



Groupe D :

Alexis Gosselin
Gabriel Rochaix--Yamamoto
Issam Alouane
Jonas Lavaur
Justin Jourdant

RAPPORT : Quel est le type de ce Pokéémon ?!



Apprentissage Profond - IMM
Département Sciences du Numérique - 2A
Année 2022-2023

Sommaire

I) Constitution de la base de données.....	3
a) Description du sujet choisi.....	3
b) Méthodologie d'acquisition et d'annotation des données.....	6
c) Méthodologie de partitionnement des images.....	7
d) Estimation de la complexité du problème et description des résultats attendus.....	8
e) Script de chargement des données.....	9
f) Sélection d'images de la base de données.....	11
II) Elaboration de la solution.....	13
a) Mono-Labelling.....	14
b) Multi-Labelling.....	16
1. Pourquoi le Multi-Labelling ? (Fonction d'activation de sortie).....	16
2. Mise-à-jour du load_data.....	17
3. Architecture du Réseau.....	18
4. Fonction Accuracy.....	18
5. Augmentation de données.....	18
III) Analyse des résultats.....	19
a) Mono-Labelling.....	19
1. Les 18 Classes.....	19
2. Les classes ayant entre 360 et 400 images.....	20
3. Quatre classes distinctes.....	21
b) Multi-Labelling.....	22
1. Les 18 Classes.....	22
2. Les classes ayant entre 360 et 400 images.....	24
3. Quatre classes distinctes.....	24
4. Trois classes proches.....	25
IV) Conclusion.....	26

I) Constitution de la base de données

a) Description du sujet choisi

L'objectif de ce projet est, comme son nom l'indique, de pouvoir déterminer le type d'un pokémon à partir d'une image de celui-ci. Pokémons est une franchise créée par Satoshi Tajiri en 1996, notamment célèbre pour ses jeux vidéos, dans des séries éditées par Nintendo. La licence a par la suite été adaptée en dessin animé, en cartes à jouer et à collectionner, en mangas, en film... Il existe même des trains pokémons !



Pour pouvoir trier ces pokémons par type, nous allons utiliser un algorithme d'apprentissage profond en utilisant une base de données constituée d'un grand nombre d'images de pokémons labellisés par type. Les types dans pokémon correspondent en quelque sorte à des "éléments" auxquels sont rattachés les pokémons. Voici la liste des types présents dans cet univers :

POKÉMON TYPE SYMBOLS

NORMAL	FIRE	WATER	ELECTRIC	GRASS	ICE
FIGHTING	POISON	GROUND	FLYING	PSYCHIC	BUG
ROCK	GHOST	DRAGON	DARK	STEEL	FAIRY

FALKE
falke2009.deviantart.com

Nous avons décidé de nous concentrer sur les quatres premières générations de pokémon pour pouvoir multiplier les images d'un même pokémon (cartes, figurine, sprite de jeu, images de dessins animés etc.)



Voici Darkrai, un pokémon de la quatrième génération (Diamant et Perle) de type ténèbre. Cette première image est une photo d'une figurine à son effigie. Ci-dessous, se trouvent deux autres images pour ce même pokémon qui l'illustre sous d'autres formats (modèle 3D et carte).



Ce pokémon est assez intéressant car sa couleur noir est un assez bon indicateur qu'il est de type ténèbre et il est assez évident pour un humain de le classer dans ce type. On aurait cependant éventuellement pu penser que Darkrai était un type spectre (voir un type poison ou psy).



Le pokémon représenté ci-dessus (tiré du dessin animé éponyme) est le plutôt célèbre Dracaufeu de type feu et vol. On voit ici facilement qu'il peut être rattaché au type feu entre celle sur sa queue et celles qu'il crache. De la même façon, ses ailes le rattachent au type vol de façon plutôt évidente. Cependant le type dragon aurait également pu lui être rattaché car il ressemble quand même assez à l'image du dragon que l'on peut avoir dans notre imaginaire européen.

Ce pokémon lève aussi une seconde question : celle des pokémons double types, c'est-à-dire un pokémon associé à deux types au lieu d'un pour une majorité. Ce problème de classement sera abordé par la suite et influencera également les résultats que l'on pourra attendre de notre algorithme.

b) Méthodologie d'acquisition et d'annotation des données

Nous nous sommes limités aux quatre premières générations pour les images d'entraînement et de validation. Cependant nous avons également cherché quelques images de pokémons de la 5e génération pour pouvoir tester à la phase de test, si nos réseaux reconnaissent les types de pokémons jamais rencontrés. A la fin du projet, nous avons réussi à rassembler une base de données de plus de 3500 images !

Pour la génération 1, les environ 530 images ont été acquises une par une à la main (via Google Images) et regroupées et triées dans des dossiers correspondant au type auquel le pokémon est rattaché. Les images ont été choisies pour représenter les pokémons dans des contextes les plus variés possibles (modèle 3D, représentation dans l'animé, dessin, peluche,...).

De plus, pour les générations 2 et 3, la même méthodologie a été appliquée. Enfin, les images de la 4ème génération étaient prises d'un archive zip contenant les images de toutes les générations et triées à la main pour mettre chaque pokémon dans sa classe correspondante.

Pour l'annotation des données, nous avons donc choisi de regrouper les images des pokémons dans des dossiers dépendant de leur type : Normal, Feu, Eau, Électrique, Plante, Glace, Combat, Poison, Sol, Vol, Psy, Insecte, Roche, Spectre, Dragon, Ténèbres, Acier, Fée.

Enfin, pour les pokémons multi-types nous avons décidé de mettre le pokémon seulement une fois dans le type qui nous semble le plus pertinent. Cependant dans ce dossier de type, nous avons créé un sous-dossier pour les distinguer. Par exemple, Florizarre est de type plante/poison, nous l'avons mis dans le dossier pour les pokémons de types plante (car cela nous semblait le plus logique) et dans ce dossier nous avons donc créé un sous-dossier poison pour "ranger" et où seront également placé les autres pokémons du double type plante-poison.

c) Méthodologie de partitionnement des images

Nous avons mis toutes nos images dans le dossier global (sauf ceux de la 5e génération) et nous avons ensuite écrit un script python qui permet de répartir toutes les images dans les dossiers train, validation ou test selon un certain pourcentage qu'on choisit. Nous avons choisi une répartition 80/10/10. Le script essaie de bien respecter cette répartition par type, pour éviter le cas où on ne teste pas une classe parce qu'on a mis toutes les images de la classe dans train ou validation.

