Formation Machine Learning

Projet 6

Classez des images à l'aide d'algorithmes de Deep Learning

Maud Charbonneau

Sommaire

1. Présentation du sujet et des données
2. Cleaning et exploration
3. Création d'un CNN
    1. From scratch
    2. Transfert Learning
4. Comparaison des modèles
5. Meilleur modèle
6. API
7. Conclusion

Présentation du sujet

- Partenaire: association de protection des animaux

Problématique:

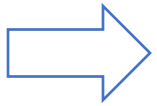
- Souhaite référencer images des animaux pensionnaires
- Base de données: trop long à faire manuellement




Mission: créer un algorithme capable de classer les images en fonction de la race du chien présent sur l'image.

Présentation des données

Base de données initiale: Stanford Dogs Dataset
Nombre de races: 120
Nombre d'images: 20 580



Nombre de races: 120	
Base train 12 000 images	Base test 8580 images

Nombre de races: 120	
Base train 9609 images	Base validation 2401 images

Utilisé pour améliorer les modèles

Nombre de races: 5	
Base train 400 images	Base test 100 images

Utilisé pour tester différents modèles

Cleaning et exploration

Data-augmentation

- Applique modifications sur batch d'image aléatoirement
- Augmente diversité des données sans risquer sur-apprentissage : pas de données supplémentaires

  <u>Exemples:</u>

  - Modification de la luminosité, rotation de l'image, flip de l'image

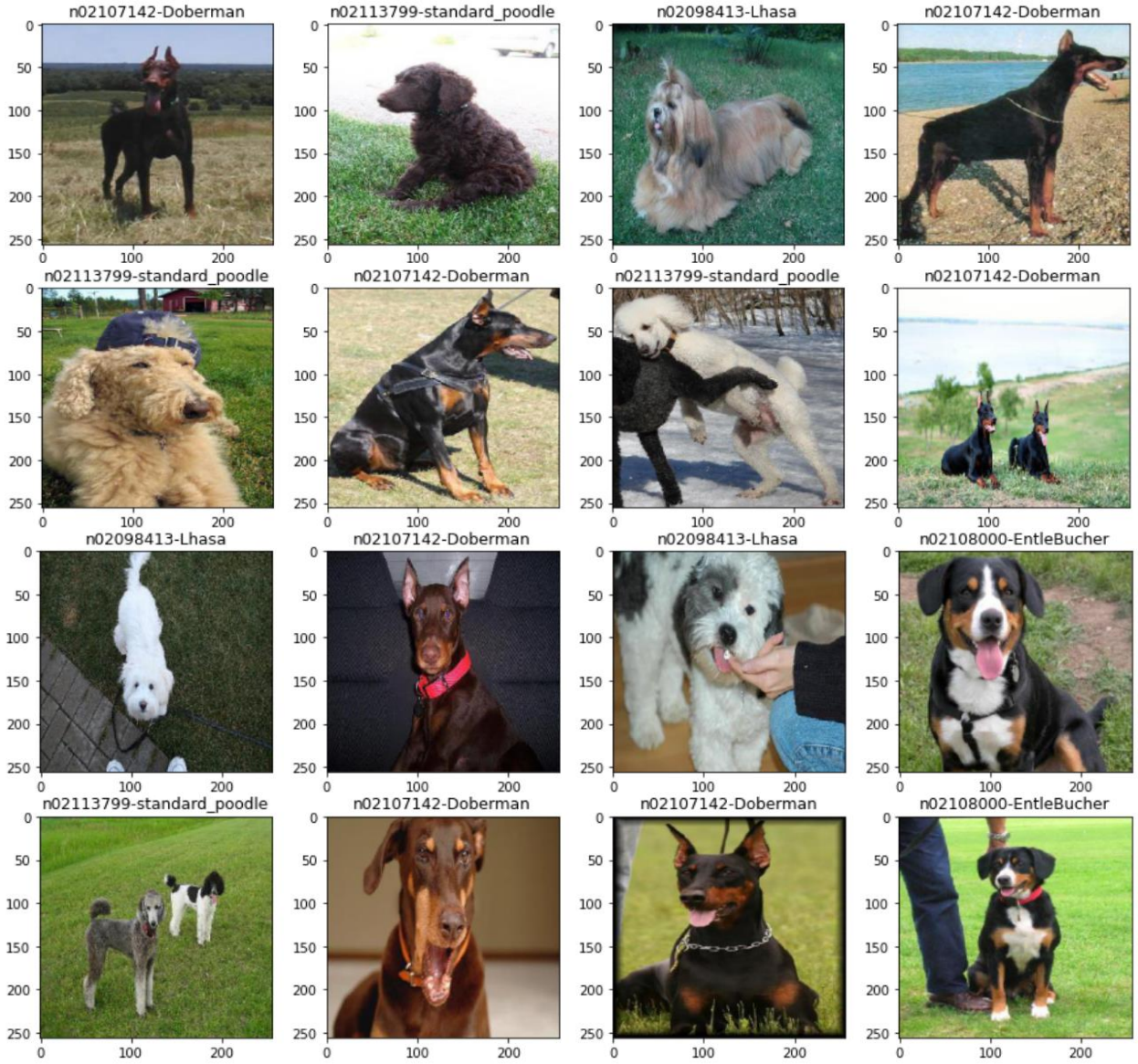
Photo original

Modification de la luminosité

Flip de l'image

Cleaning et exploration

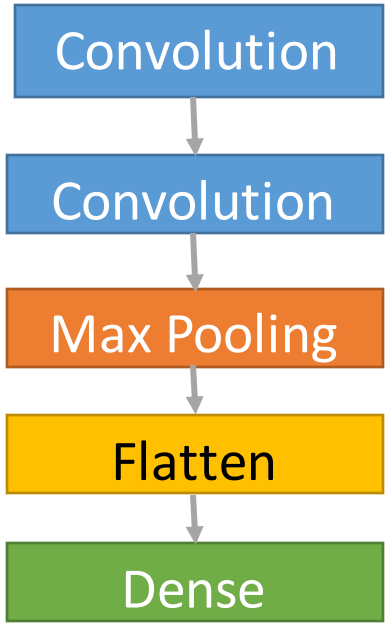
Sans augmentation



Avec augmentation

Création d'un CNN – from scratch

Premier modèle pour 5 classes:



Convolution:

- 1ère et 2ème couche: 64 filtres de taille 3x3 - repère l'ensemble des feature
- padding: same - ajout pixels autour pour ne pas rétrécir l'image
- activation: ReLU - remplace valeurs negatives par 0

Max pooling:

- réduit taille image en préservant caractéristiques

Flatten:

- Transforme matrice 3D en vecteur

Dense:

- dernière couche du CNN avec 5 classes à prédire
- activation: softmax (N=5 >2)

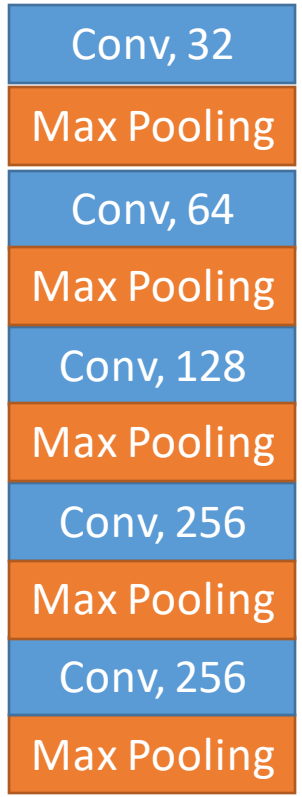
```
my_CNN = Sequential()
my_CNN.add(Convolution2D(64,(3,3), input_shape = (256,256,3), padding = 'same', activation = 'relu'))
my_CNN.add(Convolution2D(64, (3,3), padding = 'same', activation = 'relu'))
my_CNN.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
my_CNN.add(Flatten())
my_CNN.add(Dense(5, activation = 'softmax'))
```

```
Total params: 5,281,605
Trainable params: 5,281,605
Non-trainable params: 0
```

