Formation Machine Learning

Projet 7
Développez une preuve de concept

Février 2021

Maud Charbonneau

Sommaire

Introduction

- Le monde informatique évolue constemment, il faut savoir d'adapter en s'informant sur les nouveaux modèles existants.

- Les réseaux de neurones convolutifs ne cessent de s'améliorer.

- J'ai choisi de reprendre le projet 6 - Classez des images à l'aide d'algorithmes de Deep Learning:
  - Données: Stanford Dog Dataset
  - Meilleur modèle du projet sert de baseline

- Ce projet va me permettre de découvrir de nouveaux réseaux de neurones convolutifs ainsi que mieux comprendre leur fonctionnement.

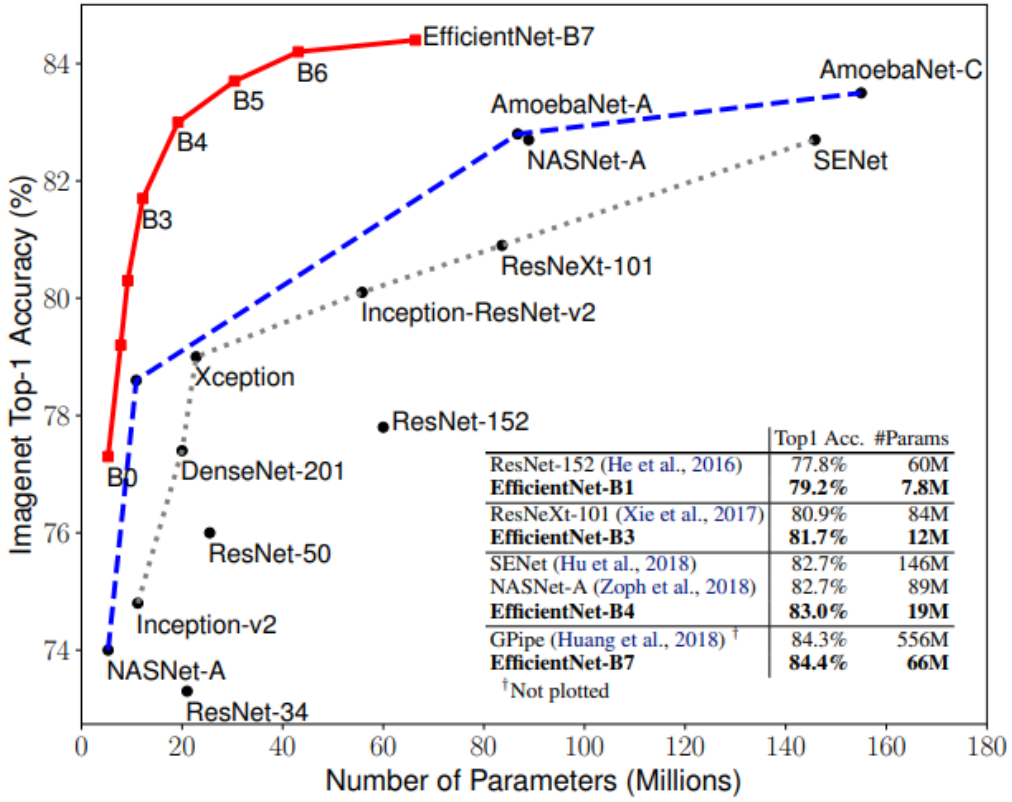
Etat de l'art: EfficientNet

EfficientNet

- Groupe de réseau de neurones convolutifs: EfficientNet-B0 --> EfficientNet-B7
  - Projet: EfficientNet-B0 --> EfficientNet-B4
- *International Conference on Machine Learning* - juin 2019
- Très performant sur différentes données

