PROJET TERMINALE 48H : IA / MACHINE LEARNING - RECONNAISSANCE AUTOMATIQUE DES MALADIES SUR LES CULTURES

Cécile TONNERRE

RÉSUMÉ :

Ce projet porte sur le thème de l'intelligence artificielle. L'objectif est de mettre au service l'IA au service de **l'agriculture de précision**. Grâce à des techniques de deep learning, la reconnaissance automatique de maladie permettrait d'utiliser de façon ciblée et limitée les produits phytosanitaires sur les cultures.

Le projet est réalisé avec le logiciel **Matlab**.

Ce projet cible uniquement la reconnaissance des maladies (traitement d'images, classification). Dans un second temps, on pourrait imaginer transférer ce code sur une carte embarquée (sur un drone par exemple) et l'associer à des actionneurs afin que le traitement soit fait automatiquement pendant le survol des cultures.

CYCLE : terminal : 1ère, terminale

AUTEURS :

- Cécile TONNERRE

Table des matières

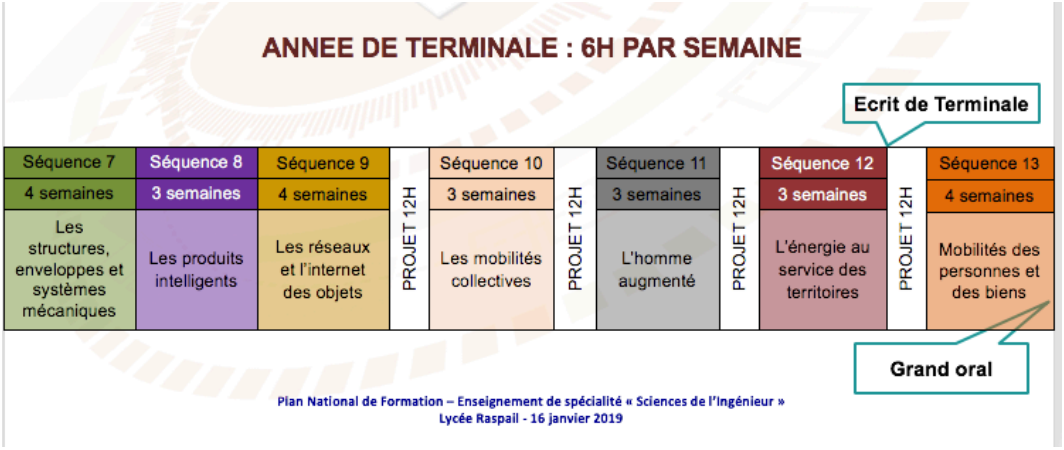
I. Présentation

1. Présentation du projet et problématique

a. Fiche Projet

Fiche présentation du projet 48h

Le projet propose de se dérouler sur quatre période de 12h selon la préconisation du PNF.



cf.

b. Problématique

Le smart farming

D'après les Nations Unies, la population mondiale atteindrait 9,7 milliards en 2050, ce qui nécessitera une forte augmentation de la production agricole, alors que les terres disponibles pour les cultures diminueront.
L'utilisation des nouvelles technologies peuvent permettre d'améliorer significativement la productivité pour répondre à ces besoins.
Le smart farming est l'utilisation des nouvelles technologies dans l'agriculture, qui peut prendre diverses formes
surveillance des cultures et des élevages
surveillance et prévisions météorologiques pour adapter la consommation d'eau, mieux planifier les tâches
surveillance de la qualité des sols, des cultures, pour optimiser la consommation d'eau, l'utilisation d'engrais et/ou de pesticides
L'intelligence artificielle permettra par exemple de surveiller les cultures, et d'utiliser les produits phytosanitaires de façon ciblée, au mètre carré près.
Dans ce cadre, ce projet se propose de reconnaître automatiquement les cultures saines des cultures touchées par une maladie ou un parasite. C'est donc la brique qui serait implantée dans un drone pour un épandage sélectif, qu'on appele aussi "agriculture de précision".

Références et sources

[Businessinsider] Smart Farming in 2020: How IoT sensors are creating a more efficient precision agriculture industry[p.]
cf.

c. Ressources matérielles

RÉSUMÉ :
Ce projet nécessite très peu de ressources matérielles, puisqu'il utilise essentiellement Matlab. La contrepartie est qu'il est important d'avoir des machines suffisamment puissantes pour utiliser Matlab et que les calculs ne prennent pas trop de temps.
Il est également possible d'utiliser Matlab online.

Logiciels

Matlab + support pour Raspberry

Pour la mise au point de ce projet, Matlab a été utilisé

- en version 2021b sous Windows
- en version 2019b sous MacOS 10.13

Le machine learning et le deep learning sont des techniques qui utilisent **énormément de calculs.** Les temps d'exécution peuvent donc être très longs, en particulier pour l'entraînement du réseau neuronal.
A titre d'exemple, pour un entraînement fait sur un problème simple, le temps nécessaire a été de l'ordre de
- 3/4 minutes sur un PC Windows 10 récent (processeur AMD Ryzen 5 2600X Six-Core, 16Go RAM)
- 14 minutes sur un Macbook pro "ancien" (mid-2012 / disque SSD / 32Go RAM)
Il peut donc être pertinent de **faire un test sur les les PC** disponibles au lycée pour valider la faisabilité du projet.
Le temps de calcul est une donnée qui fait partie du projet, les élèves devront donc apprendre à gérer cette contrainte qui impliquera de

- bien relire et vérifier son code avant de lancer l'exécution
- planifier le travail en fonction de l'emploi du temps. Par exemple, lancer un calcul avant une pause.
- Mettre à profit le temps de calcul pour avancer sur d'autres points du projet, rédiger le journal de bord etc.

Simulink

Simulink sera utilisé, ainsi que le support package pour Raspberry.

Matériels

Raspberry Pi

Pour le déploiement, j'ai utilisé une carte Raspberry Pi, de préférence une Pi 4 pour avoir de meilleurs performances (flux vidéo, calcul de la classification.

Caméra

Le projet a été mis au point avec un module caméra Raspberry Pi. Une autre caméra type USB ou port natif peut tout à fait convenir.

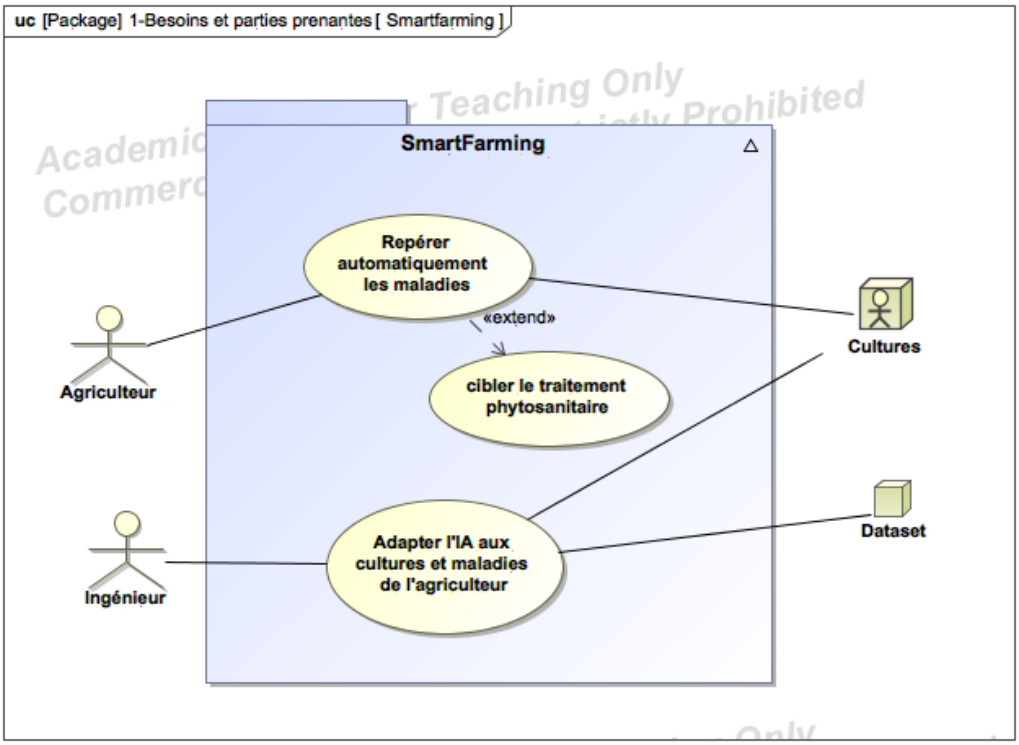
! *Attention :* Branchements de la Raspberry

Pour ce projet, la Raspberry doit être équipé d'un écran / clavier / souris car la distribution Matlab est une version avec Desktop et même si il est possible de se connecter à distance en SSH, pour voir la caméra on aura besoin de l'écran.
Attention contrairement à la Pi3 la Pi 4 a un connecteur **micro-HDMI** ! (et pour l'alimentation elle a un port USB-C et non plus micro-USB).

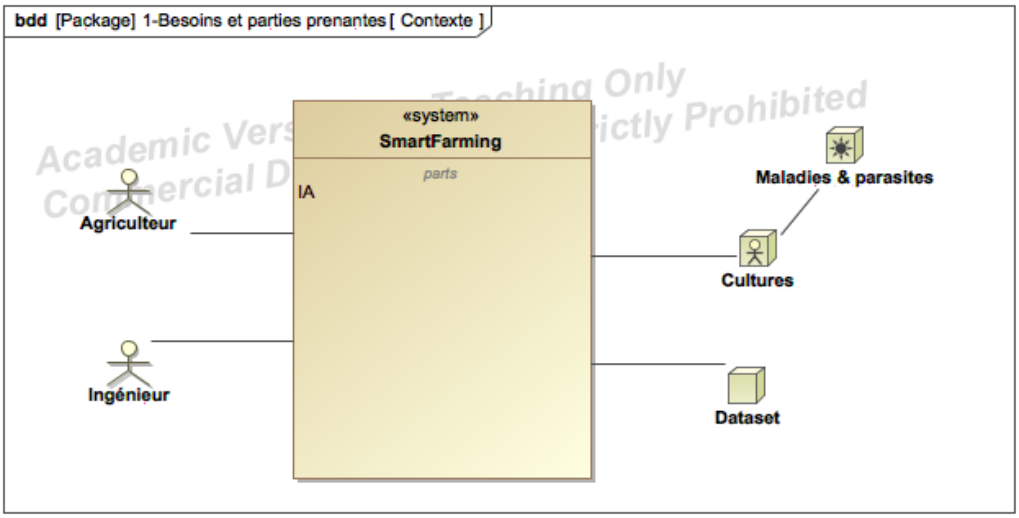
2. Cahier des charges

a. SysML

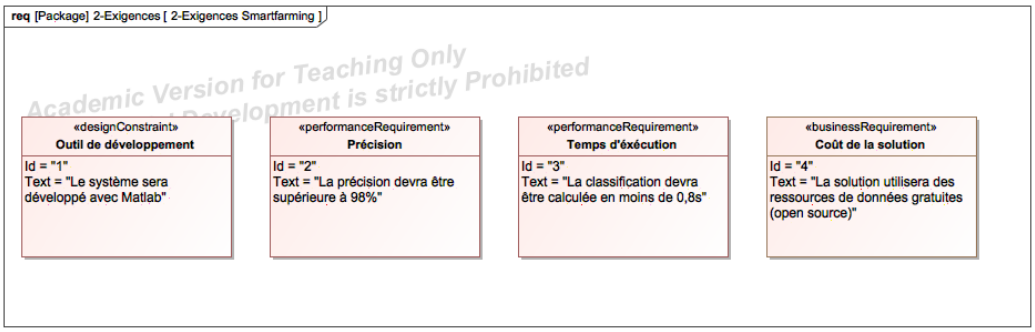
Use Case Diagram



Context Diagram



Requirement Diagram



II. Déroulement proposé

RÉSUMÉ :

Le déroulement proposé reprend la proposition de progression du PNF : il se fera sur quatre périodes de 12h.

1. Prise en main / Installation

DURÉE : 12h

RÉSUMÉ :

Les élèves prennent en main Matlab : ce temps sera variable, selon qu'ils aient déjà utilisé Matlab ou non.
Les élèves appréhendent les notions d'intelligence artificielle, de machine learning, deep learning, de classification.
Répartition :
Les élèves ont les mêmes tâches.

a. Découverte IA

RÉSUMÉ :
Les élèves appréhendent les notions d'IA, machine learning, Deep learning, de matrice de confusion

Intelligence artificielle / machine learning

Qu'est-ce que l'intelligence artificielle ?
Qu'est-ce-que le marchine learning ?
Vidéos / exercices
(non détaillé)

Deep learning et intérêt pour les images

Différences entre machine learning et deep learning.
Application du deep learning : reconnaissance d'images.
(non détaillé)
Vidéo / exercices

Workflow et matrice de confusion

L'obtention d'un réseau : quelles étapes ?
Boucler jusqu'à obtenir une précision satisfaisante.
La matrice de confusion.

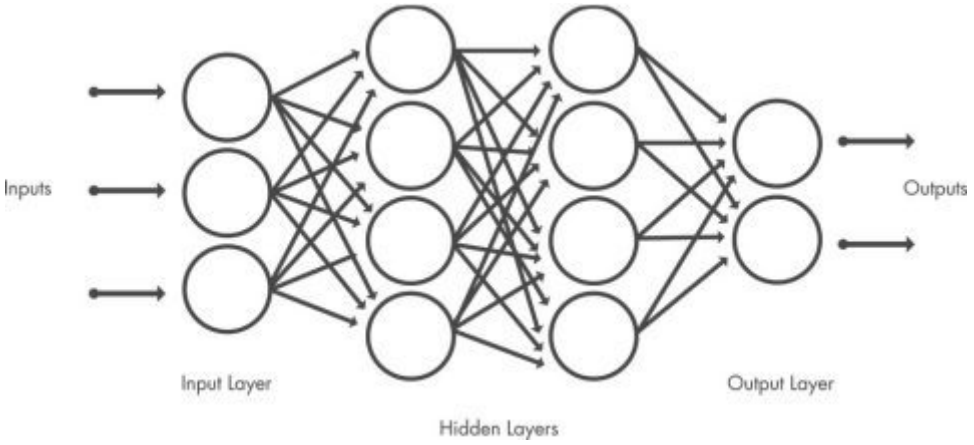
Réseau neuronal convolutif / Convolutional Neural Network (CNN)

Qu'est-ce-qu'un CNN ?

