

**8INF804 - Vision artificielle et traitement des images - Hiver 2023**

Professeur Julien Maitre, Ph.D.

**UNIVERSITÉ DU QUÉBEC À CHICOUTIMI**

Département d'Informatique et de Mathématique

# **Comparaison des réseaux de neurones convolutifs**

**BALB10020109 - Becaye Baldé**

<b>Comparaison des réseaux de neurones convolutifs</b>	<b>2</b>
<b>CNN Personalisé</b>	<b>5</b>
Architecture du modèle	5
Entraînement du modèle	6
Evaluation des performances	8
Courbes d'apprentissage	8
Score Kappa	9
Rapport de classification	10
Prediction d'une nouvelle image	10
<b>EfficientNet</b>	<b>11</b>
Aperçu	11
Architecture	13
Entraînement	13
<b>Conclusion</b>	<b>16</b>

Dans ce TP, j'ai testé des réseaux de neurones convolutifs: Un CNN personnel, et un modèle state-of-the-art: EfficientNet.

J'ai utilisé l'ensemble de données: [intel images](#) de Kaggle. Ce dataset contient 17,000 images et 6 catégories : bâtiments, forêts, mers, montagnes, glaciers et rues.

Ces images sont reparties en 3 sous ensembles:

Ces images sont reparties en 3 dossiers:

- Train: contient les images d'entraînement ;
- Test: contient les images de validation ;
- Pred: contient des images de test.

Les images dans les deux premiers dossiers sont regroupés selon leur classes:

```
PS D:\Deep Learning\Loading Custom Dataset\intel_images> dir train

Directory: D:\Deep Learning\Loading Custom Dataset\intel_images\train
              Mode LastWriteTime      Length Name
----- ---- -        ----- 
d---- 1/29/2023 4:30 PM
d---- 1/11/2023 4:06 PM
d---- 3/24/2023 11:34 AM
d---- 1/11/2023 4:07 PM
d---- 1/11/2023 4:07 PM
d---- 1/11/2023 4:07 PM

-----  def split_images_and
buildings"""
forest   Separate the ima
glacier  full_dataset=Fa
mountain """
sea      images = []
street   labels = []
```

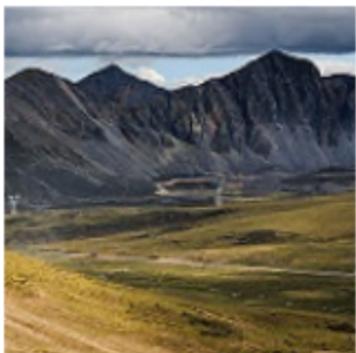
Et pred contient des images non annotées :

```
Directory: D:\Deep Learning\Loading Custom Dataset\intel_images\pred

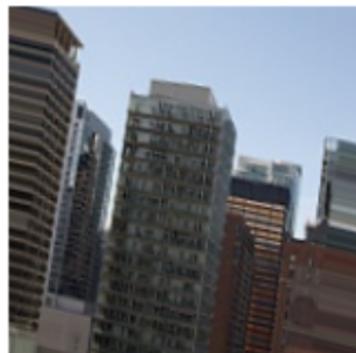
              Mode LastWriteTime      Length Name
----- ---- -        ----- 
-a--- 10/18/2019 5:19 AM    22337 10004.jpg
-a--- 10/18/2019 5:19 AM    10295 10005.jpg
-a--- 10/18/2019 5:19 AM    18143 10012.jpg
-a--- 10/18/2019 5:19 AM    12449 10013.jpg
-a--- 10/18/2019 5:19 AM    16377 10017.jpg
-a--- 10/18/2019 5:19 AM    24750 10021.jpg
```