Création d'un CNN – from scratch

Troisième modèle pour 5 et 120 classes:

```
CNN3_5C = Sequential()

CNN3_5C.add(Convolution2D(32, 3, 3, input_shape=(256,256,3), padding='same', activation='relu'))
CNN3_5C.add(MaxPooling2D((2,2), strides=(2,2)))
CNN3_5C.add(Convolution2D(64, 3, 3, padding='same', activation='relu'))
CNN3_5C.add(MaxPooling2D((2,2), strides=(2,2)))
CNN3_5C.add(Convolution2D(128, 3, 3, padding='same', activation='relu'))
CNN3_5C.add(MaxPooling2D((2,2), strides=(2,2)))
CNN3_5C.add(Convolution2D(256, 3, 3, padding='same', activation='relu'))
CNN3_5C.add(MaxPooling2D((2,2), padding='same'))
CNN3_5C.add(Convolution2D(256, 3, 3, padding='same', activation='relu'))
CNN3_5C.add(MaxPooling2D((2,2), padding='same'))

CNN3_5C.add(Flatten())
CNN3_5C.add(Dense(128, activation='relu'))
CNN3_5C.add(Dropout(0.5)) #prevent overfitting
CNN3_5C.add(Dense(64, activation = 'relu'))
CNN3_5C.add(Dropout(0.5))
CNN3_5C.add(Dense(5, activation='softmax'))
```



Dense: 5 classes

```
Total params: 1,019,973
Trainable params: 1,019,973
Non-trainable params: 0
```

Dense: 120 classes

```
Total params: 1,027,448
Trainable params: 1,027,448
Non-trainable params: 0
```

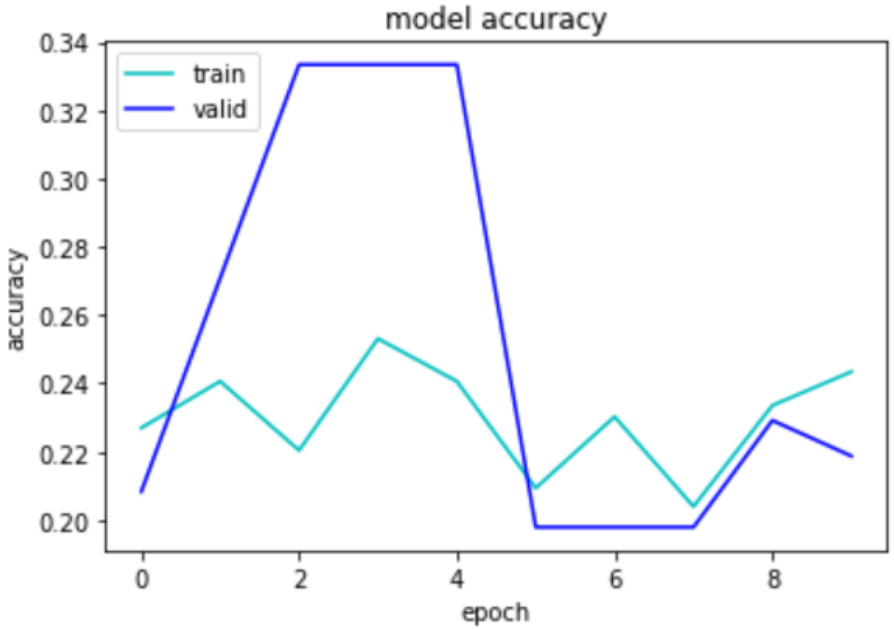
Création d'un CNN – from scratch

Paramètres modifiés et résultats pour 5 classes

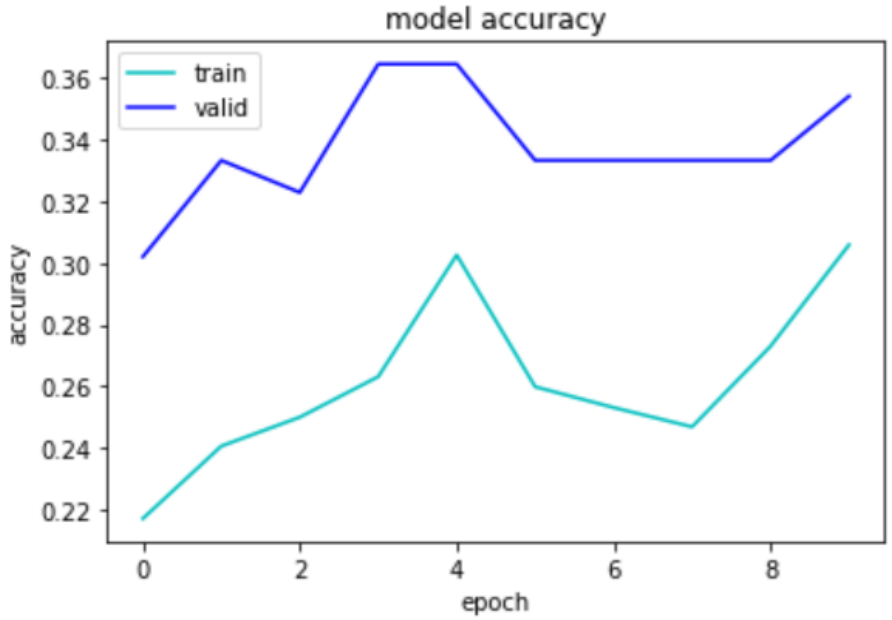
Nombres de filtres:
32, 64, 128, 256

	Sans augmentation		Avec augmentation*	
	Accuracy	Temps	Accuracy	Temps
SGD	0.2083	13.1450	0.2500	56.7250
Adam	0.3333	13.3523	0.3646	21.6229

*Augmentation = luminosité et flip



Adam – Sans augmentation



Adam – Avec augmentation

Création d'un CNN – from scratch

Paramètres modifiés et résultats pour 120 classes

Optimizer	**Learning rate**	**Accuracy**	**Temps (en s.)**
SGD	0,0001	0.0208	684,9746
Adam	0,0001	0.0469	533,2239
Adam	0,001	0.0156	408,1274
Adagrad	0,0001	0.0312	303,6105
Adagrad	0,001	0.0208	249,3780

- Précision très faible malgré les différents paramètres
- Nouveaux modèles plus performants

Transfert Learning

Transfert Learning = utilises connaissances acquises par un autre CNN pour résoudre un problème + ou - similaire

<u>Différentes stratégies:</u>

- Fine-tuning total: ré-entraîne tout le réseau
- Extraction de features: ne ré-entraîne pas les couches du réseau
- Fine-tuning partiel: entraîne couches hautes

CNN choisis:

