

# **FORMATION IA : VIGNE NETTE (SURVEILLANCE DES VIGNES /DEEP LEARNING AVEC MATLAB)**

Cécile TONNERRE

**RÉSUMÉ :**

Cette activité 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**<sup>[p.57]</sup>, la reconnaissance automatique de maladies permet d'utiliser de façon **ciblée et limitée** les produits phytosanitaires sur les cultures.

Ces techniques d'IA au service de l'agriculture peuvent être utilisées sur tout type de culture.

Ici, l'activité prend comme exemple particulier la vigne. En effet, deux maladies des bois de la vigne menacent actuellement le vignoble (en France et en Europe), en particulier l'ESCA. Il n'y a pas d'autre solution actuellement que **d'arracher les ceps d'une parcelle** où est détectée cette maladie.

L'activité se propose donc d'utiliser le deep learning pour reconnaître sur des images l'apparition de cette maladie. La surveillance se fait soit manuellement (application avec IHM) soit automatiquement (appareil avec caméra et application embarquée)

L'activité est réalisée avec le logiciel **Matlab** (déploiement en embarqué sur **Raspberry Pi** + caméra).

**CYCLE :** terminal : 1ère, terminale

**AUTEURS :**

- Cécile TONNERRE

**LICENCES :**

Creative Commons - Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions

# Table des matières

<b>I. CONTEXTE ET RESSOURCES .....</b>	<b>4</b>
1. Contexte et problématique .....	4
a. Problématique .....	4
b. Diagrammes SysML .....	5
2. Ressources logicielles et matérielles .....	8
a. Ressources matérielles et logicielles .....	8
<b>II. CRÉATION ET UTILISATION DU RÉSEAU NEURONAL « VIGNENETTE » .....</b>	<b>9</b>
1. 1-Utilisation d'un réseau neuronal (classification d'images) .....	9
a. Utiliser et comparer des CNN pour la classification d'images .....	9
b. Création d'une IHM .....	21
2. 2-Entraînement d'un réseau neuronal (transfer learning) .....	26
a. Tranfer Learning : réentraînement d'un cnn à partir d'une base d'images .....	26
b. Création d'un réseau neuronal : avec le Deep Learning Designer .....	34
c. Enregistrer et réutiliser un réseau de neurones .....	37
d. IHM : utiliser le réseau vignenette dans l'application .....	40
e. IHM : Créer une application autonome pour Windows .....	41
3. 3-Simulation - Déploiement - Alertes .....	44
a. Simulation / déploiement .....	45
<b>ANNEXES .....</b>	<b>49</b>
<b>SOLUTIONS .....</b>	<b>55</b>
<b>GLOSSAIRE .....</b>	<b>57</b>

# I. Contexte et ressources

## RÉSUMÉ :

Contexte et problématique : détection précoce des maladies des bois de la vigne  
 Diagrammes SysML

## 1. Contexte et problématique

### a. Problématique

#### Problématique générale

##### Problématique générale : 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 l'apparition de problèmes, et d'utiliser les produits phytosanitaires de façon ciblée, au mètre carré près (agriculture de précision). En effet, l'utilisation de pesticides a des impacts sur la santé des consommateurs, mais également des riverains et des ouvriers agricoles (cancers, Alzheimer...).

Cette problématique est particulièrement aiguë dans le domaine viticole.

Cette activité se propose d'utiliser l'IA sur la surveillance des vignes. En effet, plusieurs maladies menacent actuellement le vignoble français, dont l'ESCA. Pour l'instant, il n'existe pas de traitement utilisable et quand un cep est malade c'est tout une parcelle qu'il faut arracher pour éviter la propagation.

#### Références et sources

[\[Businessinsider\] Smart Farming in 2020: How IoT sensors are creating a more efficient precision agriculture industry](#)

[p.]

cf.

#### Problématique spécifique : la vigne

##### Les maladies du bois de la vigne

Cette activité se propose d'utiliser l'IA sur la surveillance des vignes. En effet, plusieurs maladies du bois de la vigne menacent actuellement le vignoble français et européen, dont l'ESCA. Pour l'instant, il n'existe pas de traitement utilisable et quand un cep est malade c'est tout une parcelle qu'il faut arracher pour éviter la propagation.

Le manque de traitement et l'arrachement des céps ont pour conséquence des pertes financières importantes. La progression de ces maladies est telle que les pépinières ont du mal à fournir suffisamment de nouveaux pieds de vigne.

Voir à ce sujet le [rapport parlementaire de 2015](#)<sup>[p.]</sup> ainsi que différents articles :

<https://www.vitisphere.com/actualite-78532-un-cout-annuel-moyen-de-2-300-ha-pour-les-viticulteurs.html>

<https://www.nouvelobs.com/planete/20150728.OBS3286/les-maladies-du-bois-tuent-le-vignoble-francais-a-petit-feu.html>

Au niveau européen, un programme de recherche a pour objectif de mieux comprendre la propagation de ces maladies et de favoriser l'échange d'information entre les acteurs.

Au-delà de l'aspect financier (manque à gagner, pertes à l'exportation des vins français, c'est tout le patrimoine viticole qui est en danger, comme à une autre époque avec le phylloxéra).

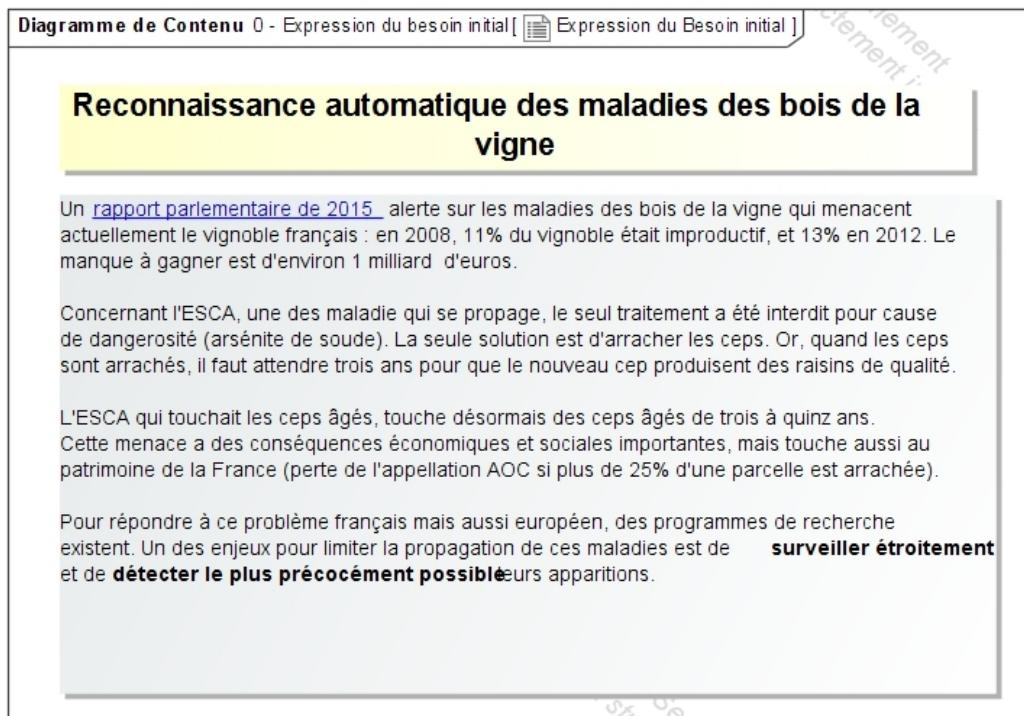
L'activité d'IA se focalise donc sur la surveillance des vignes, afin d'apporter un élément de réponse à cette problématique : comment surveiller et détecter le plus tôt possible l'apparition de maladies sur les vignes ?

## b. Diagrammes SysML

### RÉSUMÉ :

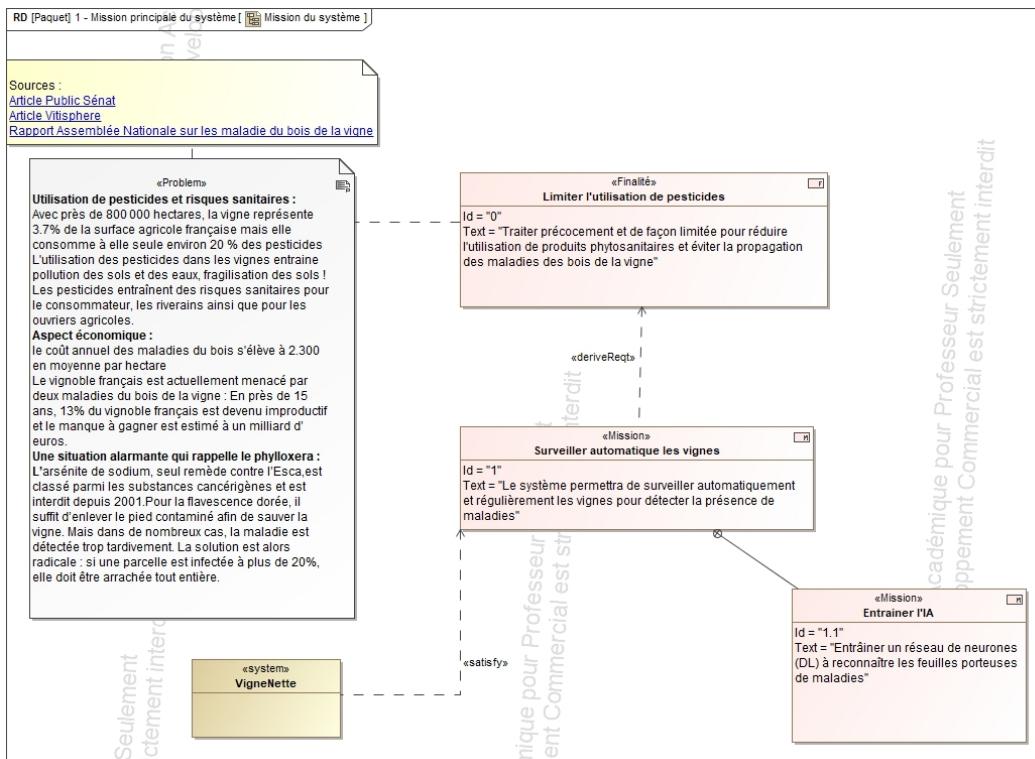
Les diagrammes SysML ont été réalisés avec le [plugin Magic Draw ISEN](#)<sup>[p.]</sup> mis au point par Yann LE GALLOU.

