

Classification de courants artistiques

Deep Learning

MURATI Vidan & VAN Cyril

2022 - 2023

Contents

Présentation et choix des catégories	2
Version de Base	3
Réseau pré-entraîné	6
Améliorations possibles et limitations actuelles	10

Présentation et choix des catégories

Dans l'apprentissage du fonctionnement et de la construction de réseaux convolutionnels, nous avons travaillé sur la classification de courants artistiques. La classification se réalise sur 9 courants différents :

- Impressionnisme
- Surréalisme
- Cubisme
- Baroque
- Pointillisme ou Divisionnisme
- Pop Art
- Expressionnisme
- Maniérisme
- Néo-Classicisme

Ces courants possèdent chacun leurs caractéristiques, plus ou moins marquées visuellement. On notera que le choix des courants a été fait pour prendre des courants similaires visuellement et d'autres très différents. Cela rend la classification plus intéressante car elle comporte de la difficulté dans la différenciation de certain courant, mais reste tout de même abordable grâce aux courants opposés.

Exemple de courants avec des caractéristiques similaires :

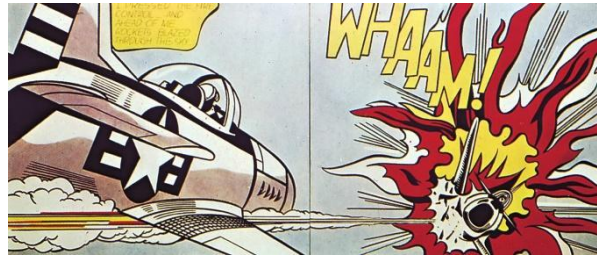


Le Correggio, *Martyrdom of Four Saints*, c. 1520, Galerie nationale de Parme - Maniérisme



John Singleton Copley, *The Forge of Vulcan*, 1754, Collection particulière - Néo-Classicisme

À contrario, ces courants sont très différents du Pop Art :



Roy Lichtenstein, *Whaam!*, 1963, Tate Modern - Pop Art

Version de Base

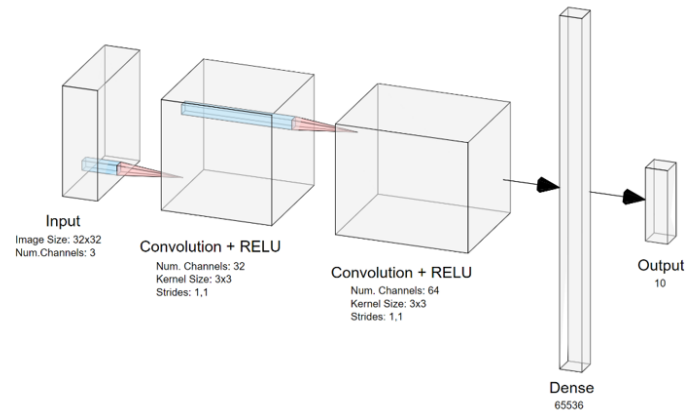
En utilisant la base construite dans l'exemple de classification de CIFAR10, nous avons construit notre premier réseau convolutionnel. En ajoutant une couche de Data Augmentation, cela a permis d'atteindre une meilleure précision. Néanmoins, la précision obtenue à la suite de nos différents modèles de base est loin d'être parfaite et reste perfectible.