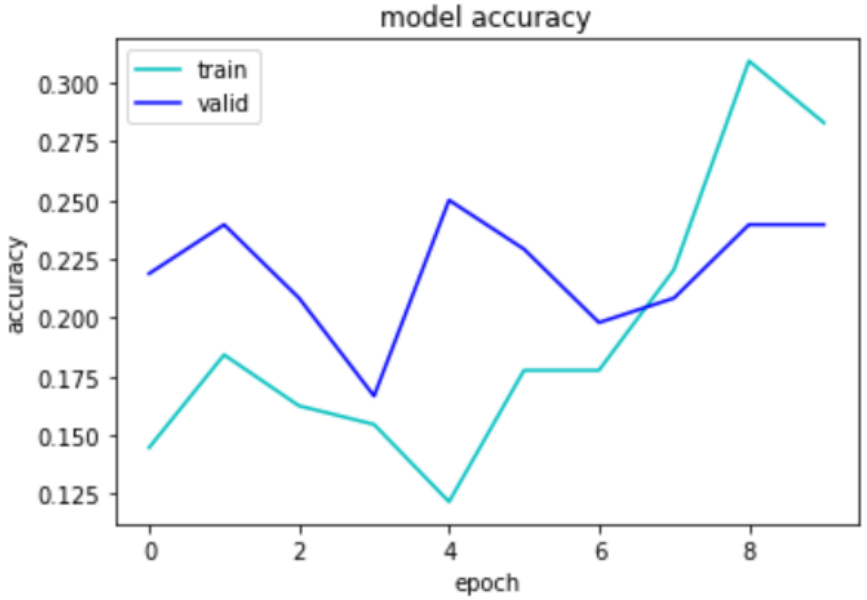
- ResNet50
- VGG-16
- InceptionV3

Création d'un CNN – Transfert Learning

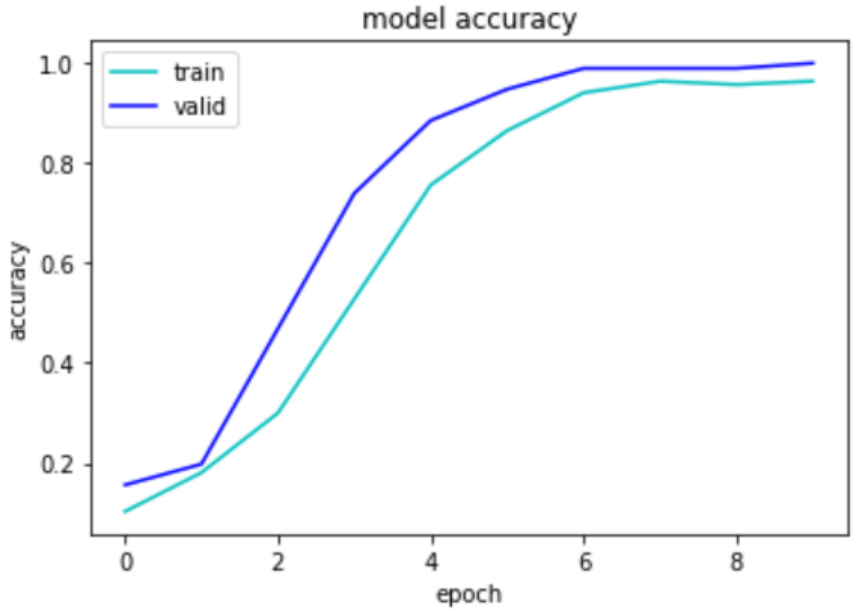
Extraction de features

Epochs: 10
Batch_size: 40

Optimizer	ResNet50		VGG-16		InceptionV3	
	Précision	Temps	Précision	Temps	Précision	Temps
SGD	0.2188	19.0068 s.	0.2188	17.2785 s.	0.9896	21.7252 s.
Adam	0.2500	18.6200 s.	0.2812	26.9603 s.	0.9967	21.7443 s.



ResNet50 – optimizer Adam



Inception V3 – optimizer SGD

- Montée à la fin de la courbe
- Tester avec plus d'epochs
- Résultats similaires: temps important et précision moyenne

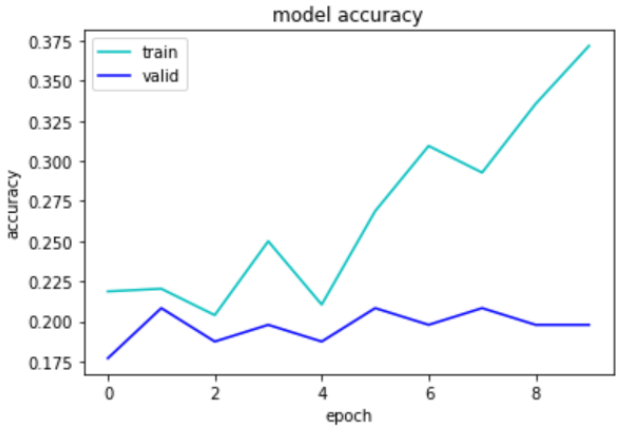
Création d'un CNN – Transfert Learning

Fine tuning

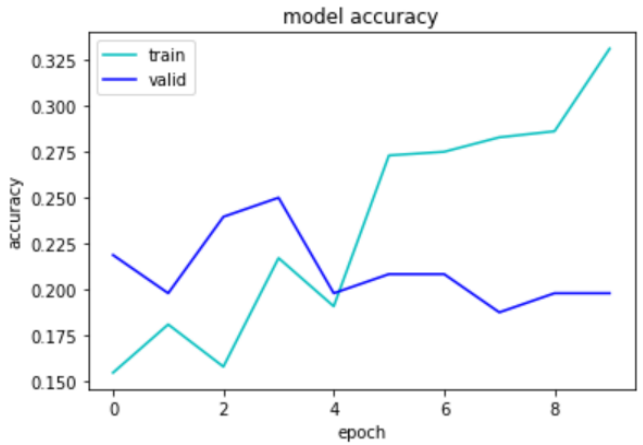
Epochs: 10
Batch_size: 40

	ResNet50		
	Précision	Temps (en s.)	Nombre de couches non-entraînée
SGD	0.2083	29.6655	10
	0.2188	29.1727	4
	0.2396	46.4502	17
Adam	0.2604	50.5133	10