### Diagramme de contenu - expression du besoin initial



Expression du besoin initial

### Diagramme spécifique - Mission du système



Mission principale du systèmes

## Diagramme de contexte

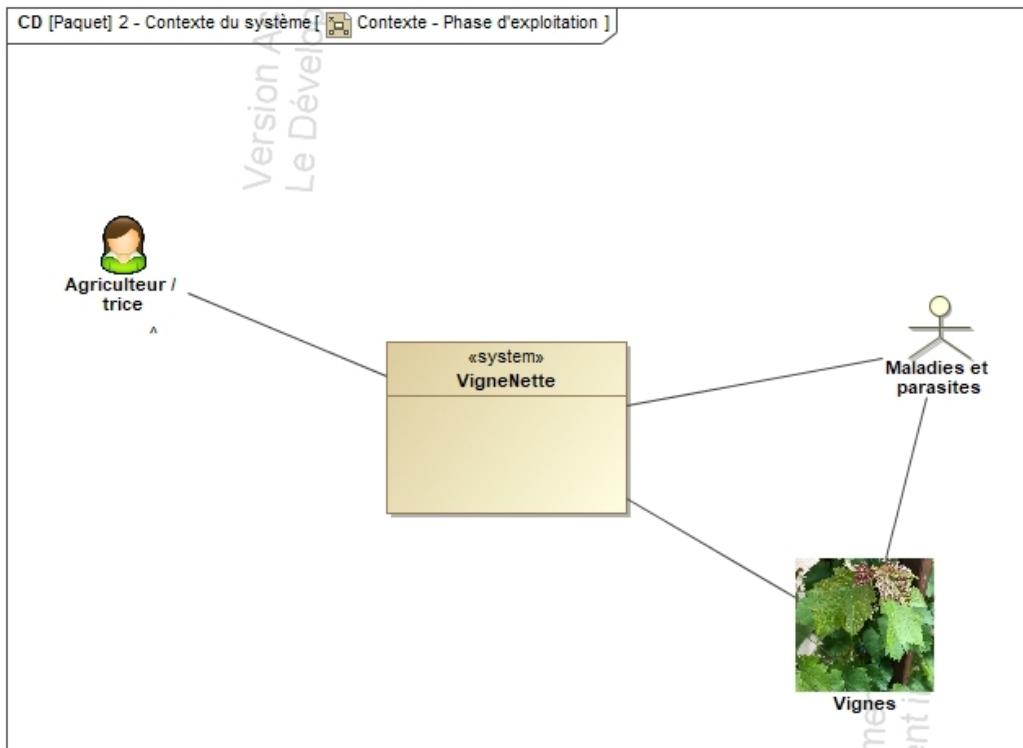


Diagramme de contexte

## Diagramme d'exigences

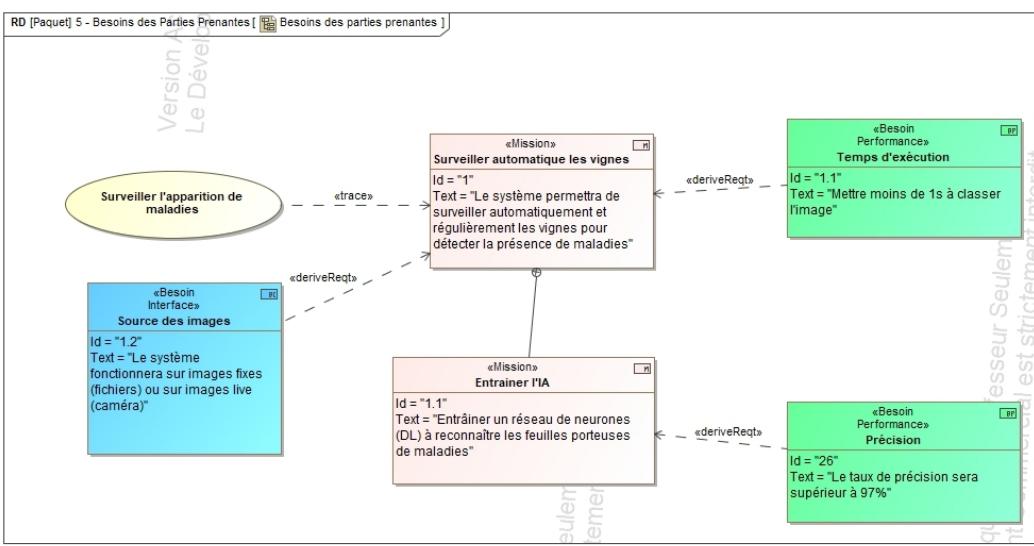


Diagramme d'exigences

## 2. Ressources logicielles et matérielles

### a. Ressources matérielles et logicielles

#### 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 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 meilleures 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).

## II. Création et utilisation du réseau neuronal « VigneNette »

### RÉSUMÉ :

- Prise en main des outils IA / Deep Learning de Matlab : classification d'images en utilisant un réseau de neurones convolutif (convulsive network neural ou CNN).
- Réentraîner un CNN : datastore d'images, paramètres, la problématique du temps de calcul, validation de la précision.
- Enregistrer et utiliser le CNN obtenu.

### 1. 1-Utilisation d'un réseau neuronal (classification d'images)

#### DURÉE : 1h

#### RÉSUMÉ :

- Utiliser un [CNN](#)[p.57] pour de la reconnaissance automatique d'images
- Comparer les résultats de plusieurs [CNN](#)[p.57]
- Réaliser une IHM de reconnaissance automatique d'images

Les fichiers sources sont sur [Github \(SI-IA-Vignenette\)](#)[p.]

#### a. Utiliser et comparer des CNN pour la classification d'images

#### RÉSUMÉ :

La première partie consiste à utiliser un [CNN](#)[p.57] pré-entraîné proposé par Matlab pour reconnaître des images, afin de se familiariser avec les termes du Deep Learning et du Machine Learning.

Un CNN est entraîné à reconnaître à quelle **catégorie** appartient une images (chien, tasse, stylo...). Le nombre de catégories est fixe pour chaque CNN (il y a un neurone par catégorie sur la couche de sortie). A titre d'exemple, le CNN MobileNet-v2 a été entraîné pour classer une image parmi 1000 catégories.

Quand un CNN « reconnaît » ce qu'il y a dans une image, en fait il donne la « catégorie » la plus probable à laquelle elle appartient. On utilise aussi les termes de « label » ou de « classe ».

Après cette prise en main, une comparaison de plusieurs réseaux de neurones sera proposée.

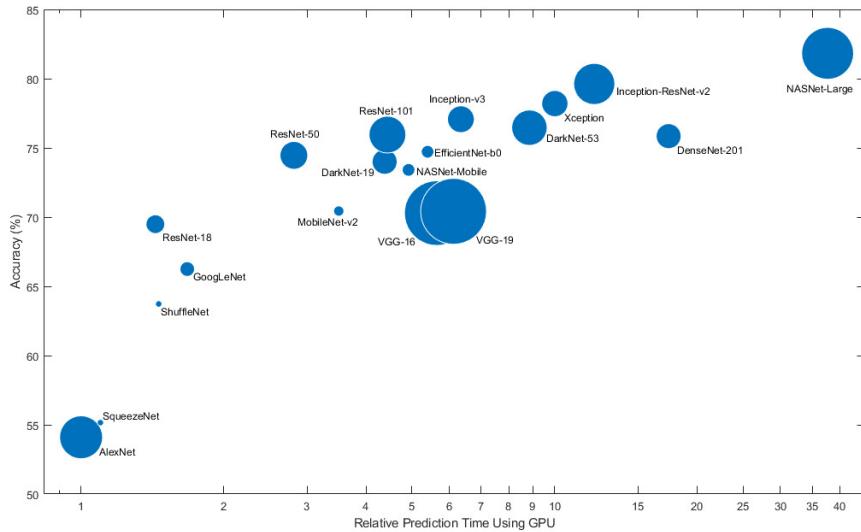
**TOUTES CES ACTIVITÉS PEUVENT ÊTRE REPRISES (ET ADAPTÉES SELON VOS BESOINS) POUR ÊTRE RÉALISÉES PAR DES ÉLÈVES.**

#### + Complément : Fichiers sources

Les fichiers sources (images et scripts) sont sur le repository [Github](#)[p.] SI-IA-Vignenette

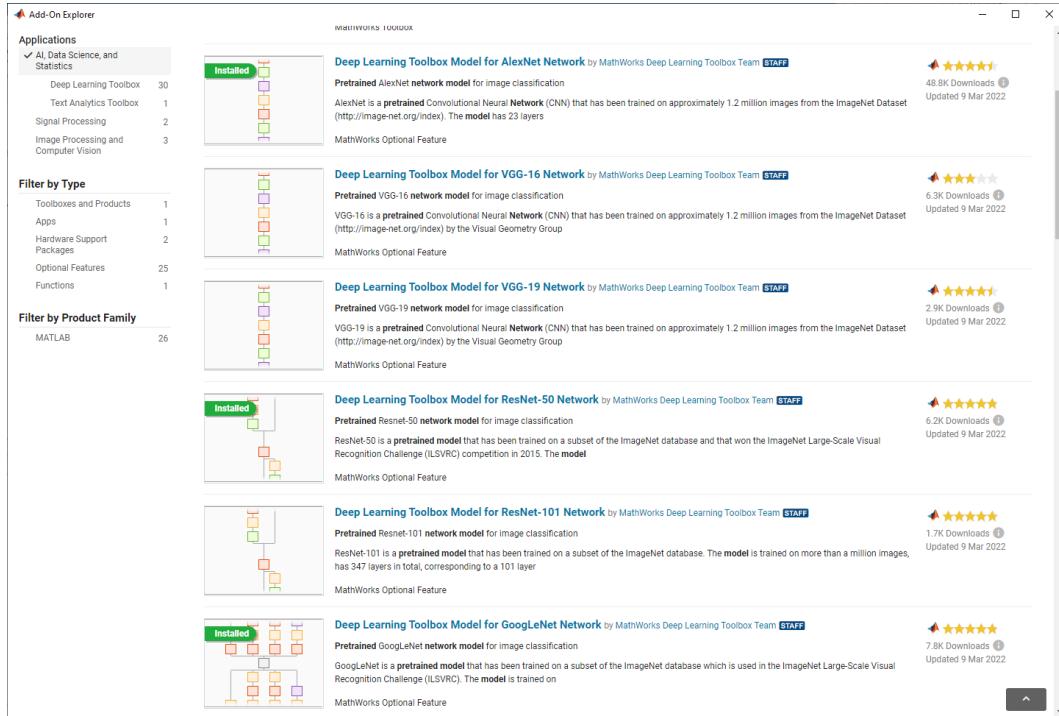
#### Pretrained Deep Neural Networks

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



Référence : [Doc Matlab : Pretrained Deep Neural Networks](#)[p.]

Les réseaux de neurones s'installent via le « Add-on manager ». On peut donc voir la liste des CNN disponibles en cherchant les mots clefs « Deep learning toolbox network ».



Réseaux de neurones entraînés pour le deep learning disponibles pour Matlab

Pour se familiariser avec l'utilisation des réseaux, la première activité est de classer une image (i.e. trouver son label) en utilisant un CNN pré-installé.

## 1ère étape : Classification simple d'une image

### Choisir une image

Sélectionner une image sur Internet, au format jpg ou prendre l'image fournie en exemple.



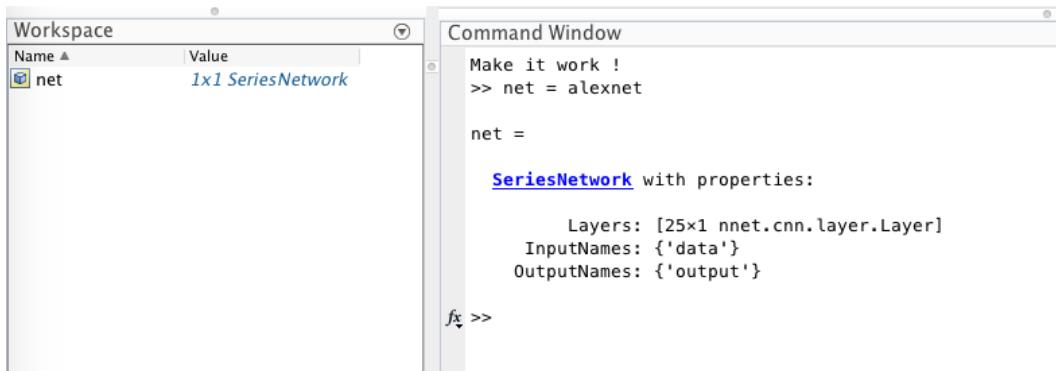


## Complément : Charger un réseau de neurones

Un CNN est chargé sous Matlab simplement en l'appelant depuis une variable, exemple

net = alexnet ;

Le réseau apparaît alors dans le Workspace.