Le script est dans le git : **partition.py**

d) Estimation de la complexité du problème et description des résultats attendus

Avec une base de données contenant plus de 3500 images, la complexité de ce projet est élevée en raison de la taille (relativement grande) de la base de données, le temps d'entraînement sera alors un peu élevé et nécessitera donc des ressources informatiques suffisantes pour avoir un temps raisonnable mais surtout nous devons classifier les images selon 18 types différents, ce qui est énorme. Nous prévoyons d'utiliser un modèle de deep learning avec une architecture convolutive pour extraire les caractéristiques clés des images. Le modèle sera entraîné sur les images de la base de données avec un algorithme de descente de gradient stochastique (SGD) pour minimiser la fonction de coût. Nous prévoyons de tester différents modèles et hyperparamètres pour trouver la meilleure combinaison pour notre problème de classification. Les résultats attendus sont une classification précise des images de Pokémons inconnus.

e) Script de chargement des données

La différence principale de notre base de données avec l'exemple donné sur moodle est que nous avons des sous-types dans des sous-dossiers. Le script à lancer est **scriptLoad.py** et les fonctions auxiliaires sont dans **loadimages.py**

Pour rendre plus lisible et modulable le code nous avons codé des fonctions auxiliaires :

- `load_image` qui permet d'enregistrer l'image à l'index courant et la classe courante
- `load_sub_type` qui teste pour chaque item du répertoire, s'il s'agit d'un fichier (image) ou d'un sous-dossier. S'il s'agit d'une image, la fonction appelle `load_image`, s'il s'agit d'un sous-dossier, la fonction s'appelle récursivement. Cela permettrait donc théoriquement également d'avoir des sous-sous-type. Pour l'instant `load_sub_type` ne change pas l'index de la classe car au départ nous allons seulement considérer les types principaux des pokémons, nous verrons plus tard lorsque l'on considérera les types secondaires.
- `count_images` qui compte récursivement toutes les images dans tous les dossiers ce qui permet d'initialiser les `x_train` et `y_train` à la bonne taille.

```
23 ▼ def load_image(path, image_size, x, y, current_index, idx_class):  
24     #<0xa0>Ouverture de l'image  
25     img = Image.open(path)  
26     #<0xa0>Conversion de l'image en RGB  
27     img = img.convert('RGB')  
28     #<0xa0>Redimensionnement de l'image et écriture dans la variable de retour x  
29     img = img.resize((image_size,image_size))  
30  
31  
32     x[current_index] = np.asarray(img)  
33     # Écriture du label associé dans la variable de retour y  
34     y[current_index] = idx_class  
35     return current_index+1  
36  
37 ▼ def load_sub_type(type_path, image_size, x, y, current_index, idx_class):  
38     dirs = os.listdir(type_path)  
39  
40 ▼     for item in dirs:  
41         itemPath = os.path.join(type_path, item)  
42  
43 ▼         if os.path.isdir(itemPath):  
44             current_index = load_sub_type(itemPath, image_size, x, y, current_index, idx_class)  
45  
46 ▼         elif os.path.isfile(itemPath):  
47             current_index = load_image(itemPath, image_size, x, y, current_index, idx_class)  
48 ▼         else:  
49             print("Unknown : " + itemPath)  
50  
51     return current_index  
  
9 ▼ def count_images(path):  
10     dirs = os.listdir( path )  
11  
12     count = 0  
13     for item in dirs:  
14         itemPath = os.path.join(path, item)  
15  
16     ▼         if os.path.isdir(itemPath):  
17             count += count_images(itemPath + "/")  
18     ▼         elif os.path.isfile(itemPath):  
19             count +=1  
20  
21     return count
```

Dans load_data il reste donc plus qu'à parcourir les dossiers en appelant load_images ou load_sub_type :

```
def load_data(data_path, classes, dataset='train', image_size=128):

    nb_images = 0;
    for i in range(len(classes)):
        #dirs = sorted(os.listdir(data_path + dataset + '/' + classes[i]) +"/")
        type_path = data_path + dataset + "/" + classes[i] +"/"
        nb_images += count_images(type_path)

    x = np.zeros((nb_images, image_size, image_size, 3))
    y = np.zeros((nb_images, 1))

    current_index = 0

    for idx_class in range(len(classes)):
        type_path = data_path + dataset + "/" + classes[idx_class] +"/"
        dirs = sorted(os.listdir(type_path))

        #nb_images_type = count_images(type_path)

        for item in dirs:
            #item = dirs[idx_img]
            itemPath = os.path.join(type_path, item)

            if os.path.isfile(itemPath):
                current_index = load_image(itemPath, image_size, x, y, current_index, idx_class)
            elif os.path.isdir(itemPath):
                current_index = load_sub_type(itemPath, image_size, x, y, current_index, idx_class)

    return x, y
```

f) Sélection d'images de la base de données

Voici une sélection d'images issues de notre base de données avec des explications de comment les trouver.

Par exemple, dans le dossier spectre on peut trouver une sélection d'images de pokémon de ce type comme :



feuforeve_3d



feuforeve_dessin



ou encore 477

(477 qui correspond au numéro du pokémon Noctunoir dans le pokédex)

On a également des sous-dossiers dans le dossier des pokémons spectres comme dragon, ténèbres, vol ou poison. Par exemple dans le sous-dossier poison on peut retrouver :



ectoplasma_3d



ectoplasma_dessin

ou encore fantominus



Dans le dossier électrique par exemple on peut trouver :



pharamp_dessin



raichu_figurine

Tandis que dans le dossier sol on peut trouver :



ossatureur_dessin



sablette_photo



472

Enfin dans le type psy par exemple on peut trouver :



mew_art



alakazam_3d2



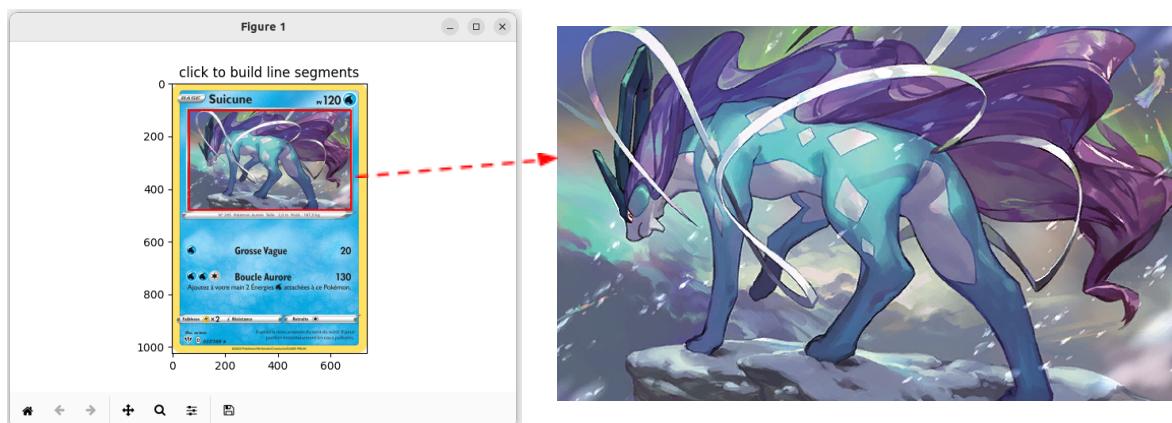
mentali_fanart

Ce qui est intéressant au travers de ces exemples est de voir que nous sommes instinctivement capables de classer les pokémons dans les types correspondants que ce soit par leur couleur ou par certains attributs.

