Projet Final: Classification d'images - Personnages de Nintendo

Par François Dewynter, Jean-Baptiste Marchetti-Waternaux , et Ludovic Sanne

Plan du projet

- Introduction
- Récupération et Nettoyage des données
- Préprocessing et Data Augmentation
- Modèle(s)
- Conclusion

Introduction

- Le But du projet est de réaliser une classification d'images permettant de reconnaître 15 personnages de l'univers Nintendo.

Récupération et nettoyage des données

- Images téléchargées via Google images, mais aussi à partir de photos prises lors de parties.
- Pour les images venant de Google, on utilise la commande Javascript suivante : **(urls=Array.from(document.querySelectorAll('.rg_i')).map(el=> el.hasAttribute('data-src')?el.getAttribute('data-src'):el.getAttribute('data-iurl')); window.open('data:text/csv;charset=utf-8,' + escape(urls.join('\n')));)** .
- Avant nettoyage des données : ***2972 données***.
- Pour nettoyer les données, Il a fallu parcourir chaque fichier dans Colab, et supprimer les images qui ne correspondaient pas à ce que l'on voulait.
- Après nettoyage de données et ajout d'autres images : ***4206 données***.

Preprocessing et Data Augmentation

1) Création de 2 datasets Train et Test

2) Data Augmentation pour les dossiers ayant moins de ***200 images*** (Donkey Kong, Captain Falcon, Phoenix Wright et Professor Layton)

3) Data splitting pour x_train, y_train, x_test et y_test

Données pour l'échantillon Train: ***3167 images***

Données pour l'échantillon Test : ***1321 images***

Data Augmentation (exemple de code)

```
import Augmentor
path = "/content/drive/My Drive/Images Projet Final/train/Output train"
path_list = os.listdir(path)
for img in path_list :
  if len(os.listdir(path+'/'+img)) < 200:
    p=Augmentor.Pipeline(path+'/'+img)
    p.rotate(probability = 1, max_left_rotation=10, max_right_rotation=10)
    p.zoom(probability=0.6, min_factor=1.2, max_factor=1.7)
    p.sample(10)
    p.process()
```

Nombre d'images par classe

	nb images avant data augmentation et avant splitting	nb images Outpout train (après data augmentation)	nb images Outpout test (après data augmentation)
Mario	454	317	137
Luigi	346	242	104
Daisy	236	175	76
Yoshi	206	152	66
Harmonie	207	154	68
Waluigi	236	175	76
Donkey Kong	113	133	39
Tiny Kong	326	227	101
Fox	500	350	150
Pikachu	402	281	121
Professor Layton	102	115	31
Toad	433	303	130
Link	453	317	136
Phoenix Wright	100	105	33
Captain Falcon	159	121	53

Modèles utilisés et modèles écartés

- Transfert learning : ResNet18, ResNet 34, **ResNet 50,** Resnet101, Resnet152, VGG16, VGG19
- Metrics utilisé: Accuracy
- Modèles écartés: Yolo, InceptionV3

	ResNet18	Resnet 34 (modifié)	**Resnet 50 (modifié)**	ResNet101	ResNet 152	VGG16 (modifié)	VGG19 (modifié)
Train	0.9363057613	0.9442675114	**0.9442675114**	0.9617834687	0.9681528807	0.9051612616	0.9041935205
Test	0.8582375646	0.8888888955	**0.9195402265**	0.9118773937	0.8773946166	0.7546153665	0.739230752

