



## Rapport final

Projet d'apprentissage profond

*Reconnaissance de circuits dans Mario Kart*

SADURNI Thomas

FAN Yanghai

MORATA Jules

ROUX Thibault

Département Sciences du Numérique - Filière Image et Multimédia  
2020-2021

# Table des matières

<b>1 Descriptif du projet</b>	<b>3</b>
1.1 Liste des différentes classes (circuits) . . . . .	3
1.2 Liste des différents jeux Mario Kart utilisés . . . . .	5
1.3 Correspondance Circuit-Jeu . . . . .	5
<b>2 Base de données</b>	<b>5</b>
2.1 Lien vers notre base de données . . . . .	5
2.2 Construction de la base de données . . . . .	5
2.3 Etapes réalisées pour construire nos images . . . . .	5
2.3.1 Utilisation d'un émulateur . . . . .	6
2.3.2 Utilisation de modificateurs de jeux . . . . .	6
2.3.3 Enregistrement vidéo des courses . . . . .	7
2.3.4 Exportation des vidéos en séquences d'images . . . . .	7
2.4 Exemples d'images présentes dans notre base de données . . . . .	7
2.5 Partitionnement des images dans les différents ensembles . . . . .	8
2.6 Attentes quant aux résultats . . . . .	9
<b>3 Résolution et Architecture du réseau</b>	<b>10</b>
3.1 Accès à notre code . . . . .	10
3.2 Première approche . . . . .	10
3.2.1 Architecture . . . . .	10
3.2.2 Autres hyperparamètres et méthodes d'apprentissage . . . . .	10
3.3 Deuxième approche : complexification du problème . . . . .	11
3.4 Troisième approche : Transfer Learning . . . . .	11
3.4.1 Fine-Tuning du réseau VGG19 . . . . .	11
3.4.2 Extraction de caractéristiques . . . . .	12
<b>4 Analyse des résultats</b>	<b>13</b>
4.1 Première approche . . . . .	13
4.1.1 Métriques obtenues . . . . .	13
4.1.2 Analyse des erreurs . . . . .	13
4.2 Deuxième approche . . . . .	17
4.2.1 Métriques obtenues . . . . .	17
4.2.2 Analyse des erreurs . . . . .	18
4.3 Troisième approche : Transfer Learning . . . . .	20
4.3.1 Fine-Tuning . . . . .	20
4.3.2 Extraction de caractéristiques . . . . .	22
<b>5 Conclusion</b>	<b>24</b>

# 1 Descriptif du projet

Notre projet est de construire un réseau de neurones capable de reconnaître différents circuits sur différents jeux Mario Kart. Le but est de reconnaître sur quel circuit on se trouve, à partir d'une capture d'écran du jeu. De plus, pour augmenter la difficulté, nous prenons des circuits qui sont présents dans au moins deux jeux Mario Kart différents. Ce qui serait intéressant, c'est également de regarder si le réseau arrive à trouver le bon circuit sur un jeu, alors que les données d'apprentissage sur ce circuit ne proviennent que d'autres jeux.

## 1.1 Liste des différentes classes (circuits)

Notre réseau devra être capable d'assigner à une image une classe. Chaque classe représente un circuit. Nous avons choisi d'utiliser 13 circuits. Chaque circuit est présent dans différents jeux Mario Kart, sur différentes consoles. Voici la liste des différents circuits que nous avons sélectionnés (noms anglais) :

1. Peach Beach
2. Ghost Valley 2
3. Maple Treeway
4. Yoshi Falls
5. SNES Rainbow Road
6. Yoshi Circuit
7. Delfino Square
8. Mario Circuit 3
9. Baby Park
10. Grumble Volcano
11. Waluigi Pinball
12. Wario's Gold Mine
13. Moo Moo Meadows

Les images suivantes ne seront pas celles utilisées lors de l'apprentissage de notre réseau de neurones et sont simplement présentes à titre d'illustration.



FIGURE 1 – Peach Beach



FIGURE 2 – Ghost Valley 2



FIGURE 3 – Maple Treeway



FIGURE 4 – Yoshi Falls

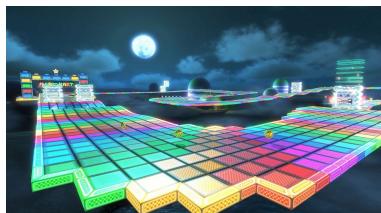


FIGURE 5 – SNES Rainbow Road



FIGURE 6 – Yoshi Circuit



FIGURE 7 – Delfino Square



FIGURE 8 – Mario Circuit 3



FIGURE 9 – Baby Park



FIGURE 10 – Grumble Volcano



FIGURE 11 – Waluigi Pinball



FIGURE 12 – Wario's Gold Mine



FIGURE 13 – Moo Moo Meadows

## 1.2 Liste des différents jeux Mario Kart utilisés

Comme expliqué précédemment, pour augmenter le challenge de notre projet, nous avons choisi d'utiliser plusieurs jeux différents car en effet certains circuits sont récurrents au fil des jeux de cette franchise. Ainsi les circuits choisis dans la section précédentes sont présents dans au moins deux jeux différents. Les jeux utilisés sont les suivants :

1. Super Mario Kart (Sorti en 1992 sur SNES)
2. Mario Kart : Double Dash (Sorti en 2003 sur Gamecube)
3. Mario Kart DS (Sorti en 2005 sur Nintendo DS)
4. Mario Kart Wii (Sorti en 2008 sur Wii)
5. Mario Kart 7 (Sorti en 2011 sur Nintendo 3DS)
6. Mario Kart 8 (Sorti en 2014 sur Wii U)

## 1.3 Correspondance Circuit-Jeu

Voici un tableau montrant dans quels jeux les différents circuits sont présents.

	SMK	MKDD	MKDS	MKW	MK7	MK8
Peach Beach		✓		✓		
Ghost Valley 2	✓			✓		
Maple Treeway				✓	✓	
Yoshi Falls			✓	✓		
SNES RR	✓				✓	✓
Yoshi Circuit		✓	✓			✓
Delfino Square			✓	✓		
Mario Circuit 3	✓			✓		
Baby Park		✓	✓			✓
Grumble Volcano				✓		✓
Waluigi Pinball			✓		✓	
Wario's Gold Mine				✓		✓
Moo Moo Meadows				✓		✓

## 2 Base de données

### 2.1 Lien vers notre base de données

Nous avons choisi de stocker notre base de données sur GitHub. Elle est accessible à cet endroit : <https://github.com/thibaultroux1/mariokart>

### 2.2 Construction de la base de données

Nous avons décidé de faire nos propres images, sans en prendre aucune sur internet, pour plusieurs raisons que nous allons développer dans cette partie.

### 2.3 Etapes réalisées pour construire nos images

Puisque nous avons choisi de réaliser nous-mêmes toutes nos images, il nous a fallu utiliser différentes méthodes afin de pouvoir jouer aux différents jeux et en extraire des images pertinentes.

### 2.3.1 Utilisation d'un émulateur

La première étape a été d'émuler les différents jeux à l'aide d'émulateurs, qui permettent de simuler le fonctionnement d'une console, et ainsi de jouer à des jeux sur son ordinateur comme si l'on jouait sur cette console. Comme nous avons décidé d'utiliser plusieurs jeux sur différentes consoles, nous avons eu besoin de différents émulateurs.