Voici quelques échantillons d'images de l'ensemble de données:

mountain



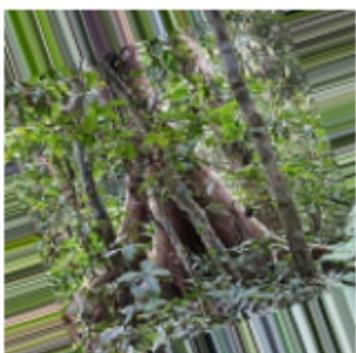
buildings



street



forest



street



glacier



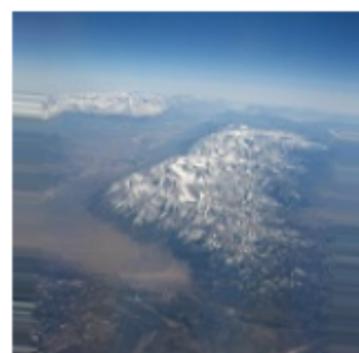
mountain



mountain



mountain



### Échantillons du dataset

On peut voir qu'elles sont très diversifiées. Cependant, le dataset est impure et contient des images mal annotées. Par exemple, la fleur suivante est annotée “glacier”:



Fausse annotation: Une fleur annoté “glacier”

Cela qui pourrait d'affecter les performances du modèle.

Passons donc à la création des modèles pour voir ce que ça donne.

## CNN Personalisé

J'ai d'abord commencé avec une architecture très simple:

- 2 couches de convolution
- 2 couches de max pooling

Cependant, le modèle n'était pas du tout performant. J'ai donc décidé de le rendre plus profond.

## Architecture du modèle

Le CNN final contient :

- 4 couches de convolution ;
- 4 couches de max pooling ;

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D )	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling 2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling 2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 256)	295168
max_pooling2d_3 (MaxPooling 2D)	(None, 7, 7, 256)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 128)	1605760
dense_1 (Dense)	(None, 6)	774
<hr/>		
Total params:	1,994,950	
Trainable params:	1,994,950	
Non-trainable params:	0	

## Architecture du modèle

## Entraînement du modèle

Puis, j'ai entraîné le modèle pour un nombre maximal de 100 epochs, avec des conditions d'arrêt au cas où l'accuracy ou la perte ne s'améliorerait pas.

```

cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

loss_stop = EarlyStopping(monitor='val_loss', patience=4, mode='min', verbose=1)
accuracy_stop = EarlyStopping(monitor='val_accuracy', patience=2)

start_time = time.time()
history = cnn.fit(train_data,
                   batch_size=batch_size,
                   epochs=100,
                   validation_data=test_data,
                   callbacks=[accuracy_stop, loss_stop])
end_time = time.time()

```

Après seulement 12 epochs, le modèle a commencé s'est stabilisé:

```

Epoch 1/100
220/220 [=====] - 259s 1s/step - loss: 1.1266 - accuracy: 0.5653 - val_loss: 0.8980 - val_accuracy: 0.6517
Epoch 2/100
220/220 [=====] - 237s 1s/step - loss: 0.7934 - accuracy: 0.7062 - val_loss: 0.6867 - val_accuracy: 0.7480
Epoch 3/100
220/220 [=====] - 241s 1s/step - loss: 0.6892 - accuracy: 0.7499 - val_loss: 0.6364 - val_accuracy: 0.7737
Epoch 4/100
220/220 [=====] - 227s 1s/step - loss: 0.6009 - accuracy: 0.7816 - val_loss: 0.5818 - val_accuracy: 0.7933
Epoch 5/100
220/220 [=====] - 228s 1s/step - loss: 0.5658 - accuracy: 0.7943 - val_loss: 0.5599 - val_accuracy: 0.7983
Epoch 6/100
220/220 [=====] - 226s 1s/step - loss: 0.5259 - accuracy: 0.8083 - val_loss: 0.5336 - val_accuracy: 0.8083
Epoch 7/100
220/220 [=====] - 226s 1s/step - loss: 0.4827 - accuracy: 0.8275 - val_loss: 0.5139 - val_accuracy: 0.8117
Epoch 8/100
220/220 [=====] - 226s 1s/step - loss: 0.4663 - accuracy: 0.8319 - val_loss: 0.4665 - val_accuracy: 0.8363
Epoch 9/100
220/220 [=====] - 227s 1s/step - loss: 0.4305 - accuracy: 0.8456 - val_loss: 0.4810 - val_accuracy: 0.8317
Epoch 10/100
220/220 [=====] - 226s 1s/step - loss: 0.4230 - accuracy: 0.8455 - val_loss: 0.4211 - val_accuracy: 0.8503
Epoch 11/100
220/220 [=====] - 226s 1s/step - loss: 0.4025 - accuracy: 0.8516 - val_loss: 0.4570 - val_accuracy: 0.8343
Epoch 12/100
220/220 [=====] - 226s 1s/step - loss: 0.3882 - accuracy: 0.8587 - val_loss: 0.4512 - val_accuracy: 0.8397
print("Elapsed time: {:.2f} minutes".format((end_time - start_time) / 60))
Elapsed time: 46.24 minutes