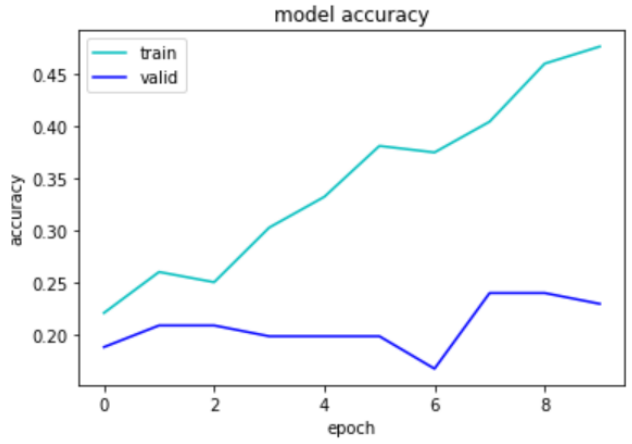
SGD – 10 couches



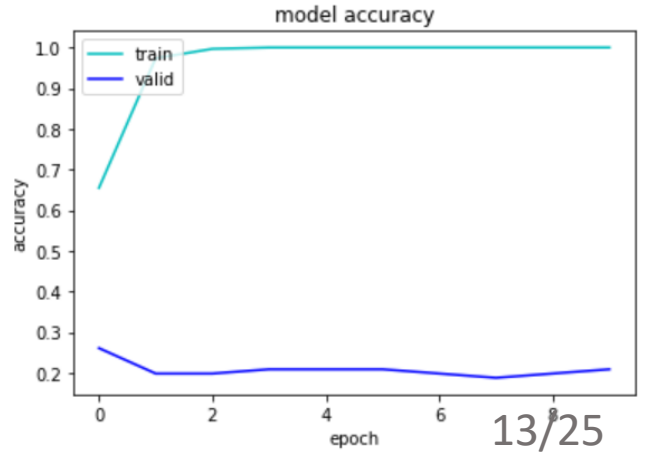
SGD – 4 couches



SGD – 17 couches



Adam - 10 couches



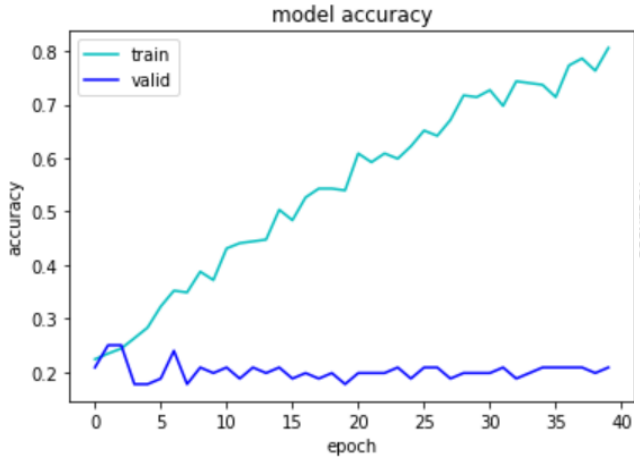
Création d'un CNN – Transfert Learning

Fine tuning

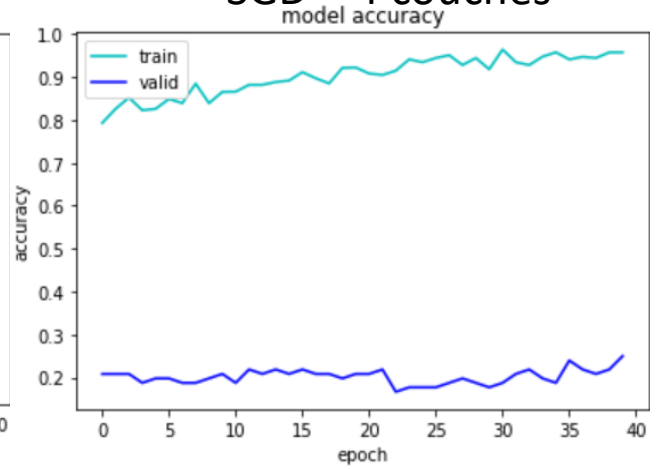
Epochs: 40
Batch_size: 40

	ResNet50			
	Précision validation	Précision	Temps (en s.)	Nombre de couches non-entraînée
SGD	0.2083	0.8059	181.1179	10
	0.2500	0.9572	181.6613	4
	0.2708	0.8224	171.2812	17
Adam	0.2083	1.0000	184.6009	10