1. *nes9x* pour émuler la Super NES
2. *Desmume* pour émuler la Nintendo DS
3. *Citra* pour émuler la Nintendo 3DS
4. *Dolphin* pour émuler la Wii et la GameCube
5. *Cemu* pour émuler la Wii U



FIGURE 14 – Logo Citra



FIGURE 15 – Logo Dolphin



FIGURE 16 – Logo Cemu

Nous avons ensuite téléchargé les jeux aux format ROM ou ISO selon les consoles pour pouvoir y jouer sur ces émulateurs.

### 2.3.2 Utilisation de modificateurs de jeux

Nous avons pensé à un problème qui aurait pu se poser lors de l'apprentissage du réseau. Dans les jeux Mario Kart, une minimap est présente et permet au joueur de voir la forme du circuit. Cela est un grand indicateur et un réseau de neurones pourrait reconnaître le circuit très simplement à partir de cette minimap. Nous avons donc trouvé un moyen de l'enlever grâce à des codes de triche que nous pouvons utiliser dans les émulateurs notamment. Ce code est appelé "No HUD" et permet d'enlever toutes les informations affichées à l'écran telles que les objets utilisables, le timer, la minimap et enfin le numéro du tour. Ces informations sont des informations parasites dans notre cas, et les enlever nous permet de nous concentrer sur l'essentiel, à savoir l'apparence du circuit.

Voici une comparaison qui montre les différences entre les deux aspects du jeu, selon si l'on garde l'HUD (17), ou si on l'enlève (18).



FIGURE 17 – Avec HUD



FIGURE 18 – Sans HUD

### 2.3.3 Enregistrement vidéo des courses

Ensuite, nous avons enregistré des vidéos sur chaque émulateur. Le principe est de faire 3 tours du circuit en explorant au mieux tous les recoins de la carte. Nous avons également choisi d'utiliser toujours le même personnage, à savoir Bébé Mario, et un kart basique, afin que l'apprentissage se fasse bien grâce au circuit et non grâce au personnage.



FIGURE 19 – Bébé Mario

### 2.3.4 Exportation des vidéos en séquences d'images

Après avoir réalisé ces vidéos, nous avons utilisé le logiciel VirtualDub qui nous a permis de prendre des images à intervalles réguliers dans ces vidéos.

## 2.4 Exemples d'images présentes dans notre base de données

Pour finir, voici quelques images de notre base de données.



FIGURE 20 – Grumble Volcano (Wii)



FIGURE 21 – Grumble Volcano (Wii U)



FIGURE 22 – Ghost Valley 2 (SNES)



FIGURE 23 – Chost Valley 2 (Wii)



FIGURE 24 – Wario's Gold Mine (Wii U)



FIGURE 25 – Wario's Gold Mine (Wii)

## 2.5 Partitionnement des images dans les différents ensembles

Nous avons choisi de partitionner les images de la façon suivante. 70% des images sont dans l'ensemble d'apprentissage, 15% dans l'ensemble de validation et les 15% restants dans l'ensemble de test. Nous ferons aussi en sorte que si un circuit est présent dans plusieurs jeux, chaque jeu est représenté de façon équitable dans chaque ensemble.

## 2.6 Attentes quant aux résultats

Nous nous attendons à reconnaître un certain nombre de circuits de façon convenable car certains des circuits ont des caractéristiques uniques, que ce soit en terme de couleur ou de formes. En revanche, d'autres circuits peuvent poser problème par leurs similarités (on peut prendre l'exemple de Maple Treeway et Grumble Volcano). On se demande également si les différences de jeux vont entraîner ou non une difficulté trop élevée en terme d'apprentissage. Nous pensons qu'il est plutôt facile d'entraîner un réseau de neurones sur tous les circuits d'un même jeu, mais la difficulté vient ici du fait que plusieurs jeux interviennent. Nous ne savons donc pas trop si notre réseau sera efficace sur tous les circuits, notamment ceux qui sont présents dans plusieurs jeux éloignés en terme de date de sortie (Super Mario Kart par exemple).

### 3 Résolution et Architecture du réseau

#### 3.1 Accès à notre code

Nous avons réalisé ce projet sur Python à l'aide de la librairie Keras de Tensorflow. Nous avons travaillé sur un notebook Google Colab. Tous nos codes sont accessibles à ce lien : <https://colab.research.google.com/drive/1SoRCT7gycEVt3AmNv9a64eAt3nG4JeYw?usp=sharing>

#### 3.2 Première approche

##### 3.2.1 Architecture

Dans un premier temps, nous avons le réseau dont l'architecture est donnée dans la figure suivante :

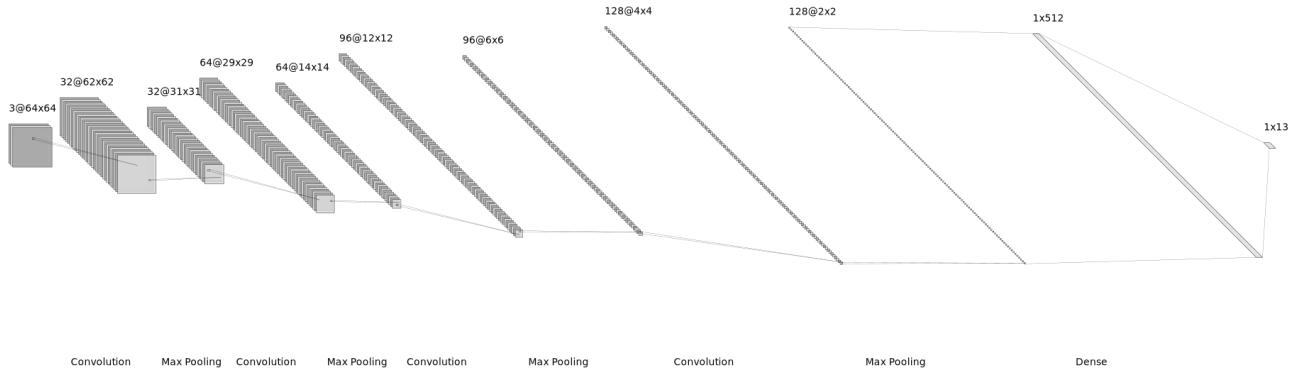


FIGURE 26 – Architecture de notre premier réseau

Notre réseau est donc composé d'un enchaînement de couches de convolution et de max-pooling. Les convolutions ont un stride de 3x3 et les couches de max-pooling de 2x2.

Sur la figure précédente, par manque de place, nous n'avons pas affiché la couche de mise à plat (Flatten) entre la dernière couche de max-pooling et la première couche dense.

##### 3.2.2 Autres hyperparamètres et méthodes d'apprentissage

Nous avons utilisé l'optimiseur Adam avec un taux d'apprentissage de 0.0003. Nous avions initialement utilisé une valeur plus haute du taux d'apprentissage mais cela engendrait un mauvais fonctionnement du réseau. En effet, ce dernier atteignait assez rapidement de très bons résultats mais ceci se dégradaient d'un coup à partir d'une certaine époque, tombant à une précision d'une trentaine de pourcents, sans jamais remonter. L'entraînement s'est fait sur 100 epochs et la taille des batchs était de 16. La fonction de coût utilisée quant à elle est l'entropie croisée.