```

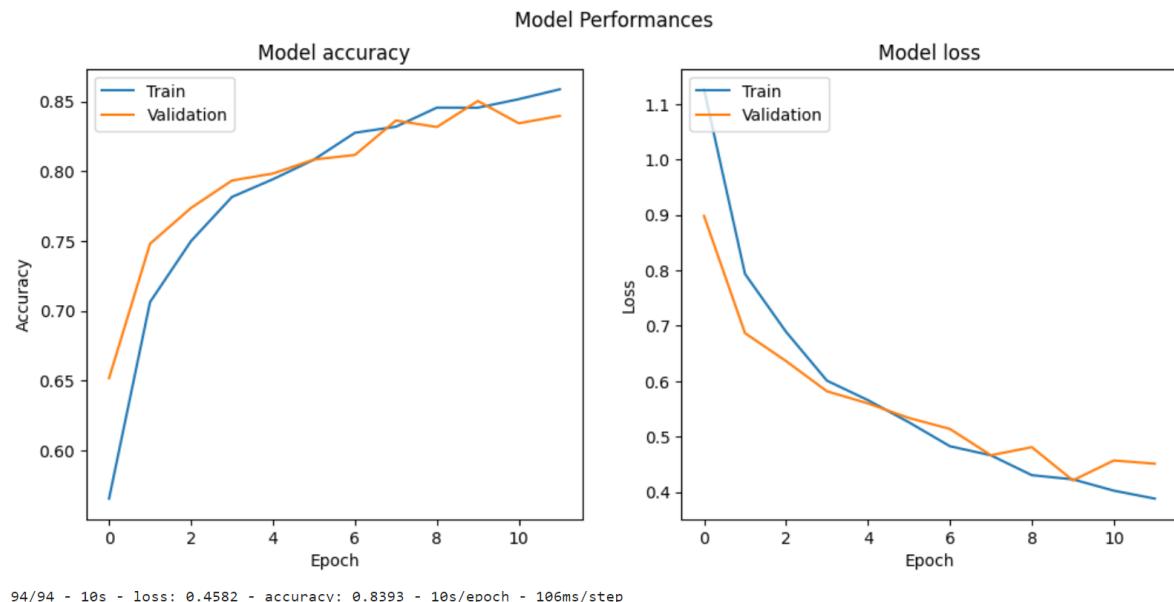
L'accuracy semble rester stable lorsque l'on relance l'entraînement. Après 8 epochs le modèle s'arrête:

```
In [3]: cnn.fit(train_data=train_data, val_data=val_data, batch_size=32, epochs=15, patience=2)
Epoch 1/15
351/351 [=====] - 503s 1s/step - loss: 0.8060 - accuracy: 0.6939 - val_loss: 0.7248 - val_accuracy: 0.7217
Epoch 2/15
351/351 [=====] - 486s 1s/step - loss: 0.6633 - accuracy: 0.7561 - val_loss: 0.6239 - val_accuracy: 0.7699
Epoch 3/15
351/351 [=====] - 485s 1s/step - loss: 0.5956 - accuracy: 0.7813 - val_loss: 0.5892 - val_accuracy: 0.7863
Epoch 4/15
351/351 [=====] - 480s 1s/step - loss: 0.5459 - accuracy: 0.7989 - val_loss: 0.5834 - val_accuracy: 0.7924
Epoch 5/15
351/351 [=====] - 479s 1s/step - loss: 0.5046 - accuracy: 0.8216 - val_loss: 0.6125 - val_accuracy: 0.7849
Epoch 6/15
351/351 [=====] - 481s 1s/step - loss: 0.4677 - accuracy: 0.8279 - val_loss: 0.4958 - val_accuracy: 0.8234
Epoch 7/15
351/351 [=====] - 504s 1s/step - loss: 0.4599 - accuracy: 0.8323 - val_loss: 0.5202 - val_accuracy: 0.8184
Epoch 8/15
351/351 [=====] - 477s 1s/step - loss: 0.4236 - accuracy: 0.8455 - val_loss: 0.4944 - val_accuracy: 0.8227
In [4]: cnn.model.save('saved_models/cnn/cnn.h5')
```