Référence : [Mathworks] Réseau neuronal convolutif : 3 choses à savoir[p.]
Un CNN est une structure composée de couches, permettant d'extraire des caractéristiques d'une image et d'en déduire à quelle classe (catégorie) appartient l'image.

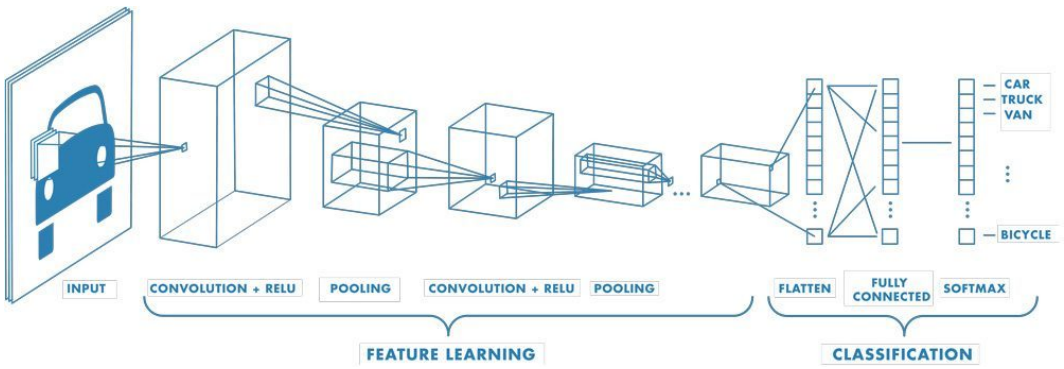
- **Input layer : couche d'entrée (unique).** L'image à analyser. Elle doit être au format attendu par le CNN (largeur hauteur et nb de couleurs)
- **Output layer : couche de sortie (unique)**. C'est la classe la plus probable pour l'image (ou catégorie)

- **Hidden layers** : couches intermédiaires cachées (multiples). Ce sont les couches de calcul, il en existe de plusieurs sortes. Ces couches font des calculs sur les images, ou des parties d'images. L'image calculée par une couche peut être réinjectée dans une autre couche.



Ici nous ne rentrons pas dans le détail des couches et des calculs effectués, qui sont d'un niveau mathématique élevé.
Les couches intermédiaires servent à extraire les informations de l'image. En fin de structure, la couche qui nous intéresse est la **couche de classification.**
La couche de classification est un réseau de **N neurones**, N étant le nombre de classes à identifier. La couche de classification la plus simple comporterait 2 neurones (output size de la couche), ce qui peut être utile par exemple en médecine, pour classer des images d'analyses cellulaires en 2 catégories

- échantillon sain
- échantillon pathologique



L'exemple d'AlexNet

Chargement dans Matlab

Le cnn alexNet comporte 25 couches (layers), la couche d'entrée s'appelle 'data', la couche de sortie s'appelle 'output'.

Le graphe des layers d'Alexnet

Chargement des layers d'Alexnet dans une variable

```
>> layersA = cnn.Layers

layersA =

  25x1 Layer array with layers:
```

```
    1   'data'     Image Input                   227x227x3 images with 'zerocenter' normalization
    2   'conv1'    Convolution                   96 11x11x3 convolutions with stride [4  4] and padding
[0  0  0  0]
    3   'relu1'    ReLU                          ReLU
    4   'norm1'    Cross Channel Normalization   cross channel normalization with 5 channels per element
    5   'pool1'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0
 0]
    6   'conv2'    Grouped Convolution           2 groups of 128 5x5x48 convolutions with stride [1  1]
and padding [2  2  2  2]
    7   'relu2'    ReLU                          ReLU
    8   'norm2'    Cross Channel Normalization   cross channel normalization with 5 channels per element
    9   'pool2'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0
 0]
   10   'conv3'    Convolution                   384 3x3x256 convolutions with stride [1  1] and padding
 [1  1  1  1]
   11   'relu3'    ReLU                          ReLU
   12   'conv4'    Grouped Convolution           2 groups of 192 3x3x192 convolutions with stride [1  1
] and padding [1  1  1  1]
   13   'relu4'    ReLU                          ReLU
   14   'conv5'    Grouped Convolution           2 groups of 128 3x3x192 convolutions with stride [1  1
] and padding [1  1  1  1]
   15   'relu5'    ReLU                          ReLU
   16   'pool5'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0
 0]
   17   'fc6'      Fully Connected               4096 fully connected layer
   18   'relu6'    ReLU                          ReLU
   19   'drop6'    Dropout                       50% dropout
   20   'fc7'      Fully Connected               4096 fully connected layer
   21   'relu7'    ReLU                          ReLU
   22   'drop7'    Dropout                       50% dropout
   23   'fc8'      Fully Connected               1000 fully connected layer
   24   'prob'     Softmax                       softmax
   25   'output'   Classification Output         crossentropyex with 'tench' and 999 other classes
```

Input layer : on voit qu'Alexnet attend en entrée des images de taille **227 x 227 pixels** en RGB (**profondeur 3**)

Dans Matlab, chargement du cnn Alexnet

```
>> cnn = alexnet

cnn =

  SeriesNetwork with properties:

         Layers: [25×1 nnet.cnn.layer.Layer]
     InputNames: {'data'}
    OutputNames: {'output'}
```

layer 23 = **couche de classification**. 1000 fully connected layer = couches à 1000 neurones, pour 1000 classes.

Ce qui est cohérent avec la classe 25 (**output** layer), qui contient 1000 classes ('tench' + 999)

Les 1000 classes d'AlexNet

Affichage de la liste des classes

```
>> layersA(25).Classes

ans =

  1000×1 categorical array

     tench
     goldfish
     great white shark
     tiger shark
     hammerhead
     electric ray
     stingray
     cock
```

```
    hen
    ostrich
    brambling
    goldfinch
    house finch
........
```

À quoi ressemblent les CNN utilisés

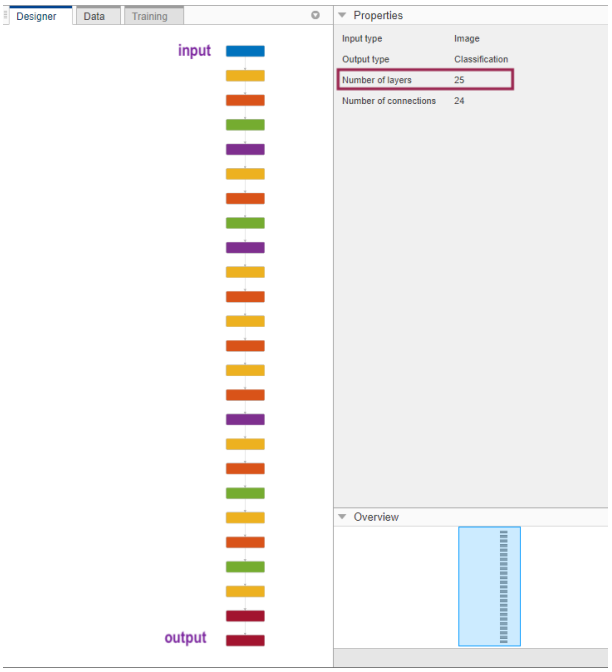
Utilisation du Deep Learning Designer

Le Deep Learning Designer permet d'ouvrir les réseaux installés dans Matlab, ce qui permet d'étudier les différentes couches d'une manière graphique, de voir leurs propriétés.
Il est ensuite possible de modifier la structure et d'enregistrer le nouveau graphe des layers pour l'utiliser, par exemple pour entraîner un cnn existant sur une banque d'images et ainsi l'adapter à un besoin spécifique.

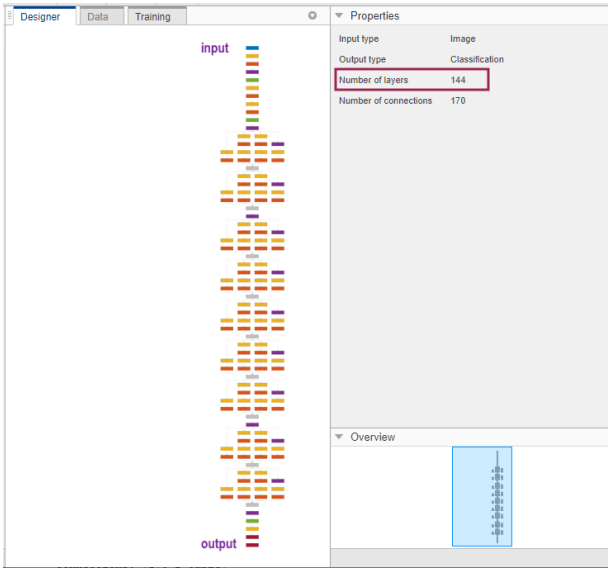
Comparaison des graphes

Ci-dessous, une comparaison rapide d'Alexnet et de Googlenet, deux cnn entraînés sur 1000 classes.

Structure des couches Alexnet : 25 couches en série



Structure des couches GoogleNet : 144 couche, structure complexe (une couche peut avoir des entrées de plusieurs couches, et envoyer la sortie à plusieurs couches

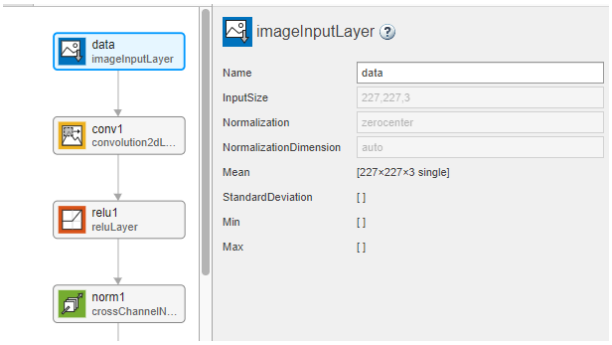


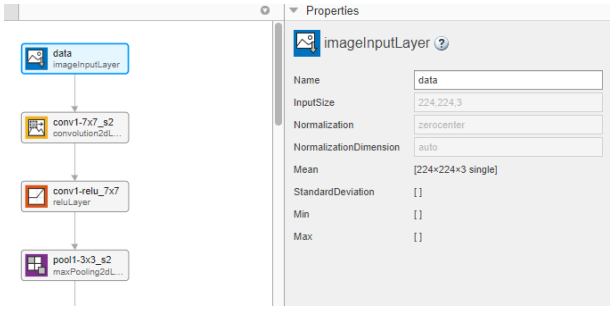
Comparaison des inputs

Il suffit de zoomer pour voir le détail de chaque couche. La couche d'entrée permet de connaître le format d'image attendu par le réseau.

AlexNet

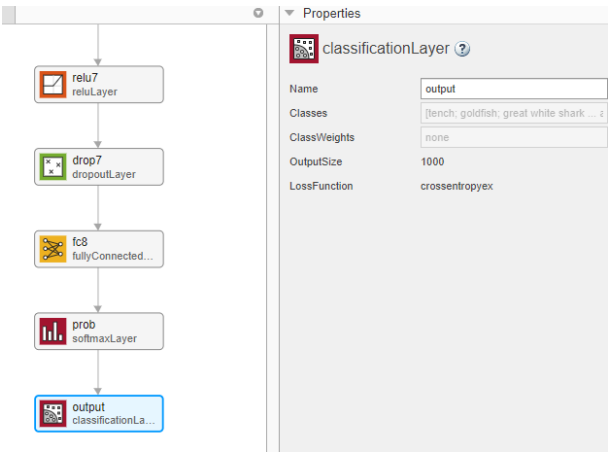
GoogleNet



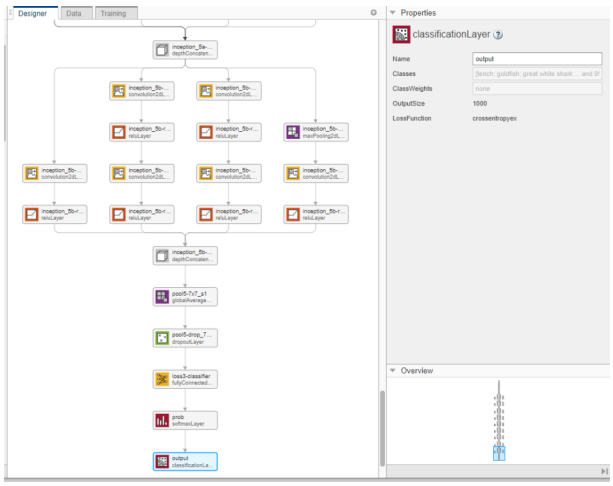


Comparaison des couches de sortie

AlexNet : 1000 classes



GoogleNet : 1000 classes



Pistes de réflexion

Comment se différentient ces deux réseaux ? Se pourrait-il que ce comparatif puisse nous aider à choisir un réseau plus à même d'acquérir de la précision sur des images spécifiques ?

Comparer les cnn proposés dans ce projet : alexnet, googlenet, resnet-18, squeezenet, mobilenetv2

b. Installation de la plate-forme de développement

RÉSUMÉ :
Installation de Matlab et add-ons

Installation des add-ons Matlab : deep learning

Réseau SqueezeNet

Add-on Deep Learning Toolbox Model for SqueezeNet Network[p.]

Réseau ResNet-18

Deep Learning Toolbox Model for ResNet-18 Network[p.][p.]

Réseau AlexNet

Deep Learning Toolbox Model for AlexNet Network[p.]

Réseau MobileNet-v2

Deep Learning Toolbox Model for MobileNet-v2 Network[p.]

Installation des add-ons Matlab : pour déploiement

Installation des add-ons pour déployer sur la Raspberry

L'installation pourra se faire au début du projet ou à l'étape de déploiement[p.36]

Support pour Raspberry Pi

Il faut ajouter deux add-ons :

MATLAB Support Package for Raspberry Pi Hardware[p.]