III) Elaboration de la solution

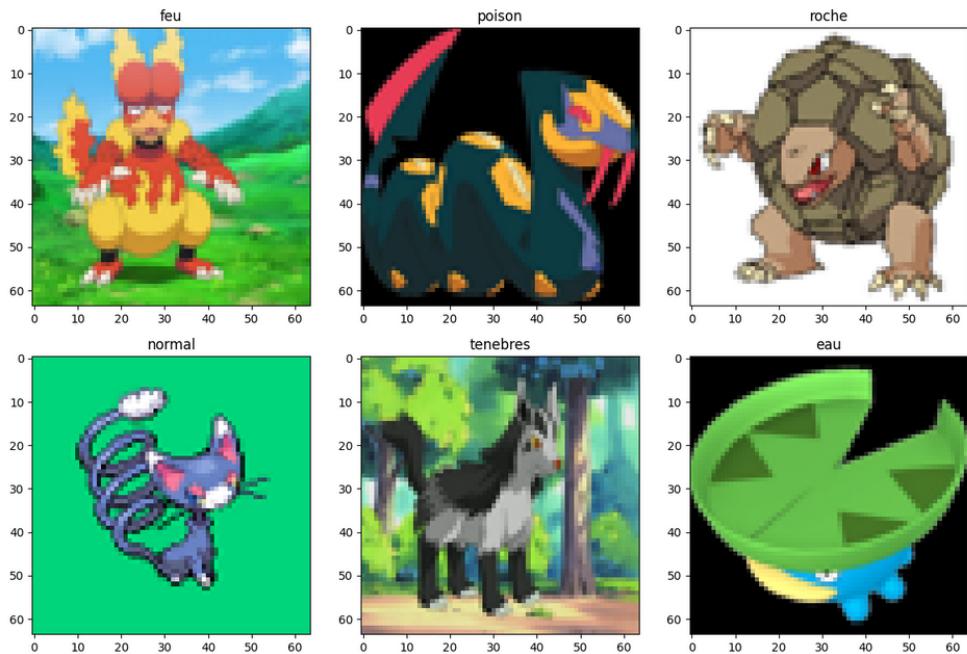
Pour commencer, nous avons regardé en groupe la base de données constituées et nous avons réfléchi à la meilleure organisation que l'on pouvait avoir pour permettre de gérer les doubles types. Nous avons opté pour la création de sous-dossiers dans les dossiers des types pour mettre les pokémons correspondant dedans. De plus, nous avons rédigé un programme python **countGlobal.py** qui compte les images de pokémon pour chaque classe. Nous avons ainsi rapidement pu constater que certaines classes comme *Dragons*, *Fées* ou *Acier* étaient largement sous-représentés tandis que des classes comme *Eau*, *Normal* ou *Plante* étaient largement sur-représentées. Nous avons donc complété notre base de données pour augmenter les classes sous-représentées jusqu'à avoir environ une centaine d'images par types.

De plus, dans notre base de données, nous avons tous mis des images de cartes pokémon pour un bon nombre de pokémons. Or, dans le coin supérieur droit de toutes ces cartes, se trouve le logo du type auquel appartient le pokémon et cela pourrait donc affecter significativement les résultats des prédictions. Ainsi, nous avons développé un script python **crop_all_cards.py** permettant de pallier ce problème. En effet, ce programme ouvre (une à une) toutes les images contenant le mot-clé "carte" dans leur nom et offre la possibilité de recadrer chaque image ainsi ouverte, en sélectionnant à la souris (rectangle rouge), la partie de l'image à conserver (cf. images ci-dessous). L'ancienne image est supprimée et la nouvelle image recadrée la remplace, en prenant le nom suivant : "<nom_ancienne_image>_cropped".



a) Mono-Labelling

Pour la visualisation des images, nous avons fait comme dans le TP3 :



D'abord, nous avons testé avec un réseau convolutif de base. Comme point de départ, nous avons pris le réseau du TP3 (classification des chiens et chats) que nous avons adapté pour qu'il convienne à notre problème. En effet, nous avons juste changé la fonction d'activation en sortie du réseau, car le problème des chiens et chats est un problème de classification binaire (utilisation de "sigmoid"), alors que le nôtre est un problème de classification à plusieurs classes (18 classes car 18 types différents). Nous avons donc utilisé la fonction d'activation "softmax" en sortie.

Ainsi, notre réseau prédisait un seul type en sortie (même si beaucoup de pokémons ont deux types cf. b) Multi-Labelling).

Pour l'entraînement, nous avons fixé les hyperparamètres suivants :

- `loss="sparse_categorical_crossentropy"`
- `optimizer=optimizers.Adam(learning_rate=3e-4)`
- `metrics=['accuracy']`
- `epochs=20`
- `batch_size=10`

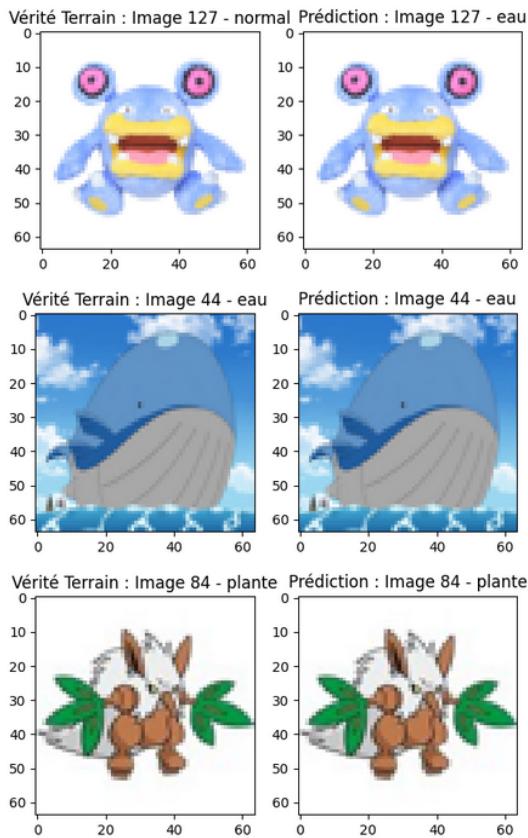
Pour essayer de corriger le surapprentissage obtenu, nous avons eu recours à la technique d'augmentation de la base de données (comme dans le TP3) en utilisant le module `ImageDataGenerator` :

```
(rotation_range=40, width_shift_range=0.2, height_shift_range=0.2,  
shear_range=0.2, zoom_range=0.2, horizontal_flip=True).
```

Puis, nous avons lancé un autre entraînement avec les hyperparamètres suivants :

- `loss="sparse_categorical_crossentropy"`
- `optimizer=optimizers.Adam(learning_rate=3e-4)`
- `metrics=['sparse_categorical_accuracy']`
- `epochs=50`
- `batch_size=10`

Comme nous avons obtenu de meilleurs résultats, nous avons décidé d'adapter le code présent dans le TP6 afin d'avoir quelques résultats visuels des prédictions :



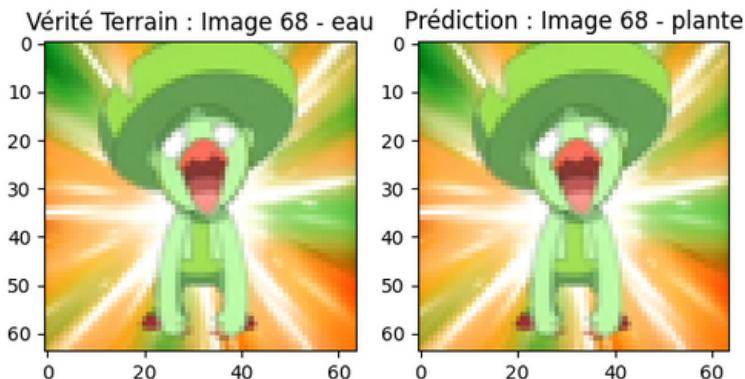
Et enfin, nous avons calculé la précision globale et les précisions pour chaque classe, ainsi que d'autres indicateurs (cf. partie III) Analyse des résultats).