### 3.3 Deuxième approche : complexification du problème

Dans un deuxième temps, nous avons décidé d'essayer de travailler avec des images en niveau de gris pour voir si nous arrivions à de bons résultats sur un problème plus difficile étant donné la réduction de la quantité d'informations disponibles. D'une part, cela nous a semblé pertinent car, comme cela sera expliqué dans la partie sur l'analyse des résultats et notamment des erreurs, notre premier réseau s'appuie très fortement sur la colorimétrie pour prédire un résultat. Nous nous sommes alors demandé si nous pouvions construire un réseau se basant plus sur les objets et formes caractéristiques des circuits. D'autre part, une telle contrainte semble pertinente dans le cas d'une application réelle, car elle permettrait de gagner une grande place de stockage au niveau des images à évaluer, car leur taille pourrait être divisée par 3. Ce n'est cependant pas le cas dans notre implémentation, car, pour des soucis de pratique, nous avons des images avec 3 canaux ayant les mêmes valeurs (le niveau de gris), néanmoins si des résultats corrects sont obtenus cela voudra dire qu'ils peuvent aussi être obtenus pour des images à un seul canal, modulo une restructuration de notre réseau pour qu'il soit adapté à la nouvelle forme des données. Dans un premier temps nous avons choisi d'essayer cette nouvelle approche avec le même réseau que celui décrit précédemment. D'autres options ont aussi été explorées comme, par exemple, l'utilisation d'un réseau de type U-Net, mais, les résultats étant moins bons, nous avons préféré rester sur l'architecture décrite ci-dessus.

### 3.4 Troisième approche : Transfer Learning

Nous avons ensuite procédé à des méthodes de Transfer Learning pour observer si nous obtenons des meilleurs résultats sur les images en niveau de gris qu'avec notre réseau initial.

#### 3.4.1 Fine-Tuning du réseau VGG19

Nous avons en premier lieu utilisé le Fine-Tuning pour adapter le réseau VGG19 à notre étude.

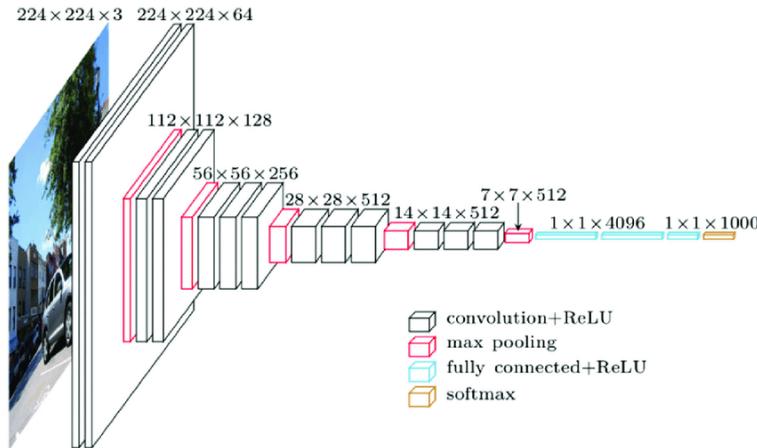


FIGURE 27 – Architecture du réseau VGG19

Nous avons alors utilisé les couches de convolution et de max pooling du réseau, puis ajouté une couche dense à 512 neurones, avant de relier cette dernière à la couche de sortie composée de 13 neurones.

La première étape consiste en le gel des couches de convolution et de max pooling de VGG19 pour ne mettre à jour que les poids de nos couches denses.

Ensuite une fois le réseau entraîné sur nos données (optimiseur Adam, taux d'apprentissage 0.001), on dégèle les 5 dernières couches de VGG, et on réentraîne le réseau avec un taux d'apprentissage très faible (optimiseur SGD, learning rate 0.0001).

Cependant, comme nous le verrons dans la partie analyse des résultats, ces derniers ne sont pas probants. C'est pourquoi nous avons choisi d'essayer l'extraction de caractéristiques, afin d'extraire les caractéristiques plus dans les premières couches de convolution et max pooling de VGG, pour ne garder que les informations les plus générales.

### 3.4.2 Extraction de caractéristiques

Nous avons alors extrait les caractéristiques de nos données grâce au réseau VGG19, en prenant les données à la sortie d'une certaine couche de convolution (ou max pooling) de VGG19. Nous avons réalisé ceci pour plusieurs des couches de convolution afin d'obtenir les meilleurs résultats.

Les couches utilisées comme sortie pour l'extraction sont les couches n°3 (pooling), n°6 (pooling), n°8 (convolution), n°11 (pooling), n°16 (pooling), n°17 (convolution), n°21 (pooling).

Une fois les caractéristiques extraites, nous avons entraîné un réseau composé de deux couches denses, l'une de 512 neurones, l'autre de 256, puis de la couche de sortie de 13 neurones, complètement connectée à la précédente couche.

Nous avons utilisé une régularisation l2 sur ces couches denses, en faisant varier l2 entre 0.001 et 0.007.

L'optimiseur utilisé est Adam avec un taux d'apprentissage de 0.0003.

## 4 Analyse des résultats

### 4.1 Première approche

#### 4.1.1 Métriques obtenues

Pour la première version de notre projet, dans laquelle nous utilisions les images RGB à disposition, nous obtenons des résultats très satisfaisants avec une précision de 100% sur les données d'apprentissage, ce qui montre que notre réseau arrive parfaitement à apprendre les données qui lui sont fournies. Pour ce qui est des données de validation ou encore des données de test, on obtient une précision autour de 99% ce qui montre la capacité de notre réseau à généraliser à partir de son apprentissage. Observons les courbes obtenues pour les données d'apprentissage et de validation :

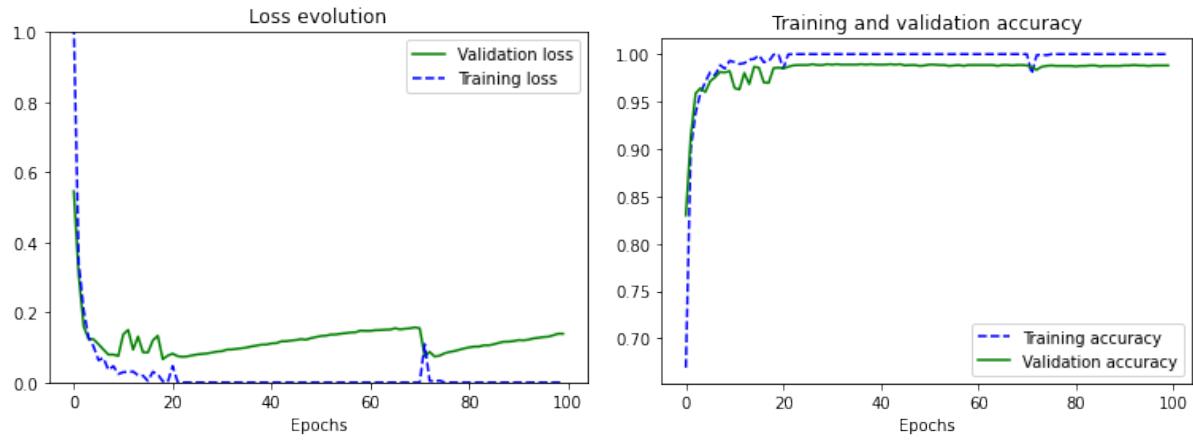


FIGURE 28 – Loss function pour les données d'apprentissage et de validation dans le premier réseau