Simulink Support Package for Raspberry Pi Hardware[p.][p.]

Simulink coder

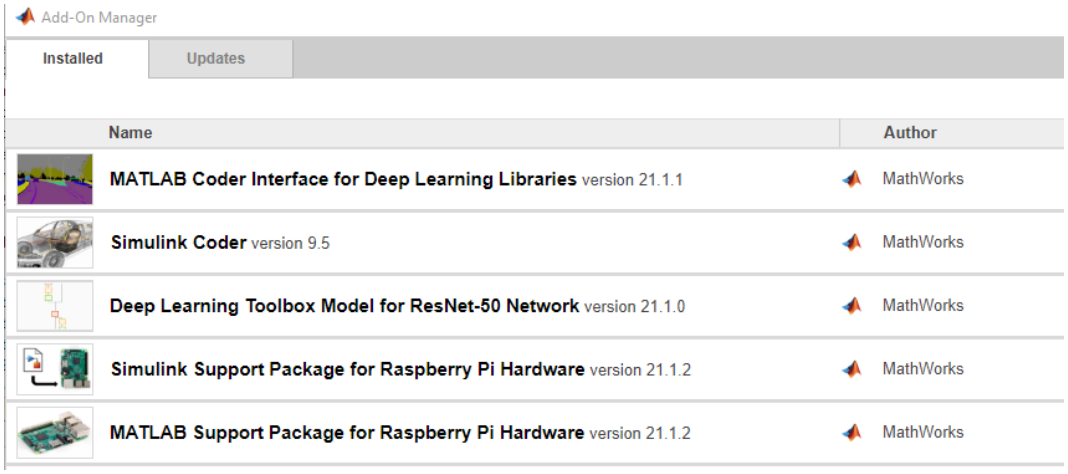
Cet add-on permet de générer du code C/C++ à partir de Simulink.
Utile pour déployer le modèle Simulink sur une Raspberry PI.

Pour fonctionner, il faut un compilateur compatible comme MinGW[p.] pour Windows.

Matlab Coder interface for Deep Learning Libraries

Nécessaires pour la compilation via Simulink Coder

Rédumé



2. Utilisation d'un réseau neuronal (classification)

DURÉE : 12h
RÉSUMÉ :

Les élèves utilisent un réseau neuronal existant pour faire de la reconnaissance automatique (classification)
Répartition :

- 2 élèves sur la programmation du script Matlab : les élèves se répartissent les cnn à tester
- 1 élève sur la création d'une IHM (App Designer)
- Classification avancée : chaque élève choisit un réseau à tester pour le comparatif

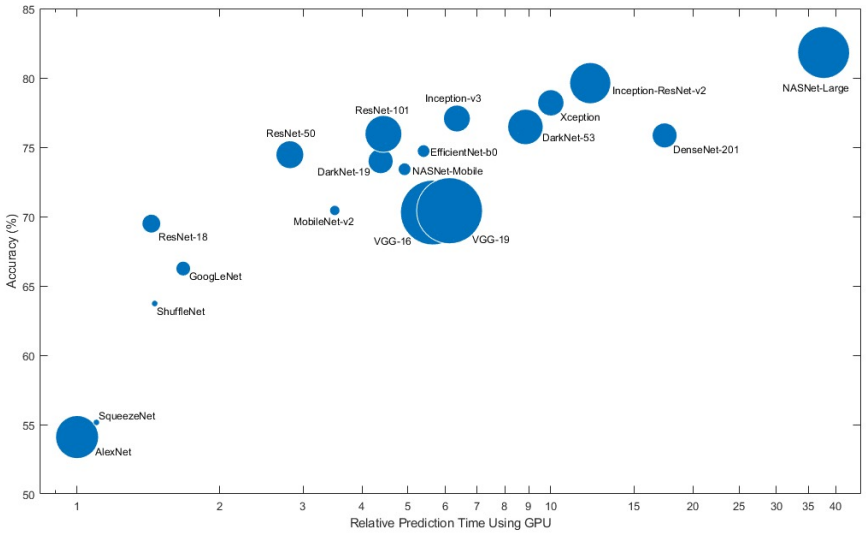
- Synthèse en commun et choix du réseau à entraîner

a. Expérimentation du deep learning : classification d'images

RÉSUMÉ :
Utilisation de CNN (convolutional Neural Networks) pré-entraînés proposés par Matlab pour classifier des images

Pretrained Deep Neural Networks

Matlab propose un grand nombre de réseaux pré-entraînés. L'image ci-dessous donne une vue de ces réseaux, avec comme critères le temps calcul de la classification et la précision.



Référence : Doc Matlab : Pretrained Deep Neural Networks[p.]
Cette étape permettra aux élèves de se familiariser avec l'utilisation des CNN et d'établir une comparaison de leurs performances. Ils pourront ensuite choisir de un à trois réseaux à étudier, ré-entraîner lors de l'étape suivante (en fonction de leur avancement).

1ère étape : Classification simple d'une image

Code complet

On donnera des indications, un code incomplet ou le code complet en fonction de l'avancement des élèves.

```
net = alexnet;  % Charge le réseau neuronal

img = imread('plage.jpg'); %lit l'image  la charge en mémoire
img =  imresize(img,[227,227]);  % mise à la taille input du réseau
label = classify(net, img);  % on récupère le label
figure  %crée la fenêtre d'affichage

imshow(img, 'InitialMagnification', 'fit')% affiche l'image
title(char(label)); % met le label en titre
```

Images proposées

imageSourcePlage.jpg[p.]

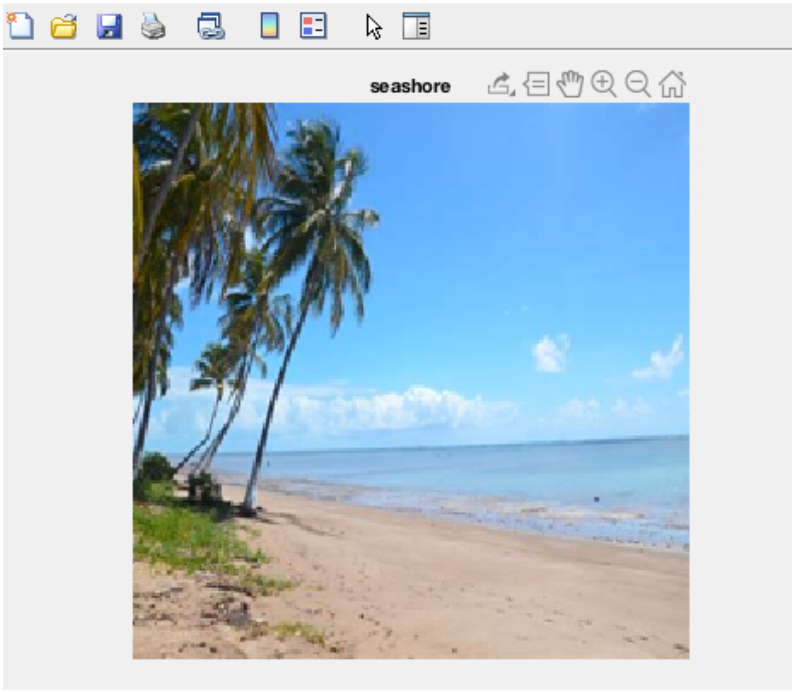
imageSourceCarlin.jpg[p.]

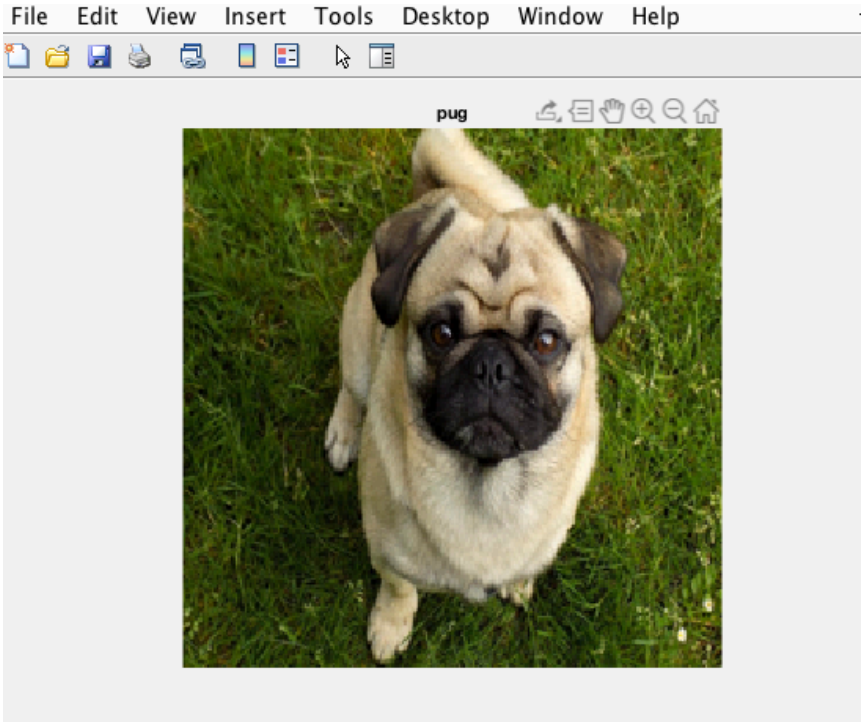
imageSourceFeuilleBouleau.jpg[p.][p.]
Pour démarrer : le plus simple est de copier les images dans le même répertoire que le script Matlab.

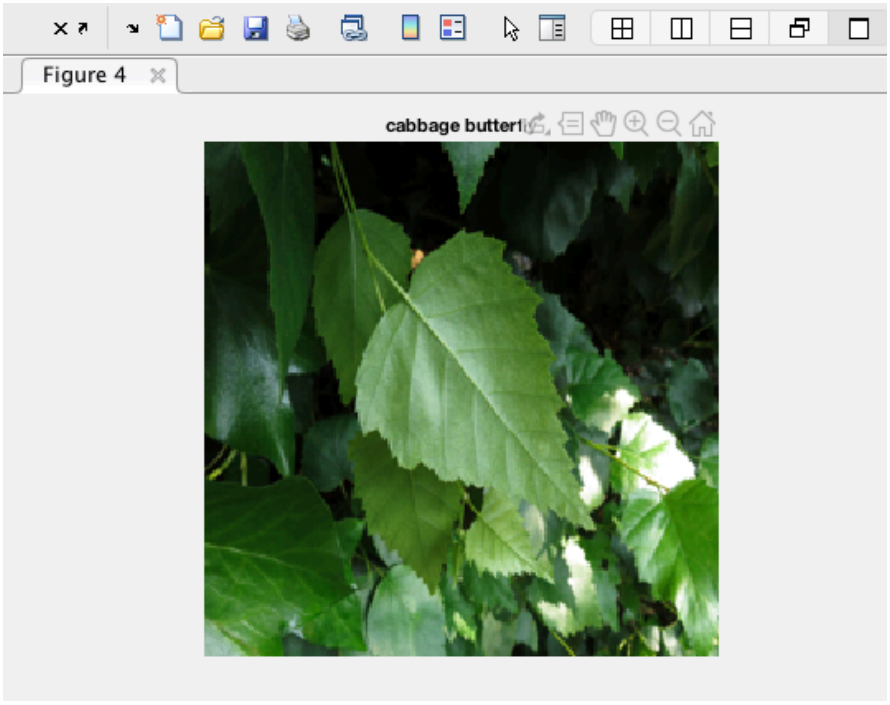
Résultats

Voici les labels attribués par alexnet :

- seashore (pour la plage)
- pug (pour le carlin)
- cabbage butterfly (pour la feuille de bouleau)







Synthèse: les élèves analysent et font des constatations.

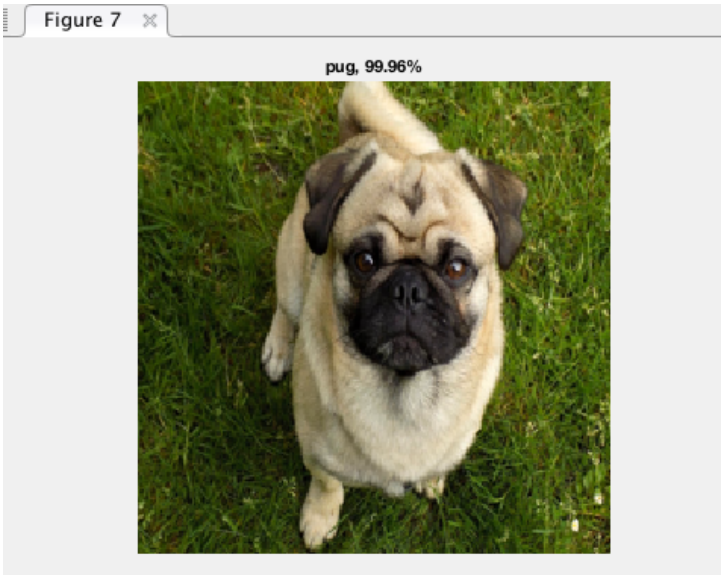
- Vérifier la traduction des mots (sur Reverso : pug = **carlin** cabbage butterfly = **piéride du chou**)
- Plus l'image est spécifique, moins le résultat est bon
- La fonction "resize" modifie le ratio de l'image --> à quel point cela peut-il influencer le résultat ? Il faudrait faire d'autres tests.

2ème étape : Classification avancée d'une image

Reprendre le code et ajouter la probabilité de confiance attribuée au label

Référence : Doc Matlab : Classify Image Using GoogLeNet[p.]