Pour augmenter la taille de notre dataset de manière virtuelle, une couche de Data Augmentation a été réalisée. Elle consiste en une modification simple mais significative des échantillons de base et permet d'ajouter des images différentes dans notre dataset.

```
data_augmentation = keras.Sequential(  
    [  
        keras.layers.RandomFlip("horizontal"),  
        keras.layers.RandomRotation(0.1),  
        keras.layers.RandomZoom(0.2)  
    ]  
)  
  
inputs = keras.Input(shape=(64, 64, 3))
```

Couche de Data Augmentation

En dessous de cette couche, on peut apercevoir une instruction *inputs*. Elle permet de spécifier la taille des images en entrée du réseau convolutionnel. On va pouvoir recréer notre modèle en suivant le schéma suivant :



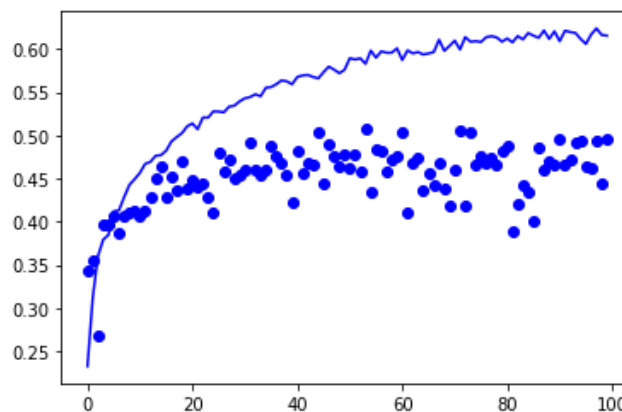
Cela nous permet d'obtenir le modèle ci-dessous, avec quelques couches supplémentaires :

```
x = data_augmentation(inputs)
x = keras.layers.Rescaling(1./255)(x)
x = keras.layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = keras.layers.MaxPooling2D(pool_size=2)(x)
x = keras.layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = keras.layers.MaxPooling2D(pool_size=2)(x)
x = keras.layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = keras.layers.MaxPooling2D(pool_size=2)(x)
x = keras.layers.Flatten()(x)
x = keras.layers.Dropout(0.5)(x)
outputs = keras.layers.Dense(9, activation="sigmoid")(x)
model_9 = keras.Model(inputs=inputs, outputs=outputs)
```

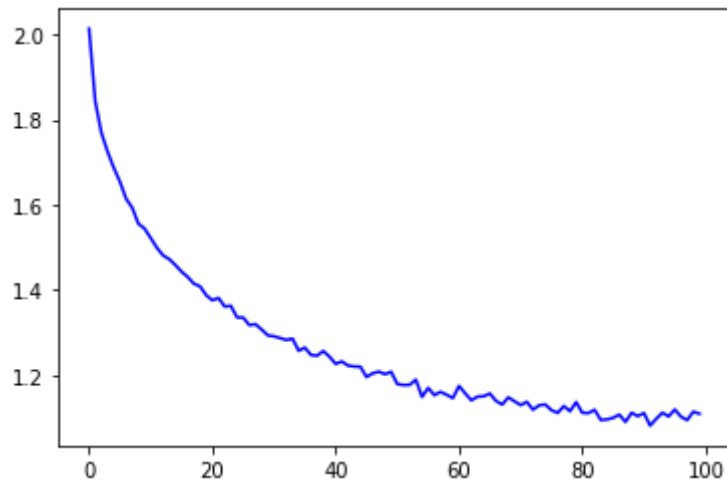
En testant le modèle sur le dataset de validation, il nous permet d'obtenir un précision d'environ 49% ce qui reste intéressant.

```
Test accuracy: 0.4961787164211273
Test loss: 1.6683247089385986
```

Précision d'apprentissage et précision de validation

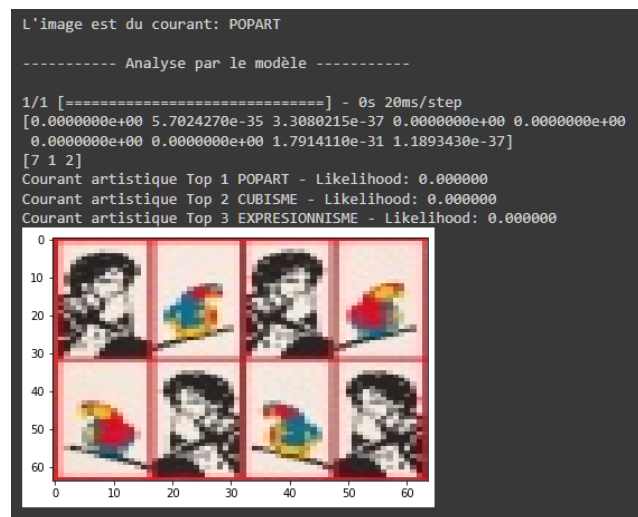


(En pointillés la précision sur le set d'entraînement / En continu sur le set de test)