FIGURE 29 – Accuracy function pour les données d'apprentissage et de validation dans le premier réseau

On observe sur ces courbes que la progression du réseau se fait en 3 étapes : d'abord une première phase assez courte pendant laquelle le réseau progresse très vite (environ 10 époques), ensuite on remarque une légère période d'oscillations, surtout au niveau de la validation, et enfin les différentes fonctions stagnent une fois ces oscillations passées. Néanmoins, ce comportement est moins respecté pour la fonction de coût (loss) des données de validation qui a plutôt tendance à augmenter une fois le palier d'oscillations franchi. Ceci montre un léger surapprentissage de notre réseau où lequel la généralisation devient plus difficile. Cela se confirme au vu de l'évolution simultanée des 2 courbes aux alentours de la 70<sup>ème</sup> époque, on remarque en effet que la courbe des données d'entraînement augmente, entraînant au même moment une diminution de la courbe des données de validation, ce qui confirme qu'en se détachant un peu des données d'apprentissage le réseau généralise mieux.

#### 4.1.2 Analyse des erreurs

Nous avons ensuite regardé quelles images notre réseau classait mal pour comprendre et expliquer ces quelques erreurs de classification.

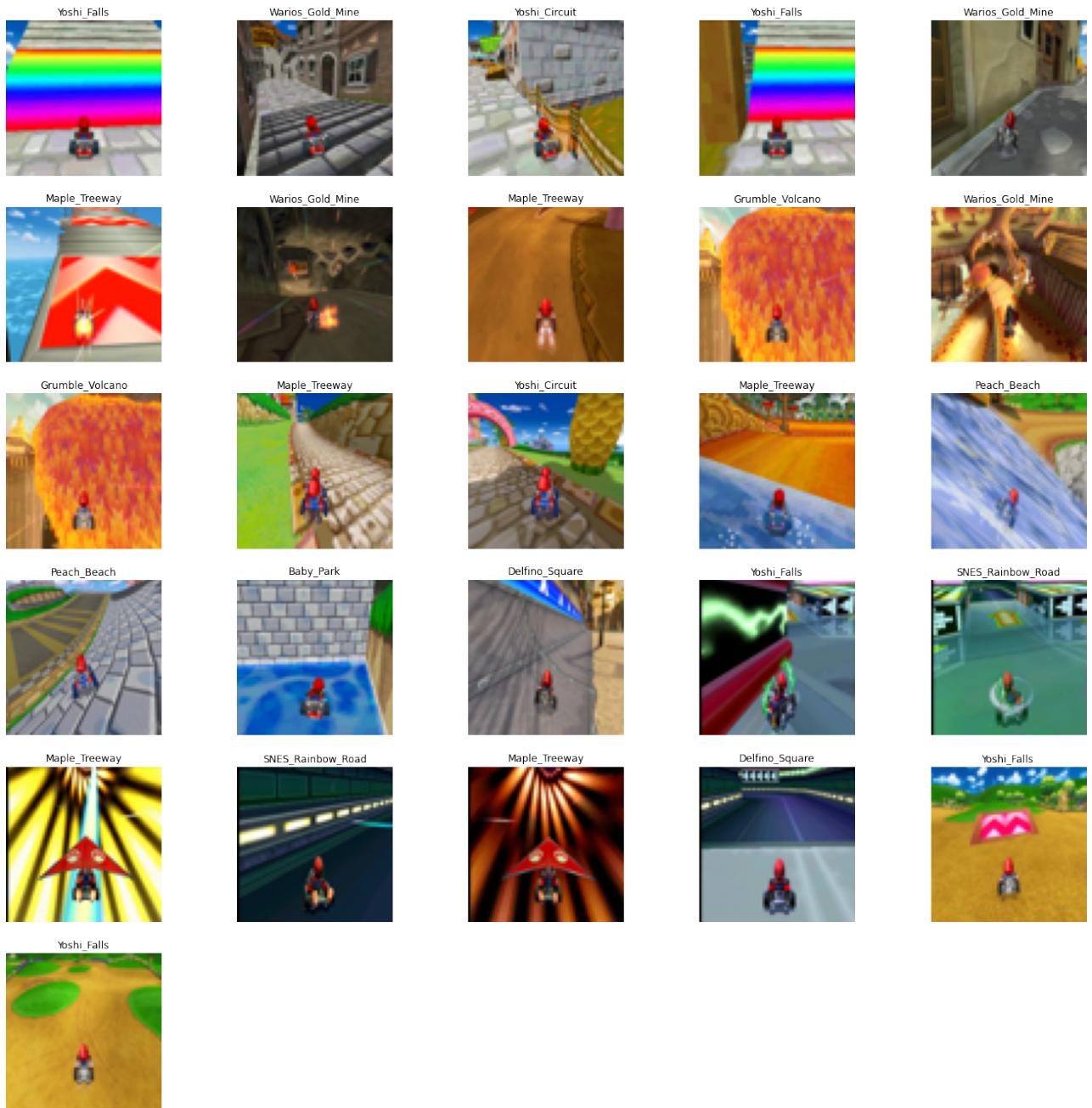


FIGURE 30 – Images de l'ensemble de validation mal classifiées

La figure 30 affiche l'ensemble des images (de l'ensemble de validation) mal classées, ainsi que la (mauvaise) classe à laquelle chaque image a été attribuée. On remarque en analysant les erreurs de classification que notre réseau se base énormément sur les couleurs des images. En effet, essayons d'expliquer ces différentes erreurs.



FIGURE 31 – Image de Delfino Square de l’ensemble de validation, mal classée (dans Yoshi Falls)



FIGURE 32 – Image de Yoshi Falls de la base d’apprentissage

L’image de la figure 31 est classée en Yoshi Falls alors qu’elle appartient à la classe Delfino Square. Cela peut être dû au fait qu’il existe des boosters couleur arc-en-ciel dans les deux circuits.



FIGURE 33 – Image de Grumble Volcano, classée dans Maple Treeway



FIGURE 34 – Image de Maple Treeway, classée dans Grumble Volcano

De plus, les deux images précédentes (Figures 33 et 34) sont inversées en terme de classification. Cela peut s’expliquer grâce à la couleur assez proche des deux circuits que sont Maple Treeway et Grumble Volcano, tous deux comprenant une teinte orangée caractéristique.



FIGURE 35 – Image de Waluigi Pinball, classée dans Maple Treeway



FIGURE 36 – Image de Maple Treeway

On remarque également que l’image en Figure 35 est classée de façon erronée dans la classe Maple Treeway. Cette erreur est sûrement due à la présence du deltaplane. En effet dans le circuit Maple

Treeway, il existe deux passages avec le deltaplane déployé (Figure 36), contrairement à Waluigi Pinball où il n'y a qu'un passage. La base d'entraînement comprend alors logiquement plus d'images comprenant des deltaplanes dans le circuit Maple Treeway, d'où la mauvaise classification de l'image. Cette observation montre que le réseau de neurones est sensible à certains détails, certaines formes, et ne se contente pas seulement de faire sa reconnaissance sur les couleurs, ce qui est un point positif.