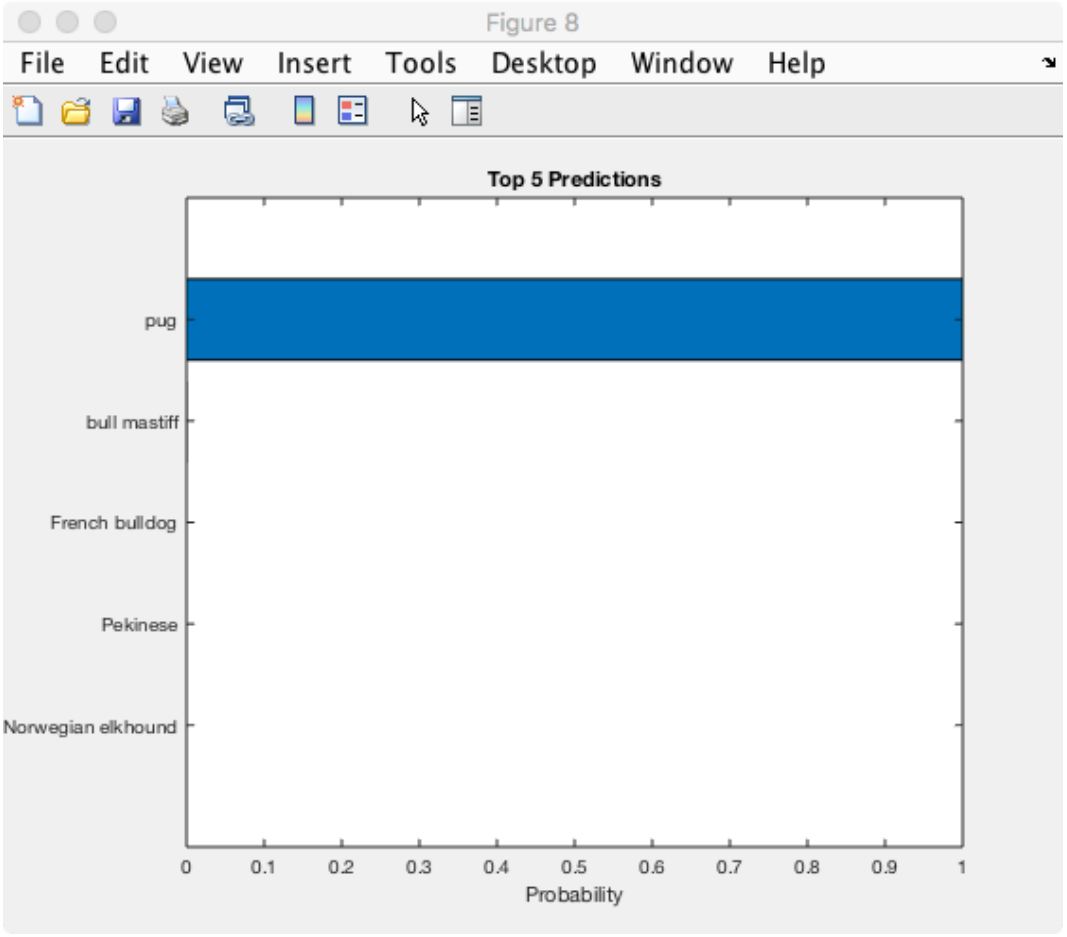
```
net = alexnet;  % Charge le réseau neuronal
classNames = net.Layers(end).ClassNames; % récupération de toutes les classes

img = imread('carlin.jpg'); %lit l'image  la charge en mémoire
img =  imresize(img,[227,227]);  % mise à la taille input du réseau

[label, scores]  = classify(net, img);  % on récupère le label et les probabilités

figure  %crée la fenêtre d'affichage

imshow(img, 'InitialMagnification', 'fit')% affiche l'image

% titre de l'image : label + probabilité calculée
scores(classNames == label)
title(string(label) + ", " + num2str(100*scores(classNames == label), '%.2f') + "%");
```



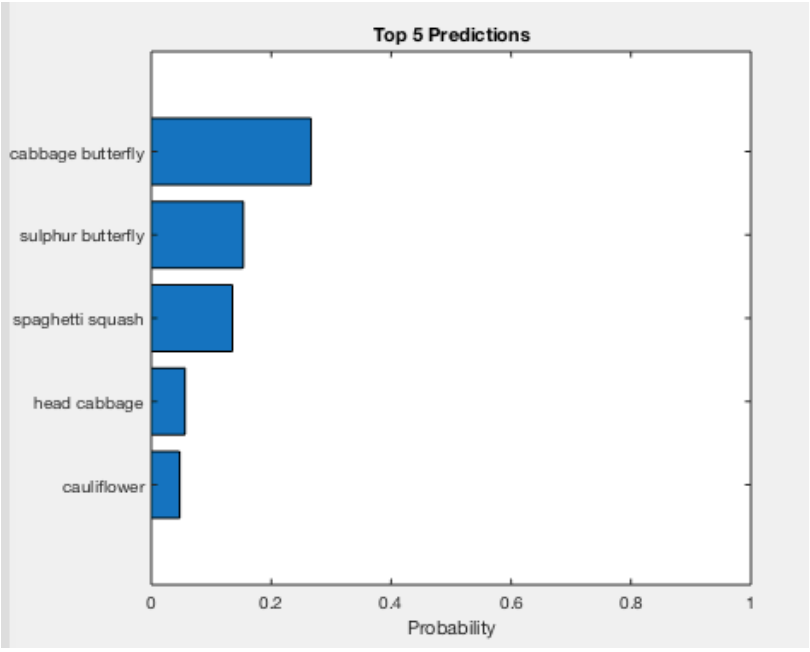
Reprendre le code et afficher également les 5 labels ayant les plus forts scores

```
net = alexnet;  % Charge le réseau neuronal
classNames = net.Layers(end).ClassNames; % récupération de toutes les classes

img = imread('carlin.jpg'); %lit l'image  la charge en mémoire
img =  imresize(img,[227,227]);  % mise à la taille input du réseau

[label, scores]  = classify(net, img);  % on récupère le label et les probabilités

figure  %crée la fenête d'affichage

imshow(img, 'InitialMagnification', 'fit')% affiche l'image

% titre de l'image : label + probabilité calculée
scores(classNames == label)
title(string(label) + ", " + num2str(100*scores(classNames == label), '%.2f') + "%");

% affiche les 5 plus fortes probabilités
[~,idx] = sort(scores,'descend');
idx = idx(5:-1:1);
classNamesTop = net.Layers(end).ClassNames(idx);
scoresTop = scores(idx);

figure
barh(scoresTop)
xlim([0 1])
title('Top 5 Predictions')
xlabel('Probability')
yticklabels(classNamesTop)
```

Exemple de résultat pour le carlin



On constate que les 5 scores les plus élevés donnent des races de chien.
Exemple de résultat pour la feuille de bouleau



On constate que le résultat est moins "sûr". (le réseau est moins confiant quant à son résultat).

Reprendre le code et afficher le temps de calcul nécessaire à la classification

Référence : Doc Matlab : Measure the Performance of Your Code[p.]
Utilisation de "tic" et "toc" juste avant et juste après la fonction pour connaître le temps d'exécution

```
tic
   % The program section to time.
toc
```

```
net = mobilenetv2;  % Charge le réseau neuronal
%à tester avec alexnet, squeezenet, googlenet, mobilenetv2, resnet18

classNames = net.Layers(end).ClassNames; % récupération de toutes les classes

img = imread('carlin.jpg'); %lit l'image  la charge en mémoire
img =  imresize(img,[224,224]);  % mise à la taille input du réseau

tic
[label, scores]  = classify(net, img);  % on récupère le label et les probabilités
toc

figure  %crée la fenêtre d'affichage
```

```
Command Window
>> labelImageCompareCnn
Elapsed time is 2.298138 seconds.
```

b. Expérimentation du deep learning : comparaison des cnn

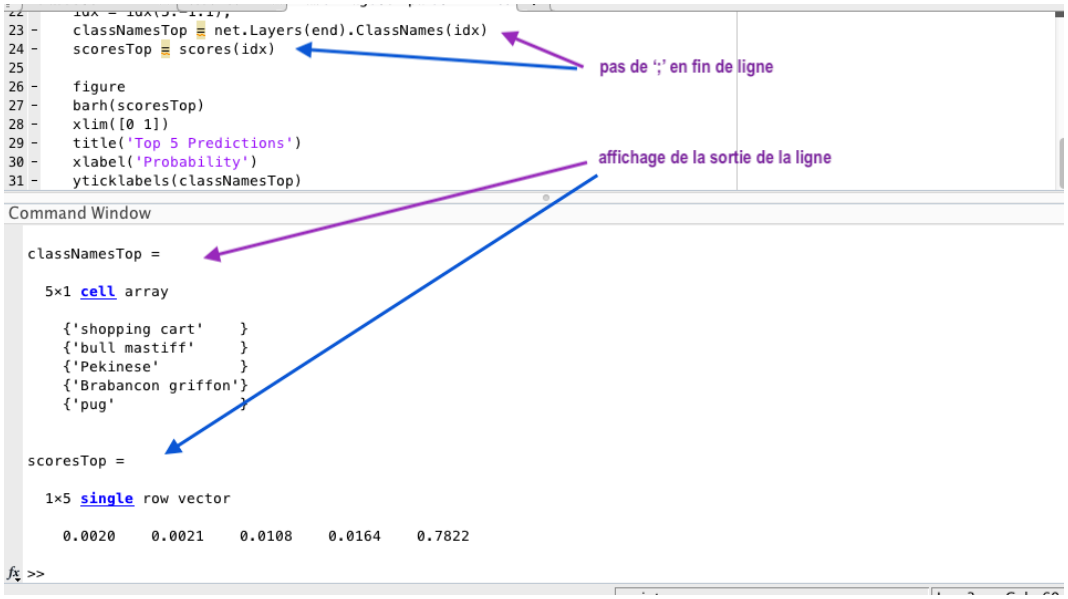
RÉSUMÉ :
Comparer les performances de plusieurs CNN

3ème étape : Classification comparée de plusieurs images (comparaison qualitative et quantitative)

> ! *Attention :* Input size
>
> Les 5 réseaux utilisés n'ont pas tous la même taille d'image en entrée ! Il faut adapter le code en conséquence.
>
> - 227 x 227 pour alexNet et squeezeNet
> - 224 x 224 pour googLeNet, MobileNetv2 et Res-Net18

L'objectif est d'analyser et de comparer le résultat de plusieurs réseaux, sur un panel de plusieurs images (ici 3)

Remarque : en enlevant les ' ;' en fin de ligne, Matlab affiche la valeur de la variable dans la fenêtre de commande.



	A	B	C	D	E	F	G
1		carlin					
2		alexnet	squeezenet	googlenet	mobilenetv2	res-net18	
3	score	99,96	99,98	98,91	78,22	98,61	
4	temps	0.212721	0.477291	0.833171	2.913375	1.101561	
5	Top1	pug	pug	pug	pug	pug	
6	top2	Brabancon griffon	bull mastiff	Pekinese	Brabancon griffon	Norwegian elkhound	
7	top3	French bulldog	French bulldog	Brabancon griffon	Pekinese	French bulldog	
8	top4	Pekinese	Pekinese	bull mastiff	bull mastiff	Pekinese	
9	top5	bull mastiff	Norwegian elkhound	Norwegian elkhound	shopping cart	bull mastiff	
10							
11		feuille de bouleau					
12		alexnet	squeezenet	googlenet	mobilenetv2	res-net18	
13	score	26,63	28,34	29,46	18,27	46,92	
14	temps	0.234208	0.404576	0.2946	2.187315	1.095072	
15	Top1	cabbage butterfly	acorn	buckeye	cucumber	head cabbage	
16	top2	sulphur butterfly	cabbage butterfly	custard apple	pot	buckeye	
17	top3	spaghetti squash	head cabbage	fig	buckeye	cucumber	
18	top4	head cabbage	cauliflower	bell pepper	head cabbage	spaghetti squash	
19	top5	cauliflower	walking stick	acorn	cauliflower	head cabbage	
20							
21							

Fichier obtenu :

Comparaison des résultats pour les 5 CNN[p.]

Analyse qualitative

- Rechercher la traduction et les images correspondants aux différents labels (exemple : comparer les images de pékinois, bulldog français etc. et estimer quel réseau donne les meilleurs résultats en terme de ressemblance)
- Classer les CNN en fonction du label proposé mais aussi des labels du top5 (exemple : mobilenetv2 qui fait apparaître "shopping cart" pour le chien).

Analyse quantitative : précision

- Comparer et classer les réseaux en fonction des scores attribués au label (pour un label pertinent)
- Quels sont les réseaux les plus susceptibles de répondre aux exigences du cahier des charges ?

Analyse quantitative : temps de calcul

- Comparer et classer les réseaux en fonction des temps de calcul

- Quels sont les réseaux les plus susceptibles de répondre aux exigences du cahier des charges ?

c. Création d'une IHM

IHM avec Matlab AppDesigner

A partir de l'exemple Create App that Uses Multiple Axes to Display Results of Image Analysis[p.], créer une IHM qui

- permet à l'utilisateur de sélectionner et uploader une image
- affiche le label prédit par l'IA.

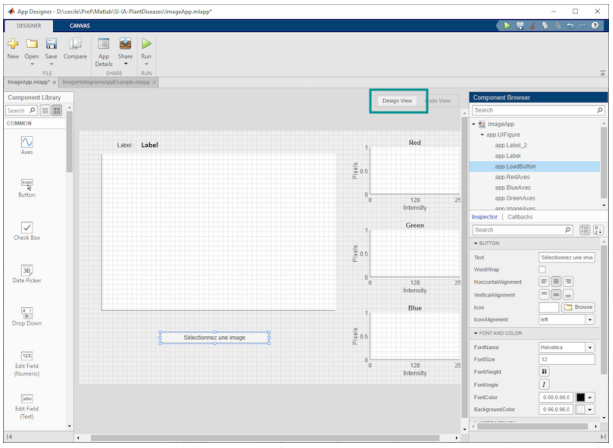
+ *Complément :* Présentation de AppDesigner (programmation événementielle un peu comme AppInventor)

Le designer présente deux vues

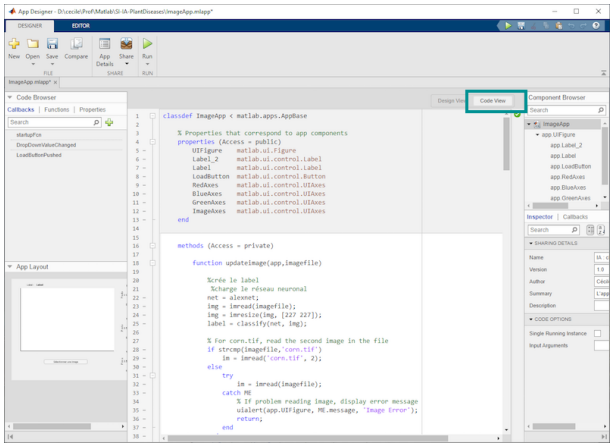
- une vue "design" pour l'IHM, on ajoute sur une grille des composants (image, bouton, liste déroulante etc.)
- une vue "code" avec le code Matlab.