## Evaluation des performances

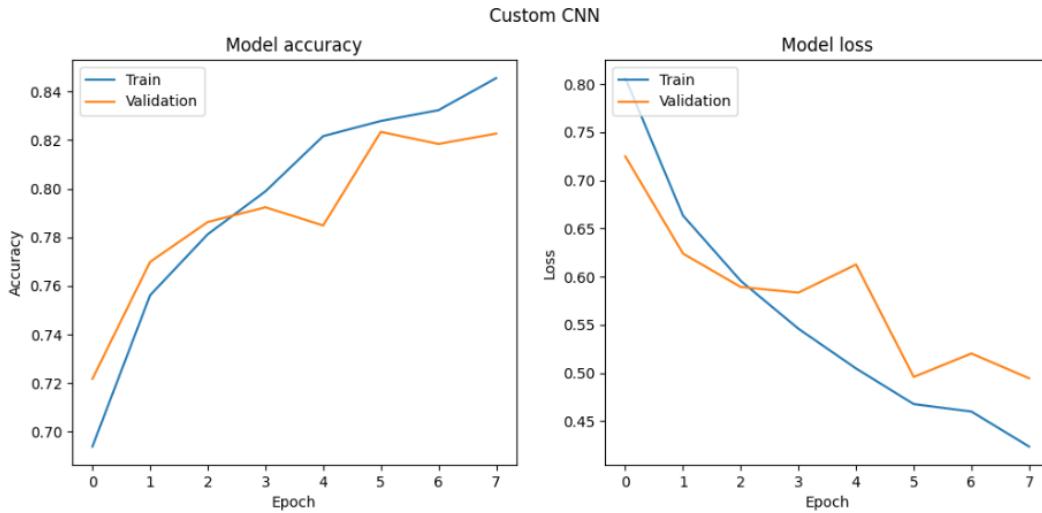
Finalement, on obtient des performances similaires sur les données d'entraînement et de validation (83% à peu près):

### Courbes d'apprentissage



```
94/94 - 10s - loss: 0.4582 - accuracy: 0.8393 - 10s/epoch - 106ms/step
Test loss: 0.45824167132377625
Test accuracy: 83.93 %
```

### Test 1 : Justesse et perte du modèle



### Test 2: Justesse et perte du modèle

#### Score Kappa

On obtient un score kappa de 0.778:

```
In [41]: cnn.compute_metrics(x_test, y_test)
94/94 [=====] - 23s 242ms/step
94/94 - 28s - loss: 0.5157 - accuracy: 0.8157 - 28s/epoch - 294ms/step
Metrics: {'loss': 0.5156731605529785, 'accuracy': 0.815666675567627,
In [42]: kappa = cnn.metrics.get("kappa")
...: print(f"Kappa: {kappa}")
Kappa: 0.7788129173256282
```

Kappa score sur les données de test

#### Rapport de classification

Le rapport montre de légères différences entre la précision et le rappel:

```
In [8]: report = print(cnn.metrics.get("report"))
      precision    recall  f1-score   support

          0       0.77      0.85      0.81      437
          1       0.92      0.98      0.95      474
          2       0.84      0.71      0.77      553
          3       0.81      0.68      0.74      525
          4       0.71      0.90      0.79      510
          5       0.88      0.80      0.84      501

   accuracy                           0.82      3000
macro avg       0.82      0.82      0.82      3000
weighted avg    0.82      0.82      0.81      3000
```

### Rapport de classification sur les données de test

### Prediction d'une nouvelle image

Lorsqu'on lui envoie cette image d'une forêt, le modèle est à 99,93% sûr que c'est une forêt, plutôt pas mal !