Perte de la phase d'apprentissage

En regardant le Top 3 des catégories obtenues, on retombe sur le bon courant artistique dans la majorité des tirages.



Réseau pré-entraîné

Choix du modèle

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6

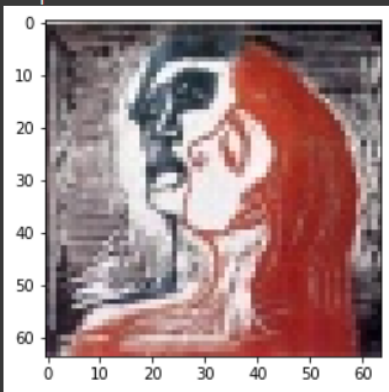
Keras propose un bon nombre de modèles aux spécificités différentes. Nous sommes partis sur le modèle VGG16 qui permet une précision satisfaisante. C'est un réseau convolutionnel composé de 13 couches convolutionnelles, 5 couches max-pooling et 3 couches FC. Il a été utilisé pour gagner la compétition ILSVR (ImageNet Large Scale Visual Recognition Challenge) en 2014 et est toujours à ce jour un réseau très performant.

Nous sommes partis sur un pré entraînement des poids du réseau sur la base d'images ImageNet et avons effectués quelques tests de prédictions sur notre dataset

```
P = decode_predictions(preds)
for (i, (imagenetID, label, prob)) in enumerate(P[0]):
    print("{}: {:.2f}%".format(i + 1, label, prob * 100))

plt.imshow(x_test[500], interpolation='nearest')
plt.show()
```

1. cowboy_boot: 20.12%
2. throne: 12.11%
3. comic_book: 4.29%
4. book_jacket: 3.86%
5. pedestal: 3.55%



Pour les stratégies de réentraînement je suis tout d'abord parti sur du FineTuning, soit l'idée de modifier un réseau pré entraîné pour qu'il soit adapté à notre besoin. J'ai donc pris une base du réseau VGG16 avec les poids ImageNet sans les dernières couches qui ne nous intéressent pas.

On veut garder les premières couches du réseau car plus une couche est basse dans le réseau plus ce qu'elle en ressort est "général" / Utilisable pour d'autres besoins. Ces couches concernent donc surtout la partie Feature Extractions qui consiste à extraire des caractéristiques principales de notre jeu de données.

```
base_model = VGG16(weights="imagenet",include_top=False,input_shape=(64,64,3))
```

Le paramètre include_top = False permet de spécifier que je ne veux pas des 3 dernières couches FC du réseau. (Parce que nous voulons pouvoir ajouter mes propres couches)

```
x = base_model.output
x = keras.layers.Flatten()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(9, activation='softmax')(x)
```

Nous avons ensuite ajouté 3 couches au réseau et avons commencé un entraînement du réseau en "freezant" toutes les autres couches du réseau d'origine : Pour ne pas avoir à tout réentraîner et d'économiser du temps de calcul. Nous atteignons 48.38% de précision à cette étape.

Nous avons ensuite rajouté une couche de Data Augmentation en entrée du réseau, car nos datasets sont de tailles relativement petites pour cette tâche, donc nous avons entraîné notre réseau de la même manière que précédemment mais cette fois avec cette couche de Data Augmentation.