On peut ensuite associer du code (action) à des événements (quand on clique sur un bouton par exemple). La programmation est en objet, on parlera de **propriétés** plutôt que de variables et de **méthodes** plutôt que des fonctions.

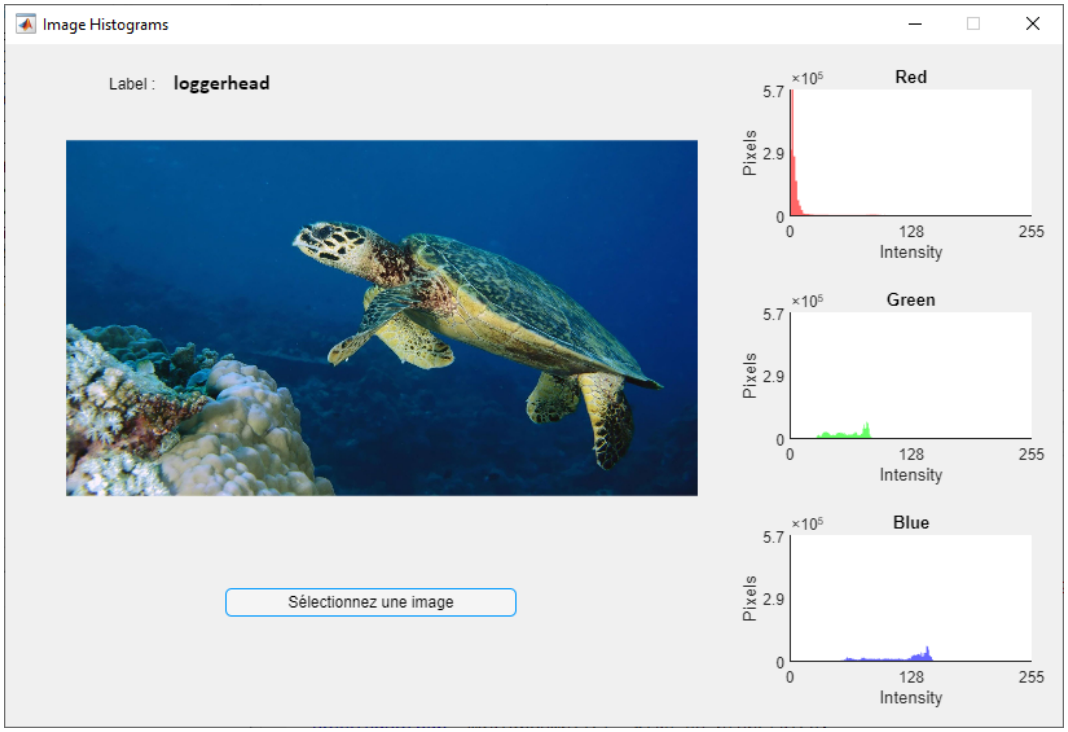
Design View



Code View



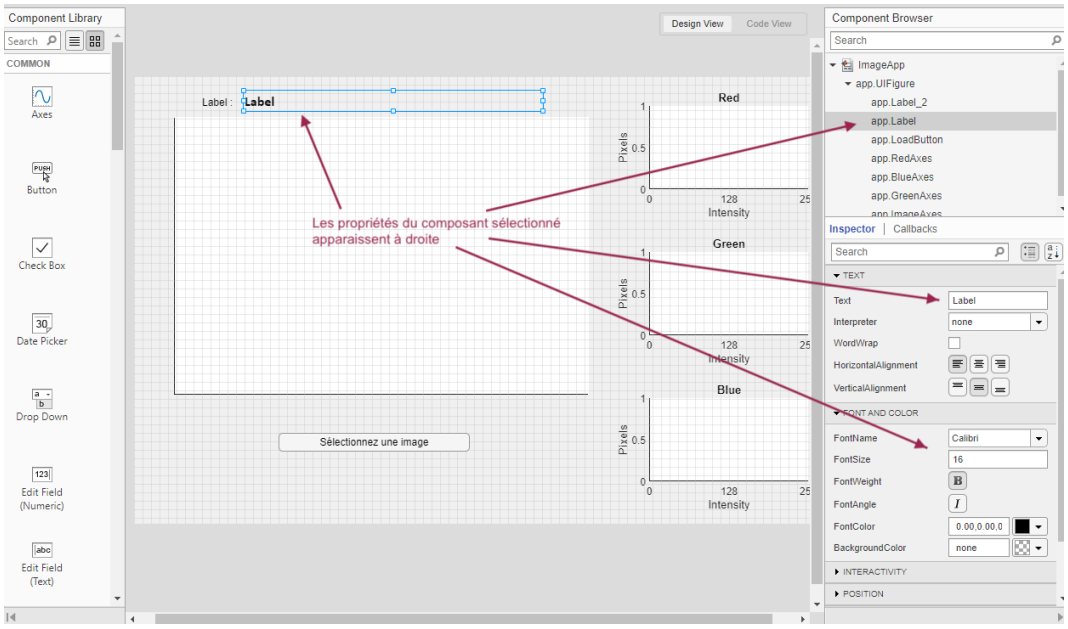
Exemple (avec alexnet)



Modifier l'interface (layout)

Ajouter le label

- Ajouter un composant texte pour l'affichage du label (modifié par le programme)
- Facultatif : ajouter un composant texte à afficher à côté (non modifiable)



Supprimer le menu déroulant avec les deux images préchargées

Facultatif, permet de simplifier l'interface.

Pour ne pas avoir d'erreur, il faut aussi supprimer le code correspondant.

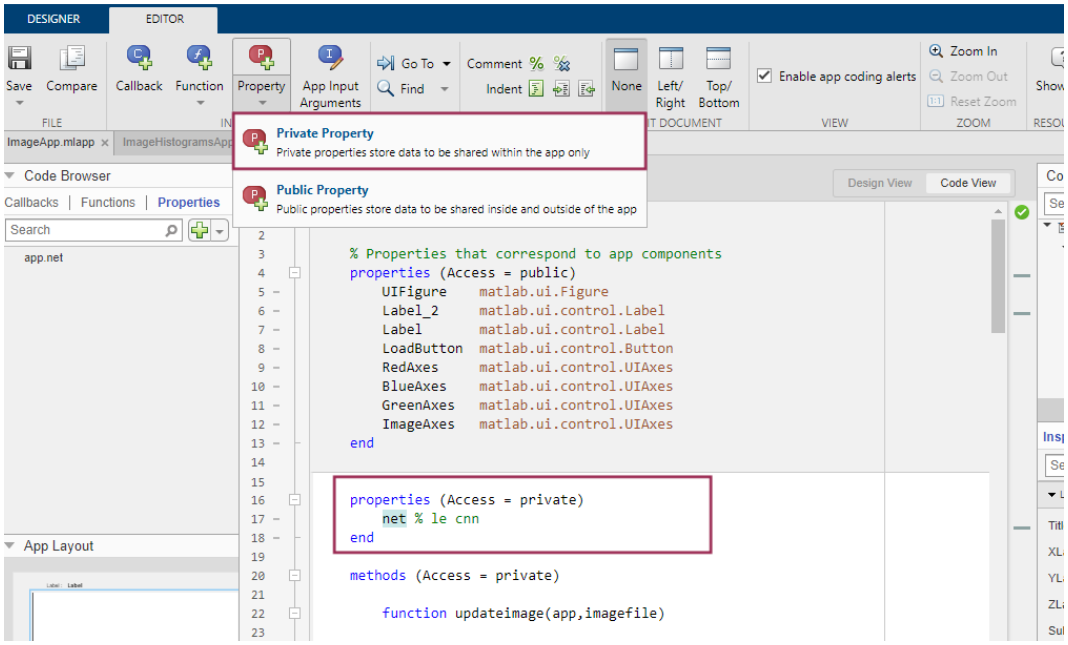
```
% Callback function
```

```
        function DropDownValueChanged(app, event)


        end
```

Modification du code

Chargement du cnn

Créer une propriété avec le menu dans l'éditeur. Ainsi le réseau sera accessible dans toute l'application.



Définir le réseau :

```
properties (Access = private)
        net = alexnet % le cnn
end
```

Les propriétés définies pour l'application sont accessibles dans tout le code en préfixant avec **app.**
Ici le réseau sera donc accessible avec **app.net**
Charger le réseau au démarrage de l'application (startupFcn)

```
function startupFcn(app)
            % Configure image axes
            app.ImageAxes.Visible = 'off';
            app.ImageAxes.Colormap = gray(256);
            axis(app.ImageAxes, 'image');

            % Chargement du cnn
            app.net = alexnet;

            % Update the image and histograms
            updateimage(app, 'peppers.png');
        end
```

Affichage du label

Le calcul et l'affichage du libellé se fait quand l'utilisateur a sélectionné une image. Code sur l'action "loadButton"

```
function LoadButtonPushed(app, event)
```

```
            % Display uigetfile dialog
            filterspec = {'*.jpg;*.tif;*.png;*.gif','All Image Files'};
            [f, p] = uigetfile(filterspec);

            % Make sure user didn't cancel uigetfile dialog
            if (ischar(p))
               fname = [p f];
               updateimage(app, fname);
            end
        end
```

On peut donc afficher le label au même endroit que l'affichage de l'image, dans "updateimage"

```
%crée le label
img = imread(imagefile);
img = imresize(img, [227 227]);
label = classify(app.net, img);
% le composant "Label" reçoit comme texte le label prédit par le réseau
app.Label.Text = label;
```

Exporter l'application

Créer une application standalone

Le menu "Share" permet de créer une application autonome, qui pourra s'exécuter sur un autre PC, même si Matlab n'est pas installé. Ce n'est pas détaillé ici.

3. Transfer Learning : Ré-entraînement d'un réseau neuronal

DURÉE : 12H
RÉSUMÉ :

Training et validation du réseau neuronal sur le dataset. Compte tenu des temps de calcul, il sera judicieux de valider l'entraînement et le choix des paramètres du training sur un sous-dataset.
Répartition :

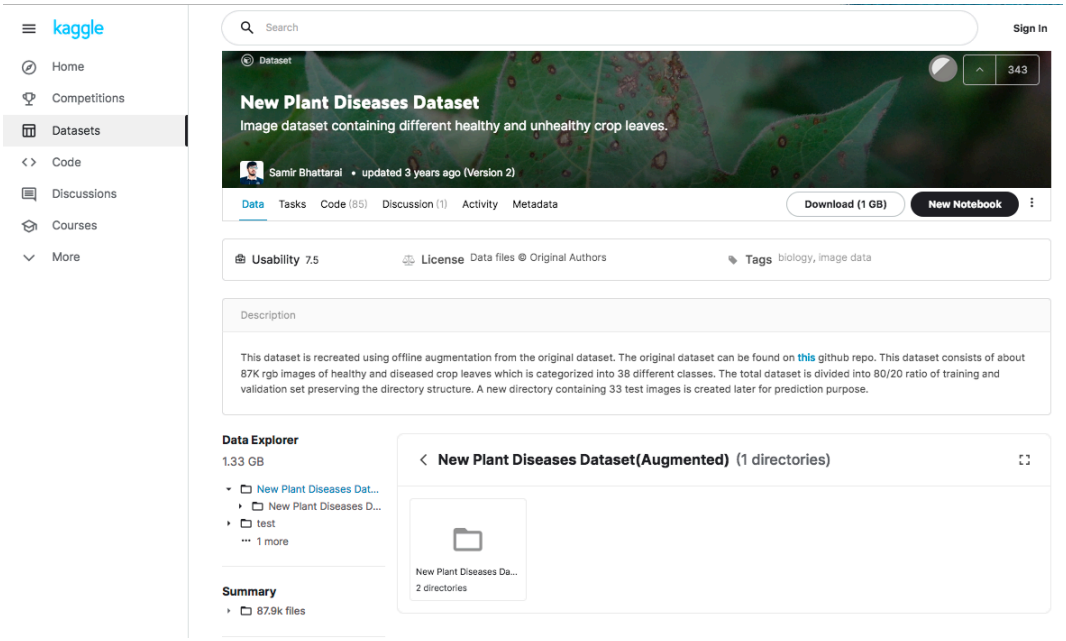
- 2 élèves sur la programmation
- 1 élève sur l'IHM
- Entraînement : répartition des paramètres à tester. Chaque élève pourra travailler sur un sous-dataset différent
- Comparatif et synthèse en commun
- Piste possible : modification du dataset avec labels en français (noms des répertoires)

a. Tranfer Learning 1 : réentraînement d'un cnn à partir d'une base d'images

Dataset utilisé

Ce projet utilise un dataset téléchargeable gratuitement sur Kaggle.

Kaggle : New Plant Diseases Dataset[p.]

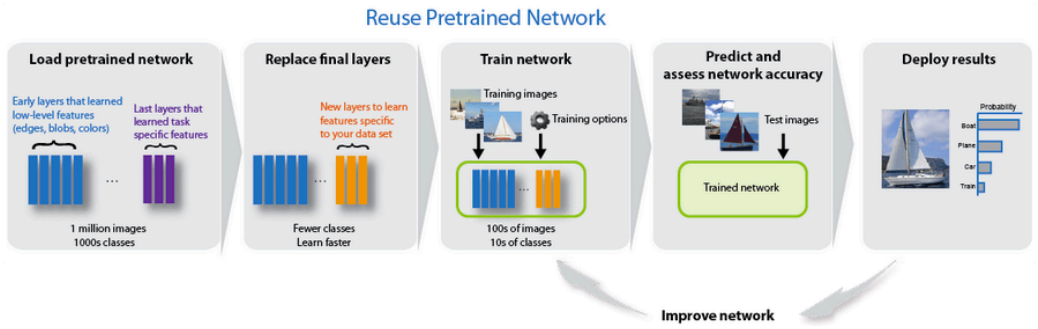


Special thanks to Samir Bhattarai for this amazing dataset !