I am 99.93% sure this a forest  
(-0.5, 149.5, 149.5, -0.5)

forest



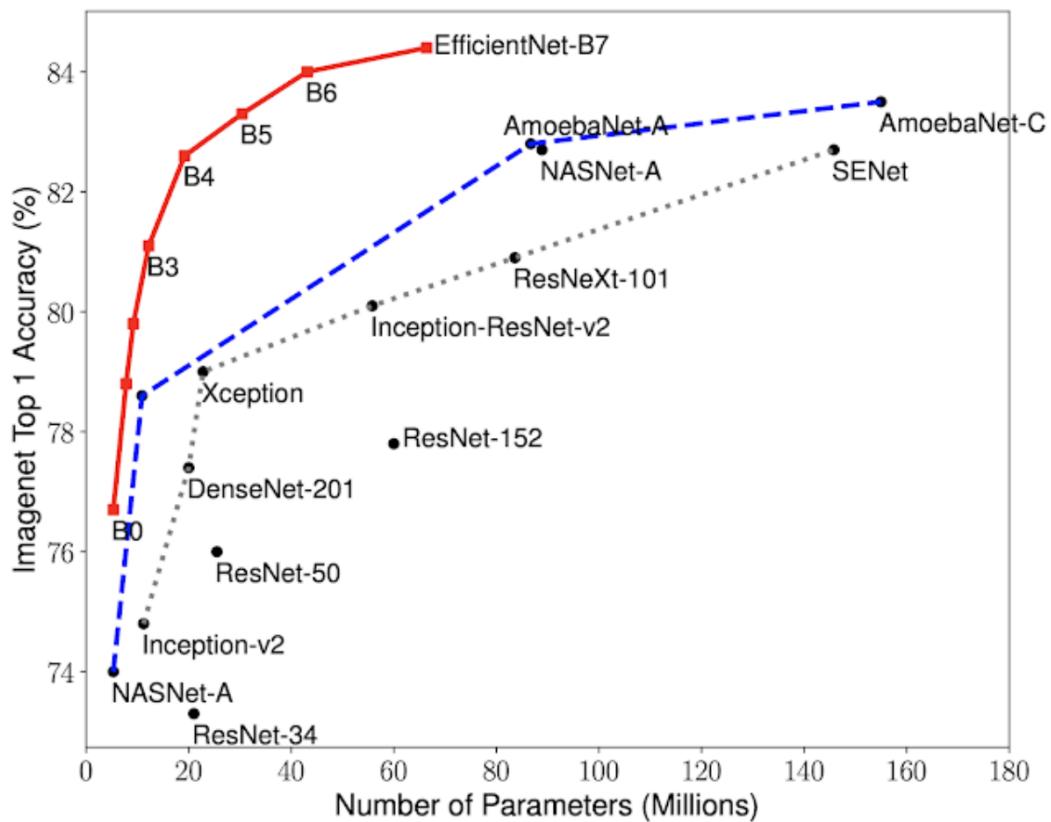
Prédiction d'une nouvelle image

Maintenant comparons les performances de notre CNN avec ceux d'un modèle state-of-the-art.

## EfficientNet

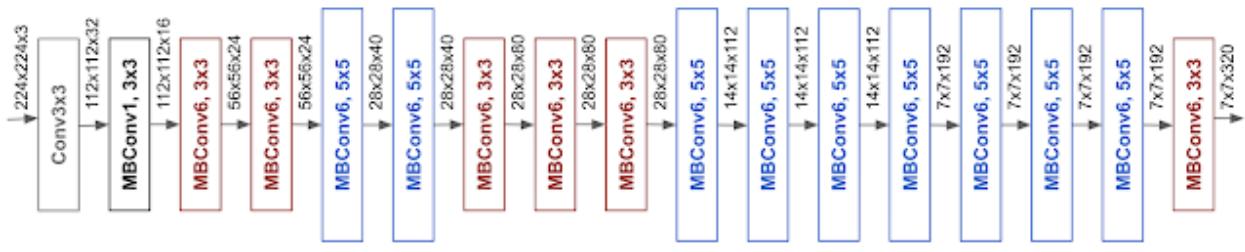
### Aperçu

J'ai choisi le modèle pré-entraîné EfficientNet. [EfficientNetV2L](#) est le modèle le plus performant une accuracy (justesse) de 97.5% (source: [keras.io](#)).