On peut faire des analyses similaires sur les erreurs de classification des images de l'ensemble de test. (Figure 37)

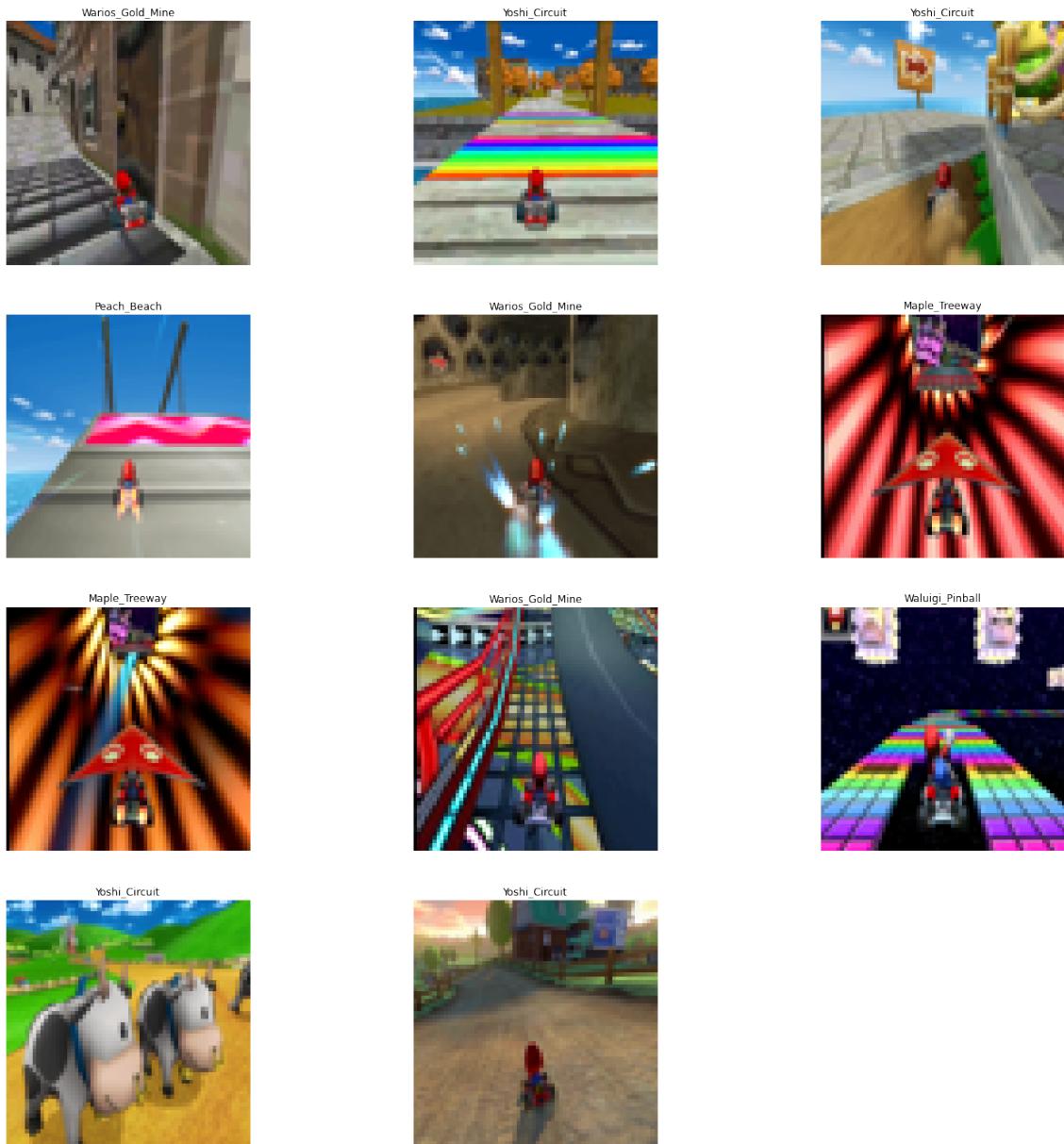


FIGURE 37 – Images de l'ensemble de test mal classifiées

En outre, la plupart des erreurs que fait le réseau sont explicables, que ce soit par la similarité de couleurs entre certaines parties de différents circuits, ou encore à la présence d'objets identiques sur ces différents circuits.

## 4.2 Deuxième approche

Analysons désormais les résultats obtenus avec le même réseau, mais avec les données en noir et blanc.

### 4.2.1 Métriques obtenues

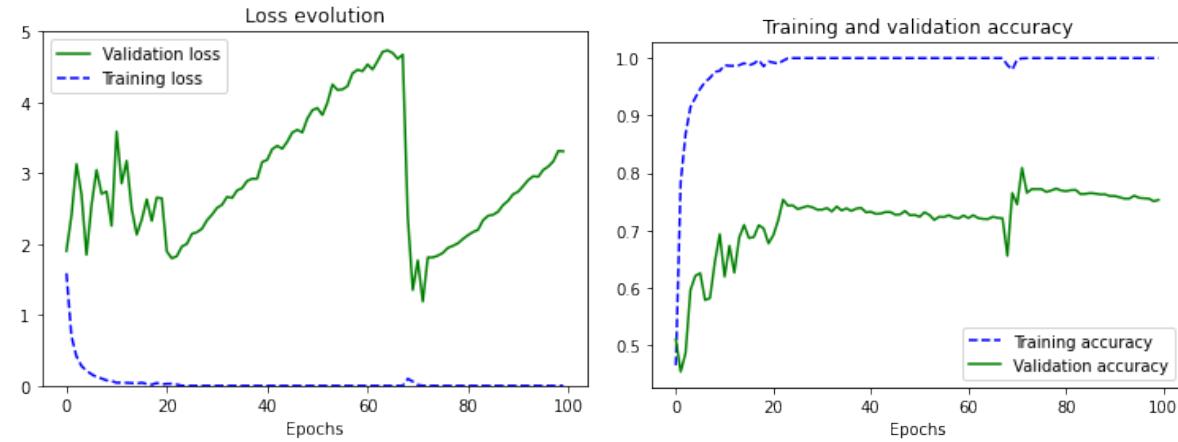


FIGURE 38 – Fonction de coût pour l'ensemble d'entraînement et de validation, premier réseau, données en noir et blanc

FIGURE 39 – Précision pour l'ensemble d'entraînement et de validation, premier réseau, données en noir et blanc

On observe dans cette deuxième approche des résultats bien moins convaincants. On peut déjà noter un fort sur-apprentissage dès le début de l'entraînement. La précision pour les données de validation plafonne alors à 80%, ce qui est un début, mais nous pouvons essayer de faire mieux, en limitant le surapprentissage.

Pour ce faire, nous ajoutons dans un premier temps une régularisation l2 de valeur 0.01.