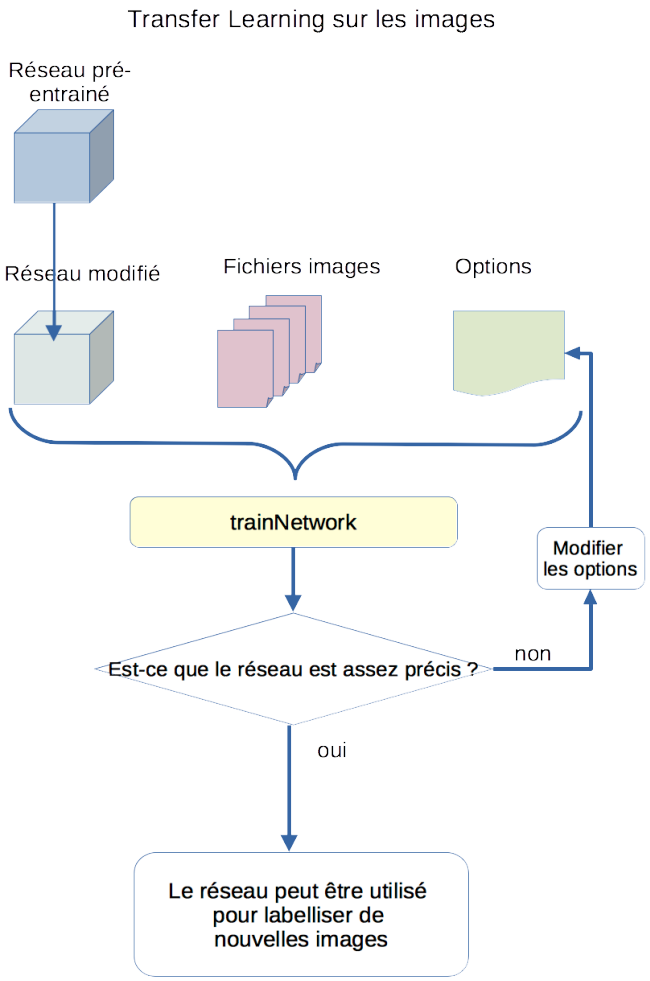
Découvrir le Transfer Learning

Principe

Voir également Découverte IA[p.8]



Transfer Learning Using Pretrained Network (Mathworks)



Étape 1 : Réseau modifié

Matlab permet de charger un réseau pré-entraîné existant, et de le modifier pour l'adapter à nos besoins.
La modification se limite à modifier les deux couches

- classification layer : remplacer la couche existante par une nouvelle couche de classification vierge avec en output size le nombre de classes à identifier
- output layer : remplacer la couche existante par une nouvelle couche output vierge le nombre de classes à identifier

Étape 2 : Fichiers images

Les images sont le plus souvent organisées dans une arborescence. Sous Matlab nous pourrons utiliser les "imageDatastore", une variable qui regroupe les images situées dans un répertoire (avec ou non des sous-répertoires).
L'utilisation des imageDatastore ont plusieurs avantages intéressants

- si il faut retailler les images pour les adapter au cnn, le datastore permet de retailler automatiquement chaque image avant le traitement
- les images sont tirées une par une du datastore de façon transparente au moment du traitement, ce qui évite de charger toutes les images en mémoire
- Matlab sait répartir les images pour avoir un ensemble pour le training et un ensemble pour le test

Étape 3 : options

On peut très bien lancer l'entraînement avec les options par défaut au début.

Étape 4 : entraînement et évaluation

Le temps de calcul peut être très long (plusieurs heures). Il faut donc penser à enregistrer le nouveau réseau en fin de calcul !
Après l'entraînement, le réseau peut être utilisé, on cherchera à contrôler la matrice de confusion pour valider la précision du réseau.

Comment faire ?

Explquer : datastore, imagedatastore, pool pour training / pour test etc...

Entraîner un cnn : epochs, iteration, mini-batch...

Explication des concepts
comment afficher la progression du training.

Exercice

Entraînement d'un réseau

Entraînement d'un réseau neuronal : ligne de commande

Étape 1 : réseau modifié

```
% chargement du cnn googlenet
net = googlenet
% récupération des layers
lay = net.Layers
% voir les détails de la input layer
inLayer = lay(1)
% connaître le format d'image en entrée
inSize = inLayer.InputSize
% récupération des layers (structure du réseau)
lgraph = layerGraph(net);
% création d'une couche neuronale pour 38 classes
newFc = fullyConnectedLayer(38,"Name","new_fc");
% on remplace la couche existante par la nouvelle
lgraph = replaceLayer(lgraph,"loss3-classifier",newFc)
% création d'une couche output vierge
newOut = classificationLayer("Name","new_out");
% on remplace la couche existante par la nouvelle
lgraph = replaceLayer(lgraph,"output",newOut)
```

! *Attention :* classification layer / output layer

La couche de classification est l'avant-dernière couche : elle attribue un score à chaque classe pour l'image traitée, c'est le réseau de neurones. Son type est "fully connected layer" car ce sont des neurones connectés.
La classe de sortie (output layer) traite les scores, les traduit en probabilités pour chaque classe, et donne en sortie la classe la plus probable. Son type est "classification output"
Il y a parfois ambiguïté entre la "output layer" et la "classification layer" car Matlab a attibué le type classificationLayer a la couche Classification Output.

Étape 2 : Image Datastore

```
% création du datastore à partir du répertoire "imgAll"
% les labels seront extraits des noms des répertoires
% le datastore prend "imgAll" et tous les sous-répertoires
imds = imageDatastore("imgAll", "IncludeSubfolders",true, "LabelSource","foldernames")
imds.Labels
% Répartition aléatoire des images,
% 70% pour le training
% 30% pour le test
[trainAllImgs, testAllImgs] = splitEachLabel(imds, 0.7, "randomized")
```

```
% augmentedDatastore permet d'ajouter une fonction de pré-traitement
% ici on adapte la taille de l'image au cnn
trainds = augmentedImageDatastore([224 224],trainAllImgs);
testds = augmentedImageDatastore([224 224],testAllImgs);
```

Etape 3 : Options

```
options = trainingOptions("adam",...
    'InitialLearnRate', 0.01, ...
    'LearnRateDropFactor',0.2, ...
    'LearnRateDropPeriod',5, ...
    'MaxEpochs',10, ...
    'MiniBatchSize',64, ...
    'Plots','training-progress',...
    'Shuffle','every-epoch');
```

Pour plus de détails : Doc Matlab : trainingOptions[p.]

Etape 4 : Training et évaluation

```
% entraînement du cnn. Temps de calcul important !!
plantDiseasesNet = trainNetwork(trainds,lgraph,options)
% ne pas oublier de sauvegarder !!!!
save plantDiseasesNet
% calcule la classe prédite sur les images de test
predictions = classify(plantDiseasesNet,testds)
% Trace la matrice de confusion
confusionchart(testAllImgs.Labels, predictions)
```

Ré-utiliser le cnn

Matlab a enregistré le nouveau réseau ré-entrâiné (ici plantDiseasesNet.mat).
Ce réseau peut être réutilisé avec la commande
load plantDiseasesNet

Entraînement d'un réseau neuronal : avec le Deep Learning Designer

Designer : charger le réseau à ré-entraîner

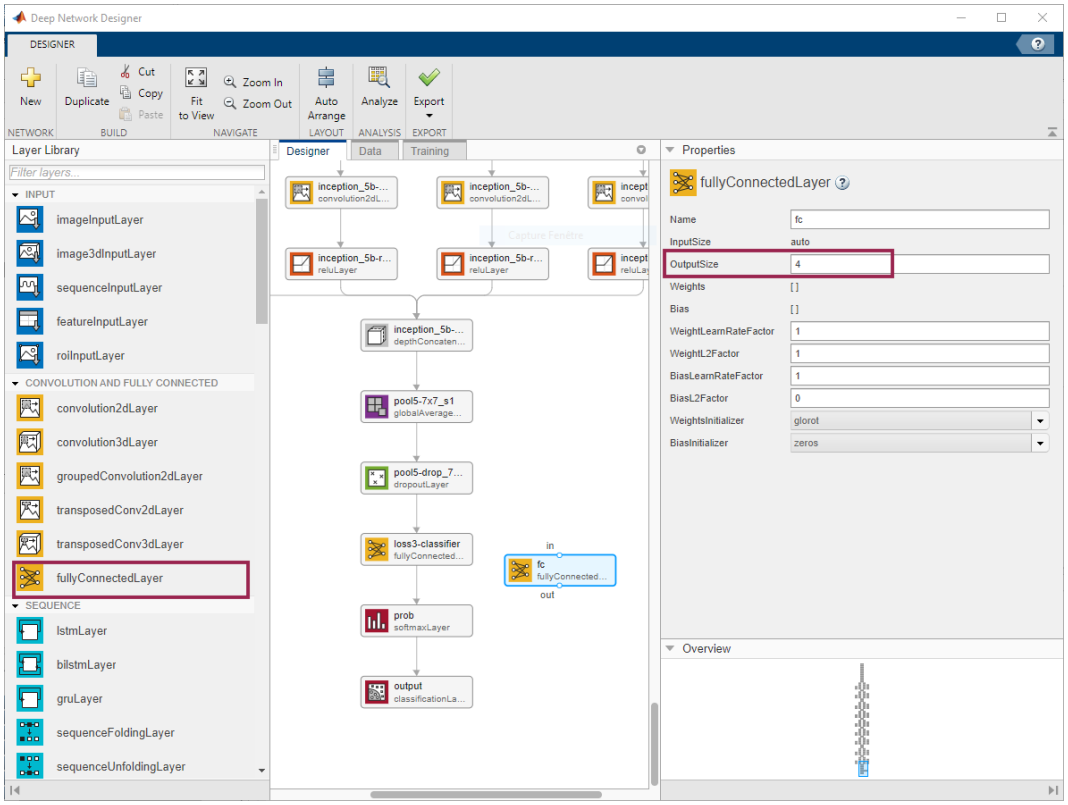
Créer un nouveau CNN. Le menu donne le choix entre

- créer un réseau vide
- charger un réseau existant (Alexnet,...)
- charger un réseau depuis le workspace

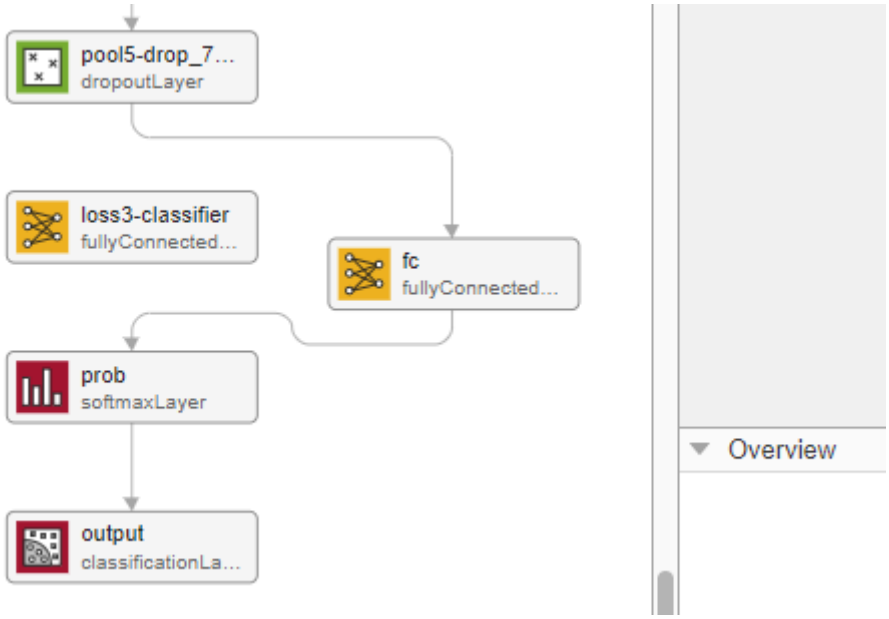
On choisira le réseau que l'on souhaite modifier pour faire le transfer learning.

Modifiation de la couche 'fully connected"

En zoomant sur les couches finales, on peut voir la couche classifiante "fully connected".
Il suffit de glisser depuis la palette à gauche une nouvelle couche de type "fully connected"

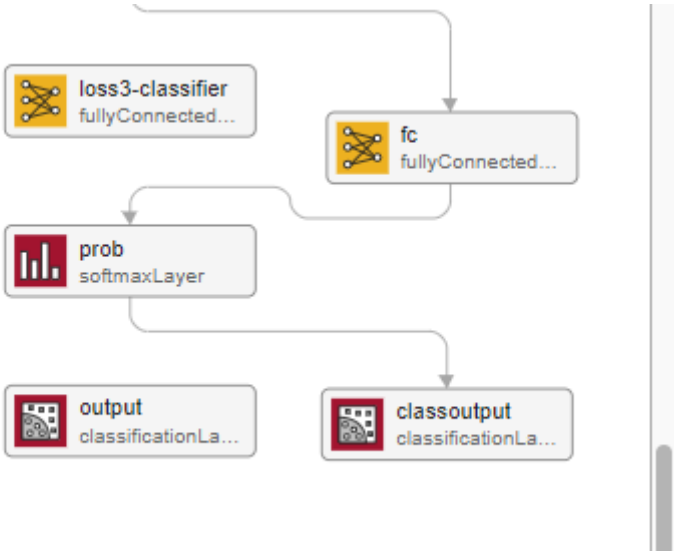


Output size : doit correspondre aux nombre de classes
Il suffit ensuite de supprimer les flèches qui connectent la couche "fully connected" initiale et de connecter la nouvelle à la place



Modification de la couche de sortie

Même principe pour la couche de sortie



Suite du processus

Une fois le réseau importé modifié, il suffit de choisir "Export -> Workspace" : la structure "lgrah" est accessble depuis le workspace.
Il est possible de l'enregistrer mais ce n'est pas obligatoire.
Le graphe est ce qui permet de lancer la fonction "trainNetwork", en spécifiant en plus les images et les options. Le résultat sera le nouveau réseau ré-entraîné, qu'il suffira d'enregistrer.