Chargement d'un CNN sous Matlab

Les CNN installés pour la formation sur le PC sont :

alexnet, googlenet, resnet-18, resnet-50

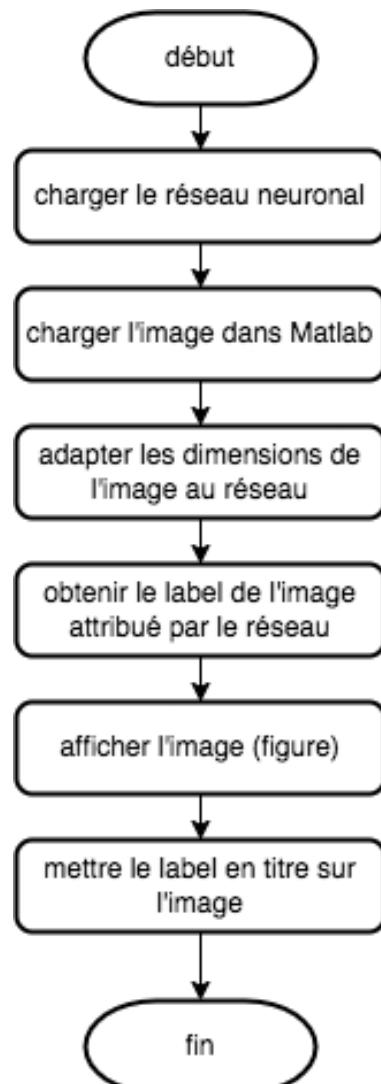
Chaque collègue pourra choisir les CNN à installer en fonction de ses besoins.

?

## Afficher une image avec le label

### Question

En utilisant les fonctions Matlab, écrire le programme qui affiche une image, ainsi que son label (classe) attribué par le réseau.



### Fonctions Matlab utilisées :

imread<sup>[p.]</sup> (Read image from graphics file) . Ex :

```
1 A = imread('ngc6543a.jpg');
```

imresize<sup>[p.]</sup> (Resize image). Ex :

```
1 B = imresize(A,[numrows numcols]);
```

classify<sup>[p.]</sup> (Classify data using trained deep learning neural network)

```
1 labels = classify(net,images)
```

figure<sup>[p.]</sup> (Create Figure Window)

```
1 figure; % ouvre une fenêtre d'affichage (fenêtre d'affichage courante)
2 figure(1); %ouvre une fenêtre numérotée
```

imshow<sup>[p.]</sup> (Display Image)

(on pourra utiliser le paramètre « **InitialMagnification** »

```
1 imshow(monImage);
```

```
2 imshow(monImage, 'InitialMagnification', 'fit');
```

title<sup>[p.]</sup> (Add title)

```
1 % plusieurs façons d'écrire le titre de la figure
```

```
2 title('Le titre de la figure');
```

```
3 title(char(leTitre));
```

```
4 title(string(leTitre));
```

```
5
```

[solution n°1 p.55]

## Test du CNN sur une image de feuille de vigne

Refaire la classification sur une image spécifique comme une feuille de vigne.

Figure 2



Feuille de vigne reconnue comme « poncho »

**Conclusion :** plus une image est spécifique, plus le cnn a du mal à deviner.

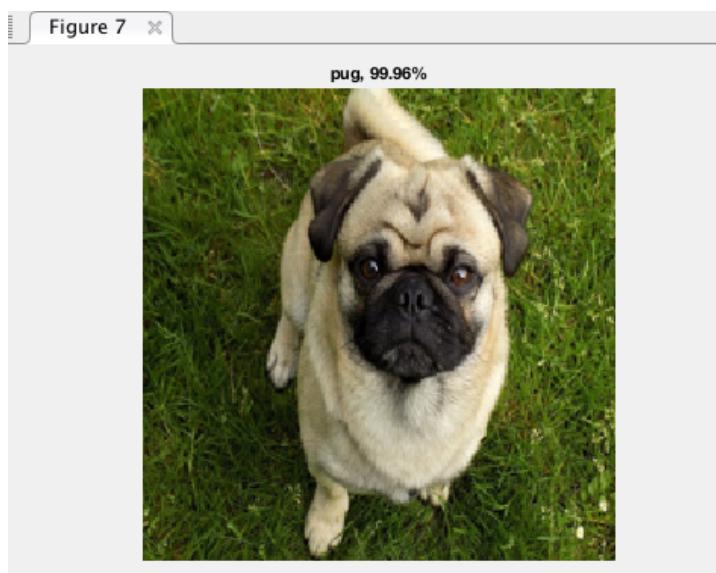
## 2ème étape : Classification avancée d'une image

### Label et probabilité (score de confiance)

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

On peut obtenir des informations supplémentaires avec la fonction 'classify', notamment le score (taux de confiance pour chaque catégorie)

Figure 7



Sur cet exemple, le score de confiance pour le label « pug » (carlin) est de 99,99%.

?

### Affichage du score

## Question

Améliorer le script précédent en ajoutant le score de confiance à côté du label

### Références Matlab :

Propriété [Layers](#)<sup>[p.]</sup> du réseau (accessible en lecture uniquement)

La dernière couche du réseau autant de neurones que de classes. Chaque neurone porte la probabilité que l'image corresponde à sa classe.

Il est possible de récupérer le **nom des classes** via les propriétés de cette dernière couche.

```
1 % récupération de toutes les classes du
  CNN (dernière couche )
2 classNames = net.Layers(end).ClassNames;
```

	1	2
1	tench	
2	goldfish	
3	great white shark	
4	tiger shark	
5	hammerhead	
6	electric ray	
7	stingray	
8	cock	
9	hen	
10	ostrich	
11	brambling	
12	goldfinch	
13	house finch	
14	juncos	
15	indigo bunting	
16	robin	
17	bulbul	
18	jay	
19	magpie	
...	...	...

Exemple de variable 'classNames' obtenue avec la fonction 'classify'.

[classify](#)<sup>[p.]</sup> (Classify data using trained deep learning neural network)

La fonction classify permet également d'obtenir les scores pour chaque classe (chaque neurone de la dernière couche).

```
1 % on récupère le label et toutes les
  probabilités
2 [label, scores] = classify(net, img);
```

	1	2	3	4
1	2.6158e-15	4.5208e-17	1.7356e-15	1.7529e-17
2				
3				

Exemple de variable 'scores1' obtenu avec la fonction 'classify'

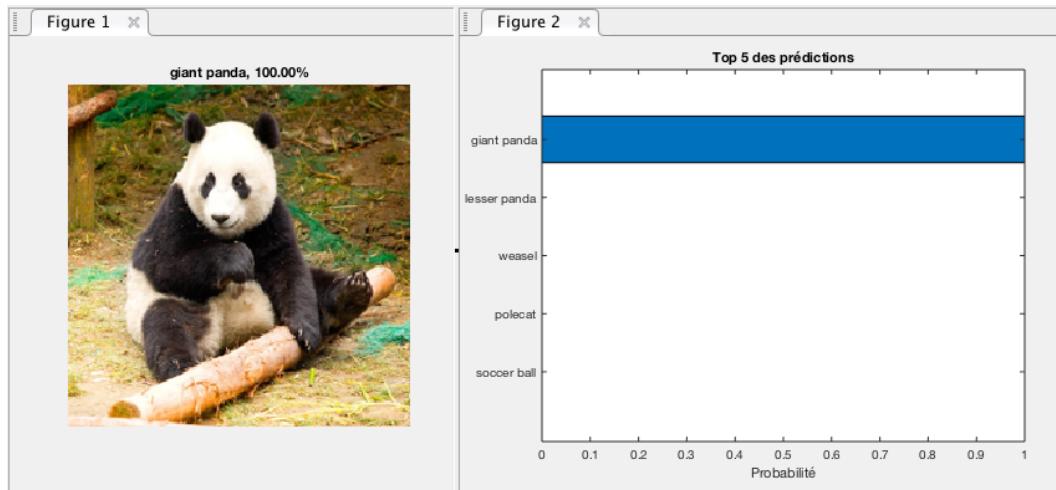
Exemple : pour obtenir le score du label renvoyé par 'classify' :

```
1 % Sélection de la cellule de 'scores' avec comme index
2 % le classNames = le label rentré par 'classify'
3 scores( classNames == label )
```

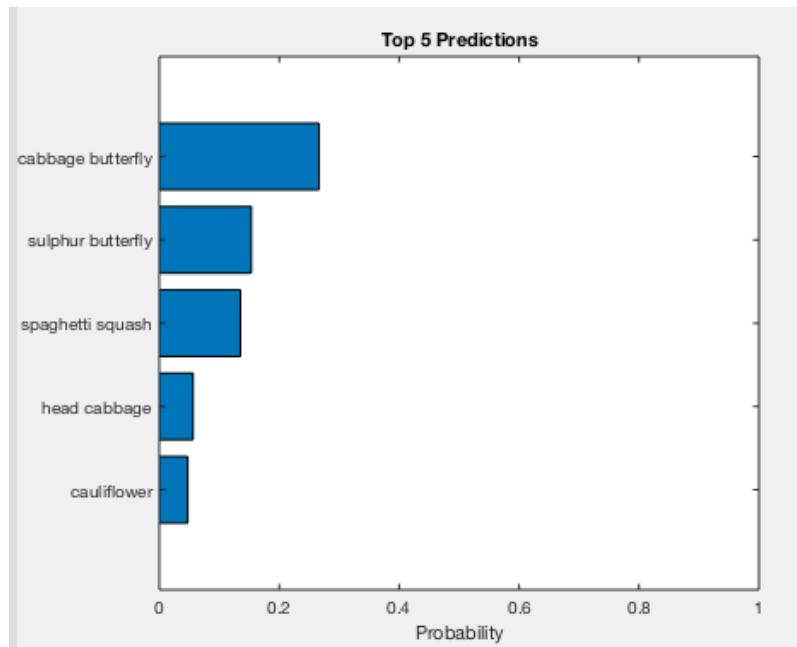
?

**Bonus : Afficher les 5 labels ayant les plus forts scores****Question**

Reprendre le code et afficher une figure avec les 5 labels ayant les plus forts scores et leurs probabilités.

**Exemple de résultat pour le panda**

Résultats des scores les plus élevés avec alexnet

**Autre exemple de résultat pour une feuille de bouleau**

On constate que le résultat est moins "sûr". (le réseau est moins confiant quant au label renvoyé par 'classify').

### 3ème étape : comparaison des CNN

#### Activités bonus

Cette section propose des activités complémentaires qui pourraient être réalisées avec les élèves. Elles pourront être faites pendant la formation en fonction du temps disponible et des priorités de chacun.

L'objectif est de comparer les performances de plusieurs CNN. Cette comparaison peut être qualitative ou quantitative et pourra servir à choisir le CNN qui sera réentraîné sur le dataset d'images spécifiques (ici les feuilles de vigne, saines ou porteuses de maladies).