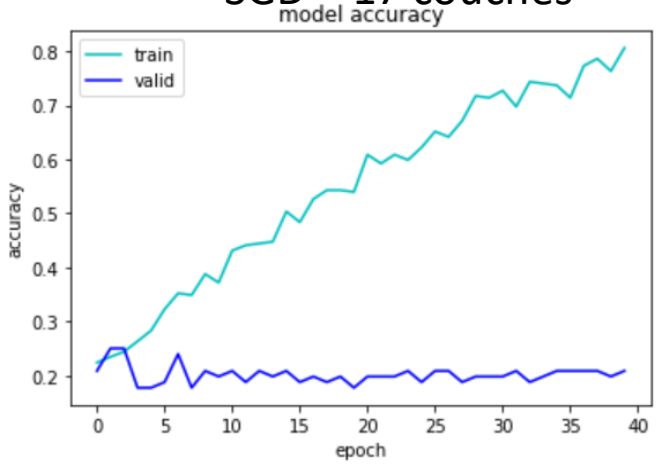
SGD – 10 couches



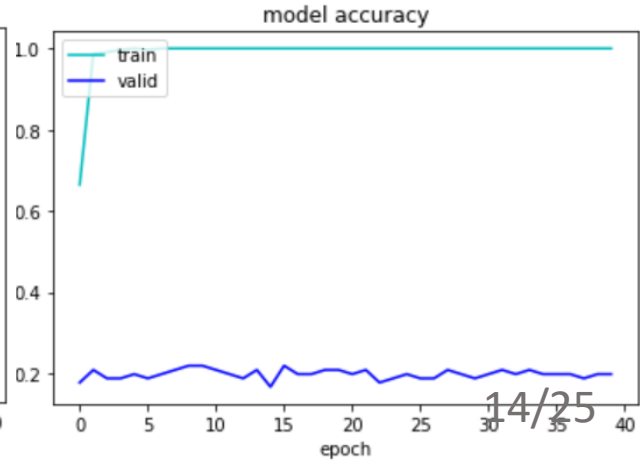
SGD – 4 couches



SGD – 17 couches



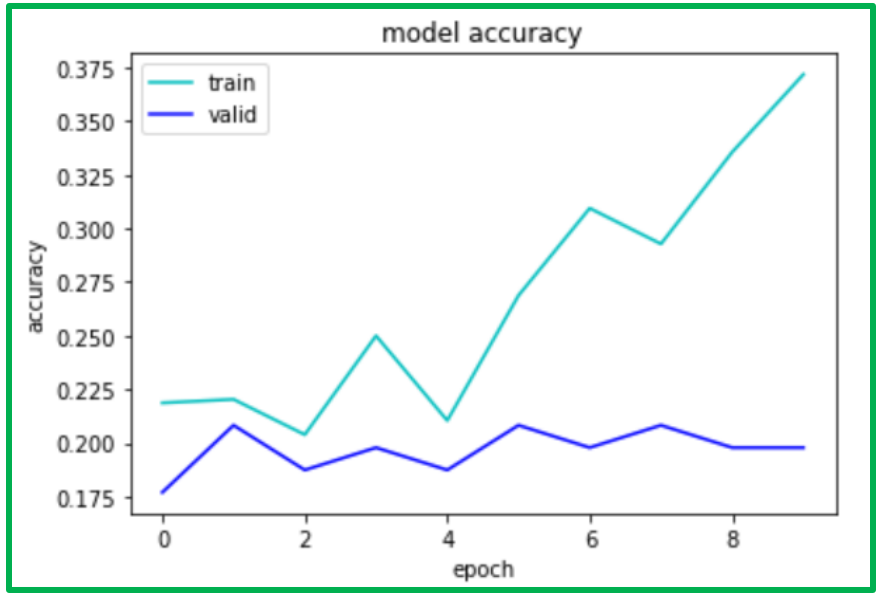
Adam - 10 couches

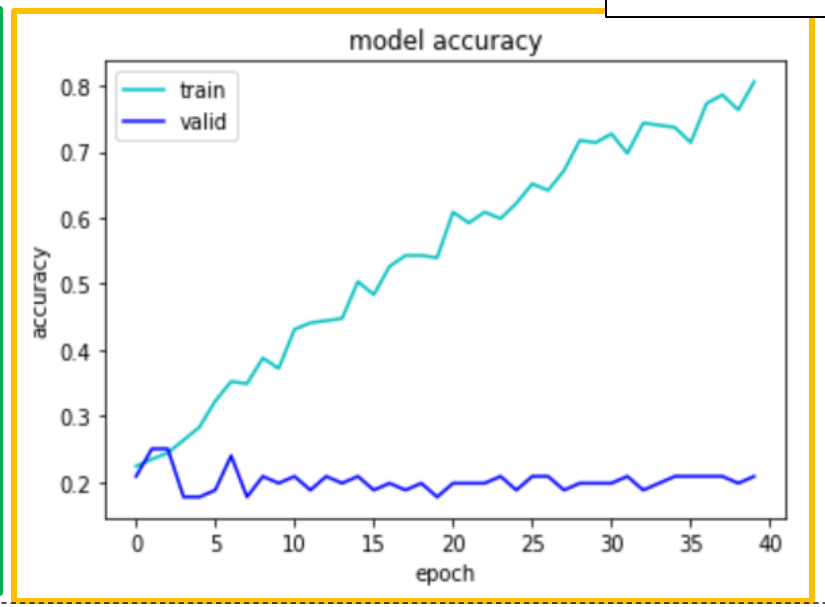


Création d'un CNN – Transfert Learning

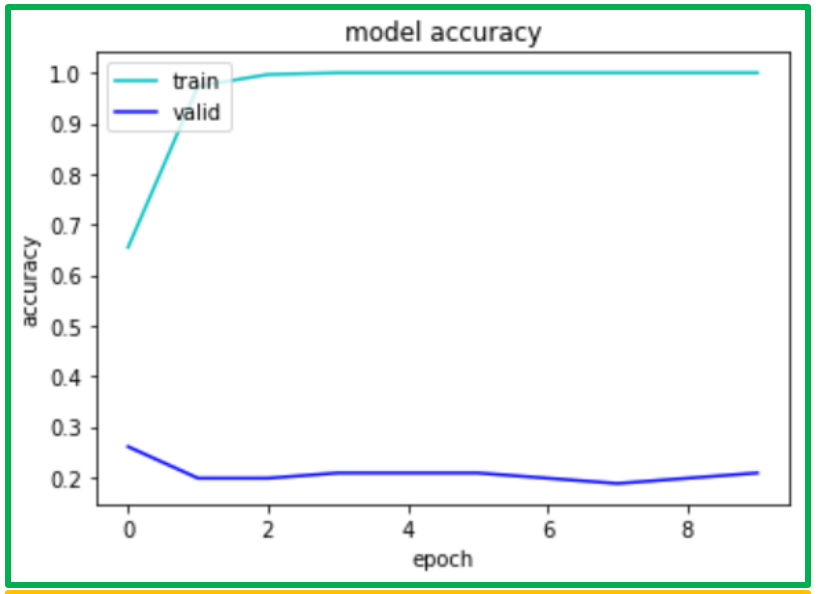
10 epochs
40 epochs

SGD – 10 couches

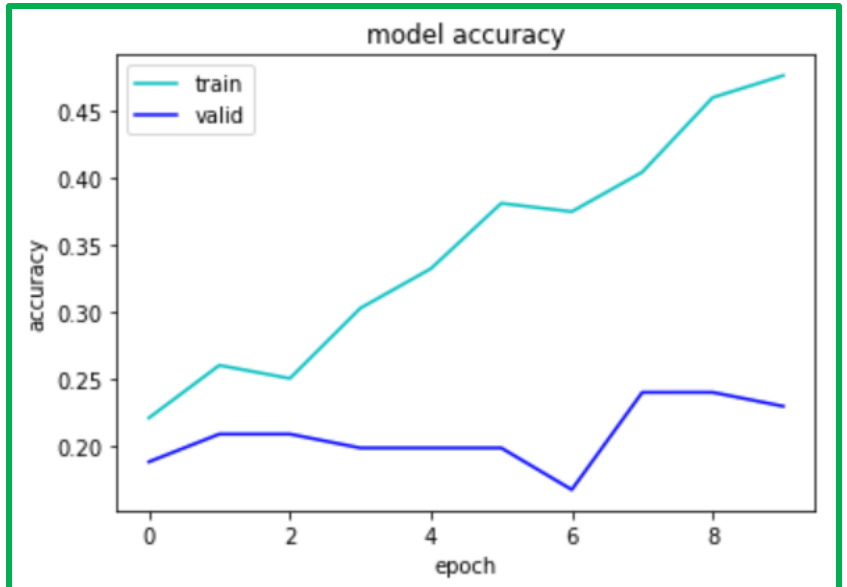


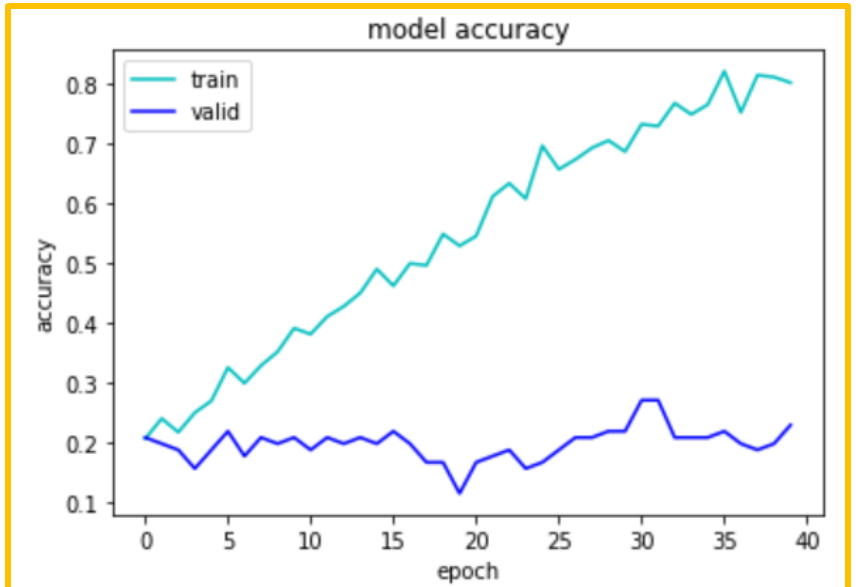


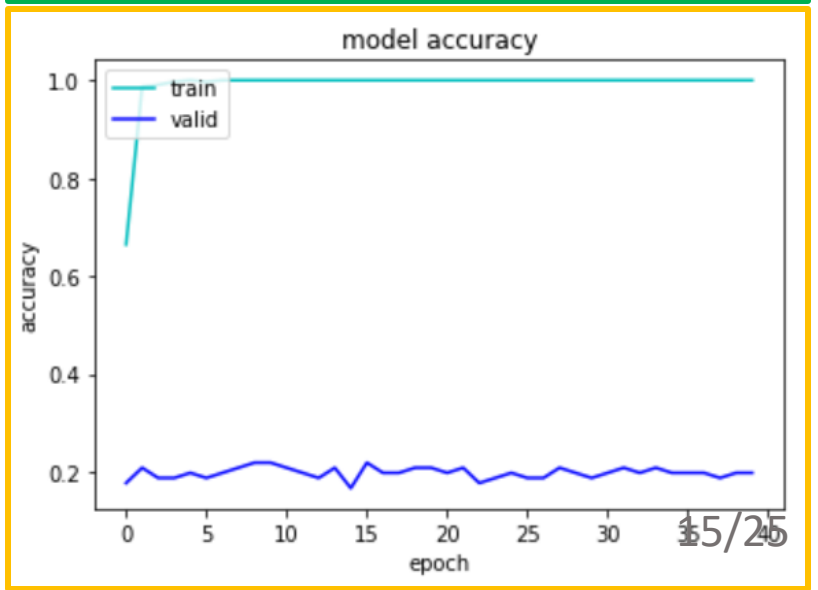
Adam - 10 couches



SGD – 17 couches







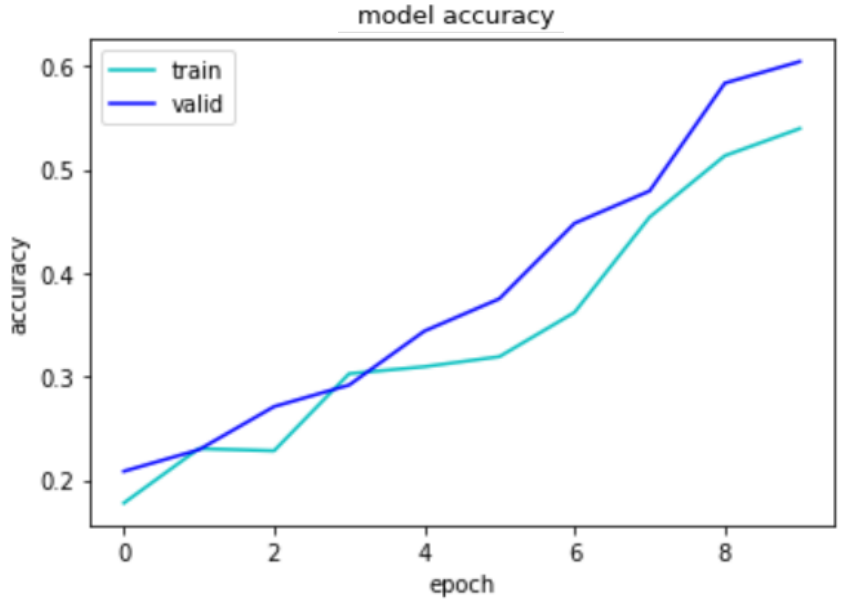
Création d'un CNN – Transfert Learning

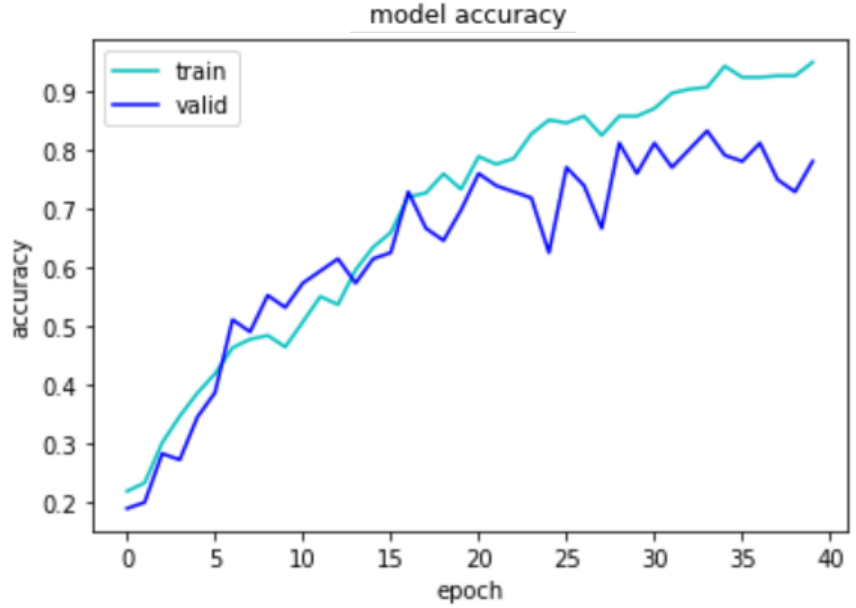
Fine tuning

Epochs: 40
Epochs: 20
Batch_size: 16

NCNE = Nombre de couches non-entraînées

	VGG-16			InceptionV3		
Optimizer	Précision	Temps	NCNE	Précision	Temps	NCNE
SGD	0.8333	172.8461	4	0.9896	71.4795	3
	0.8021	131.7296	8			
Adam	0.8854	185.3149	4	0.9792	80.4870	3





VGG-16: Optimizer SGD, 4 couches - 10 vs 40 epochs

- Tests avec différents epochs: 10, 20, 40
- Amélioration des précisions
- Temps d'exécution important

Comparaison des différents CNN

Différents CNN du projet:

- From scratch → Précision trop faible
  Max accuracy des modèles = 0.3646
- Extraction de features → Précision similaire au fine-tuning
  Temps d'exécution beaucoup plus faible
  Nombre d'epochs = 10
- Fine-tuning partiel → Précision similaire
  Temps d'exécution plus long
  Nombre d'epochs = 40

	InceptionV3	
Optimizer	Précision	Temps
SGD	0.9896	71.4795
Adam	0.9792	80.4870

	InceptionV3	
Optimizer	Précision	Temps
SGD	0.9896	21.7252 s.
Adam	0.9967	21.7443 s.

Comparaison des meilleurs modèles

4 meilleurs modèles (précision) = InceptionV3

A. Extraction de features: Optimizer SGD
B. Extraction de features: Optimizer Adam
C. Fine-tuning partiel: 3 couches non entraînées - SGD
D. Fine-tuning partiel: 3 couches non entraînées - Adam

Précision	Temps	
0.9896	21.7252 s.	10 epochs
0.9967	21.7443 s.	
0.9896	71.4795	40 epochs
0.9792	80.4870	

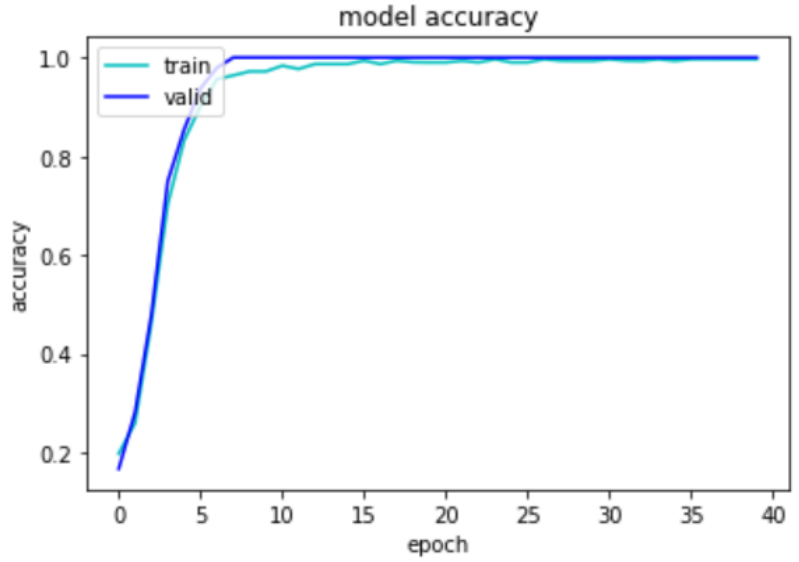
10 epochs	Précision	Temps
Modèle A	0.9896	21.7252 s.
Modèle B	0.9967	21.7443 s.
Modèle C	0.9479	47.0188
Modèle D	0.9896	41.7289

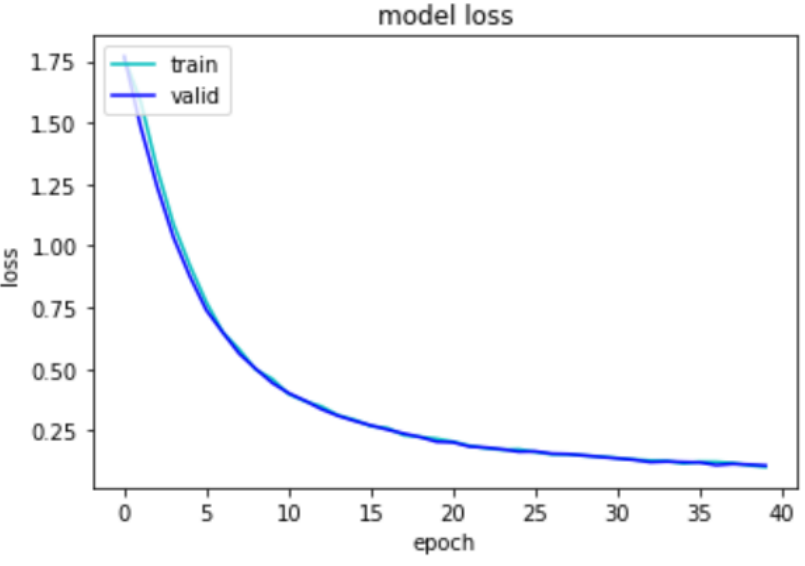
40 epoch	Précision	Temps
Modèle A	1.0000	72.5857
Modèle B	0.9896	41.7289
Modèle C	0.9896	71.4795
Modèle D	0.9792	80.4870