Source: [ai.googleblog.com](#)

Il introduit le “compound scaling”, ce qui améliore l’accuracy.



Source: ai.googleblog.com

## Architecture

Pour revenir au TP, c'est la version la plus petite d'EfficientNet qui a été utilisé. J'ai rajouté deux couches au modèle de base:

- Une couche d'AveragePooling afin de réduire les dimensions.
- Une couche dense avec une activation "softmax" pour prédire les classes.

```
In [21]: enet.model.summary()
Model: "sequential_1"
-----
Layer (type)          Output Shape         Param #
=====
efficientnetb0 (Functional)  (None, 7, 7, 1280)      4049571
global_average_pooling2d (GlobalAveragePooling2D)
dense_2 (Dense)        (None, 6)            7686
=====
Total params: 4,057,257
Trainable params: 4,015,234
Non-trainable params: 42,023
=====
```

### Architecture EfficientNet personnalisé

J'avais une accuracy très basse. J'ai donc "dégelé" les couches d'EfficientNet afin de mieux personnaliser l'entraînement.

```
def unfreeze(self):
    # Unfreeze base model layers
    for layer in self.base_model.layers:
        layer.trainable = True
```

### Unfreezing des couches

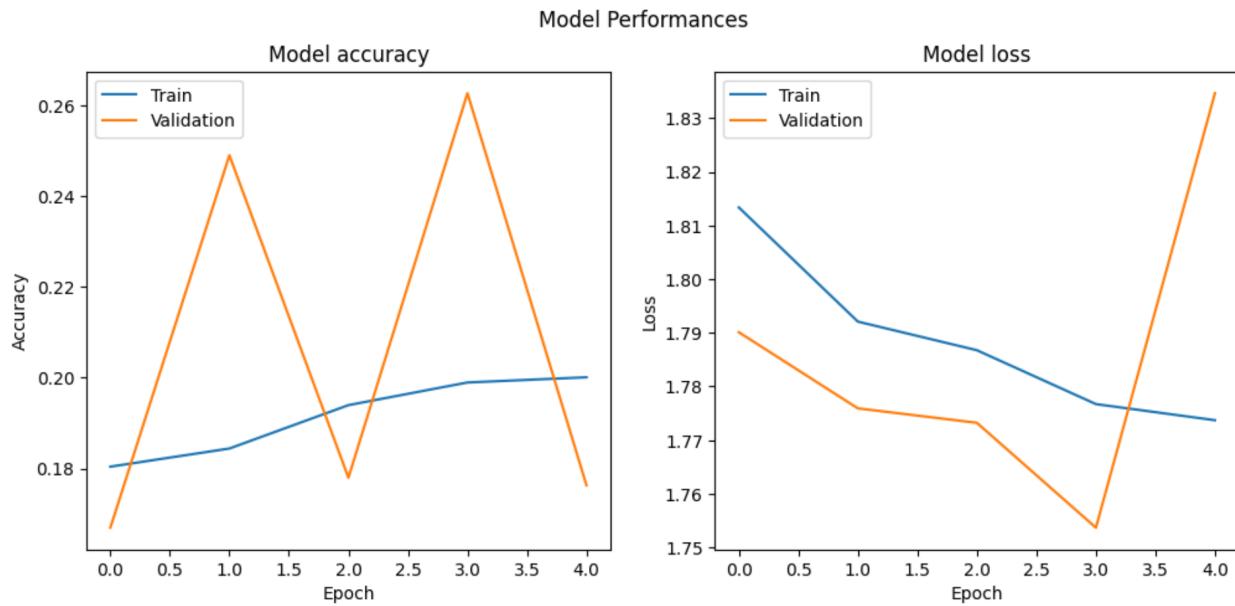
## Entraînement

J'ai lancé l'entraînement pour 15 epochs mais il s'est arrêté après 3 epochs.