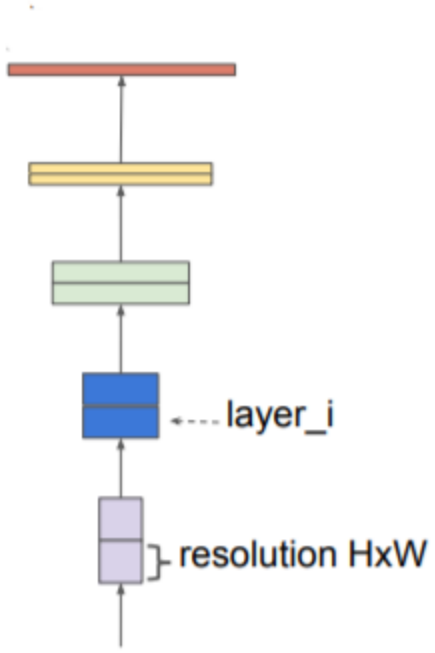
Compound method scaling

- Augmenter les 3 dimensions proportionnellement

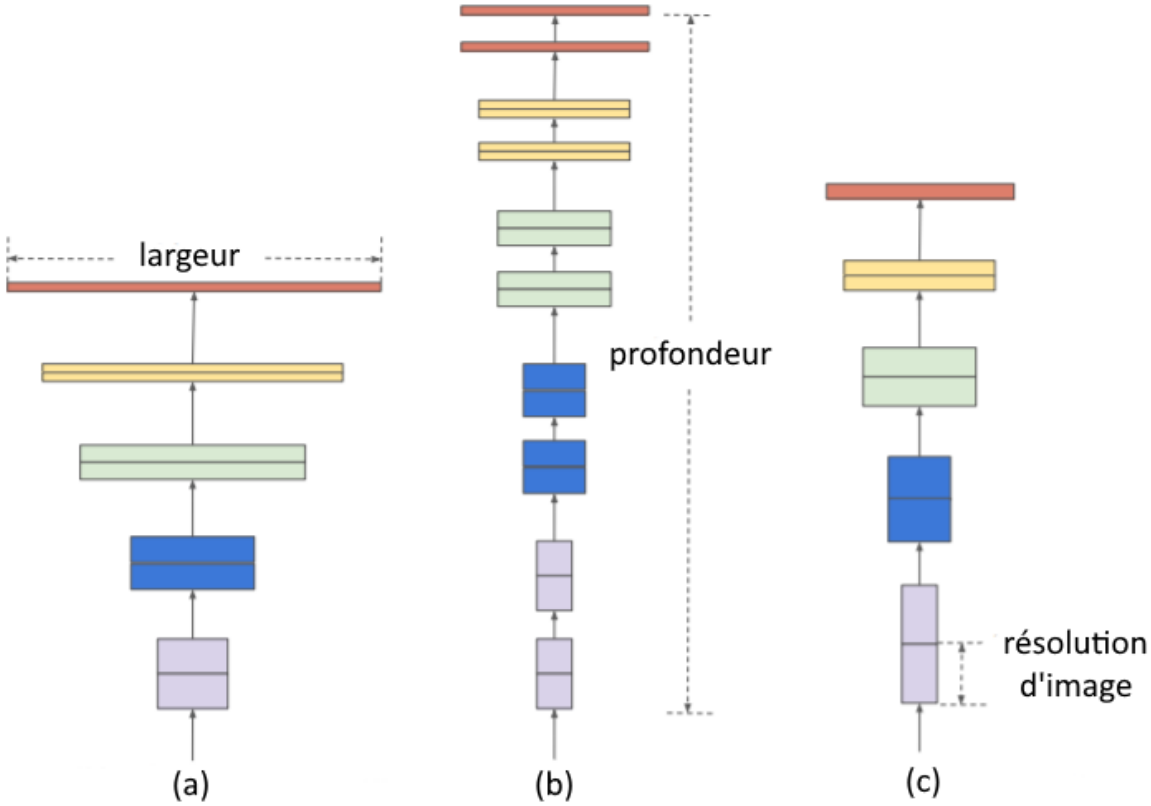


Etat de l'art: EfficientNet

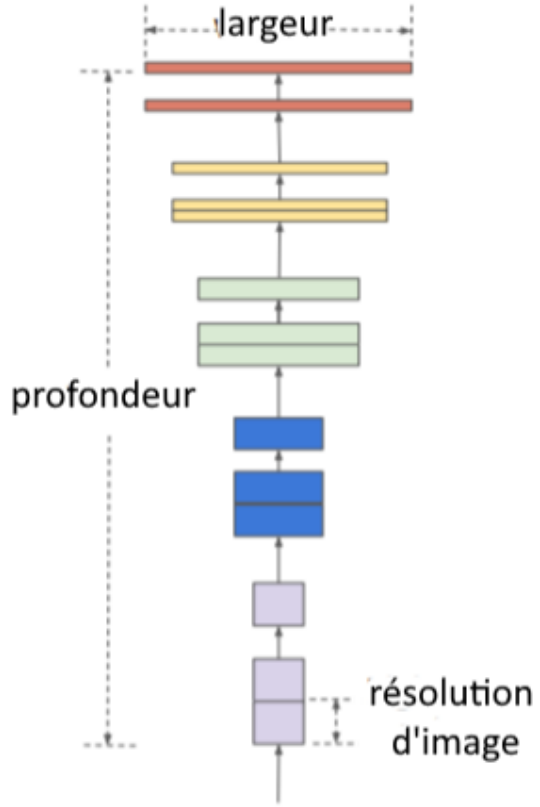
- Augmenter des dimensions pour améliorer la performance
- Technique communément utilisée : Augmentation arbitraire une ou plusieurs des 3 dimensions
- Technique de EfficientNet: Augmenter les trois dimensions de manière proportionnelle



Architecture d'un CNN



Augmentation de différentes dimensions



Augmentation des trois dimensions

Jeu de données et modèle de baseline

Jeu de données:

Stanford Dog Dataset: 120 classes

- 20 580 images

Train et validation: 120 classes

- Train: 9609 images
- Validation: 2401 images

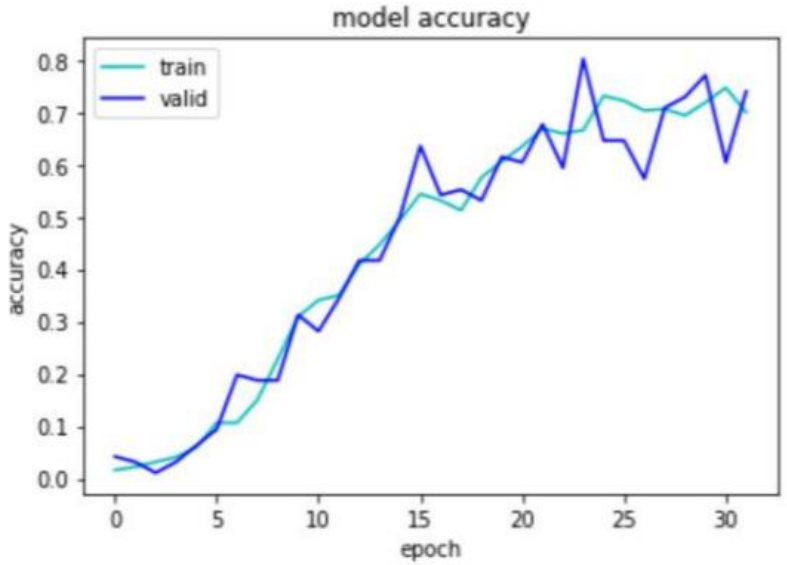
Test: 120 classes

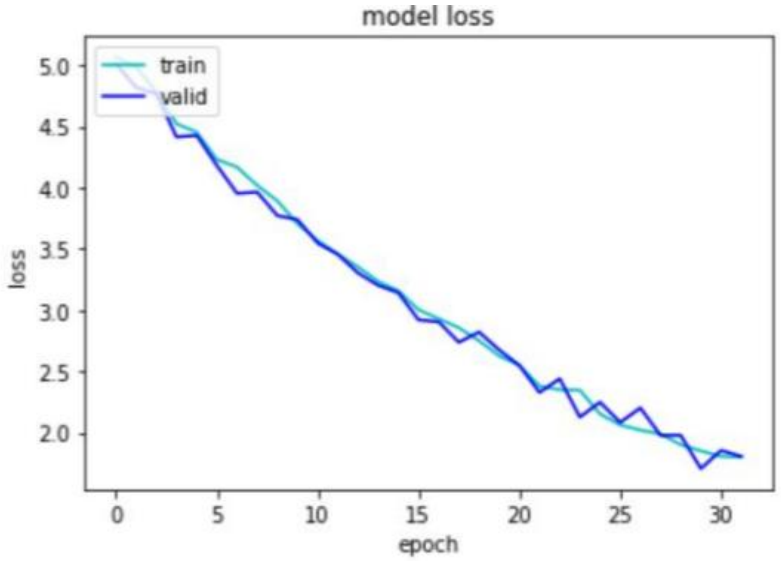
- 8580 images

Modèle de baseline:

- InceptionV3 : troisième édition du Google's Inception Convolutional Neural Network
- Extraction de features