Meilleurs modèles choisis: A, B et C

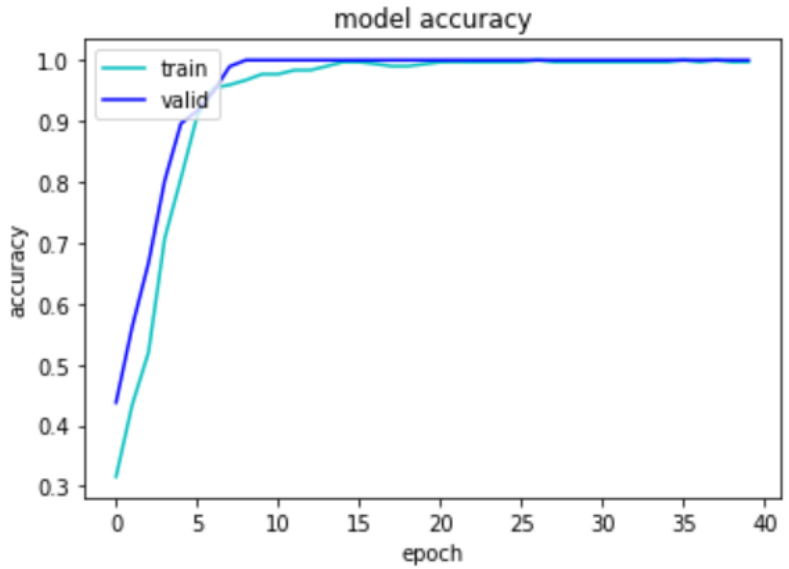
Comparaison des meilleurs modèles

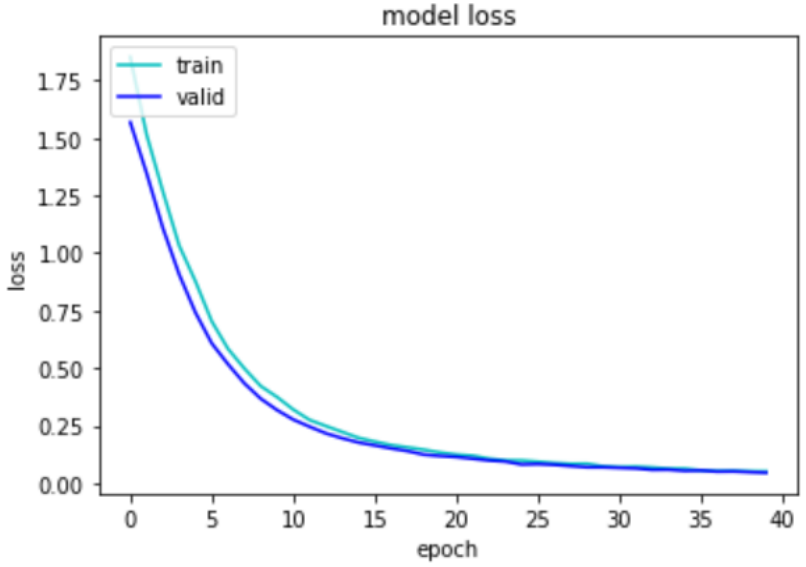
Modèle A



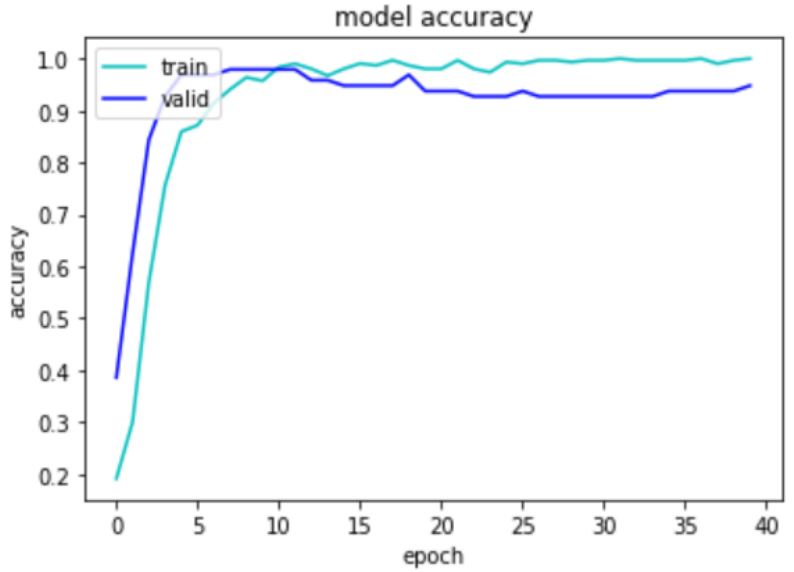


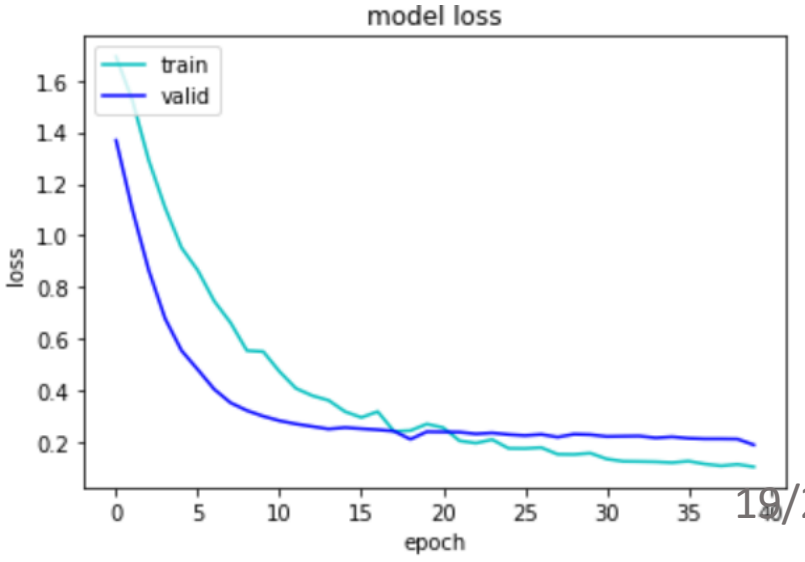
Modèle B





Modèle C





Comparaison des meilleurs modèles

Pour choisir le meilleur modèle, on teste les modèles avec et sans augmentation sur 120 classes et on compare les résultats.
Travail avec 120 classes = Recherche de gain de temps ----> EarlyStopping

EarlyStopping: s'arrête si la valeur val_accuracy n'augmente pas après un nombre X=8 d'epoch.

```
early_stopping_cd = EarlyStopping(monitor = 'val_accuracy', patience=8)
```

Pour confirmer le meilleur modèle, j'ai testé avec différents paramètres:

- Nombre d'epoch:
  - 30, 50, 60
- Batch size:
  - 40, 80
- Avec et sans augmentation