#### ! *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



**Exemple :** Analyser qualitative des résultats (pertinence) 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.

```

22 - idx = 1:5,1:1);
23 - classNamesTop = net.Layers(end).ClassNames(idx)
24 - scoresTop = scores(idx) ← pas de ';' en fin de ligne
25 -
26 - figure
27 - barh(scoresTop)
28 - xlim([0 1])
29 - title('Top 5 Predictions')
30 - xlabel('Probability')
31 - yticklabels(classNamesTop)

Command Window
classNamesTop =
5x1 cell array
{'shopping cart'    }
{'bull mastiff'     }
{'Pekinese'          }
{'Brabancon griffon'}
{'pug'               }

scoresTop =
1x5 single row vector
0.0020    0.0021    0.0108    0.0164    0.7822
f5 >>

```

	A	B	C	D	E	F	G
1	carlin						
2		alexnet	squeezeenet	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	squeezeenet	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.]

Pour cette analyse, les élèves pourraient rechercher la traduction (et les images) correspondants aux différents labels et estimer quel réseau donne les meilleurs résultats en terme de pertinence sur les N plus forts scores. Dans le cas des animaux (races de chien, l'analyse peut se faire également en terme de ressemblance).



## Exemple : Analyse quantitative : performance des CNN

Sur l'étude de N plus forts scores et de leurs classes, il serait possible d'attribuer des points aux CNN

- points positifs pour des classes proches de l'image analysée
- points négatifs pour des classes éloignées ou sans rapport avec l'image (exemple : shopping cart pour le chien)

Cette note globale servirait à comparer les CNN pour le choix du réseau à réentraîner.



## Exemple : 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 à une exigence du cahier des charges ? (exemple : taux de précision > 97%).



## Méthode : Afficher le temps de calcul nécessaire à la classification

Référence : [Doc Matlab : Measure the Performance of Your Code](#)<sup>[p.]</sup>

Utilisation de "tic" et "toc" juste avant et juste après la fonction pour connaître le temps d'exécution

```

1
2
3 tic
4 % The program section to time.
5 toc
6
7
8
9
10 - net = mobilenetv2; % Charge le réseau neuronal
11 - %à tester avec alexnet, squeezenet, googlenet, mobilenetv2, resnet18
12 - classNames = net.Layers(end).ClassNames; % récupération de toutes les classes
13 - img = imread('carlin.jpg'); %lit l'image la charge en mémoire
14 - img = imresize(img,[224,224]); % mise à la taille input du réseau
15 - tic
16 - [label, scores] = classify(net, img); % on récupère le label et les probabilités
17 - toc
18 -
19 - figure %crée la fenêtre d'affichage
20

```

Command Window

```

>> labelImageCompareCnn
Elapsed time is 2.298138 seconds.

```

## Choix du CNN

Ces analyses peuvent aider à choisir le CNN qui sera entraîné sur le dataset d'images.

## b. Création d'une IHM

### RÉSUMÉ :

A partir de l'exemple [Create App that Uses Multiple Axes to Display Results of Image Analysis](#)<sup>[p.]</sup>, créer une IHM qui

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

Cette activité peut être reprise et faite par les élèves telle quelle ou adaptée pour un projet.

Les fichiers sources (fichier de démarrage AppClasseStart.mlapp et fichier complet AppClasse.mlapp) sont sur [Github SI-IA-VigneNette](#)<sup>[p.]</sup>

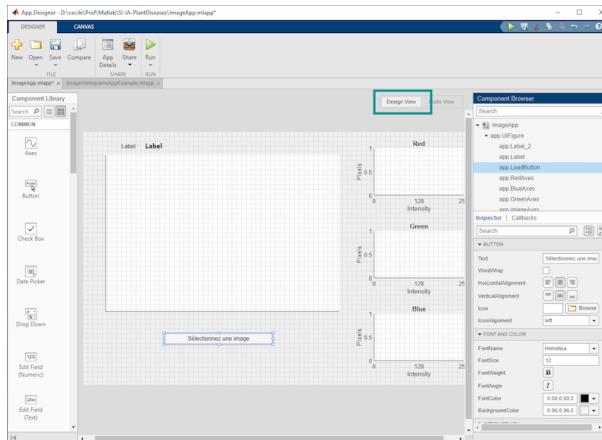
### + Complément : Présentation de AppDesigner (programmation événementielle un peu comme AppInventor)

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 orientée objet, on parlera de **propriétés** plutôt que de variables et de **méthodes** plutôt que des fonctions.

Design View



Code View

```

classdef ImageApp < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access = public)
        UIFigure matlab.ui.Figure
        app.Label matlab.ui.control.Label
        Label1 matlab.ui.control.Label
        Label2 matlab.ui.control.Label
        RedAxes matlab.ui.control.UITaxes
        BlueAxes matlab.ui.control.UITaxes
        GreenAxes matlab.ui.control.UITaxes
        ImageAxes matlab.ui.control.UITaxes
    end

    methods (Access = private)
        function updateImage(app,imagefile)
            %Crée le label
            %Initialise le tableau neuronal
            net = alexnet;
            img = imread(imagefile);
            img = im2double(img);
            img = im2uint8(img([127 227]));
            label = classify(net, img);

            %For corn.tif, read the second image in the file
            if strcmp(imagefile,'corn.tif')
                im = imread('corn.tif', 2);
            else
                try
                    im = imread(imagefile);
                catch ME
                    if isproblem reading image, display error message
                    uiAlert(app.UIFigure, ME.message, 'Image Error');
                    return;
                end
            end
        end
    end
end

```



## Méthode : Ouvrir le fichier de démarrage Matlab App Designer

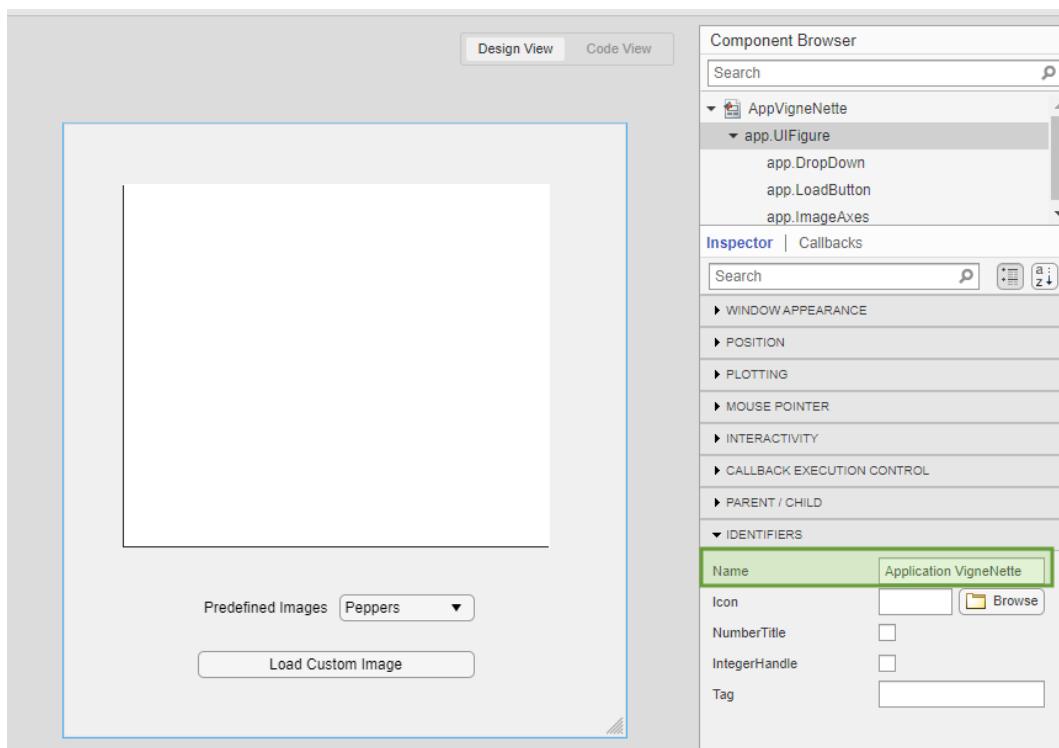
Ouvrir le fichier 'VigneNetteStart.mlapp' sous Matlab AppDesigner. (C'est une version simplifiée de l'exemple 'Analyze an Image')



## Modifier l'interface (layout)



### Méthode : Changer le titre de l'application



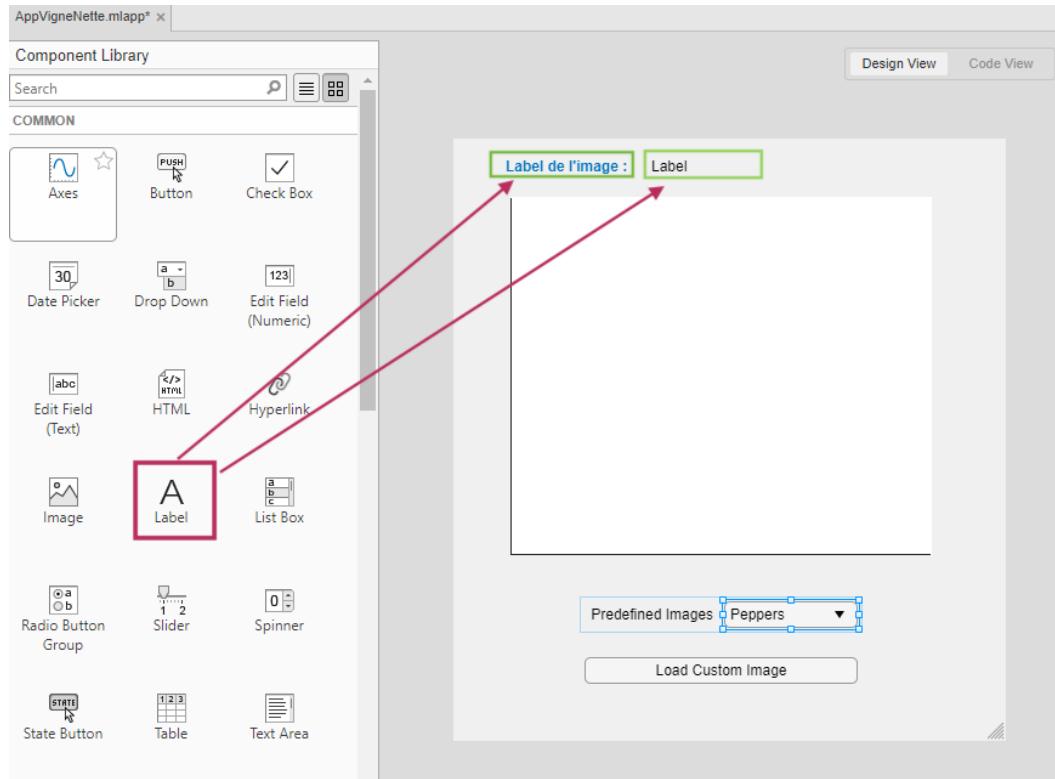
### Attention : « label » sous AppDesigner

une zone de texte courte est appelée « label » sous AppDesigner, ce qui peut prêter à confusion avec le « label » de l'image à trouver (sa catégorie).



## Méthode : Ajouter le label

- Se mettre en « Design View »
- Ajouter un composant texte pour l'affichage du label (qui sera modifié par le programme)
- Ajouter un composant texte à afficher à côté (non modifiable)



## Méthode : Facultatif : supprimer le menu déroulant avec les deux images préchargées

Cela permet de simplifier l'interface.