Cependant, comme dit précédemment, beaucoup de pokémons possèdent en réalité deux types différents, c'est pourquoi nous avons décidé de partir sur du multi-labelling.

b) Multi-Labelling

1. Pourquoi le Multi-Labelling ? (Fonction d'activation de sortie)

L'un des premiers problèmes que nous avons rencontré lors de ce projet est les pokémons à 2 types. Une des limites du mono-labelling était les pokémons à 2 types et nous les avons considéré comme ayant un seul type principal et un type secondaire, ce qui n'existe pas. Nous pensions alors que passer au double-labelling nous permettrait de s'affranchir de cette limite et d'améliorer les résultats. De plus le réseau mono-labelling donnait parfois le label secondaire à une image (exemple ci dessous avec Lombre) et il nous semblait bizarre de considérer cela comme une erreur alors que quelque part le réseau avait raison :



Cependant pour passer au double-labelling il nous fallait changer la représentation du label. En mono-labelling un label est représenté par un entier, ici nous avons pensé qu'il était intéressant de représenter un label par un tableau de 0 et de 1, avec les positions des 1 représentant l'appartenance à un certain type.

Il nous fallait donc maintenant changer la fonction d'activation de sortie. Et nous étions face à un problème, comment dire qu'on veut pas juste 2 types, mais 1 ou 2 types ? Le problème s'est avéré complexe et nous avons finalement décidé de faire du "multi-labelling". C'est-à-dire que le réseau peut associer une image à un nombre de labels qu'il veut. Cela nous a permis d'utiliser une sigmoid sur chaque classe et d'utiliser une Binary Cross Entropy pour la loss (Nous avons utilisé la Binary Focal Cross Entropy pour des raisons citées plus bas).

Nous pensions que cette solution amènerait le réseau à multiplier le nombre de types associés mais à l'inverse cela a amené le réseau à donner aucun type à certaines images. Cette solution n'est donc pas totalement satisfaisante mais c'était la meilleure solution que nous avions trouvé. De plus, la plupart du temps le réseau attribue au moins 1 type et jamais plus de 2 types.

2. Mise-à-jour du load_data

Ensuite il a fallu coder une nouvelle fonction `load_data`. Nous avons totalement abandonné l'idée du type principale et type secondaire car cela aurait complexifié encore plus le projet. Nous avons donc considéré qu'un pokémon rangé dans le dossier feu/combat/ et qu'un pokémon rangé dans le dossier combat/feu/ avaient la même labellisation. Nous avons écrit une nouvelle fonction `load_data` respectant ces caractéristiques, qui est dans le git, dans le dossier `/multi_labels/` :

```
def load_labels(label_path, image_size, x, y, current_index, labels_indexes, labels, classes):
    dirs = os.listdir(label_path)
    label_bool = is_label_considered(labels, classes)

    for item in dirs:
        itemPath = os.path.join(label_path, item)

        if os.path.isdir(itemPath):
            if labels != []:
                sub_labels = labels.copy()
            else:
                sub_labels = []
            sub_labels.append(item)

            sub_label_index = get_label_index(item, classes)
            sub_labels_indexes = labels_indexes.copy()
            if sub_label_index != -1:
                sub_labels_indexes.append(sub_label_index)

            current_index = load_labels(itemPath, image_size, x, y, current_index, sub_labels_indexes, sub_labels, classes)

        elif os.path.isfile(itemPath) and label_bool:
            current_index = load_image(itemPath, image_size, x, y, current_index, labels_indexes)

    return current_index

def load_data(data_path, classes, dataset='train', image_size=128):
    set_path = os.path.join(data_path, dataset)
    dirs = os.listdir(set_path)

    nb_images = count_images(set_path, [], classes)
    x = np.zeros((nb_images, image_size, image_size, 3))
    y = np.zeros((nb_images, len(classes)))
```

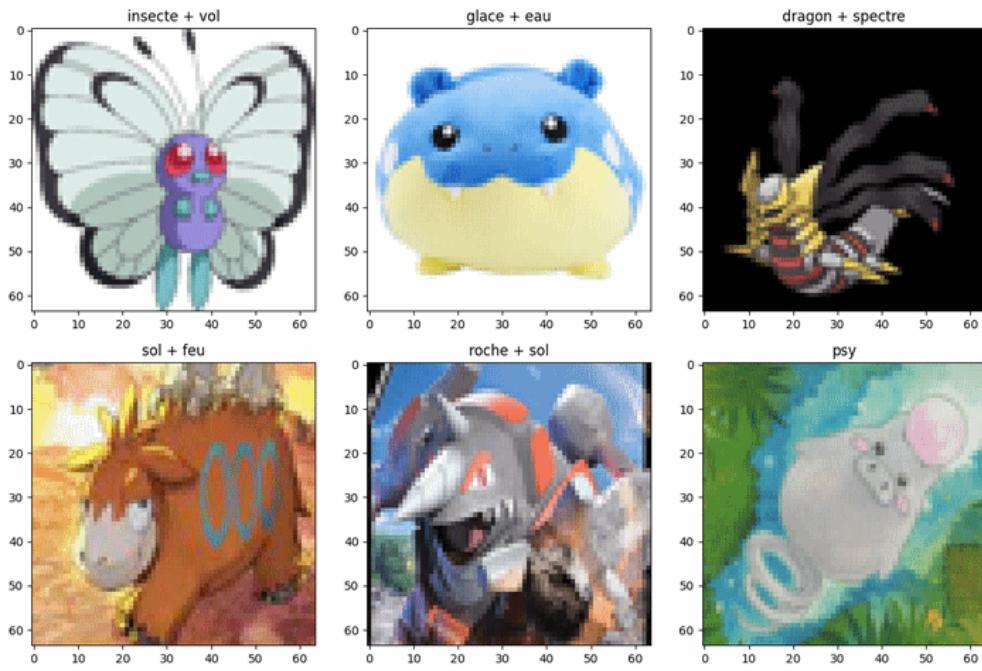
En considérant cette nouvelle labellisation on peut recompter le nombre d'images par types :

fée	:	148
électrique	:	160
dragon	:	169
glace	:	169
spectre	:	173
combat	:	174
acier	:	205
feu	:	225
tenebres	:	241
roche	:	263
poison	:	292
insecte	:	297
sol	:	360
normal	:	371
psy	:	374
plante	:	387
vol	:	393
eau	:	611
'Total'	:	5012

Les images des pokémons à double type sont comptées 2 fois ici. Nous avons donc "artificiellement" augmenté le nombre d'images par types pour arriver à un total de 5000 images (ici les 250 images de la 5e génération ne sont pas comptabilisées).