Pour le fine tuning nous avons essayé de voir les différentes couches qui composent notre réseau de base pour savoir lesquelles "défreeze" / "ajouter dans l'entraînement"

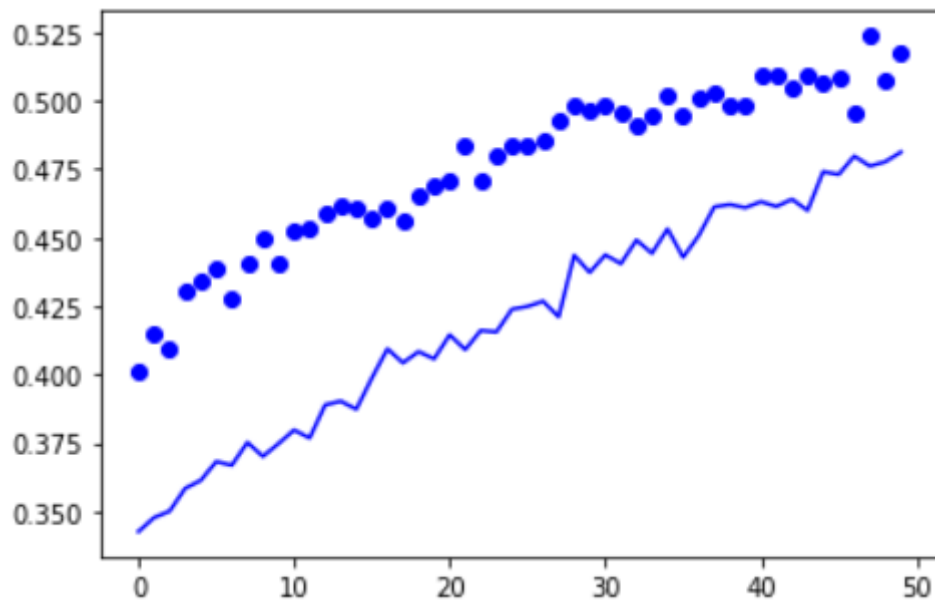
```
0 input_9
1 block1_conv1
2 block1_conv2
3 block1_pool
4 block2_conv1
5 block2_conv2
6 block2_pool
7 block3_conv1
8 block3_conv2
9 block3_conv3
10 block3_pool
11 block4_conv1
12 block4_conv2
13 block4_conv3
14 block4_pool
15 block5_conv1
16 block5_conv2
17 block5_conv3
18 block5_pool
```


Nous sommes donc partis sur les 3 dernières couches convolutionnelles et nous avons entraîné encore une fois notre modèle. Nous atteignons ici une précision d'un peu plus de 51%

Finalement nous avons essayé une autre stratégie en ajoutant d'autres couches au modèle de base, et en réitérant le processus