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

```

1 % Callback function
2     function DropDownListValueChanged(app, event)
3
4
5     end

```

## Modification du code



### Méthode : Accéder rapidement à une méthode

Le menu « Go To permet d'accéder rapidement à une méthode.

Une autre possibilité est de choisir à droite le composant (ici l'application) et de cliquer à gauche sur 'Callbacks'.

La méthode 'startupFcn' apparaît dans la liste et il suffit de cliquer dessus

```

44 -
45 -
46 -
47 -
48 -
49 -
50 -
51 -
52 -
53 -
54 -
55 % callbacks that handle component events
56 % methods (Access = private)
57 -
58 -
59 function startupFcn(app)
60     % Configure image axes
61     app.ImageAxes.Visible = 'off';
62     app.ImageAxes.Colormap = gray(256);
63     axis(app.ImageAxes, 'image');
64 -
65     % Update the image and histograms
66     updateImage(app, 'peppers.png');
67 end

```

Accéder rapidement à la fonction startupFcn

### Zones blanches / zones grises

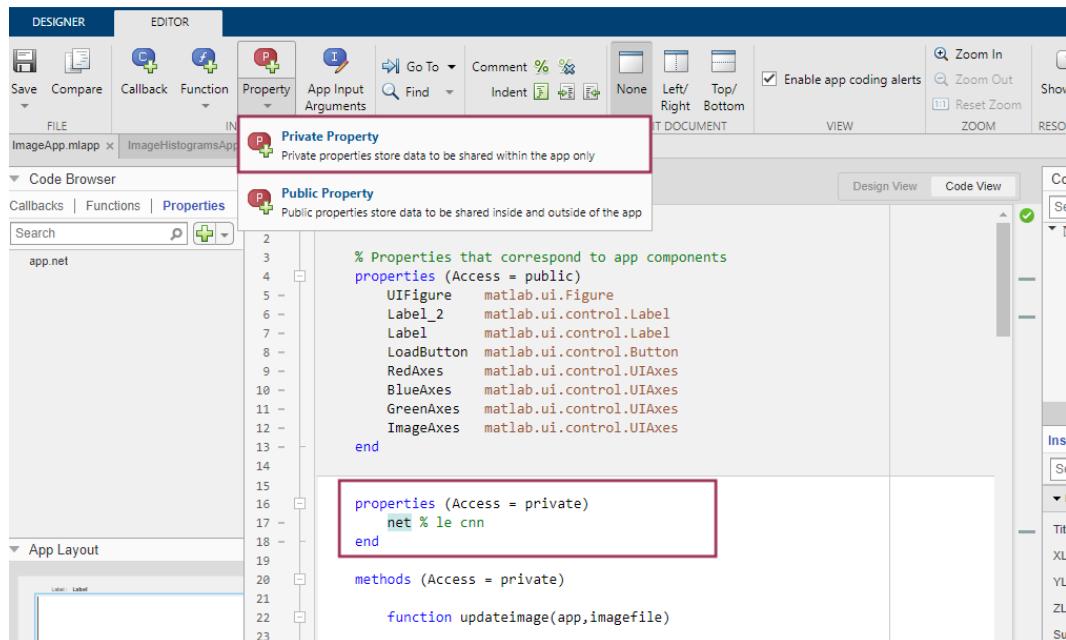
Les zones grises ne sont pas éditables, elles sont générées automatiquement Matlab.



## Méthode : Chargement du cnn

Se placer en 'Code View'

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



### Définir le réseau

```

1 properties (Access = private)
2     net % le cnn
3 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)

```

1 function startupFcn(app)
2     % Configure image axes
3     app.ImageAxes.Visible = 'off';
4     app.ImageAxes.Colormap = gray(256);
5     axis(app.ImageAxes, 'image');
6
7     % Chargement du cnn
8     app.net = alexnet;
9
10    % Update the image and histograms
11    updateimage(app, 'peppers.png');
12 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"

```

1 function LoadButtonPushed(app, event)
2
3     % Display uigetfile dialog
4     filterspec = {'*.jpg;*.tif;*.png;*.gif','All Image Files'};
5     [f, p] = uigetfile(filterspec);
6

```

```

7      % Make sure user didn't cancel uigetfile dialog
8      if (ischar(p))
9          fname = [p f];
10         updateimage(app, fname);
11     end
12 end

```

On peut donc afficher le label juste sous l'affichage de l'image, dans "updateimage"

```

1 %crée le label
2 img = imresize(im, [227 227]);
3 label = classify(app.net, img);
4
5 % le composant "Label" reçoit comme propriété Text le label prédict par le réseau
6 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.

Share > Standalone Desktop App

permet de créer un exécutable qui installera une application sous Windows.

## 2. 2-Entraînement d'un réseau neuronal (transfer learning)

DURÉE : 1h30

RÉSUMÉ :

Training et validation du réseau neuronal sur le dataset  
 Enregistrement et réutilisation du réseau neuronal  
 Réalisation d'une IHM pour obtenir le label d'une image  
 Création de l'application Windows VigneNette

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

RÉSUMÉ :

- Le principe du transfer learning
- Modifier un réseau existant (couches de sorties)
- Entraîner le réseau sur un dataset d'images

Le réseau utilisé pour VigneNette est le réseau **googleNet**.

Tous les fichiers sont sur [Github SI-IA-VigneNette](#)<sup>[p.]</sup>

### Dataset utilisé

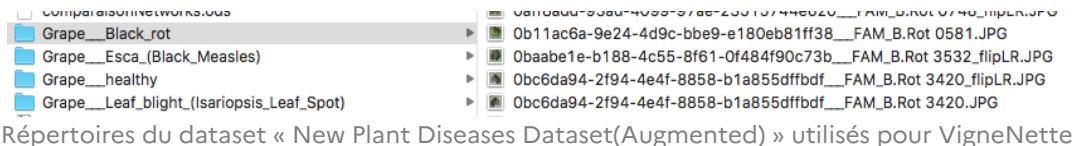
Le dataset utilisé vient de Kaggle.

Source : [Kaggle : New Plant Diseases Dataset](#)<sup>[p.]</sup>

Special thanks to [Samir Bhattacharai](#) for this amazing dataset !

Pour cette activité de détection des maladies des bois de la vigne, nous utiliserons une partie du dataset : les répertoires concernant les feuilles de vigne.

Un répertoire contient des photos de feuilles de vigne saines, trois autres répertoires contiennent des photos de feuilles de vigne atteintes de maladies.

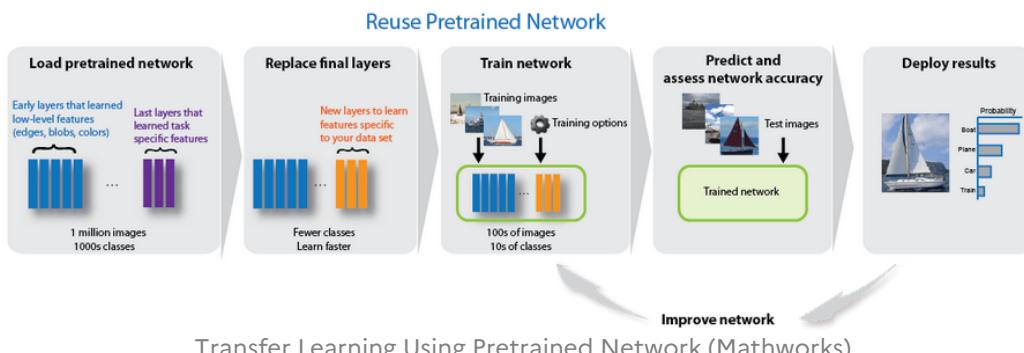


Répertoires du dataset « New Plant Diseases Dataset(Augmented) » utilisés pour VigneNette

Cette activité est réalisée avec deux répertoires : « healthy » et « ESCA ».

## Découvrir le Transfer Learning

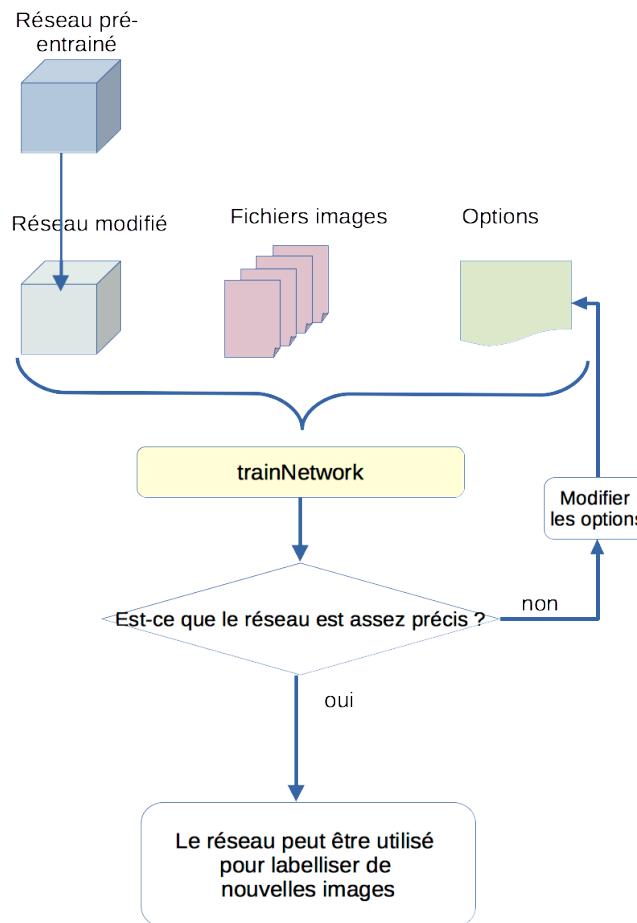
### Principe du transfer learning



Le transfer learning consiste à partir d'un réseau de neurones déjà entraîné à reconnaître des images, et à le réentraîner sur des images. L'avantage est que le réseau est déjà existant (couches de neurones de différents types reliées entre elles), avec des paramètres fonctionnels pour certaines images.

Il sera beaucoup plus facile d'entraîner ce réseau qui devra recalculer et modifier ses paramètres que de créer complètement un nouveau réseau de neurones.

### Transfer Learning sur les images





## Méthode : Étape 1 : Modifier un réseau existant

Matlab permet de charger un réseau pré-entraîné existant, et de le modifier pour l'adapter à nos besoins. Pour cette activité nous utiliserons le cnn **GoogleNet**.

La modification se limite à **modifier 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

Couche « fully connected » : chaque neurone est connecté à tous les neurones de la couche suivante



Fully connected layer : chaque neurone est connecté à tous les neurones de la couche suivante



## Méthode : Étape 2 : Préparer les fichiers images

Les images sont le plus souvent organisées dans une arborescence. Sous Matlab nous pourrons utiliser les "[imageDatastore](#)"<sup>[p.]</sup>, une variable qui regroupe les images situées dans un répertoire (avec ou non des sous-répertoires).

L'utilisation des imageDatastore présente plusieurs avantages intéressants

- si il faut retailler les images pour les adapter au cnn, le datastore permet de retailler automatiquement chaque image à la volée.
- 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



## Méthode : Étape 3 : Définir les options d'entraînement

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

Parmi les options :

- l'algorithme de descente de gradient (sgdm, adam, rmsprop)
- le nombre maximum d'époques
- la taille des min-batches

...

Personnellement j'ai eu de bien meilleurs résultats avec l'algorithme « sgdm » qu'avec « adam » qui est mis par défaut.

Référence : [Documentation sur les options d'entraînement des cnn](#)<sup>[p.]</sup>



## Méthode : Étape 4 : Entraînement et évaluation du réseau de neurones

Le temps de calcul d'un entraînement peut être très long (plusieurs heures). Il faut donc penser à enregistrer le nouveau réseau en fin de calcul !

Il est possible d'entraîner le réseau sur une sous-partie des images et/ou en limitant le nombre d'époques, pour valider les options, avant de lancer l'entraînement global.

Ici, le dataset est modeste et l'entraînement prend entre 6 et 12 minutes.

Après l'entraînement, le réseau peut être utilisé, on cherchera à contrôler la matrice de confusion pour valider la précision du réseau.

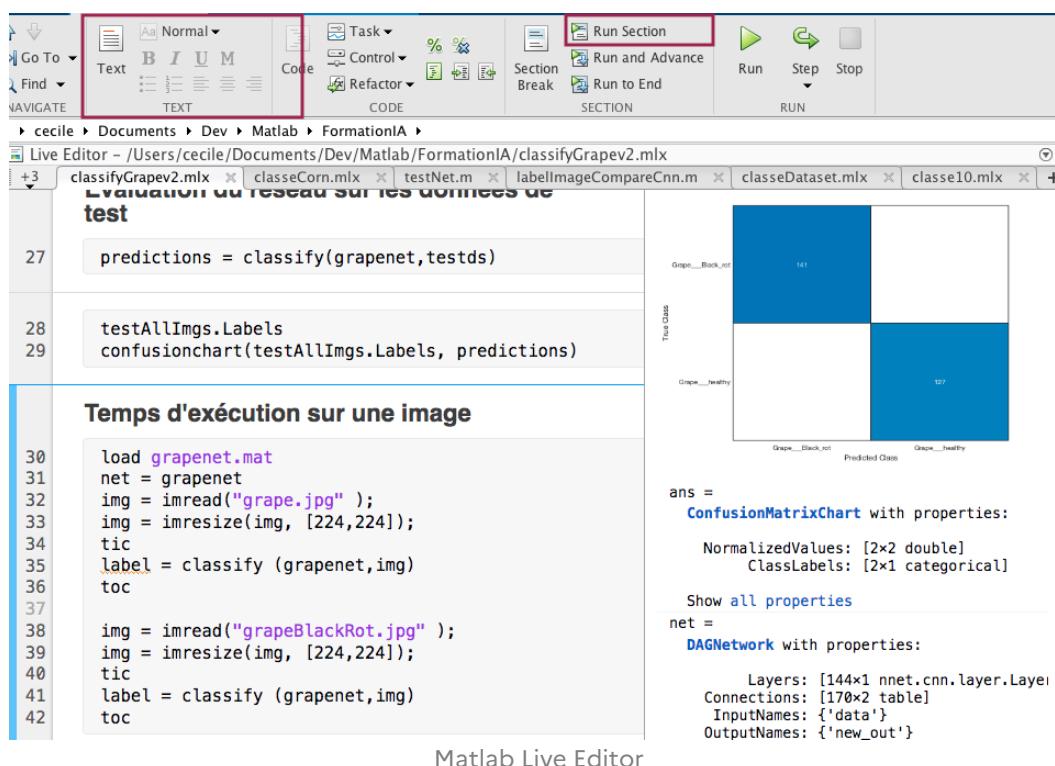
### Entraînement d'un réseau neuronal en ligne de commande



#### Conseil : Matlab Live Editor

Le live editor de Matlab permet d'enregistrer le code ainsi que le résultat de l'exécution, et permet également de créer des sections qui peuvent être exécutées de façon indépendante.

Le live editor permet également d'améliorer la lisibilité avec des blocs de texte insérés dans les sections.



### Étape 1 : Modifier un réseau existant