On remarque qu'il y a toujours un fort écart du nombre d'images par type. Le type eau est loin devant avec 200 images de plus que le type 2e type vol. Et loin derrière le type fée avec 150 images. Nous avons donc utilisé la Binary Focal Cross Entropy pour minimiser ce problème.

Plus bas nous avons inséré 6 exemples de labellisations.



Pour la taille des images, nous avons décidé de rester à 64x64. Nous avions testé des tailles 128x128 et 256x256 mais ces tailles n'ont pas suffisamment amélioré les résultats pour compenser le temps d'exécution ajouté.

3. Architecture du Réseau

Nous n'avons pas changé le réseau VGG en lui-même par rapport au mono-labelling car les caractéristiques qu'on veut que le réseau trouve dans les images sont les mêmes.

4. Fonction Accuracy

Cependant nous avons dû écrire notre propre fonction accuracy. En effet, avec une simple Binary Accuracy, les True Negatives sont beaucoup trop valorisées. Une image d'un pokémon feu, mais prédit comme eau aurait une accuracy de 16/18 = 0.88. Dans ce cas on voudrait une accuracy de 0. Nous avons donc plutôt choisi une accuracy qui calcule la formule suivante : $TP / (TP + FP + FN)$, avec TP : True Positive ; FP : False Positive ; FN : False Negative. Cette formule permet de défavoriser les oubliés de types et les mauvais types prédits tout en favorisant les bonnes prédictions.

5. Augmentation de données

Nous avons utilisé la même augmentation de données que pour le mono-labelling.

III) Analyse des résultats

a) Mono-Labelling

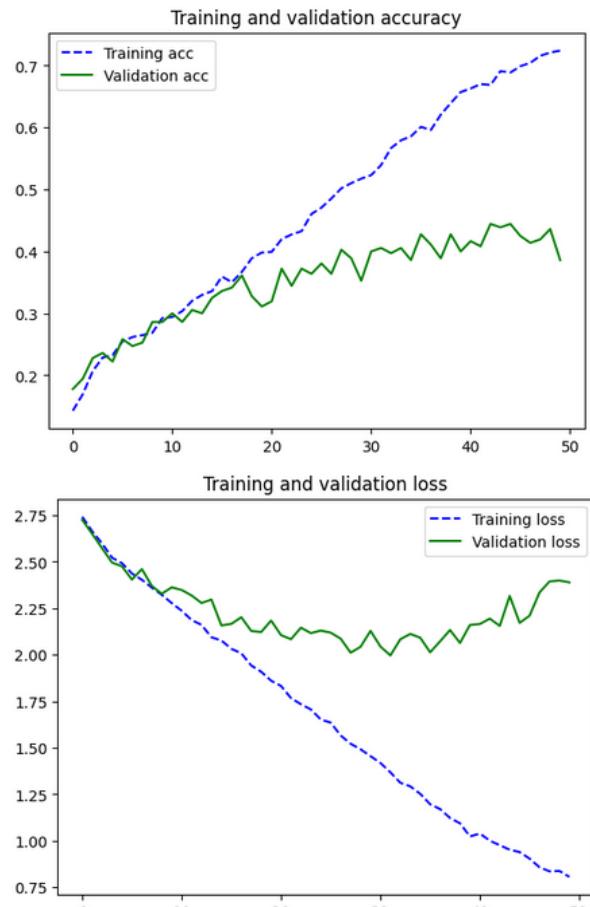
Notre fonction **load_data** nous permet de choisir les classes qu'on veut considérer. Nous en avons donc profité pour obtenir différents résultats.

1. Les 18 Classes

En considérant toutes les classes, après augmentation des données nous avions une accuracy d'environ 40%.

La précision globale est de 39.8%

Classe	Précision	Rappel	F1-score
acier	0.44	0.37	0.40
combat	0.19	0.33	0.24
dragon	0.00	0.00	0.00
fée	0.27	0.43	0.33
glace	0.36	0.27	0.31
insecte	0.72	0.52	0.60
normal	0.36	0.35	0.36
plante	0.45	0.60	0.51
poison	0.36	0.42	0.38
psy	0.50	0.19	0.28
roche	0.26	0.37	0.30
sol	0.31	0.24	0.27
spectre	0.37	0.47	0.41
tenebres	0.40	0.43	0.41
vol	0.33	0.25	0.29
électrique	0.41	0.47	0.44
feu	0.38	0.28	0.32
eau	0.51	0.59	0.55

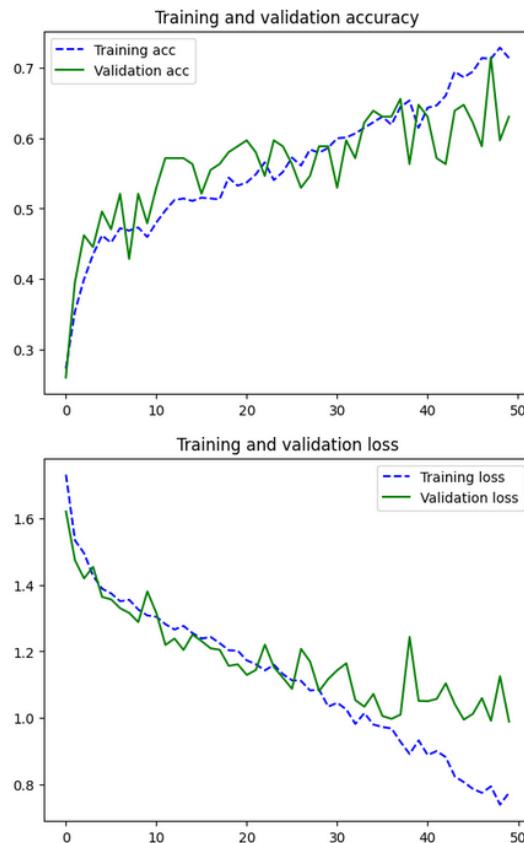


2. Les classes ayant entre 360 et 400 images

Pour obtenir de meilleurs résultats, nous avons voulu prendre en compte uniquement les classes ayant à peu près le même nombre d'images et après augmentation des données nous sommes arrivés à 60% :

La précision globale est de 60.5%

Classe	Précision	Rappel	F1-score
vol	0.45	0.42	0.43
plante	0.54	0.90	0.68
psy	0.69	0.43	0.53
normal	0.68	0.62	0.65
sol	0.78	0.41	0.54