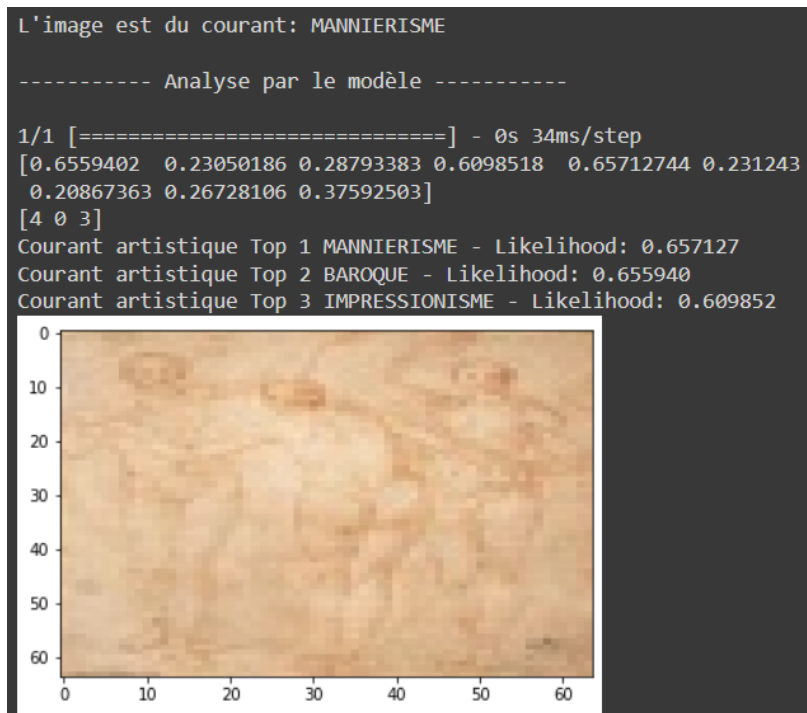
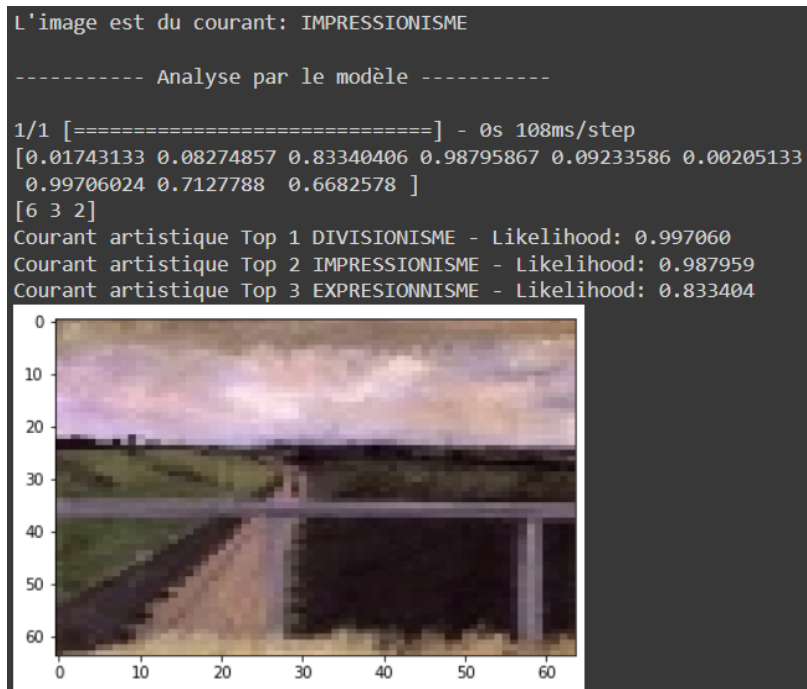
```
# Data augmentation
x = data_augmentation(inputs)
x = base_model(x, training=False)
x = keras.layers.MaxPooling2D(pool_size=2)(x)
x = keras.layers.Flatten()(x)
x = Dense(128, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)
predictions = Dense(9, activation='sigmoid')(x)
```

Mais même ici la précision ne dépasse pas 51%



(En pointillés la précision sur le set d'entraînement / En continu sur le set de test)

Sur ces prédictions le courant qui doit être reconnu est donné en haut. On a ensuite un top 3 des mouvements dans lequel il situe le + l'œuvre. Ici nous n'avons que 2 exemples mais le mouvement à reconnaître est très souvent dans le top 3.



Améliorations possibles et limitations actuelles

Concernant notre jeu de données, il faut savoir que les œuvres qui le composent sont des œuvres de supports très variés, dessins / peintures / sculptures. C'est une des difficultés de notre sujet puisque notre réseau intègre dans son apprentissage les supports en question. Alors que la représentation ou surreprésentation d'un support dans un mouvement peut suggérer une mauvaise interprétation d'une œuvre uniquement parce que son support est surreprésenté dans un autre mouvement de notre datasets. Et la phase de Data Augmentation ne permet pas de pallier ce problème.

Aussi due aux limitations de google drive nos images sont d'une taille relativement petite (64x64) ne facilitant pas la reconnaissance de features.