Précision train: 85,94 %
Précision test: 79,33 %

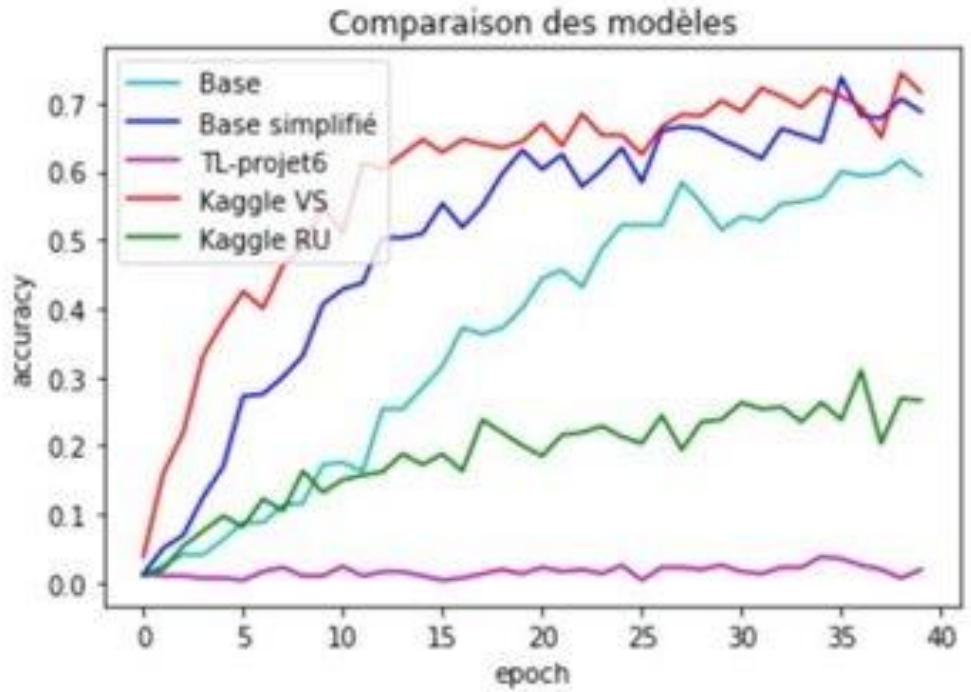


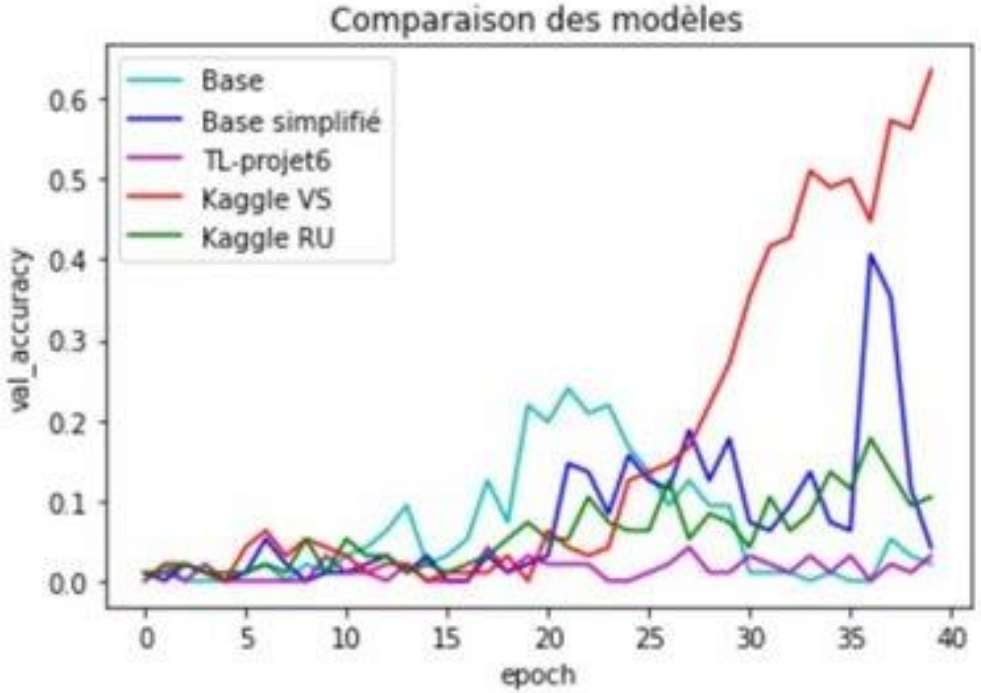


EfficientNet: implémentation et paramétrage

Transfert Learning

- Différentes architectures (modèles plus détaillés: voir rapport)
- Kaggle RU et Kaggle VS: fine tuning partiel avec différents initialiseurs
- Epochs = 40 et batch size = 40
- Optimiseur = Adam et learning rate = 0,0001
- EfficientNet-B0





Meilleurs performances: Kaggle VarianceScaling:

- précision ≃ 60%

EfficientNet: implémentation et paramétrage

Kaggle VarianceScaling

```
def tl_effnet_1(model, nb_classes):
  initializer = VarianceScaling(scale=0.1, mode='fan_in', distribution='uniform')
  x = model.output

  x = GlobalAveragePooling2D()(x)
  # x = Dropout(rate = .2)(x)
  x = BatchNormalization()(x)
  x = Dense(1280, activation='relu',  kernel_initializer=glorot_uniform(3), bias_initializer='zeros')(x)
  # x = Dropout(rate = .2)(x)
  x = BatchNormalization()(x)
  predictions = Dense(nb_classes, activation='softmax', kernel_initializer=initializer, bias_initializer='zeros')(x)

  new_model = tf.keras.Model(inputs=model.input, outputs=predictions)

  #for layer in new_model.layers:
  #   layer.trainable = True
  for layer in new_model.layers[-2:]:
      layer.trainable = True
  new_model.summary()
  return new_model
```

Modèle choisi

EfficientNet: implémentation et paramétrage

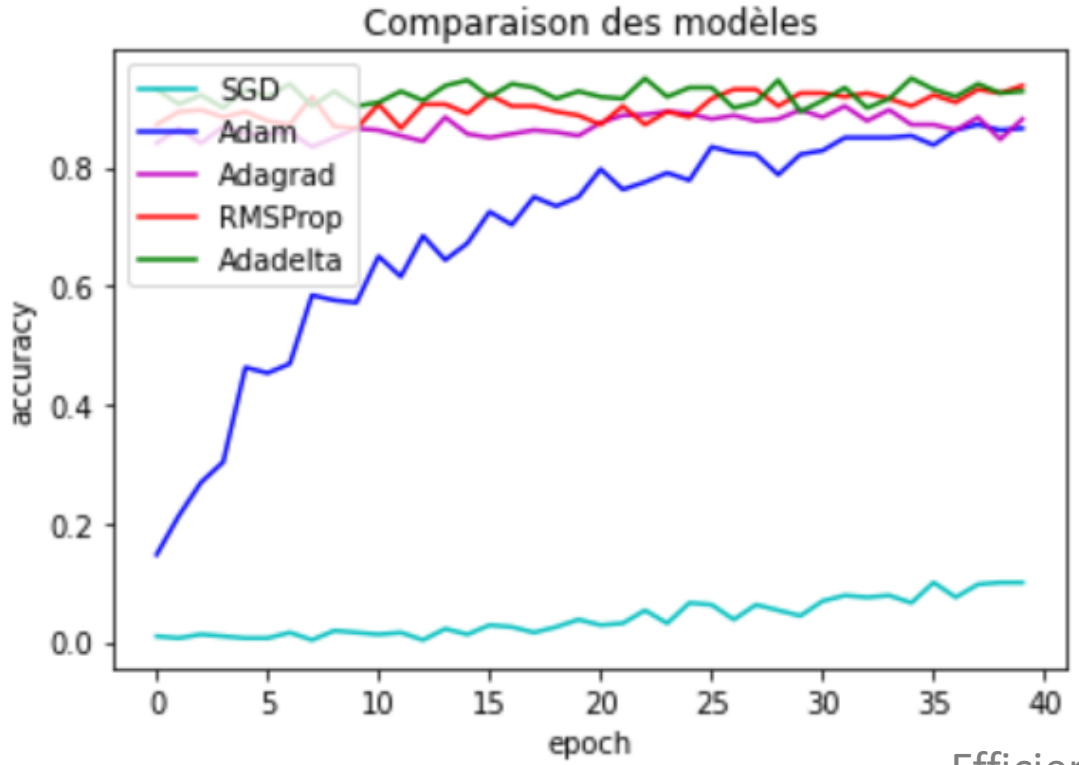
Paramétrage:

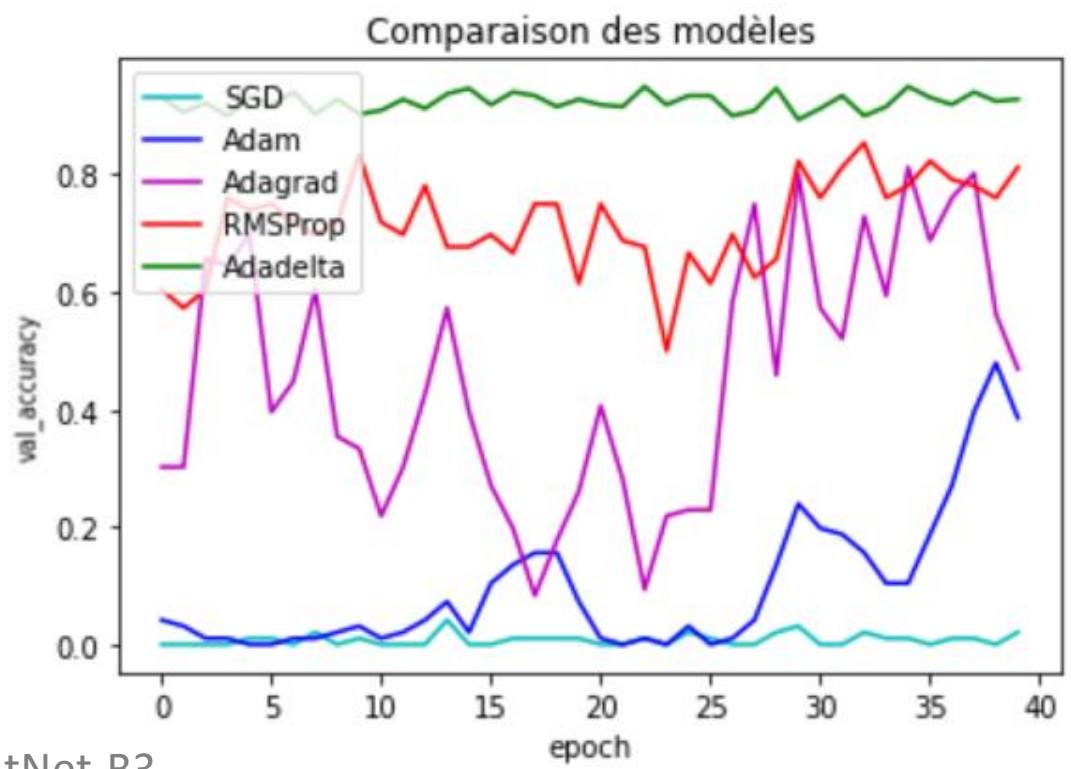
Optimiseurs :

- 5 optimiseurs
- Epoch = 40
- Batch size = 40

Résultat:

- Adadelta et RMSprop meilleure précision
- Uniquement Adam apprend





EfficientNet-B3

EfficientNet: implémentation et paramétrage

Paramétrage: EfficientNet-B0 --> EfficientNet-B4

- Nombre d'epochs:
  - 40, 80, 100
- Taille de batch :
  - 40, 80
- Learning rate :
  - 1e-2, 1e-3, 1e-4, 1e-6
- Callbacks
  - EarlyStopping
  - ReduceLROnPlateau

Meilleurs paramètres:

- 80 epochs
- Batch size = 40 ou 80
- Learning rate = 0,0001
- Callbacks: ReduceLROnPlateau

Analyse des résultats

Résultats pour 40 epochs et 40 de batch size

Modèle	Optimiseurs	Learning rate	Précision	Temps en s.
EficientNet-B0	Adam	0.0001	0,2705	207
EficientNet-B1	Adam	0.0001	0,3304	287
EficientNet-B2	Adam	0.0001	0,3285	299
EficientNet-B3	Adam	0.0001	0,5286	368
EficientNet-B4	Adam	0.0001	0,3573	463

Augmentation du nombre d'epoch:
pas d'amélioration de la précision

Meilleur précision:

1. EfficientNet-B3
2. EfficientNet-B4

EfficientNet-B4 > EfficientNet-B3 ?

- Variantes plus grandes ≠ meilleures performances
  - moins de données
  - moins de classe
- Plus EfficientNet grand, plus **difficile à paramétrer.**

Analyse des résultats

	Modèle	Epoch	Batch size	Précision train	Précision validation
A	EfficientNet-B3	80	40	0,9132	0,8281
B	EfficientNet-B3	100	80	0,8717	0,8906
C	EfficientNet-B4	100	40	0,8511	0,8750
D	EfficientNet-B4	100	80	0,8312	0,8750

Perte et précision sur données test :

A: Résultat: [0.5082215070724487, 0.8503900170326233]

B: Résultat: [0.7371186017990112, 0.824077308177948]

C: Résultat: [0.7219782471656799, 0.8211666345596313]

D: Résultat: [0.6364232897758484, 0.8269879817962646]