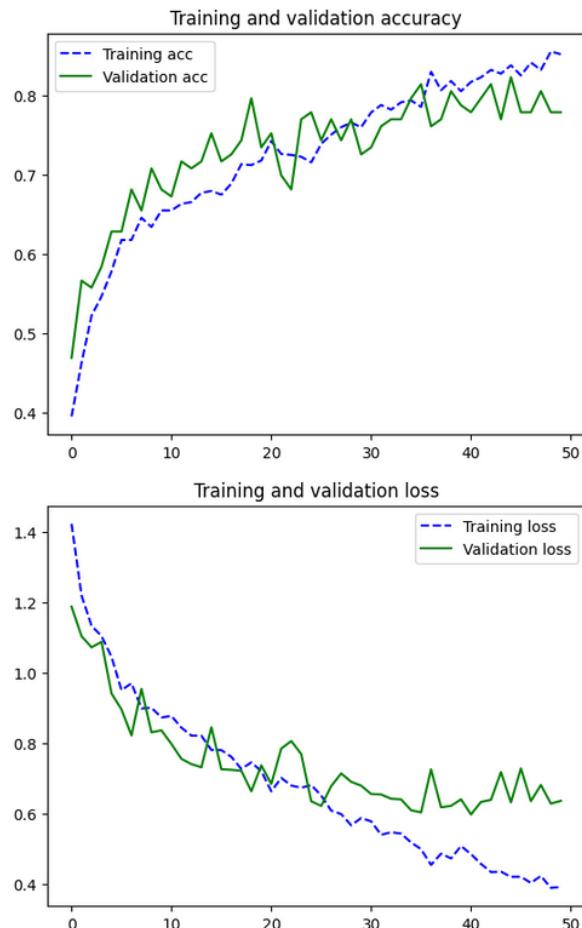


3. Quatre classes distinctes

Nous voulions voir comment le réseau réagissait s'il était seulement confronté à 4 classes facilement labellisables. Nous avons choisi les classes feu, eau, plante et électrique. Après augmentation des données, nous avons obtenu une précision globale de plus de 75% :

La précision globale est de 76.1%

Classe	Précision	Rappel	F1-score
feu	0.82	0.50	0.62
plante	0.72	0.97	0.83
eau	0.80	0.80	0.80
électrique	0.67	0.53	0.59



b) Multi-Labelling

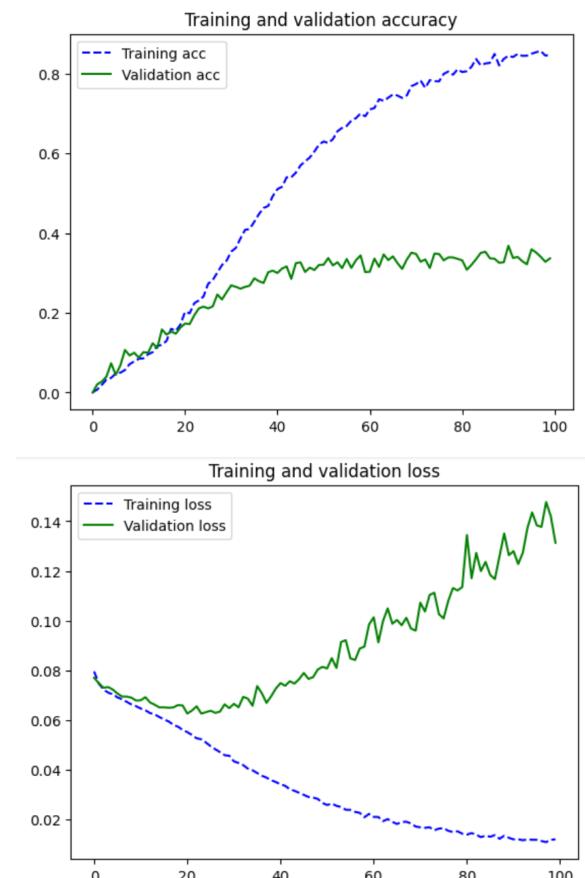
En faisant varier `load_data` comme précédemment :

1. Les 18 Classes

Nous avons d'abord considéré toutes les classes. Après une augmentation des données nous avions une accuracy d'environ 30%. Nous avons ensuite testé sur les données de test et calculé plusieurs métriques :

La précision globale est de 25.7%

Classe	Précision	Rappel	F1-score	TP	FP	FN
acier	0.43	0.41	0.42	9	12	13
combat	0.40	0.22	0.29	4	6	14
dragon	0.53	0.47	0.50	8	7	9
fée	0.50	0.18	0.27	2	2	9
glace	0.60	0.35	0.44	6	4	11
insecte	0.52	0.42	0.46	13	12	18
normal	0.33	0.36	0.35	14	28	25
plante	0.64	0.36	0.46	14	8	25
poison	0.71	0.35	0.47	12	5	22
psy	0.64	0.22	0.33	9	5	32
roche	0.30	0.26	0.28	7	16	20
sol	0.45	0.23	0.30	10	12	34
spectre	0.67	0.30	0.41	6	3	14
ténèbres	0.53	0.42	0.47	10	9	14
vol	0.29	0.28	0.29	11	27	28
électrique	0.50	0.25	0.33	4	4	12
feu	0.68	0.52	0.59	13	6	12
eau	0.57	0.52	0.54	31	23	29



Le tableau est très intéressant car il illustre bien la capacité de notre réseau à classer les pokémons par types selon chacun d'entre eux. Il apparaît ainsi que les types poisons, feu et spectres sont relativement facile à classer. Dans le cas du feu, cela est peu surprenant car pour les humains, il est souvent plutôt simple de repérer ce genre de pokémon : ils sont de couleur rouge orangé, ont souvent des flammes sur leur corps etc. à l'image de Galopa ou Dracaufeu. Pour ce qui est des pokémons poisons, c'est davantage surprenant, ils sont régulièrement de couleurs violette mais des pokémons d'autres types également (psy, ténèbres, spectres)... Cependant dans le cas du type poison, le violet reste souvent plus sombre et les pokémons ont également une apparence "toxique".