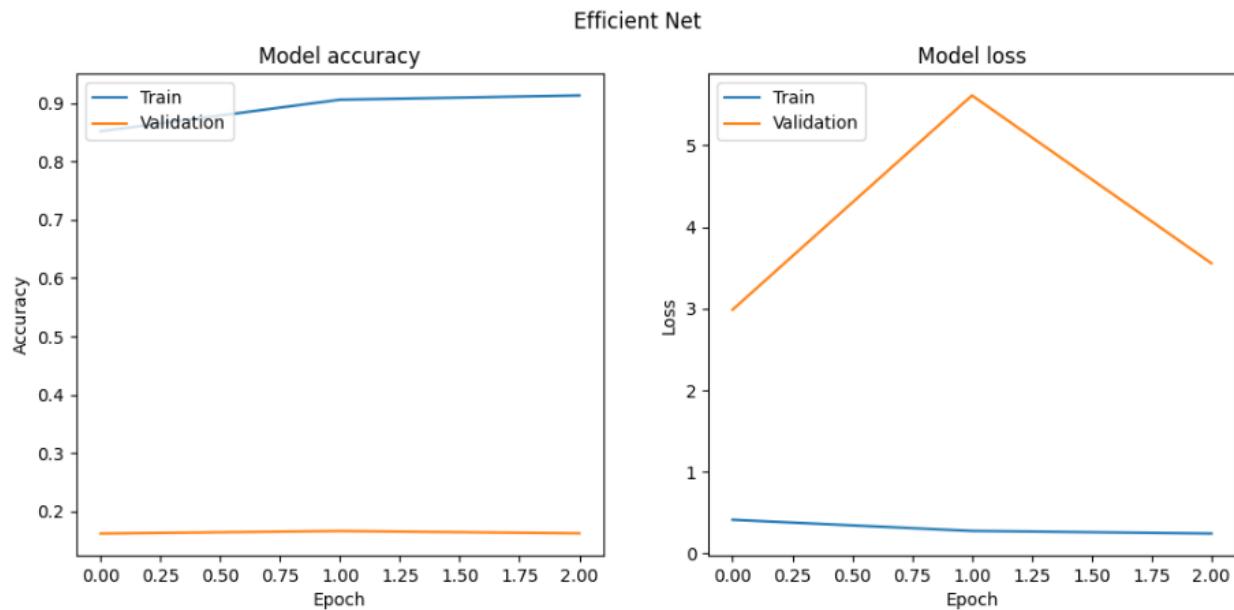
```
In [18]: enet.fit(train_data=train_data, val_data=val_data, epochs=15, patience=2)
Epoch 1/15
351/351 [=====] - 1697s 5s/step - loss: 0.4139 - accuracy: 0.8518 - val_loss: 2.9872 - val_accuracy: 0.1620
Epoch 2/15
351/351 [=====] - 1686s 5s/step - loss: 0.2773 - accuracy: 0.9060 - val_loss: 5.6120 - val_accuracy: 0.1663
Epoch 3/15
351/351 [=====] - 1708s 5s/step - loss: 0.2453 - accuracy: 0.9133 - val_loss: 3.5560 - val_accuracy: 0.1623
```

## Entraînement d'EfficientNet

Les résultats produits ont une accuracy très basse: moins de 20%. Le modèle est très instable et la fonction perte varie drastiquement avec le nombre d'epochs.



## Test 1: Justesse et perte du modèle



Test 2: Justesse et perte du modèle

# Conclusion

Ce TP m'a permis d'avoir une expérience pratique sur les réseaux de neurones convolutionnels. En outre, j'ai pu comprendre plus en profondeur le fonctionnement de Keras, et à quel point cette bibliothèque facilite la création des CNNs.

En expérimentant, j'ai découvert que plus les données d'entraînement était nombreuses, plus le modèle était précis.

Il est à noter que la qualité des données est encore plus importante. Je pense que le fait d'avoir un dataset pas très limpide a affecté les performances de mon modèle. Cependant, il m'était impossible de vérifier ces 17,000 images manuellement.

Comme vous l'avez mentionné, le maxpooling a ses inconvénients. J'aurais bien voulu présenter les performances du model en remplaçant les couches de maxpooling par convolutions avec strides. Malheureusement l'entraînement prend énormément de temps. Ces tests n'apparaîtront donc pas dans ce rapport.

Il faut aussi souligner que je n'ai pu lancer l'entraînement des modèles finaux que deux fois, car mon ordinateur était en surchauffe, après plusieurs jours de tests.