- 2 modèles sélectionnés

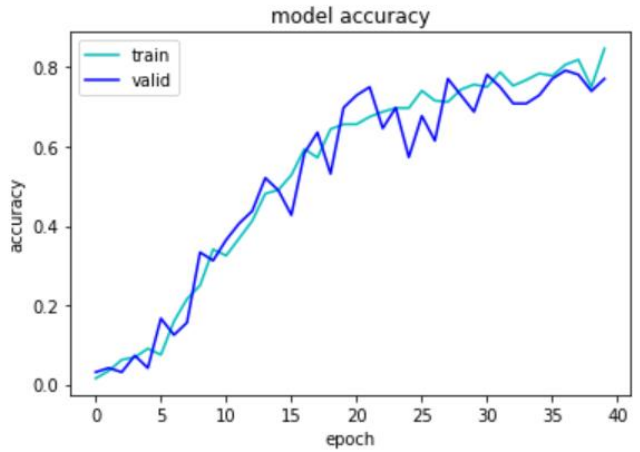
Suite:

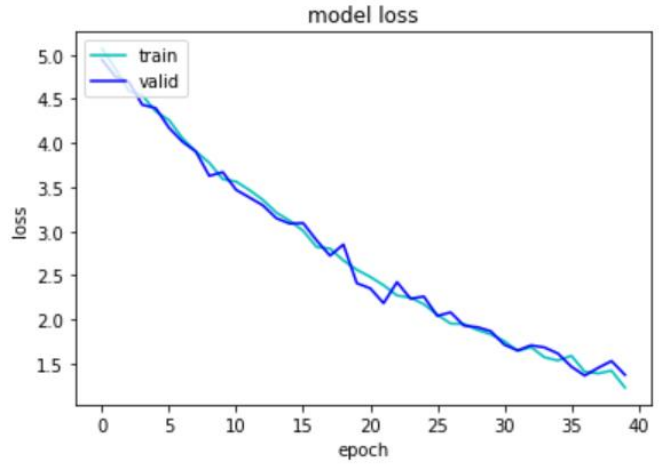
- Observation des courbes d'apprentissage
- Comparaison avec la précision baseline

Comparaison avec la baseline

Baseline: InceptionV3

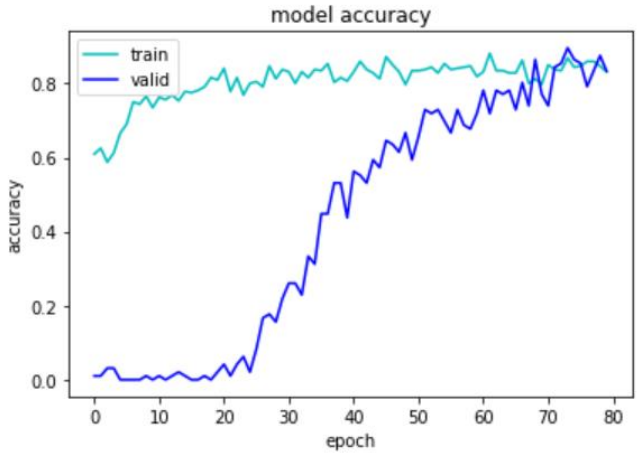
- Batch size = 40
- Extraction de features
- No callbacks
- Précision = 79,33%

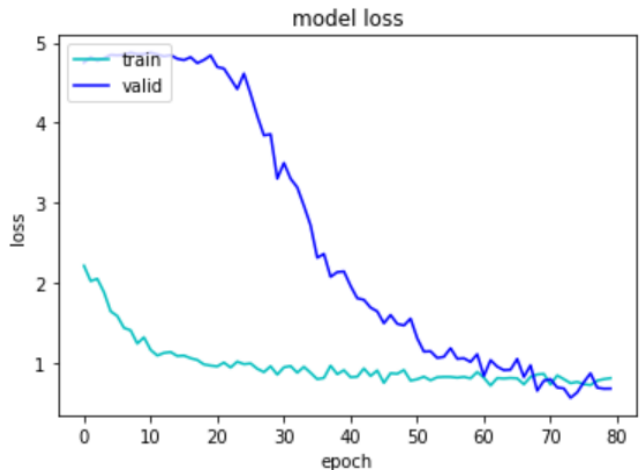




EfficientNet-B3

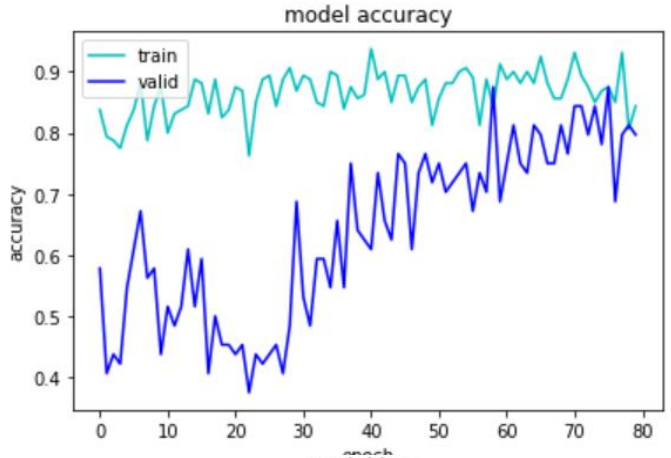
- Batch size = 40
- Fine tuning partiel
- ReduceLROnPlateau
- Précision = 85,04%