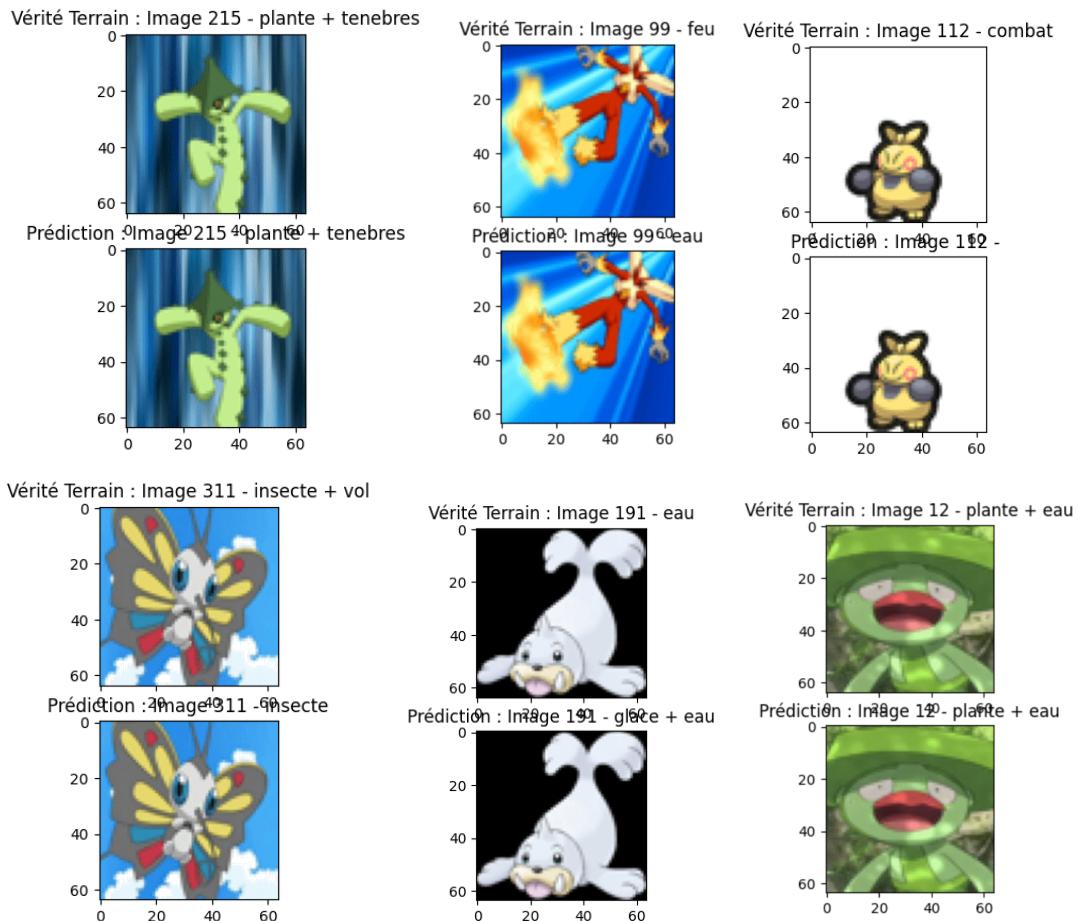
A l'inverse, les pokémons de types Vol, Roche et Normal sont à l'inverse assez difficiles à détecter pour le réseau de neurone. Ce n'est pas surprenant pour les pokémons de types normal car ce sont les pokémons "sans types" et on les reconnaît d'avantages à leur absence de caractéristiques propres. De même, les types vols peuvent être assez difficiles à classer même si la présence d'ailes est souvent un bon indicateur du type. C'est pour ça

qu'utiliser un réseau à base d'animaux pour pré-apprendre aurait pu être utile dans cette situation. Enfin, en ce qui concerne les pokémons de type roches, on peut supposer que la difficulté vient de la proximité d'apparence avec le type sol.

Enfin, la précision globale de 25.7% est assez faible en apparence. Cependant, il ne faut pas oublier qu'avec 18 classes, en les classant au hasard, on aurait environ 5.5% de bonnes réponses ce qui rend ces premiers résultats relativement satisfaisants.

Même en testant sur les pokémons de génération 5 et 6 (des pokémons que le réseau n'a jamais vu), la précision globale tombe à 20.2%, ce qui reste mieux qu'une labellisation aléatoire.

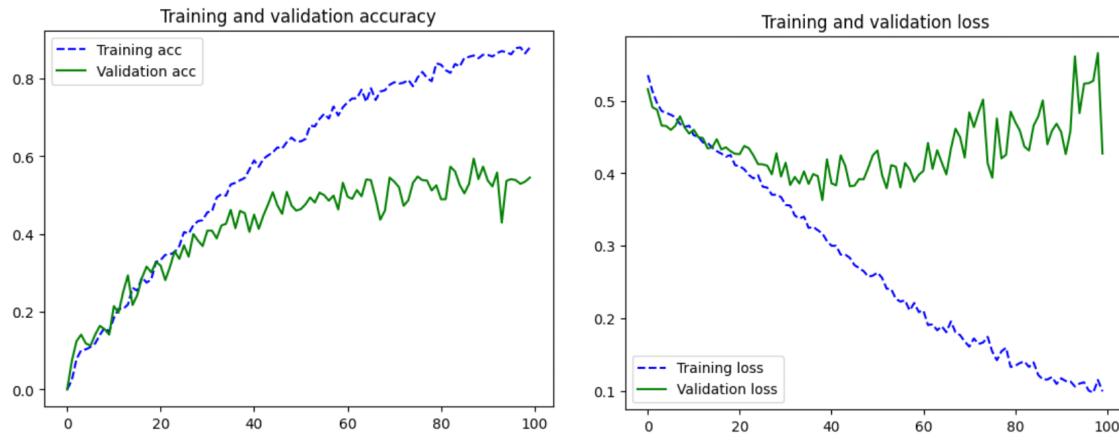
Quelques exemples de prédictions (en bas) avec la vérité terrain (en haut) :



D'ailleurs nous pouvons remarquer que nous avons réussi à donner les 2 bons labels au pokémon lombre.

2. Les classes ayant entre 360 et 400 images

Pour avoir d'encore meilleurs résultats nous avons voulu considérer des classes ayant à peu près le même nombre d'images. Dans cette partie nous avons bien pensé à changer la loss par une simple Binary Cross Entropy. Après augmentation des données nous sommes arrivés à ces résultats :

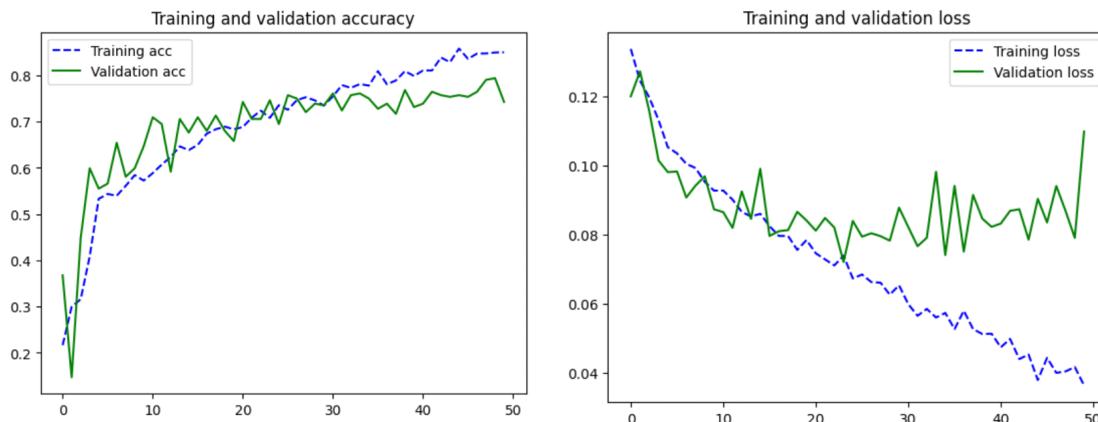


La précision globale est de 36.3%

Classe	Précision	Rappel	F1-score	TP	FP	FN
vol	0.61	0.50	0.55	22	14	22
plante	0.69	0.73	0.71	33	15	12
psy	0.45	0.51	0.48	20	24	19
normal	0.59	0.42	0.49	16	11	22
sol	0.46	0.29	0.35	11	13	27