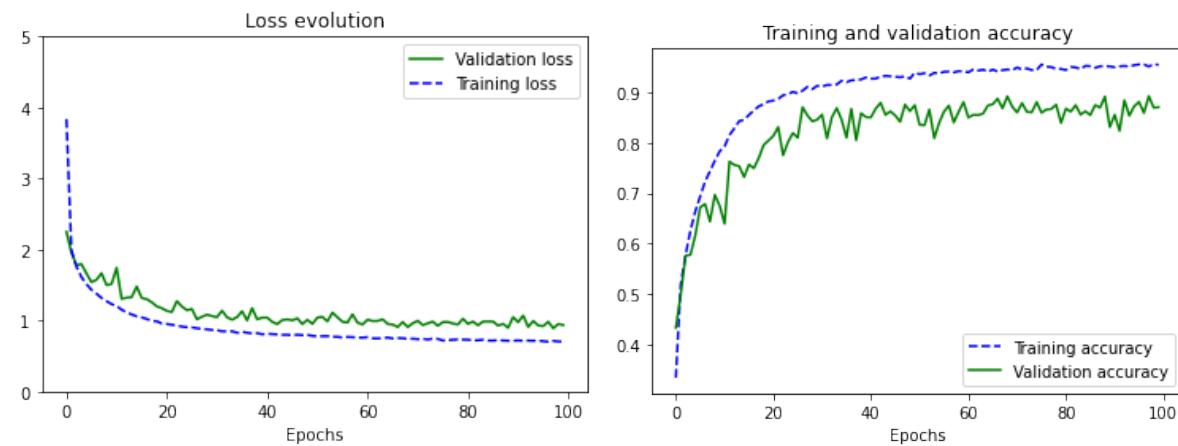


FIGURE 40 – Fonction de coût pour l'ensemble d'entraînement et de validation, premier réseau, régularisation l2 (0.01)

FIGURE 41 – Précision pour l'ensemble d'entraînement et de validation, premier réseau, régularisation l2 (0.01)

On remarque que la régularisation l2 permet de limiter le surapprentissage, qui est tout de même toujours présent.

On obtient alors de meilleurs résultats pour la précision de validation, qui avoisine les 85% à son maximum, tout comme la précision de test.

Après avoir fait quelques tests, nous ne sommes pas arrivé à obtenir au mieux 85.5% de précision pour les données de test avec ce réseau, obtenu avec 200 epochs, une régularisation l1 de 0.0001 et une régularisation l2 de 0.01, dont voici les détails de l'entraînement :

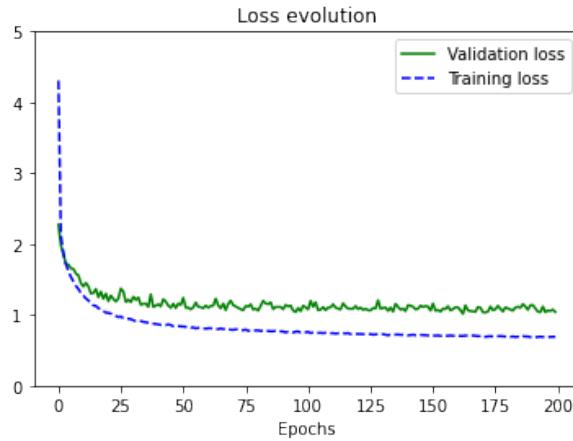


FIGURE 42 – Régularisation  $l1=0.0001$ ,  $l2=0.01$

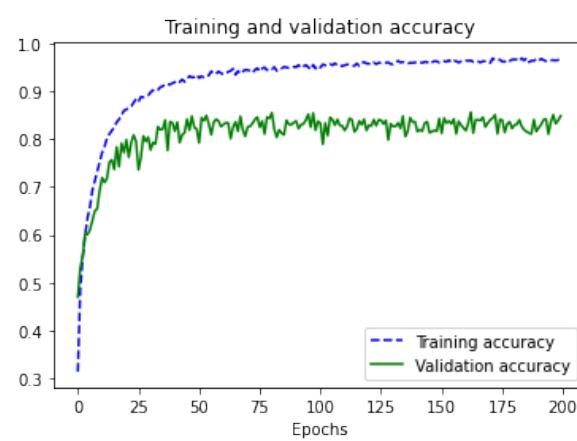


FIGURE 43 – Régularisation  $l1=0.0001$ ,  $l2=0.01$

#### 4.2.2 Analyse des erreurs

Examinons les images mal classées avec ce réseau.