Entraînement avec le Deep Designer

Il est possible de finaliser le processus avec le Deep Learning Designer

- Spécifier les datas
- Définir les options du training
- Lancer le training

Ce n'est pas détaillé ici.

Le problème du temps de calcul

Entraînement sur tout le dataset

Exécution d'un script avec transfer learning sur toutes les images du dataset (plus de 5000).

```
Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:15 |       22.66% |       2.6129 |          0.0010 |
|       2 |          50 |       00:11:26 |       97.66% |       0.0859 |          0.0010 |
|       3 |         100 |       00:22:52 |       98.44% |       0.0375 |          0.0010 |
|       4 |         150 |       00:34:18 |       97.66% |       0.0703 |          0.0010 |
|       5 |         200 |       00:46:00 |       98.44% |       0.0367 |          0.0010 |
|       7 |         250 |       00:57:28 |       97.66% |       0.0523 |          0.0010 |
|       8 |         300 |       01:08:57 |       98.44% |       0.0200 |          0.0010 |
|       9 |         350 |       01:20:22 |       99.22% |       0.0234 |          0.0010 |
|      10 |         400 |       01:31:46 |       99.22% |       0.0180 |          0.0010 |
|      12 |         450 |       01:43:10 |      100.00% |       0.0092 |          0.0010 |
|      13 |         500 |       01:54:39 |      100.00% |       0.0083 |          0.0010 |
```

Presque 2h de calcul pour 13 Epochs (30 Epochs par défaut).
La fonction "train" est très gourmande en ressources et en particulier en temps de calcul. L'utilisation de GPU peut permettre de gagner du temps. Par rapport aux contraintes horaires, les élèves vont devoir proposer des solutions pour faire des tests (algorithmes, options, nb d'epochs, mini-batch etc.) avant de lancer le transfer learning sur toute la base des images.

> *Méthode :* Limiter le volume des images d'entraînement
>
> - Limiter les images d'entraînement (ex : 500 pour le training, 150 pour le test)
> - Étudier les résultats (matrice de confusion, précision, temps d'exécution d'une classification)
> - Utiliser ces paramètres pour lancer le training sur tout le dataset

Entraînement sur un sous-ensemble des images

Se limiter à un sous-ensemble des dossiers (exemple avec le maïs : 4 répertoires)

Utilisation de la fonction "subset"

Doc Matlab : subset[p.]
[cf. Doc Matlab : subset]
Proposition de solution pour créer les deux sous-ensembles d'images : les images doivent être choisies de façon aléatoire et de façon équitable depuis tous les dossiers.

```
% Creation des datastores pour training et test du cnn
imds = imageDatastore("img", "IncludeSubfolders",true, "LabelSource","foldernames")
imds.Labels
[trainAllImgs, testAllImgs] = splitEachLabel(imds, 0.7, "randomized")

%pour travailler sur une partie des images et pas sur la totalité
nFiles = length(trainAllImgs.Files);
RandIndices = randperm(nFiles); %mélange aléatoire des indices des fichiers images
indices = RandIndices(1:500) % on ne prend que les 500 premières images (après mélange)
trainPartImg = subset(trainAllImgs,indices) %création d'un datastore extrait

%idem pour le datastore dédié au test du cnn
nFiles = length(testAllImgs.Files);
RandIndices = randperm(nFiles);
indices = RandIndices(1:200)
testPartImg  = subset (testAllImgs, indices)

% si besoin de modifier la taille de l'image pour l'adapter
trainds = augmentedImageDatastore([224 224],trainPartImg);
testds = augmentedImageDatastore([224 224],testPartImg);
```

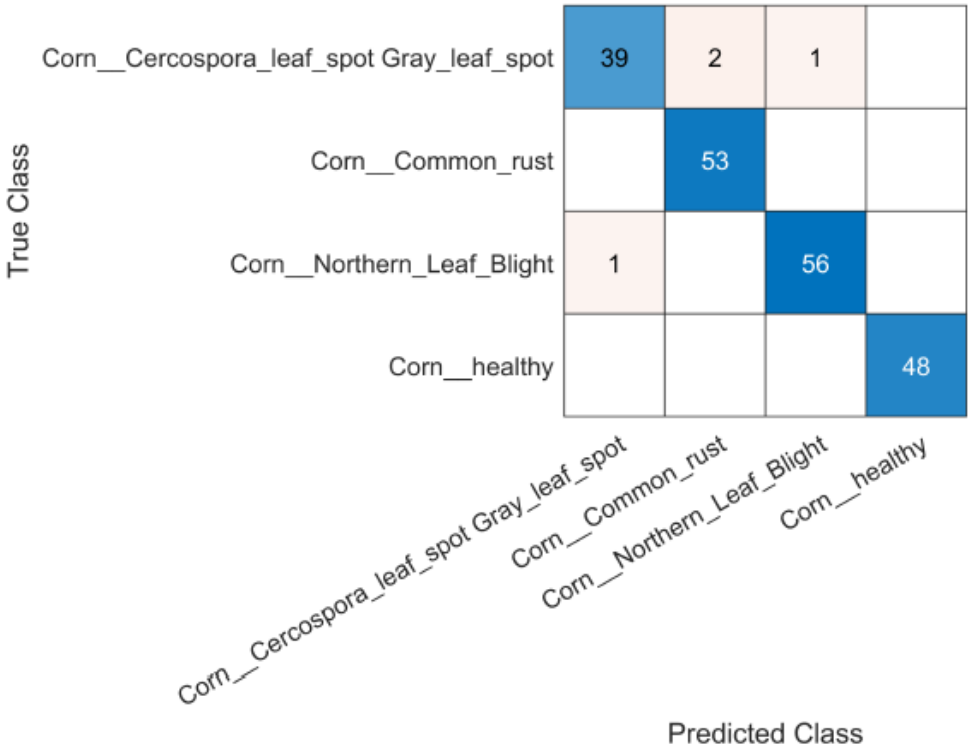
Les options de la fonction "train"

Référence : Doc Matlab : trainingOptions[p.]
[cf. Doc Matlab : trainingOptions]

Choix de l'algorithme "solver" : la matrice de confusion

Algorithme "**adam**"
Problème : toutes les images sont classées comme "healthy" !

Algorithme "**sgdm**"
Le réseau donne de très bons résultats !



Voir le code : Github dédié au projet[p.]

b. Transfer Learning 2 : Mise à l'échelle et validation

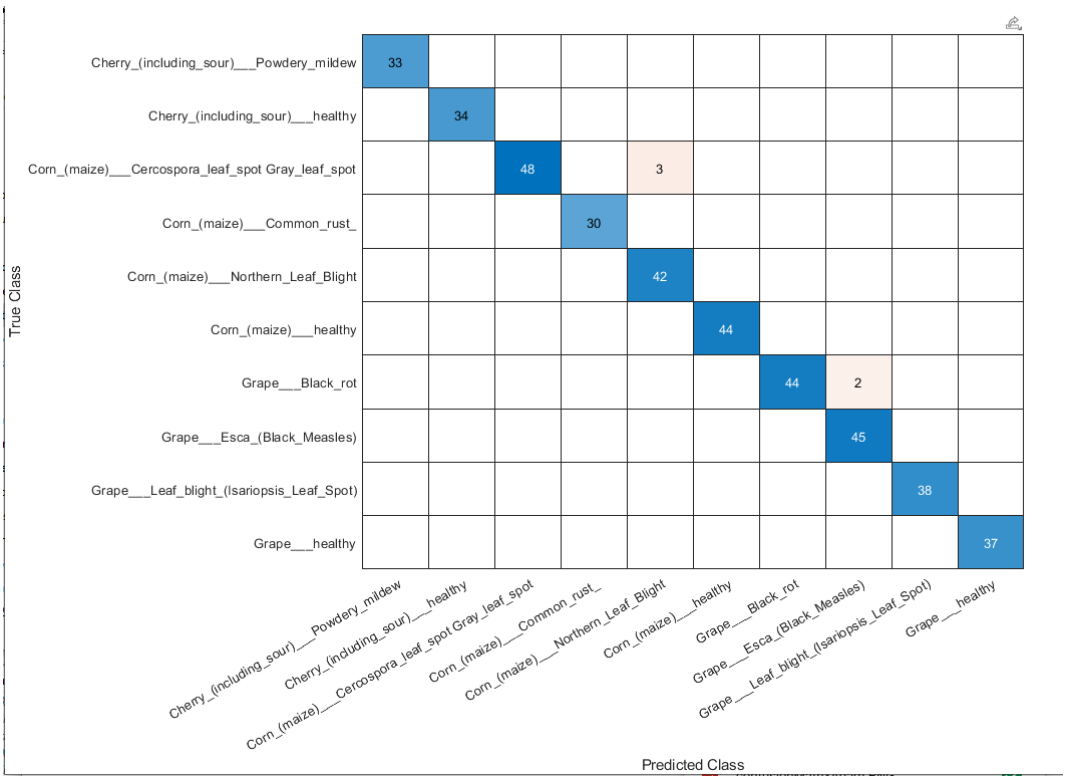
Training sur tout le dataset

Un premier training est fait sur 10 classes, pour évaluer la précision et voir si les options du training sont bonnes, avant de lancer l'entraînement sur tout le dataset.

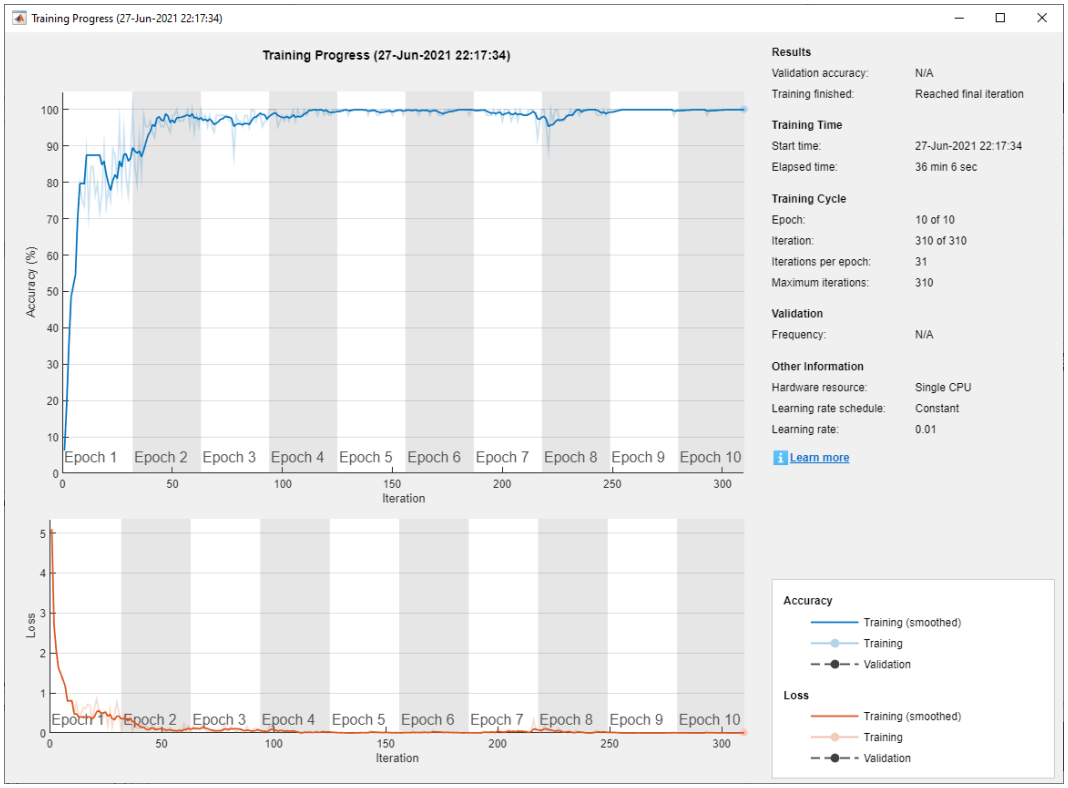
Les codes Matlab Livescript sont sur Github dédié au projet[p.]

Training sur 10 classes

Exemple : pour 10 classes



Pour 10 classes, 2000 fichiers (training) / 400 fichiers (test) : 36min.



Options (training parameters)

```
options = trainingOptions("sgdm",...
    'InitialLearnRate', 0.01, ...
    'LearnRateDropFactor',0.2, ...
    'LearnRateDropPeriod',5, ...
    'MaxEpochs',10, ...
    'MiniBatchSize',64, ...
    'Plots','training-progress',...
    'Shuffle','every-epoch');
```

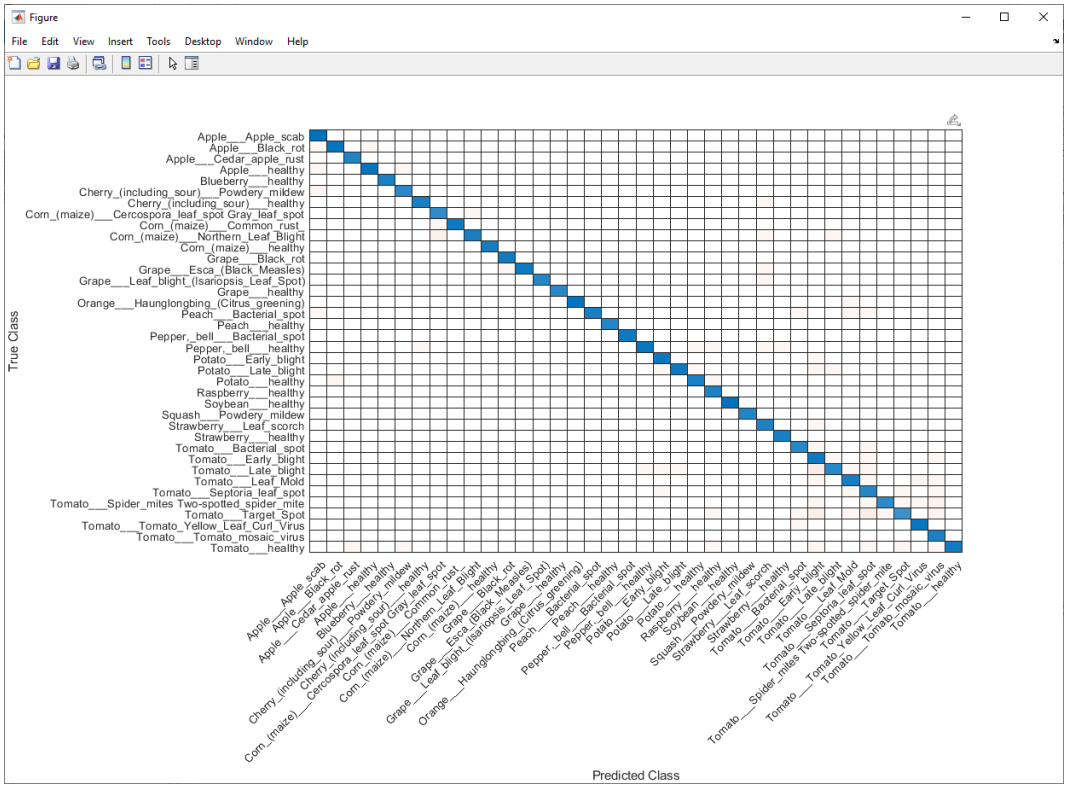
Training sur tout le dataset

Code et résultat du LiveScript Matlab

Remarque : le calcul a duré près de 9h30

cf.

