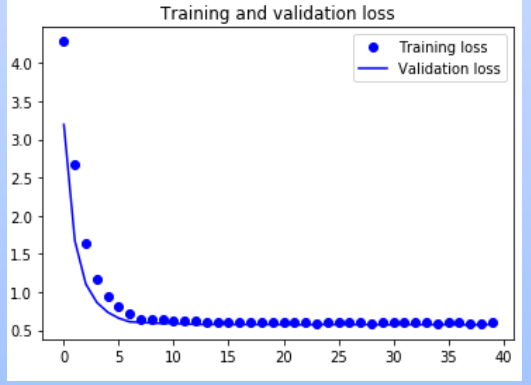




ResNet50 (PyTorch)
- *Batch size = 100*
- *Extraction de features*
- *No Early stopping*

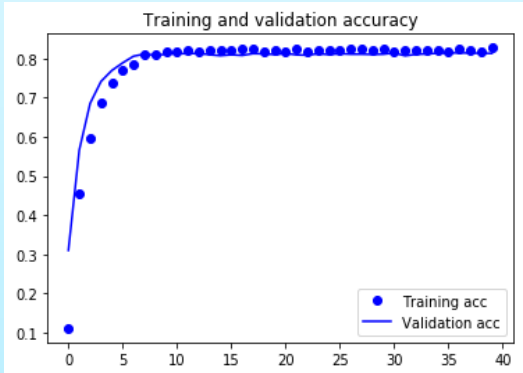
• Temps d'exécution : 48 min
• Précision : 85 %

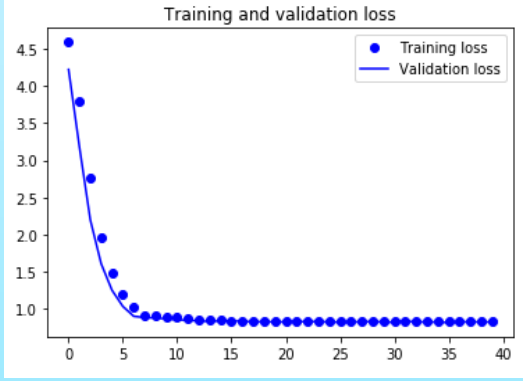




SE-ResNet50 (PyTorch)
- *Batch size = 100*
- *Extraction de features*
- *No Early stopping*

• Temps d'exécution : 51 min
• Précision : 82 %





6. CONCLUSION

6. Conclusion

- Le projet m'a permis de :
  - Me familiariser davantage avec les algorithmes CNN récents
  - Découvrir une nouvelles API PyTorch

- Les modèles testés dans ce projet ne m'ont pas permis d'améliorer les résultats de performance de modèle de baseline en termes de précision, mais ils sont plus performants en termes de vitesse d'apprentissage

- Pour améliorer les résultats et avoir une meilleure base de comparaison :
  - Exécuter le modèle de baseline avec le même batch-size (32 vs. 100)
  - Utiliser la même API pour exécuter les modèles
  - Paramétrer une fonctionnalité « Early stopping » dans PyTorch
  - Paramétrer et tester fine-tuning partiel dans PyTorch

7. BIBLIOGRAPHIE

7. Bibliographie

Articles :

[1] Wu, N. et al. Deep neural networks improve radiologists' performance in breast cancer screening. *IEEE Trans. Med. Imaging* https://doi.org/10.1109/TMI.2019.2945514 (2019).

[2] Dalca, A. et al. Learning Conditional Deformable Templates with Convolutional Networks. *NeurIPS.* http://papers.nips.cc/paper/8368-learning-conditional-deformable-templates-with-convolutional-networks.pdf (2019).

[3] Gong, L., Jiang, S., Yang, Z. *et al.* Automated pulmonary nodule detection in CT images using 3D deep squeeze-and-excitation networks. *Int J CARS* **14,** 1969–1979 (2019) doi:10.1007/s11548-019-01979-1

[4] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV] (2015).

[5] Chollet, F. Deep Learning with Depthwise Separable Convolutions. arXiv:1610.02357 [cs.CV] (2016).

[6] Jie Hu and Li Shen and Samuel Albanie and Gang Sun and Enhua Wu. Squeeze-and-Excitation Networks. arXiv:1709.01507 [cs.CV] (2017).

Blogs :

https://towardsdatascience.com/review-senet-squeeze-and-excitation-network-winner-of-ilsvrc-2017-image-classification-a887b98b2883

Livres :

Aurélien Géron : Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow

Liens :

[7] https://github.com/moskomule/senet.pytorch

[8] https://pytorch.org/tutorials/