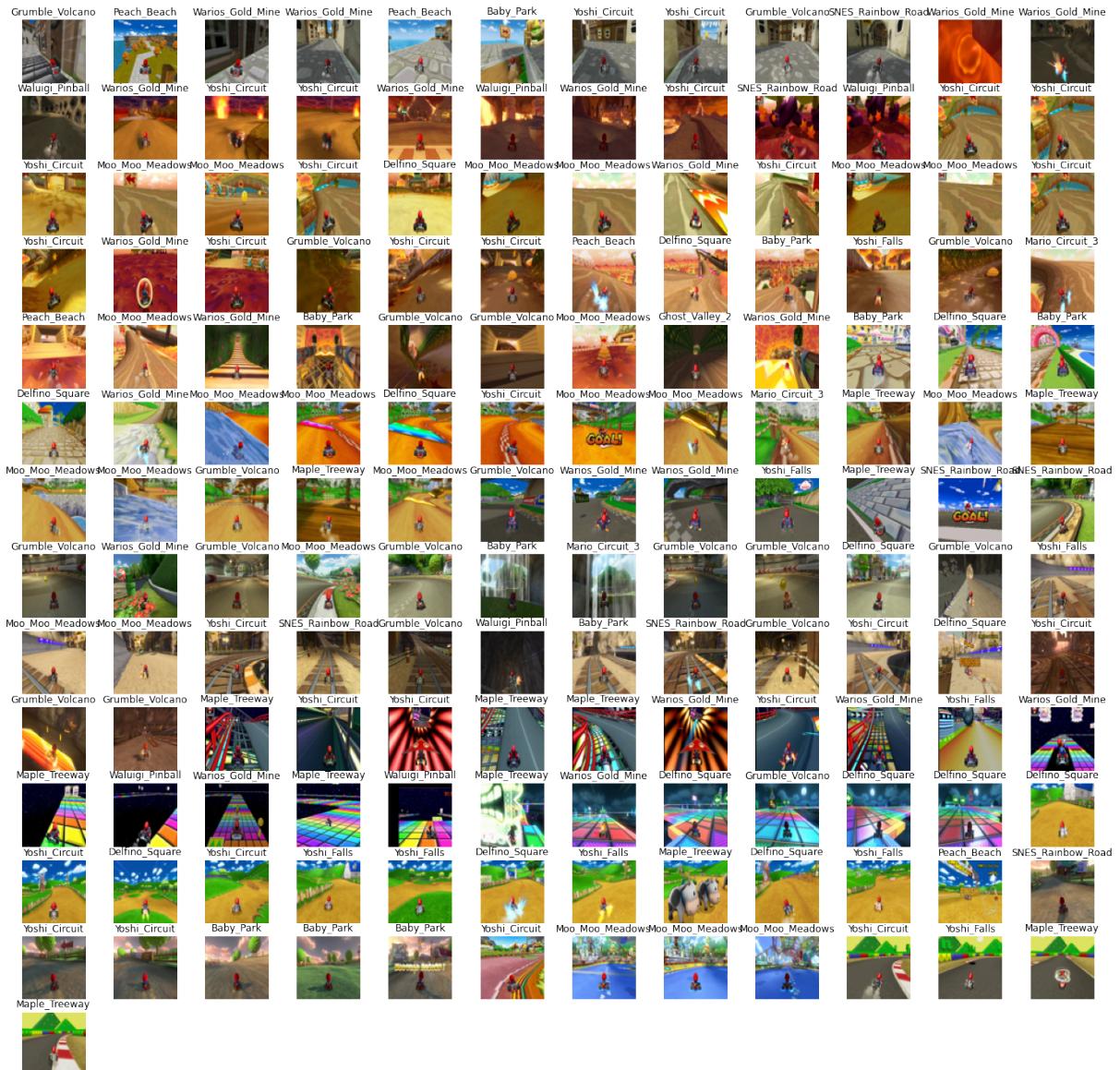


FIGURE 44 – Images de l'ensemble de test mal classifiées

A première vue, les erreurs sont beaucoup plus difficile à expliquer que précédemment, où la couleur influait grandement les prédictions. Ici les erreurs semblent quasiment aléatoires, et nous pouvons analyser la matrice de confusion suivante :

Réalité\Prédiction	DS	GV2	GV	MT	PB	YF	YC	WGM	WP	RR	MMM	BP	MC3
Delfino Square	0	0	2	0	2	0	2	2	0	1	0	1	0
Ghost Valley 2	0	0	0	0	0	0	0	0	0	0	0	0	0
Grumble Volcano	0	0	0	0	0	0	3	5	2	0	0	0	0
Maple Treeway	2	1	4	0	2	1	10	4	1	1	8	2	1
Peach Beach	2	0	0	0	0	0	0	1	0	0	0	2	0
Yoshi Falls	1	0	1	3	0	0	1	0	3	0	8	0	1
Yoshi Circuit	1	0	6	1	0	1	0	3	0	2	1	1	1
Wario's Gold Mine	1	0	5	0	0	1	3	0	1	2	2	1	0
Waluigi Pinball	0	0	0	3	0	1	3	2	0	0	0	0	0
Rainbow Road	3	0	1	3	0	0	0	3	2	0	0	0	0
Moo Moo Meadows	4	0	0	1	1	4	4	0	0	1	0	3	0
Baby Park	0	0	0	0	0	0	1	0	0	0	3	0	0
Mario Circuit 3	0	0	0	2	0	1	1	0	0	0	0	0	0

La matrice précédente représente les erreurs que fait le réseau. Si on appelle M cette matrice, M(i,j) représente le nombre de fois où le réseau a classé une image du circuit i (réalité) dans la classe du circuit j (prédiction).

Remarque : Seules les données **mal classées** sont présentes sur cette matrice de confusion, d'où le fait que la diagonale soit nulle.

On observe que le réseau n'arrive pas bien à reconnaître certains circuits, par exemple Maple Treeway avec 37 erreurs. Parmi ces 37 erreurs, le circuit pense reconnaître 10 fois Yoshi Circuit et 8 fois Moo Moo Meadows pour une raison qui nous échappe.

En revanche, il reconnaît parfaitement le circuit Ghost Valley 2, avec aucune erreur de classification. Cela est sûrement dû au fait que ce circuit est très sombre, avec un fond noir sur toute la course, ce qui le rend facilement reconnaissable. A noter que le réseau le reconnaît une seule fois à tort, sur l'image de la Figure 46, qui est une image assez sombre, ce qui va dans le même sens que précédemment



FIGURE 45 – Ghost Valley 2, un circuit sombre



FIGURE 46 – Image de Maple Treeway, classée dans Ghost Valley 2

### 4.3 Troisième approche : Transfer Learning

Nous parvenons à obtenir des résultats satisfaisants sur les données en noir et blanc, et avons essayé de faire mieux en utilisant le transfer learning.

#### 4.3.1 Fine-Tuning

Dans un premier temps, nous utilisons le fine-tuning avec le réseau VGG19, entraîné sur ImageNet. Voici les résultats que nous obtenons lors du premier apprentissage, c'est à dire l'étape où les couches

de VGG sont gelées et où seuls les poids de la partie dense du réseau sont actualisés. Nous avons testé ce premier entraînement avec différentes régularisations l1 l2 dont les valeurs étaient comprises entre 0.0001 et 0.1. La meilleure précision a été obtenue pour l1 = 0.001 et l2=0.01.

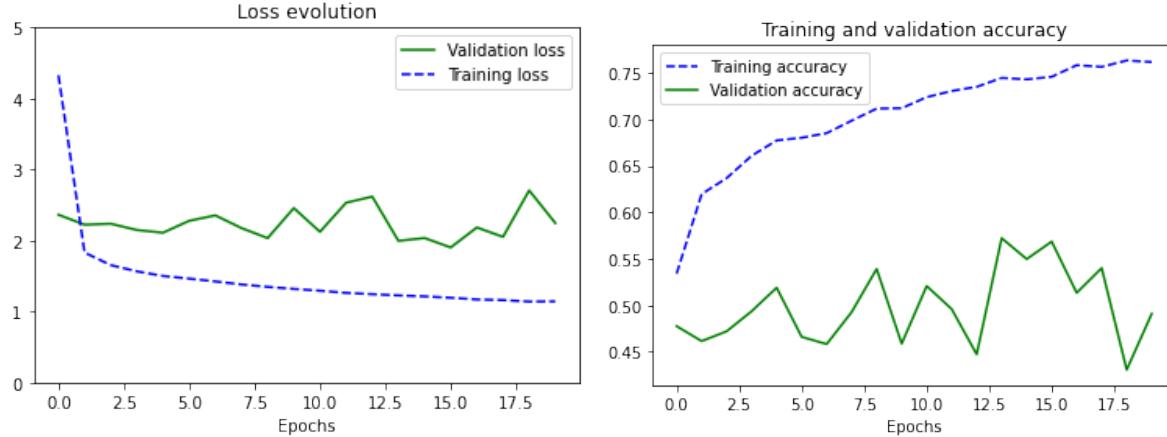


FIGURE 47 – Loss function lors du premier appren- FIGURE 48 – Accuracy function lors du premier apprentissage

Comme on peut le voir, les courbes pour la validation sont très chaotiques et un très grand surapprentissage est observé. Nous n'avons pas réussi à le diminuer plus que cela en testant différentes valeurs pour la régularisation.

Voici les résultats obtenus lors du second entraînement, c'est à dire l'étape où l'on dégèle les 4 dernières couches de convolution et la dernière couche de max pooling de VGG19.