3. Quatre classes distinctes

Après augmentation des données nous sommes arrivés à ces résultats :

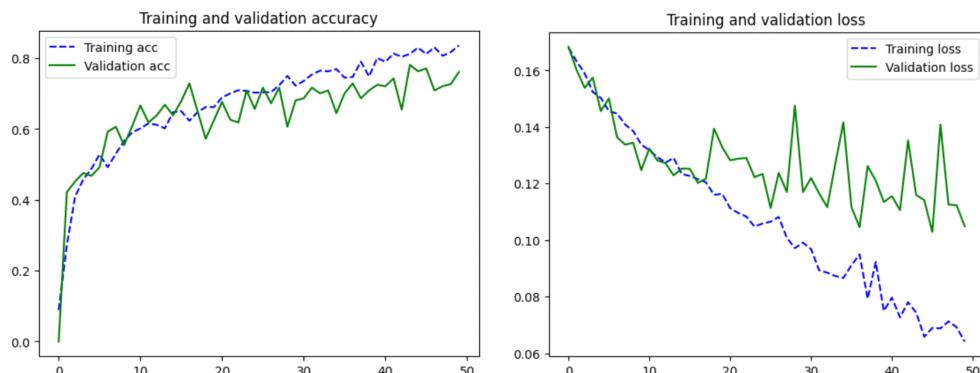


La précision globale est de 67.0%

Classe	Précision	Rappel	F1-score	TP	FP	FN
feu	0.71	0.60	0.65	15	6	10
plante	0.89	0.89	0.89	40	5	5
eau	0.85	0.83	0.84	55	10	11
électrique	0.73	0.50	0.59	8	3	8

4. Trois classes proches

Nous voulions maintenant voir si nous arrivons toujours à d'aussi "bon" résultats avec des classes qui sont difficilement distinctifs. Nous avons choisi les classes insectes, plantes et poison. Après augmentation des données nous sommes arrivées à ces résultats :



La précision globale est de 69.7%

Classe	Précision	Rappel	F1-score	TP	FP	FN
plante	0.90	0.84	0.87	38	4	7
insecte	0.77	0.90	0.83	27	8	3
poison	0.78	0.67	0.72	18	5	9

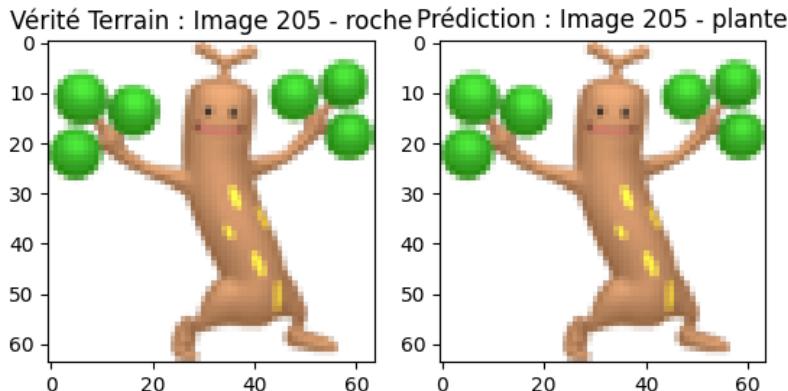
Globalement le Mono-Labelling a de meilleurs résultats que le Multi-Labelling. On peut expliquer cela par le fait qu'avec le Multi-Labelling, le réseau a beaucoup plus de chance de faire des erreurs sachant qu'il peut choisir entre 0 et toutes les classes.

IV) Conclusion

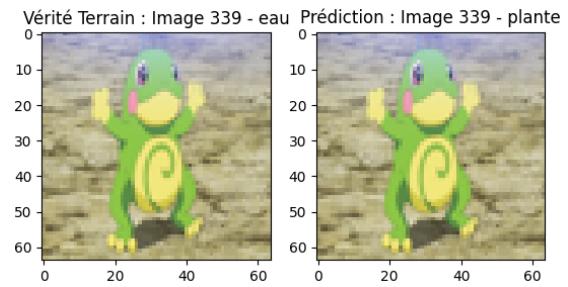
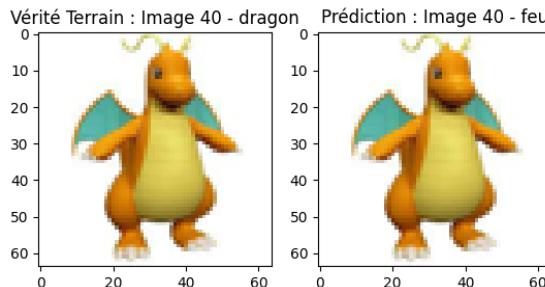
Nous remarquons donc que la grande complexité du problème vient du fait d'avoir 18 classes, et parmi ces 18 classes la répartition des pokémons n'est pas uniforme. En diminuant le nombre de classes, la précision globale augmente rapidement, même si on ne garde que des classes qui se ressemblent à l'œil nu.

De plus, le Mono-Labelling présente globalement de meilleurs résultats que le Multi-Labelling. Nous pensons que forcer le réseau de Multi-Labelling à choisir entre 1 et 2 types est une bonne piste d'amélioration pour diminuer le nombre de choix possibles qu'il peut faire.

Ce réseau met également en évidence qu'on ne peut pas trouver les types des pokémons seulement grâce à leurs apparences. Un pokémon comme 'simularbre' simule un arbre et arrive à tromper notre réseau, on devrait donc peut-être se réjouir que notre réseau reconnaissse une plante.



On peut également noter que le design des pokémons est artificiel, ils ont été créés par des humains et qu'il n'y a pas forcément de logique à l'association entre l'apparence et le type. De plus, à chaque nouvelle génération le style des pokémons évoluent et ajoute de nouvelles caractéristiques aux types et ainsi beaucoup de types partagent des caractéristiques. Par exemple concernant la caractéristique de la couleur, Dracolosse qui est rouge/orange mais qui n'est pas de type feu, ou Tarpaud qui a une peau verte mais n'est pas de type plante :



Cependant, une personne pourrait prédire assez facilement le type des 2 pokémons précédents en s'appuyant sur d'autres caractéristiques qu'uniquement la couleur.

De plus, nous aurions bien aimé avoir le temps de partir d'un réseau qui sait déjà reconnaître les caractéristiques des animaux. On ne pense pas que le résultat obtenu aurait été forcément meilleur car les types partagent beaucoup de caractéristiques, certains pokémons sont des mélanges d'animaux réels, que les caractéristiques sont très variés dans un même type, et qu'elles ne sont jamais nécessaires ou suffisantes pour caractériser un type. Cependant il aurait été intéressant de comparer les résultats.

Enfin, nous aurions également aimé tester une architecture telle que mobileNet pour voir si on n'obtient pas de meilleurs résultats.

Globalement nous sommes contents de nos résultats face à un problème si complexe.