Matrice de confusion



On constate que la matrice de confusion est très satisfaisante : la diagonale montre que les classes prédites correspondent aux classes réelles des images.

4. Simulation - Déploiement - Alertes

DURÉE : 12h
RÉSUMÉ :

Simulation de l'application sur Simulink et déploiement sur Raspberry Pi.

a. Simulation / déploiement

Organisation

- 1 élève : mise en œuvre de la Raspberry Pi avec Matlab, installation de la caméra, test de simulation / déploiement sur la Raspberry (exemple LEB blink)
- 1 élève : adaptation de l'exemple avec le cnn entraîné et des images du dataset (fichiers informatiques ou images imprimées à placer devant la caméra)
- 1 élève : publication d'alertes en cas de reconnaissance de maladies (Thingspeak MQTT)

Simulink

Les élèves pourront partir de l'exemple proposé par Matlab : Classify Objects Using Deep Learning Algorithm on Raspberry Pi Hardware[p.]
[cf. Classify Objects Using Deep Learning Algorithm on Raspberry Pi Hardware]

Mise en œuvre de la Raspberry avec Matlab / Simulink

Support package for Raspberry Pi et add-ons complémentaires

Si ce n'est pas déjà fait, installer le support pour la Raspberry

- MATLAB Support Package for Raspberry Pi Hardware[p.]
- Simulink Support Package for Raspberry Pi Hardware[p.][p.]

Ainsi que les modules complémentaires. (Cf. Installation[p.12])

Installation de la Raspberry

Suivre le tutoriel pour installer la Raspberry Pi. Le plus simple est de graver une nouvelle carte SD en prenant bien soin de choisir l'image **AVEC l'option Deep Learning**.

> ! *Attention :* si problème de connexion avec la carte
>
> Après plusieurs essais infructueux de connexion avec la Raspberry en mode réseau LAN, j'ai réussi à établir la connexion au setup en branchant la Raspberry directement au PC avec un âble RJ45.
> Une fois que la connexion avec la Raspberry est établie, on peut remettre le PC et la Raspberry sur le réseau et se connecter depuis Matlab avec l'adresse IP de la Raspberry. Exemple :

```
pi = raspi('192.168.0.126');
```

Simulink : exemple blink

Suivre l'exemple pour simuler le programme qui fait clignoter la LED et le déployer. Modifier la fréquence de clignotement.
Getting Started with Simulink Support Package for Raspberry Pi Hardware[p.]

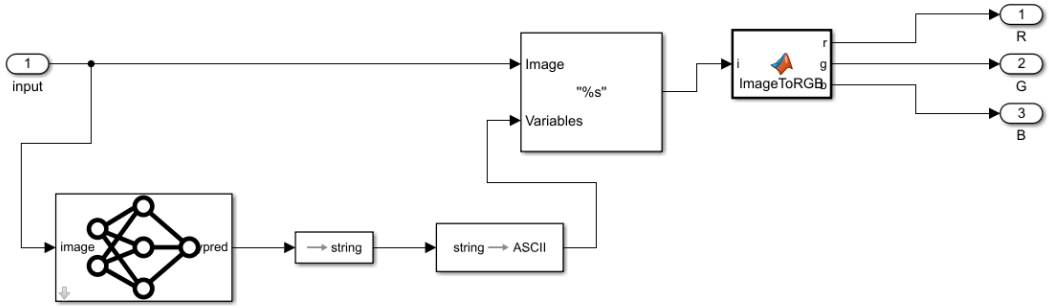
Installation de la caméra

Connecter la caméra et valider avec raspistill et raspivid.

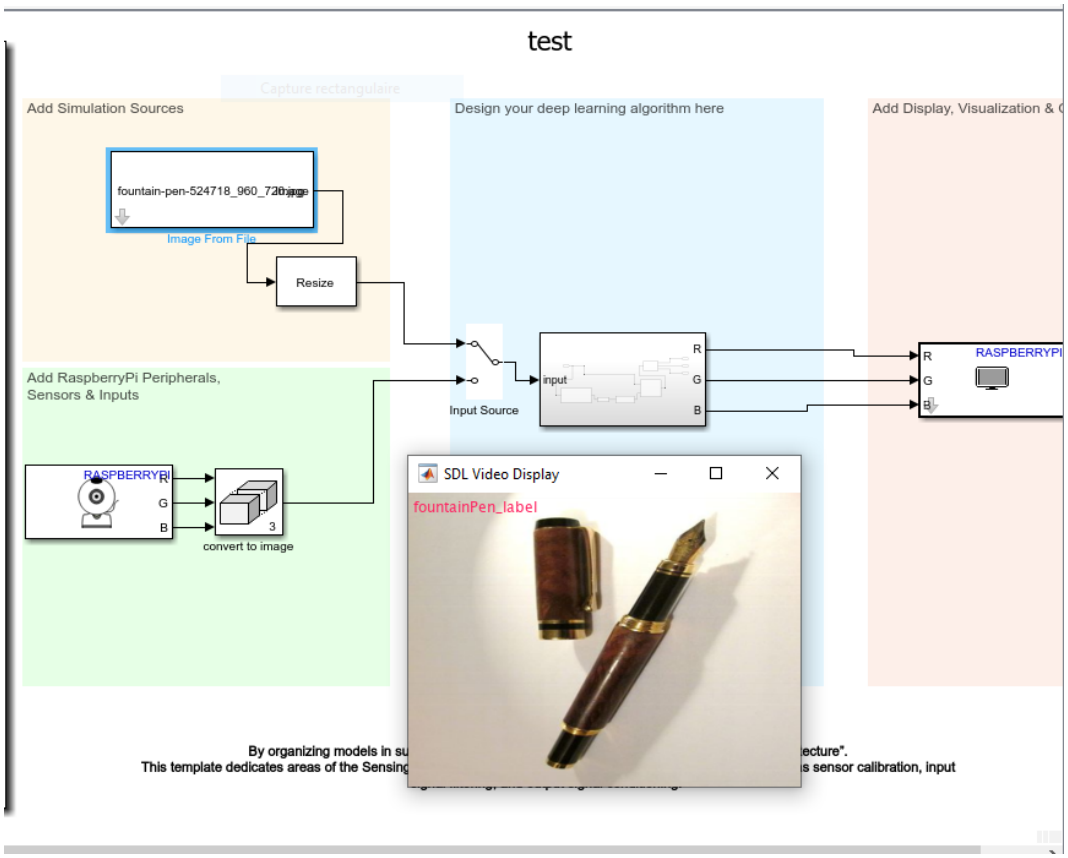
Simulation / Déploiement

Adaptation de l'exemple

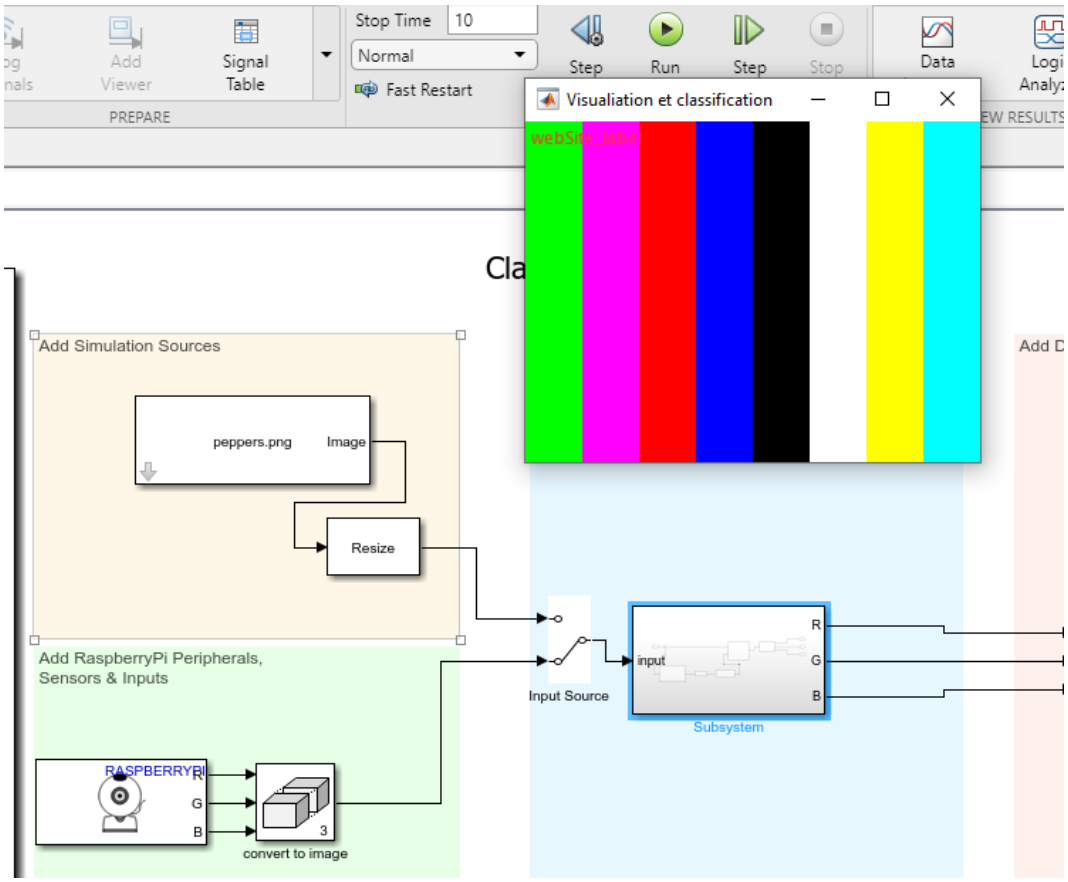
Le sous-système



Mettre en œuvre l'exemple (ici avec squeezenet)
Le tester avec quelques images fixes choisies par l'utilisateur. (Simulation - Run)



Remarque : La simulation avec l'image de la caméra donne une mire



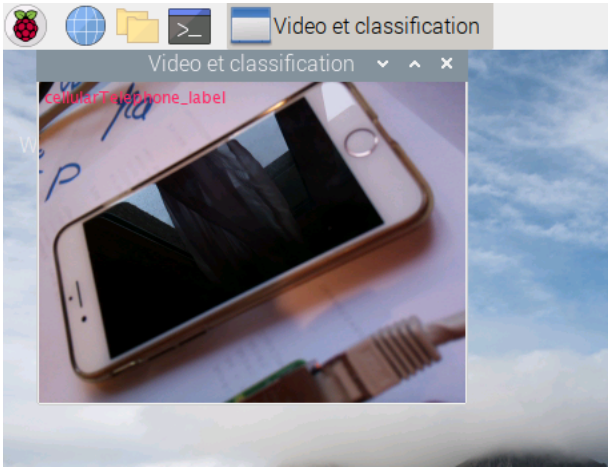
Le tester en déploiement avec l'image de la caméra (Harware - Build and Deploy). Attention cela prend quelques minutes le temps que le programme soit compilé, copié sur la Raspberry.
Le programme se lance automatiquement sur la Raspberry (ouverture du display caméra).

! > ### *Attention :* Echantillonage des images
>
> Dans l'exemple de départ, le paramètre d'échantillonage de la caméra est à 0.1. Ce qui fait rapidement planter la Raspberry car sa puissance de calcul ne sui pas. On pourra modifier ce paramètre par exemple à 1 (une image par seconde).
> Ce paramètre sera affiné en fonction du matériel et des tests.

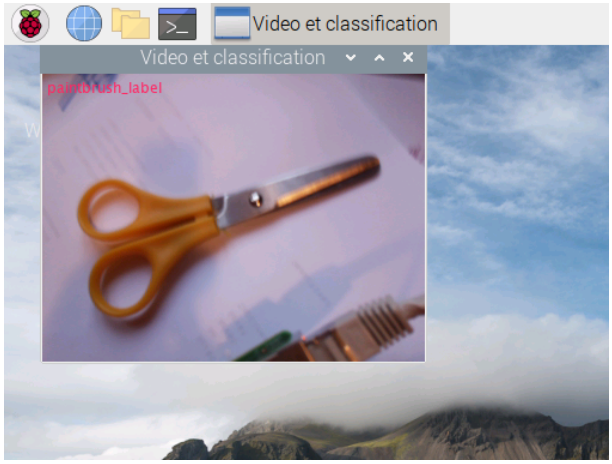
Déploiement

Déployer sur la Raspberry une fois que celle-ci est mise en œuvre.
Tester avec la caméra.

Le réseau squeezenet hésite entre "iPod" et "cellular phone" : plutôt bon



le réseau squeezenet a du mal avec les ciseaux



Déployer sur la Raspberry en remplaçant le réseau Matlab par le réseau entraîné.