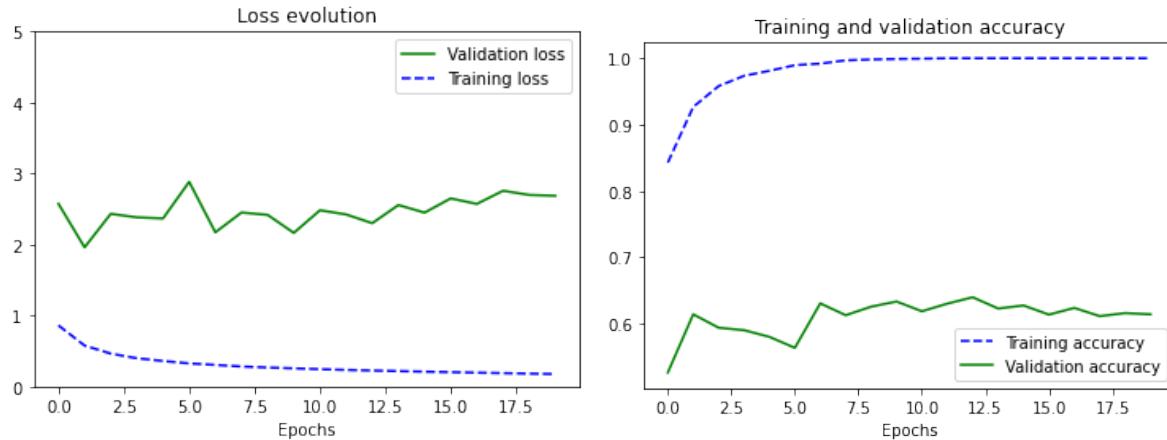


FIGURE 49 – Loss function lors du second appren- FIGURE 50 – Accuracy function lors du second apprentissage

Nous obtenons alors pour notre meilleur réseau avec cette méthode une précision sur les données de validation et de test d'environ 64%. Nous pensons que ces mauvais résultats sont dûs au fait que la sortie des dernières couches de convolution de VGG19 représente des données qui ne sont pas assez générales et déjà un peu trop spécifiques.

Pour remédier à cela, nous avons eu l'idée de n'utiliser que les premières couches de convolution de VGG, contenant des informations assez générales qui pourraient être plus adaptées à notre problème.

### 4.3.2 Extraction de caractéristiques

Nous avons extrait les caractéristiques de nos données en noir et blanc en les faisant passer dans les premières couches de convolution et max pooling de VGG19, en s'arrêtant à des couches différentes pour comparer les performances.

Le réseau VGG19 est composé de 21 couches avant ses couches denses, qui sont ses couches de convolution et de max pooling. L'optimiseur utilisé est Adam avec un taux d'apprentissage de 0.0003, et une régularisation l2 a été utilisée, entre 0.001 et 0.007. Le tableau suivant donne les précisions maximales obtenues en fonction de la couche de VGG19 utilisée comme couche de sortie pour l'extraction.

Couche de VGG utilisée pour l'extraction	Précision maximale sur l'ensemble de test
Couche n°3	83%
Couche n°6	87%
Couche n°8	79%
Couche n°11	85%
Couche n°16	93%
Couche n°17	84%
Couche n°21	67%

On observe sur le tableau précédent que les précisions ne sont pas du tout les mêmes selon les couches utilisées pour l'extraction de caractéristiques.

On remarque tout d'abord que lorsqu'on extrait les données à la 21<sup>e</sup> couche, c'est à dire juste avant les couches denses de VGG, la précision est très faible comparée aux autres couches. En réalité, extraire des données à cette couche revient à ce que l'on a fait lors de la première étape du fine tuning, et la précision est environ du même ordre.

On peut aussi noter que l'extraction de caractéristiques à partir de la couche n°16 est de loin la meilleure, c'est pourquoi nous avons décidé de tester plus particulièrement celle-ci qui nous semblait la plus prometteuse en ajustant les hyperparamètres de régularisation et de taux d'apprentissage.

En testant différentes valeurs de ces hyperparamètres, nous sommes arrivés à une précision de 93% sur l'ensemble de test, en prenant le réseau qui maximisait la précision de validation, sur 200 epochs, l'optimiseur Adam avec un taux d'apprentissage de 0.0001, et une régularisation l2 de 0.005. Voici les images mal classées, ainsi que la matrice de confusion correspondante.



FIGURE 51 – Images de l'ensemble de test mal classifiées

Réalité\Prédiction	DS	GV2	GV	MT	PB	YF	YC	WGM	WP	RR	MMM	BP	MC3
Delfino Square	0	0	0	1	1	0	0	1	0	3	0	0	1
Ghost Valley 2	0	0	0	0	0	0	1	0	0	0	0	0	0
Grumble Volcano	0	0	0	1	0	0	1	1	0	0	1	0	0
Maple Treeway	2	0	5	0	0	0	4	0	0	1	0	0	0
Peach Beach	2	0	1	2	0	0	0	0	0	2	0	0	0
Yoshi Falls	1	0	0	3	0	0	2	0	0	0	0	0	0
Yoshi Circuit	0	0	0	1	0	0	0	2	0	3	0	0	0
Wario's Gold Mine	1	0	1	0	0	0	0	0	0	0	0	0	0
Waluigi Pinball	0	0	0	0	0	0	0	0	0	0	0	0	0
Rainbow Road	0	0	0	0	0	0	0	0	0	0	0	0	0
Moo Moo Meadows	0	0	0	8	1	0	7	0	0	3	0	0	0
Baby Park	0	0	0	0	0	0	8	1	0	1	3	0	0
Mario Circuit 3	0	0	0	0	0	0	0	0	0	0	0	0	0

On remarque que la plupart des erreurs du réseau de font sur 3 circuits : Maple Treeway (12 erreurs), Baby Park (13 erreurs) et Moo Moo Meadows (19 erreurs).

En revanche il reconnaît parfaitement SNES Rainbow Road (0 erreur), Mario Circuit 3 (0 erreur), Waluigi Pinball et Ghost Valley 2 (1 erreur).

Enfin on remarque encore que deux des images mal classées contiennent un texte parasite ("Finish" et "Nouveau Record") ce qui peut être une explication au fait que le réseau se soit trompé sur ces images, car ce texte est présent dans des images d'autres circuits.

## 5 Conclusion

En conclusion, nous pouvons dire que notre objectif initial, qui était la reconnaissance d'un circuit Mario Kart à partir d'une image a bien été atteint malgré les quelques erreurs de classification qui persistent. Cependant, ces bons résultats sont à mettre en parallèle avec le fait que les images d'un circuit à l'autre sont très différentes, notamment au niveau de la colorimétrie, ce qui rend l'identification assez facile. C'est pour cela que nous avons, par la suite, essayé de résoudre le même problème en supprimant la couleur de nos images. Comme on pouvait s'y attendre, cela a considérablement réduit les performances de notre réseau, bien qu'elles restent acceptables. On pourrait probablement encore améliorer ces résultats en optimisant nos hyperparamètres, chose que nous n'avons malheureusement pas eu le temps de faire. Une autre possibilité serait aussi de trouver un réseau de neurones qui soit plus adapté à ce problème mais nos recherches dans cette direction n'ont pas été fructueuses. Ainsi l'approche en niveaux de gris ne serait, dans l'état actuel, utilisable que pour une application dans laquelle le stockage est limité et qui ne serait pas critique, c'est-à-dire avec des erreurs tolérables. Enfin, une dernière approche qui nous semble pertinente mais que nous n'avons pas eu le temps de mettre en œuvre serait de réaliser un fine-tuning du réseau VGG mais seulement en gardant les premières couches de convolution, par exemple jusqu'à la couche n°16, et non pas toutes comme nous l'avons fait dans la partie 4.3.1.