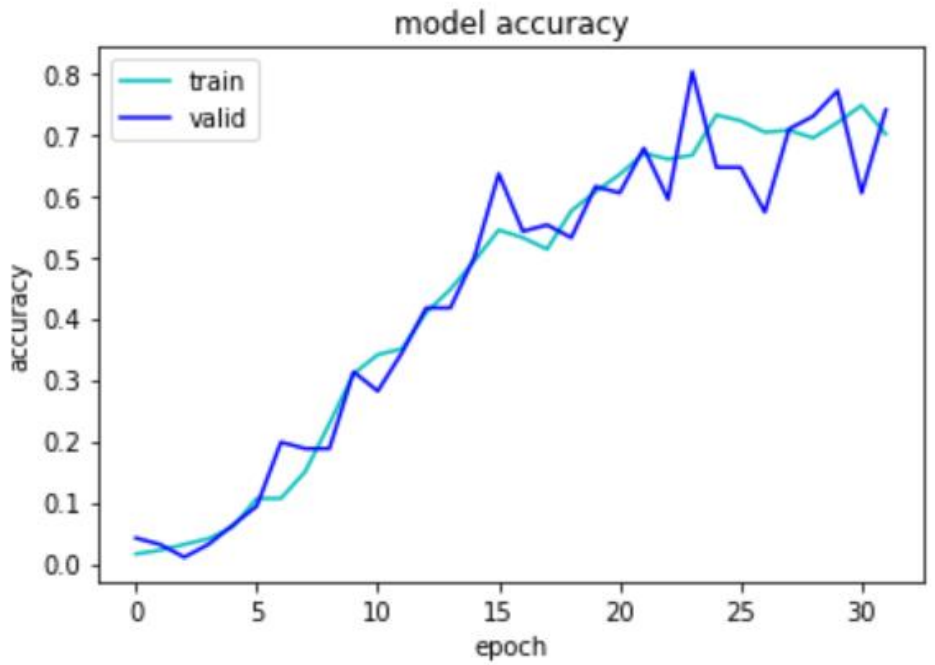
	Sans augmentation		Avec augmentation	
	Précision	Temps	Précision	Temps
Modèle A	0,0156	14 s.	0,0156	20 s.
Modèle B	0,8594	52 s.	0,7188	78 s.
Modèle C	0,0219	12 s.	0,0156	26 s.

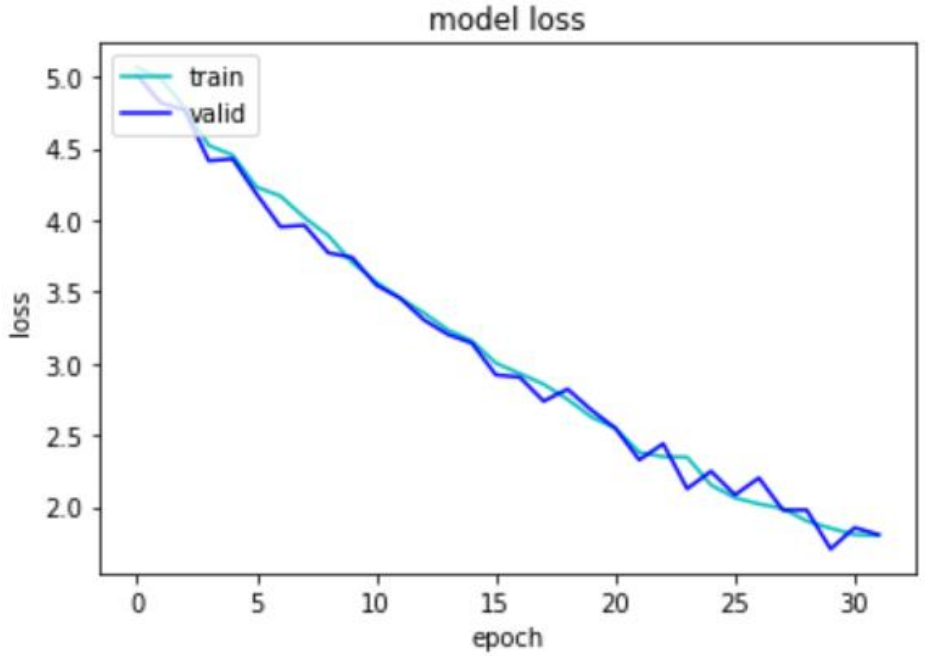
batch_size = 80, epochs = 50

Meilleur modèle

Le meilleur modèle est:
Modèle B = Inception V3 - Extraction de features - Optimizer Adam

Précision: 85,94 %
Temps: 52 s.





Evaluation sur data set de test: `Résultat: [1.1557236909866333, 0.7933403253555298]`

Précision = 79,33 %

API et prédiction

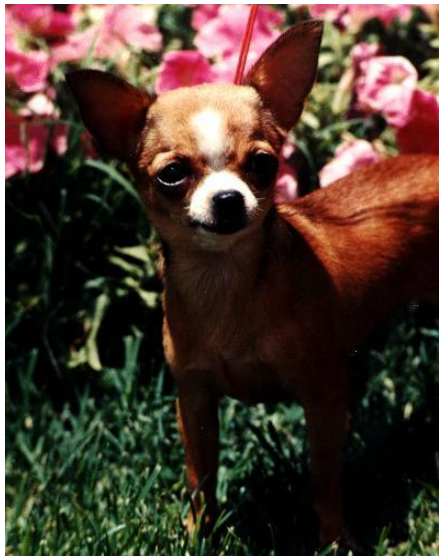
Après avoir enregistré le meilleur modèle, il est importé en 1ère ligne.
La fonction qui suit prend en entrée le chemin d'accès d'une photo, la redimensionne, transformée puis utilisé par le modèle pour la prédiction

```
# Import model
model = tf.keras.models.load_model('saved_models/best_model.h5')

# Create function that gives the breed and the percentage of matching from a photo
def predict_breed(path_file):
  # load the image from file
  image = load_img(path_file, target_size = (256,256))
  # convert the image pixels to a numpy array
  image = img_to_array(image)
  # reshape data for the model
  image = image.reshape(1, 256,256, 3)
  image = image/255.
  # predict the breed
  prediction = model.predict(image)
  name_classes = os.listdir('train')
  pos = np.argmax(prediction)

  print('Ce chien est : {} avec une probabilité de {}%'.format(name_classes[pos][10:], prediction[0][pos]*100))
```

La fonction renvoie le nom de la race prédite ainsi que la probabilité de réussite.

API et prédiction

A)



B)

C)

1.

```
predict_breed('test/n02085620-Chihuahua/n02085620_10131.jpg')
```

```
Ce chien est : Chihuahua avec une probabilité de 70.54483890533447%
```

A) ✓

2.

```
predict_breed('test/n02102177-Welsh_springer_spaniel/n02102177_1681.jpg')
```

```
Ce chien est : Welsh_springer_spaniel avec une probabilité de 71.06332182884216%
```

B) ✓

3.

```
predict_breed('test/n02108915-French_bulldog/n02108915_3520.jpg')
```

```
Ce chien est : bull_mastiff avec une probabilité de 50.591373443603516%
```

C) ✗

Conclusion

Ce que j'ai appris:

- Découverte de nouvelles librairies pour le traitement d'image
- Familiarisation avec les CNN et leurs architectures --> différence CNN from scratch et transfert learning
- Travail sur l'optimisation de modèle exigeant en ressource

Améliorations possibles:

- Modification des paramètres du modèle choisi

Ressources

Voici les liens qui m'ont aidé à travers ce projet:

https://towardsdatascience.com/google-colab-import-and-export-datasets-eccf801e2971
https://machinelearningmastery.com/image-augmentation-deep-learning-keras/
https://elitedatascience.com/keras-tutorial-deep-learning-in-python
https://www.geeksforgeeks.org/working-images-python/
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/
https://kuanhoong.medium.com/how-to-use-tensorboard-with-google-colab-43f7cf061fe4
https://blog.octo.com/classification-dimages-les-reseaux-de-neurones-convolutifs-en-toute-simplicite/
https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/
https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/
https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5097666-tp-implementez-votre-premier-reseau-de-neurones-avec-keras
https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
https://medium.com/@kenneth.ca95/a-guide-to-transfer-learning-with-keras-using-resnet50-a81a4a28084b
https://keras.io/api/preprocessing/image/
https://keras.io/api/layers/