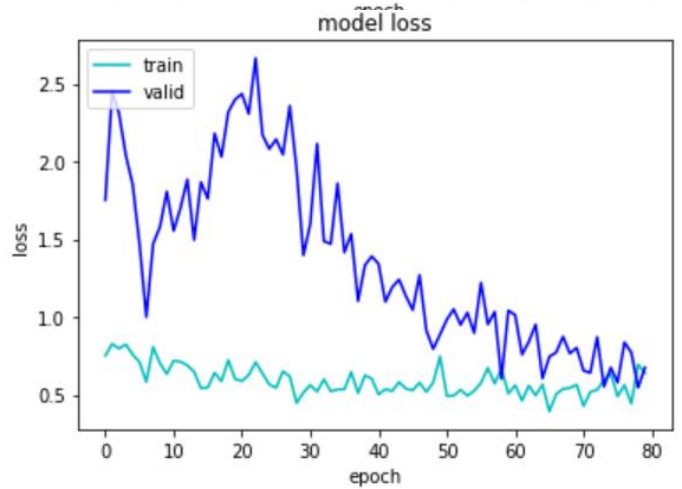




EfficientNet-B4

- Batch size = 80
- Fine tuning partiel
- ReduceLROnPlateau
- Précision = 82,70%

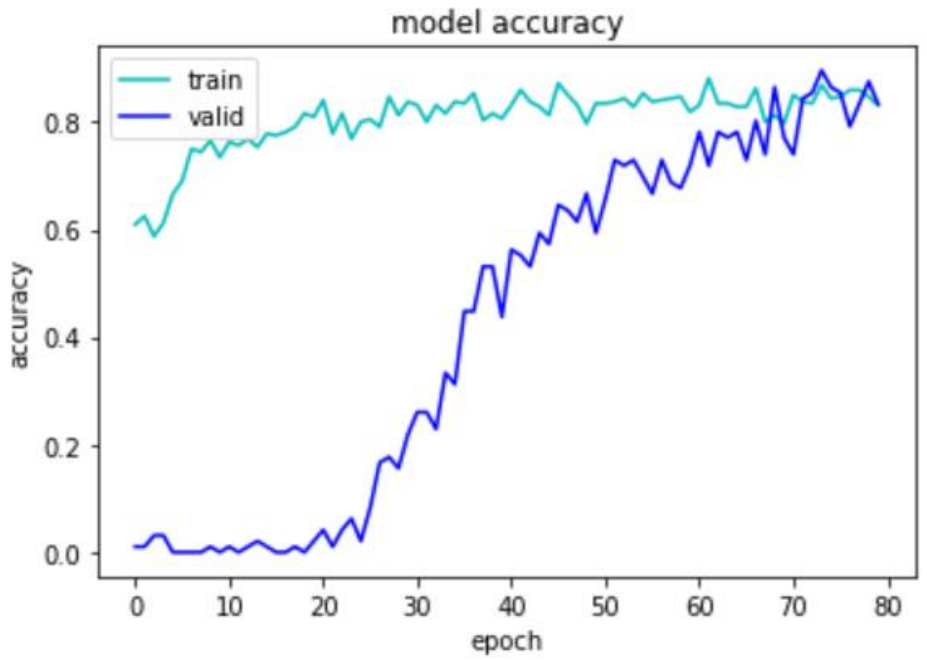


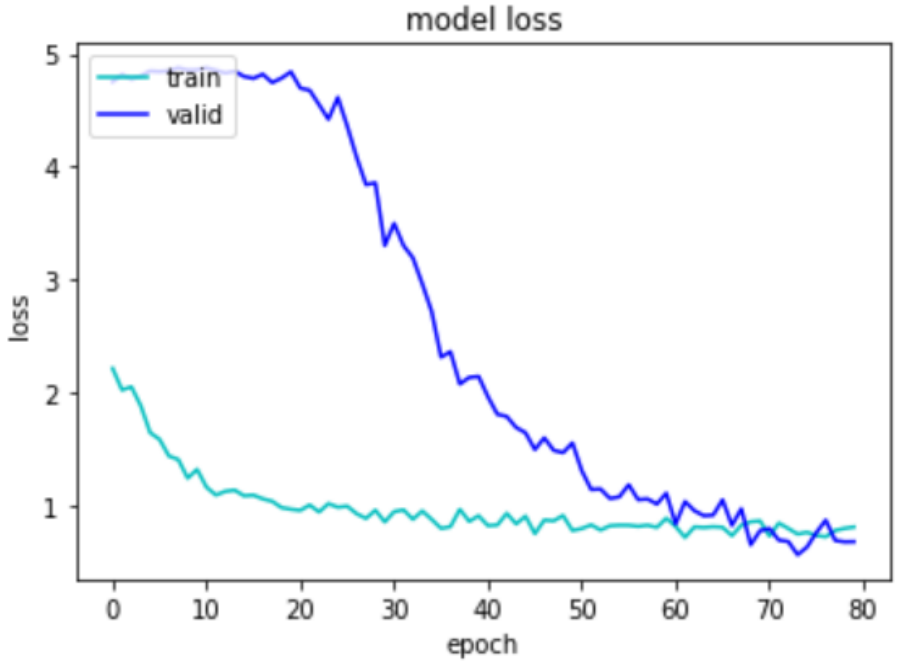


Meilleur modèle EfficientNet

Le meilleur modèle EfficientNet:
EfficientNet-B3 – Kaggle VarianceScaling – Optimiseur: Adam – Batch size: 40