ResNet 50, 101 et 152

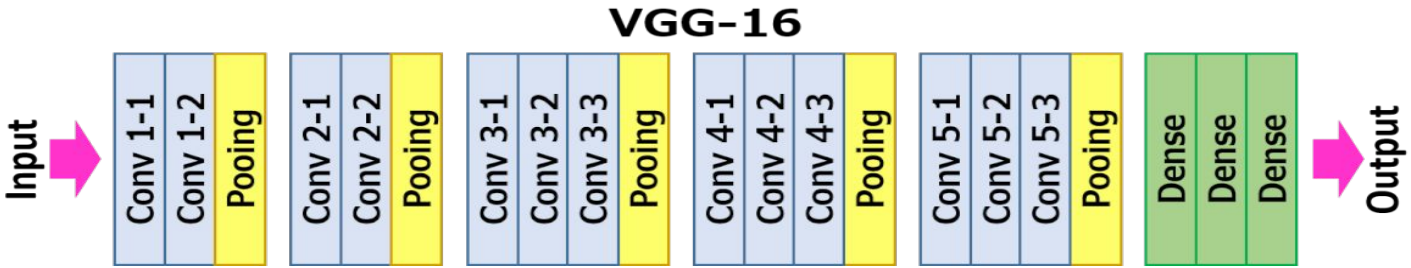
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Chaque bloc ResNet a deux couches de profondeur (utilisées dans les petits réseaux comme ResNet 18, 34) et/ou trois couches de profondeur (ResNet 50, 101, 152).

- ResNet50 : Chaque bloc à 2 couches est remplacé dans le 34-layer net par un bloc goulot d'étranglement à 3 couches, ce qui donne un ResNet à 50 couches (voir tableau ci-dessus).
- ResNet101 et 152: utilisent plus de blocs de 3 couches. Même après l'augmentation de la profondeur, le ResNet 152 (11,3 milliards de FLOPs) présente une complexité inférieure aux réseaux VGG-16/19 (15,3/19,6 milliards de FLOPs)

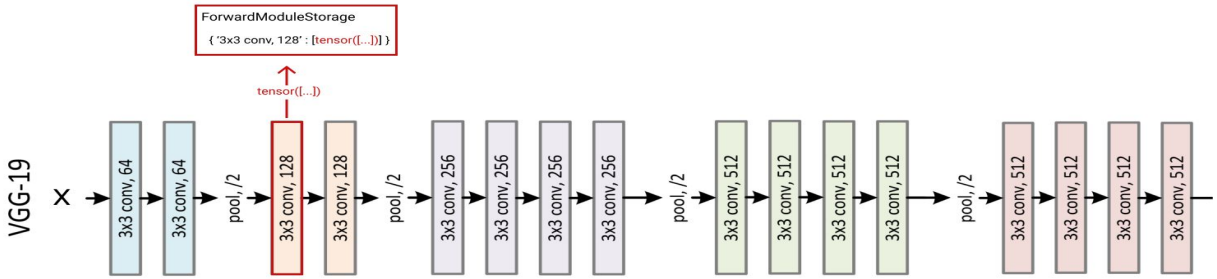
Modèle : VGG16

- modèle de réseau neuronal convolutionnel composé de 16 couches (13 couches de convolution et 3 fully-connected)
- prise en entrée d'une image en couleurs de taille 224 × 224 px, classifiée dans une des 1000 classes.
- Ce dernier renvoie donc un vecteur de taille 1000, contenant les probabilités d'appartenance à chacune des classes.
- Résultats (voir diapo des modèles utilisés).



Modèle VGG19

- modèle de réseau neuronal convolutionnel composé de 19 couches (16 couches de convolution, 3 fully-connected, avec 5 couches MaxPool, et 1 couche SoftMax)
- prise en entrée d'une image en couleurs de taille 224 × 224 px, classifiée dans une des 1000 classes. (comme pour le modèle VGG16)
- Résultats (voir diapo des modèles utilisés).



Conclusion

Points à améliorer:

- Récupérer plus d'images d'un personnage mieux connu (peu d'images pour Tiny Kong sur google, par exemple)
- Essayer d'autres modèles (ex: AlexNet,...)
- Dans le cas des modèles VGG16, et VGG19, faire une cross-validatiton afin de résoudre le problème de l'overfitting
- Application Streamlit ou Flask

Bilan des compétences:

- Amélioration de la collecte et le nettoyage des données(images).
- Différentes méthodes de Data Augmentation
- Compréhension de nouveaux modèles de transfer learning