```

1 % on réutilise GoogleNet
2 net = googlenet
3 lay = net.Layers
4 % affichage de la taille d'entrée dans la command window (facultatif)
5 inLayer = lay(1)
6 inSize = inLayer.InputSize
7 % création du nouveau réseau en modifiant deux couches
8 lgraph = layerGraph(net);
9 % taille 2 car le cnn doit reconnaître 2 classes d'images
10 newFc = fullyConnectedLayer(2,"Name","new_fc");
11 lgraph = replaceLayer(lgraph,"loss3-classifier",newFc)

```

```

12 newOut = classificationLayer("Name", "new_out");
13 lgraph = replaceLayer(lgraph, "output", newOut)

```

## ! Attention : Classification layer / Output layer

La couche de **classification** est une des dernières couches : elle attribue un score à chaque classe pour l'image traitée. 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 (le label). Son type est sous Matlab est "**Classification Output**"

Il y a parfois ambiguïté entre la "classification layer" et la "output layer" car Matlab a attribué le type 'Classification Output' à la couche de sortie.

## Étape 2 : Préparer les fichiers images

```

1 % création du datastore : les labels sont les noms des répertoires
2 imds = imageDatastore("grape2classes/", "IncludeSubfolders",true, "LabelSource","foldernames")
3 imds.Labels
4 % répartition lot d'entraînement / lot de test
5 [trainAllImgs, testAllImgs] = splitEachLabel(imds, 0.7, "randomized")
6 % les images seront retaillées à la volée par le cnn, sans modification de l'image originale
7 % mise à la taille de la taille d'entrée du cnn
8 trainds = augmentedImageDatastore([224 224],trainAllImgs);
9 testds = augmentedImageDatastore([224 224],testAllImgs);

```

## Etape 3 : Définir les options d'entraînement

Algorithme = sgdm

MaxEpochs = 3 pour limiter le temps de calcul (C'est suffisant ici, on pourrait mettre aussi 5 si on a plus de temps)

MiniBatchSize = 78

J'ai pris des mini-batches de 78 images car j'ai 624 images dans le lot d'entraînement (624 = 8x78)

Shuffle = every epoch : le réseau mélange les images au début de chaque époque pour que les mini-batches ne soient pas constitués de façon identique

```

1 options = trainingOptions("sgdm',...
2     'InitialLearnRate', 0.001, ...
3     'LearnRateDropFactor',0.2, ...
4     'LearnRateDropPeriod',5, ...
5     'MaxEpochs',3, ...
6     'MiniBatchSize',78, ...
7     'Plots','training-progress',...
8     'Shuffle','every-epoch');

```

Pour plus de détails : [Doc Matlab : trainingOptions](#)<sup>[p.]</sup>

## Etape 4 : Training et évaluation

```

1 % entraînement du cnn, qui sera chargé dans la variable grapenet
2 vignenette = trainNetwork(trainds,lgraph,options)
3 % enregistrement dans un fichier .mat pour pouvoir réutiliser le cnn
4 save VigneNette
1 predictions = classify(vignenette,testds)
2 testAllImgs.Labels
3 % trace la matrice de confusion
4 confusionchart(testAllImgs.Labels, predictions)

1 % test sur des images et calcule le temps
d'exécution (affiché dans la command window)
2 img = imread("grape.jpg" );
3 img = imresize(img, [224,224]);
4 tic
5 label = classify (vignenette,img)
6 toc
7

```

```

8 img = imread("grapeBlackRot.jpg");
9 img = imresize(img, [224,224]);
10 tic
11 label = classify(vignenette,img)
12 toc

```

```

outputnames{1} = 'new_out';
label = categorical
Grape_healthy
Elapsed time is 0.883932 seconds.

label = categorical
Grape_Esca
Elapsed time is 0.206915 seconds.

```

Affichage du temps d'exécution dans la command window (tic / toc)

## ? Entrainer googleNet à reconnaître les feuilles de vigne porteuses de maladie (ESCA)

### Question

Ecrire un live script Matlab pour réentraîner le réseau googleNet à reconnaître les feuilles de vigne porteuses de maladies (ESCA)

[solution n°4 p.56]

### Le problème du temps de calcul

#### Entraînement sur tout le dataset

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, on peut lancer un entraînement sur une partie du dataset 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

Ici le dataset est réduit, ce qui permet d'avoir des temps de calcul minime. Si on travaille sur un dataset plus important, on peut faire des tests sur un volume plus réduit pour valider les paramètres avant de lancer les calculs sur tout le dataset.

Méthode :

- 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)
- Modifier les options d'entraînement (algorithme, nombre d'époques...) et relancer l'entraînement
- Quand les paramètres permettent d'approcher une bonne précision, on peut conserver ces options et lancer le training sur tout le dataset

### Complément : utiliser un sous-ensemble d'images et algorithmes

#### Se limiter à un sous-ensemble des dossiers (exemple avec la vigne : 2 répertoires)

Cela permet de travailler sur un sous-ensemble d'images pour valider l'algorithme et certains paramètres. Quand cela donne des bons résultats, le calcul peut ensuite être lancé sur tout le dataset.

Utilisation de la fonction "subset"

[Doc Matlab : subset](#)<sup>[p.]</sup>

[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.