Précision: 91,32%
Temps: 36,05 min



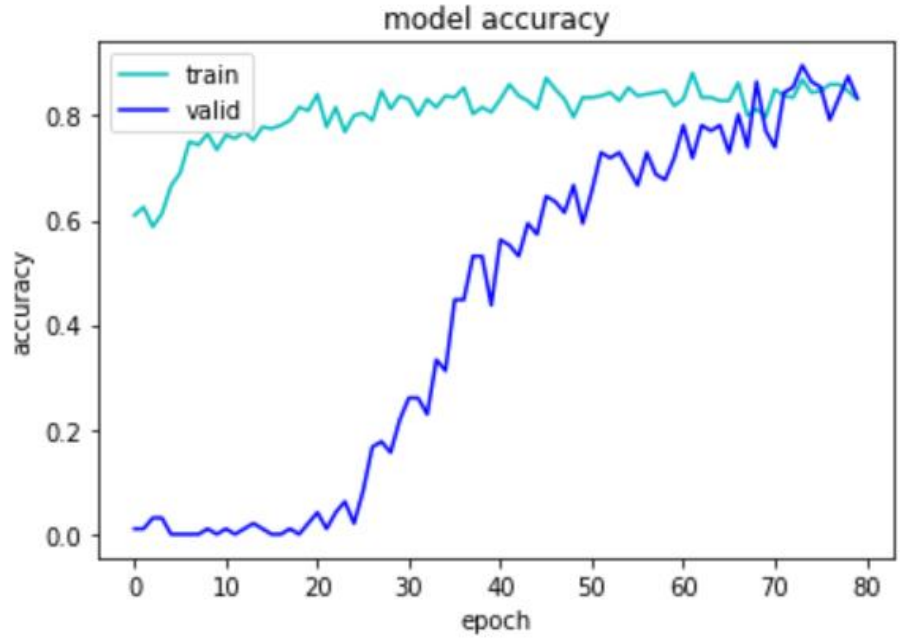


Evaluation sur data set de test: `Résultat: [0.5082215070724487, 0.8503900170326233]`

Précision = 85,03%

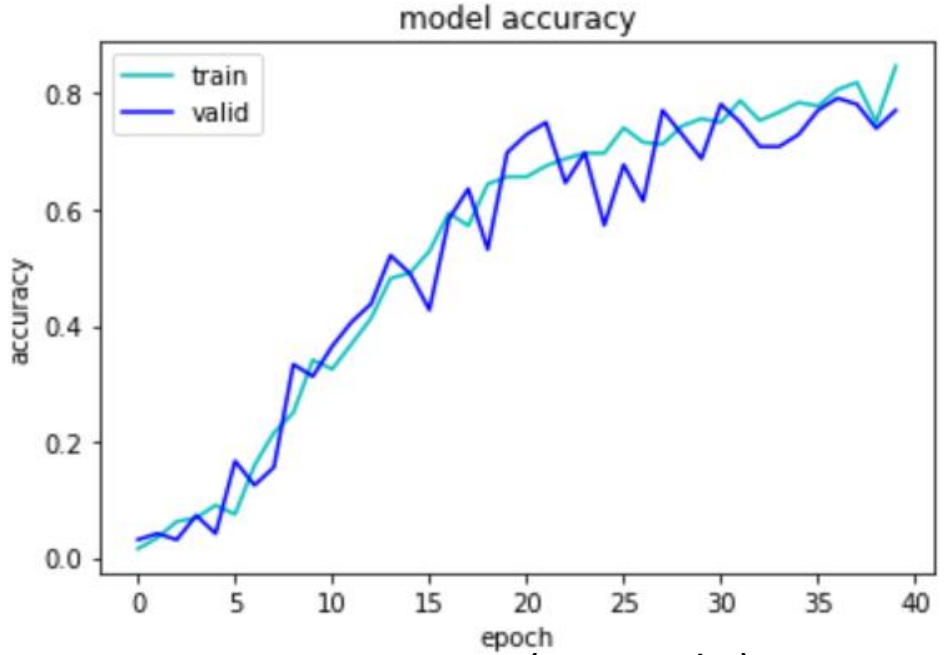
Conclusion

- Avec EfficientNet-B3 on obtient une meilleure précision que le modèle baseline InceptionV3: 85,03% contre 79,33%.
- Les courbes d'apprentissages de EfficientNet-B3 ne sont pas très stables.



EfficientNet-B3 (80 epochs)

Précision = 85,03%



InceptionV3 (40 epochs)

Précision = 79,33%

Conclusion

- Nouvelles modifications pouvant améliorer la performances de EfficientNet:
  - Augmentation linéaire de la valeur de Dropout lors du fine-tuning partiel
  - Définir drop_connect_rate pour controler la valeur du drop out
  - Freeze la couche BatchNormalization
  - Petit batch size pour accroitre la précision de la validation

- Dans le plan prévisionnel de travail, je voulais tester également EfficientNet-B5. Ce modèle ayant un très grand nombre de paramètres (35 millions sans transfert learning), je n'ai pas pu l'éxecuter.

- Ce projet m'a permis:
  - D'améliorer mes compétences de recherches
  - De me familiariser encore plus avec les réseaux de neurones convolutifs

Ressources trouvables dans le rapport