```

1 % Creation des datastores pour training et test du cnn
2 imds = imageDatastore("img", "IncludeSubfolders",true, "LabelSource","foldernames")
3 imds.Labels
4 [trainAllImg, testAllImg] = splitEachLabel(imds, 0.7, "randomized")
5
6 %pour travailler sur une partie des images et pas sur la totalité
7 nFiles = length(trainAllImg.Files);
8 RandIndices = randperm(nFiles); %mélange aléatoire des indices des fichiers images
9 indices = RandIndices(1:500) % on ne prend que les 500 premières images (après mélange)
10 trainPartImg = subset(trainAllImg,indices) %création d'un datastore extrait
11
12 %idem pour le datastore dédié au test du cnn
13 nFiles = length(testAllImg.Files);
14 RandIndices = randperm(nFiles);
15 indices = RandIndices(1:200)
16 testPartImg = subset (testAllImg, indices)
17
18 % si besoin de modifier la taille de l'image pour l'adapter
19 trainds = augmentedImageDatastore([224 224],trainPartImg);
20 testds = augmentedImageDatastore([224 224],testPartImg);

```

## Les options de la fonction "train"

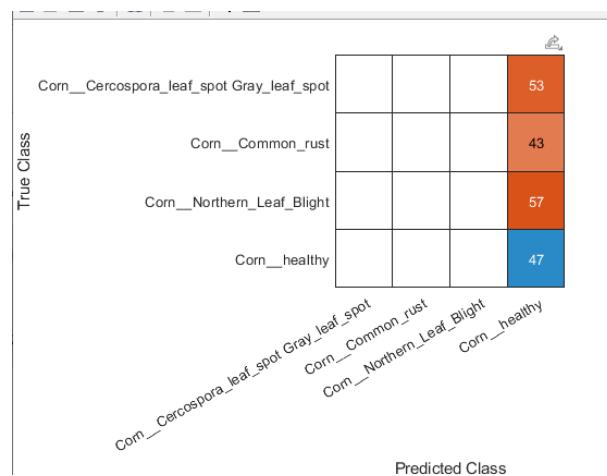
Référence : [Doc Matlab : trainingOptions](#)<sup>[p.]</sup>

[cf. Doc Matlab : trainingOptions]

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

Pour mesurer l'importance du choix de l'algorithme, voici pour info deux tests effectués sur un dataset plus complet : l'un avec l'algorithme "adam" et l'autre avec « sgdm »

avec « adam »



avec « sgdm »

		Corn_Cercospora_leaf_spot_Gray_leaf_spot	Corn_Common_rust	Corn_Northern_Leaf_Blight	Corn_healthy
True Class	Corn_Cercospora_leaf_spot_Gray_leaf_spot	39	2	1	
	Corn_Common_rust		53		
	Corn_Northern_Leaf_Blight	1		56	
	Corn_healthy				48

Corn\_Cercospora\_leaf\_spot\_Gray\_leaf\_spot    Corn\_Common\_rust    Corn\_Northern\_Leaf\_Blight    Corn\_healthy

Predicted Class

L'algorithme « sgdm » a donné de biens meilleurs résultats c'est pourquoi je l'ai utilisé pour VigneNette.

## b. Création d'un réseau neuronal : avec le Deep Learning Designer

### RÉSUMÉ :

La création du réseau à partir d'un cnn existant peut se faire graphiquement avec le module « Deep Network Designer » sous Matlab, ce qui limite les opérations en ligne de commande Matlab.

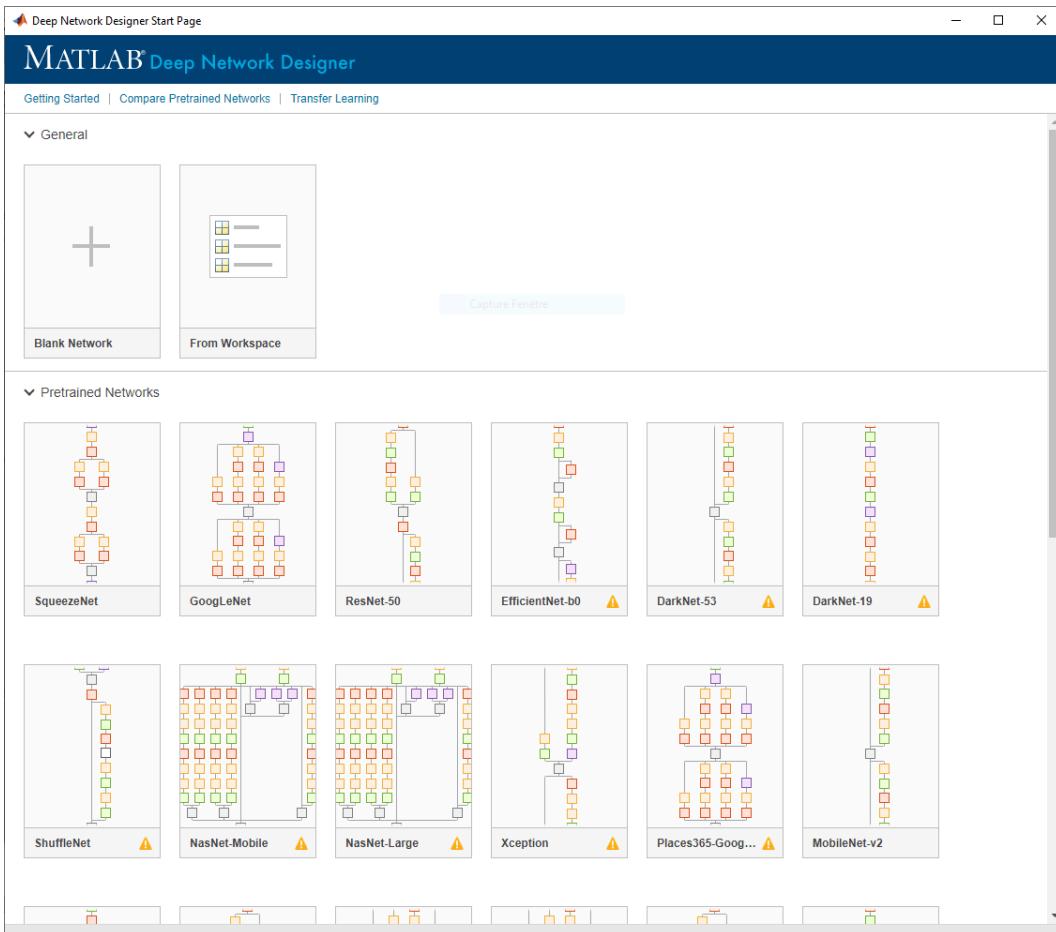
L'utilisation du Deep Network Designer permet aussi une visualisation des réseaux de neurones (couches de neurones de différents types et interconnexion des couches de neurones), ce qui est intéressant et permet de rendre plus concret ces concepts.

### Designer : charger le réseau à ré-entraîner

Créer un nouveau CNN. Le menu donne le choix entre

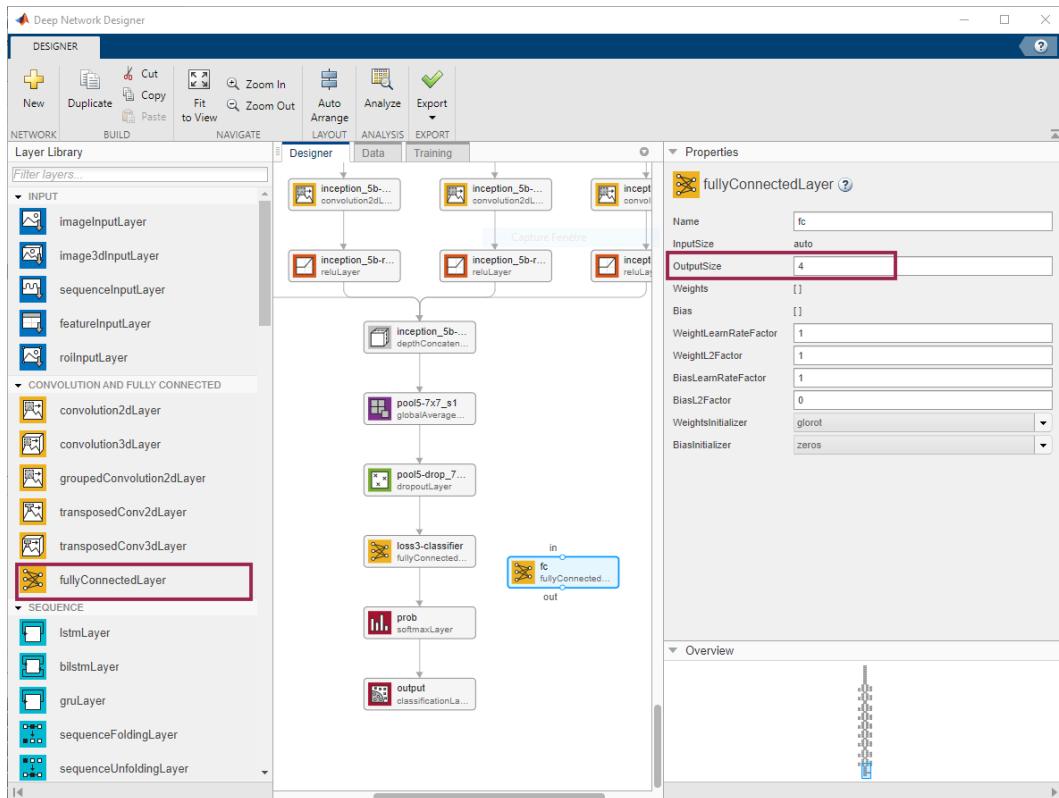
- créer un réseau vide
- charger un réseau existant (Alexnet,...)
- charger un réseau depuis le workspace

On choisira le réseau que l'on souhaite modifier pour faire le transfer learning (ici googlenet)



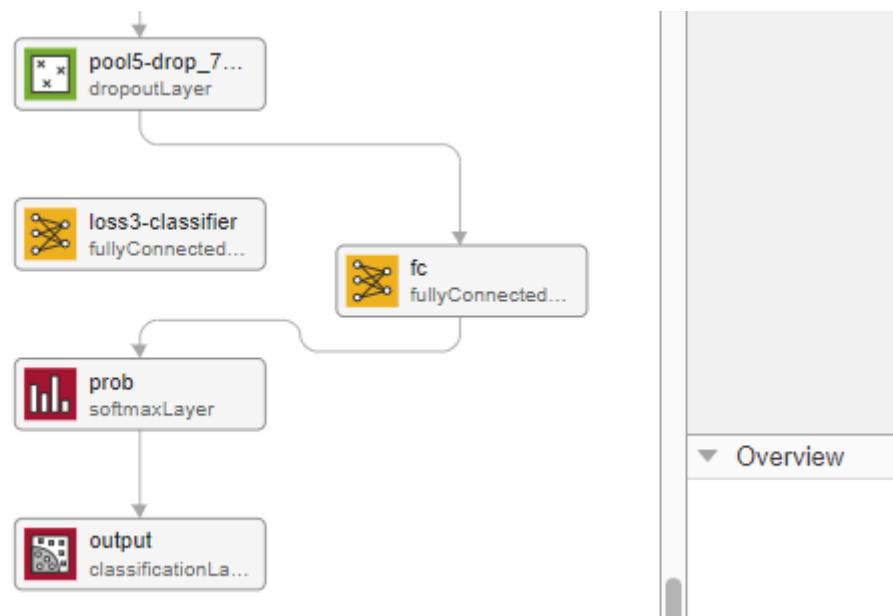
### Modification de la couche 'fully connected'

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"



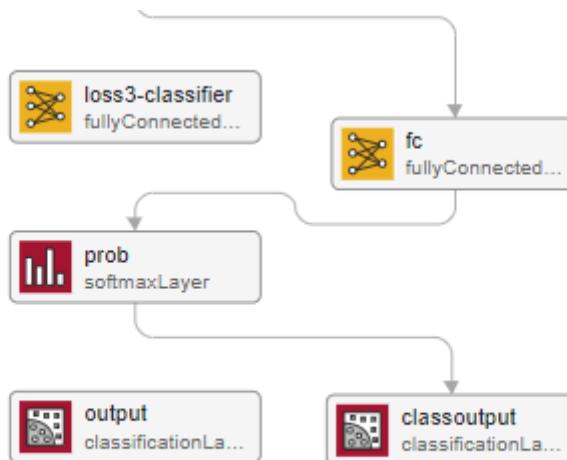
Output size : doit correspondre aux nombre de classes

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 "lgraph" est accessible 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.

## c. Enregistrer et réutiliser un réseau de neurones

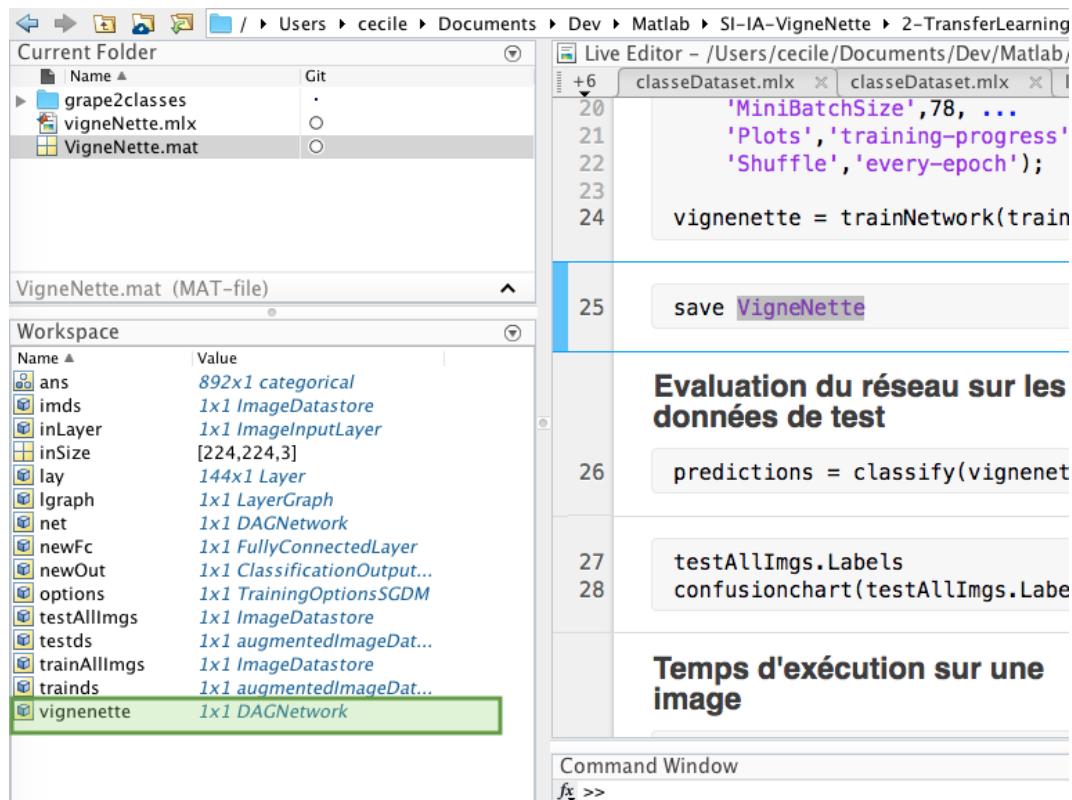
### RÉSUMÉ :

Comprendre comment enregistrer le cnn obtenu après l'entraînement  
Comment réutiliser le cnn sous Matlab et Simulink

### ! *Attention : Sauvegarder le cnn dans un fichier*

A la fin de l'exécution du livescript, Matlab a enregistré le nouveau réseau dans un fichier .mat ainsi que **tout le workspace**.

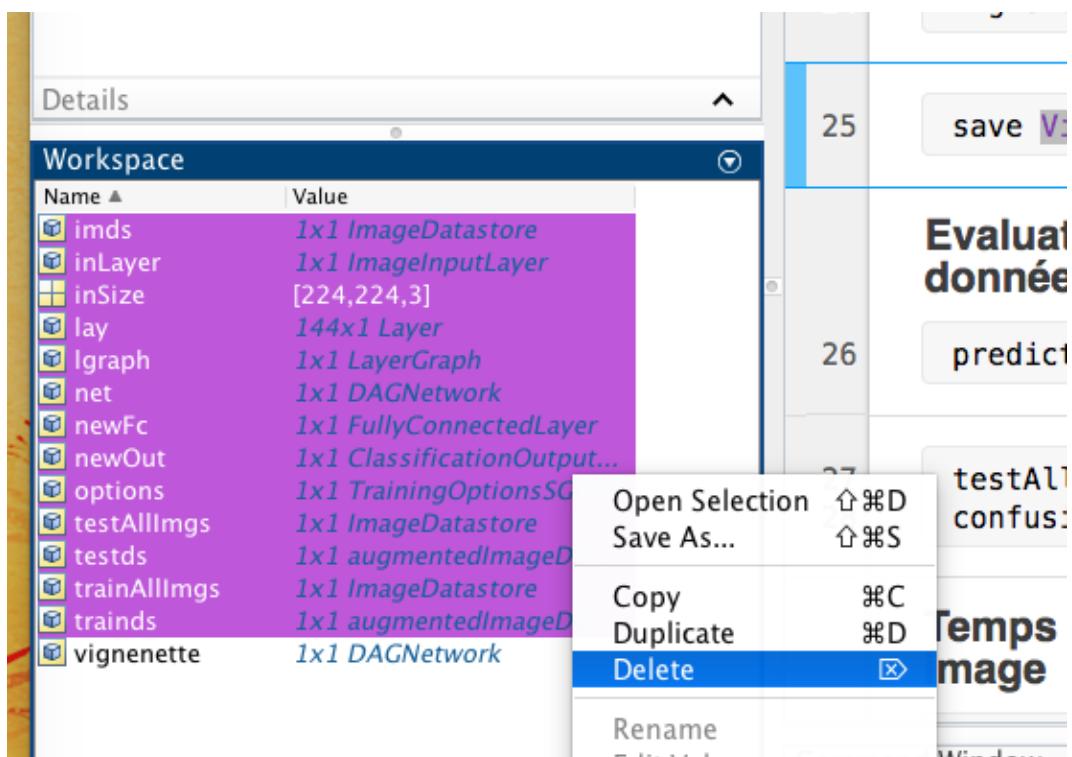
Pour s'en rendre compte, il suffit d'effacer le workspace, et de charger le fichier « VigneNette.mat » en double-cliquant dessus.



## Comment enregistrer le cnn pour pouvoir le réutiliser ?

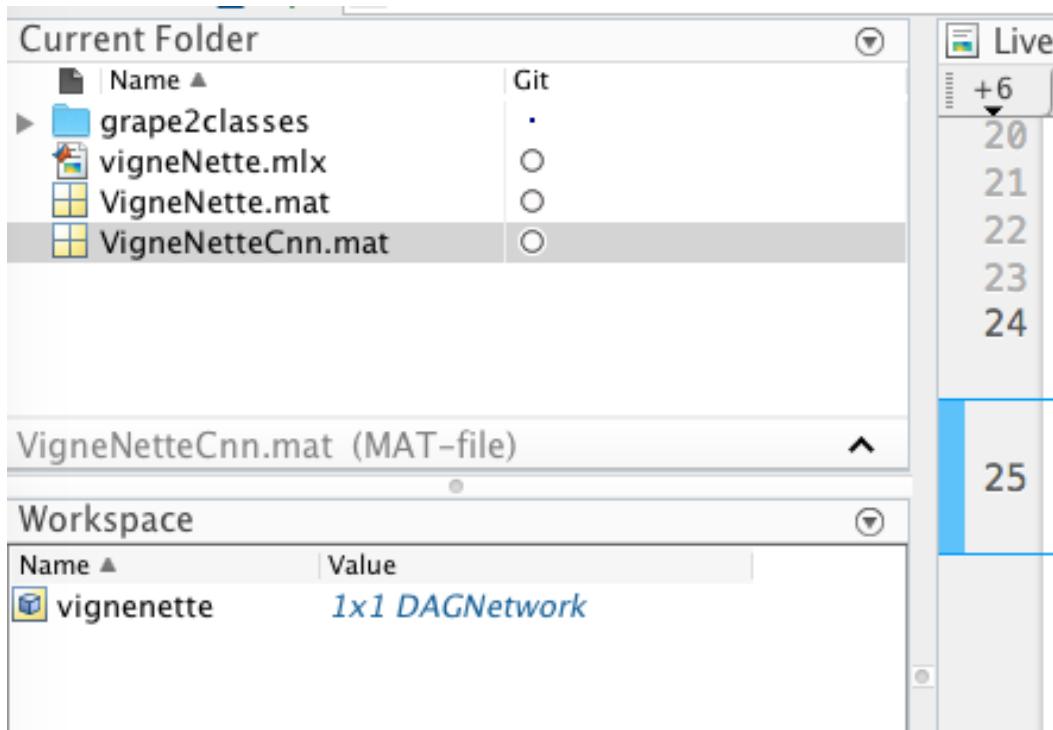
Pour utiliser facilement le cnn dans une autre application (sous Simulink par exemple) on aura besoin de créer un fichier .mat contenant uniquement le réseau de neurones.

Pour ce faire :supprimer toutes les variables du workspace sauf le réseau de neurones :



Suppression des variables du workspace pour ne garder que le réseau de neurones

Puis enregistrer la variable de type DAG dans un nouveau fichier matlab .mat (ici **VigneNetteCnn.mat**)  
En vidant le workspace et en cliquant sur le nouveau VigenNetteCnn.mat on doit charger uniquement le cnn



Créer un fichier .mat ne contenant que le réseau de neurones

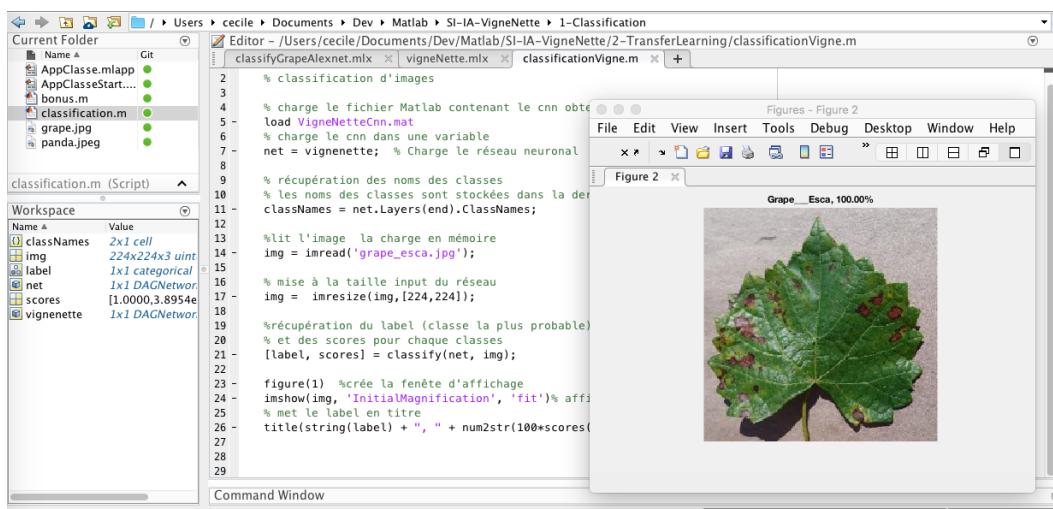
## Utiliser le nouveau réseau neuronal dans un autre script

Pour utiliser le réseau neuronal, il suffit de charger le fichier Matlab puis de charger le réseau de neurones dans une variable.

```

1 % charge le fichier Matlab contenant le cnn obtenu par transfer learning
2 load VigneNetteCnn.mat
3 % charge le cnn dans une variable
4 net = vigenette; % Charge le réseau neuronal
5
6

```



Utilisation du cnn vigenette dans un script Matlab

## + Complément : Fichiers sources

Les fichiers (LiveScript, images, script d'utilisation, fichiers .mat) sont disponibles sur [Github \(SI-IA-VigneNette\)](#).

### d. IHM : utiliser le réseau vignenette dans l'application

#### RÉSUMÉ :

Reprise de l'IHM faite dans la première partie (classification).

L'application est modifiée pour donner le label donné par le cnn vignenette.

#### Méthode : Fichier AppVigneNette

Le fichier AppClasse.mlap créé dans l'étape « 1-classificaation » est conservé, et dupliqué dans 2-TransferLearning/AppVigneNette.mlap

Le code sera modifié pour remplacer le réseau alexnet par le nouveau réseau vignenette obtenu précédemment.

#### Charger le cnn vignenette

Charger le fichier Matlab contenant le cnn (PATH à adapter, ici tout est dans le même répertoire) : dans les propriétés de l'application.

```
1 properties (Access = private)
2     vigneMat = load('VigneNetteCnn.mat');
3     net % Le cnn
4 end
5
```

Charger le réseau neuronal dans la variable net (dans la fonction startup)

```
1 % Chargement du CNN
2 app.net = app.vigneMat.vignenette;
3
```

La taille des images est à adapter car alexnet prend des images en 227x227 alors que vignenette (obtenu à partir de GoogleNet) prend des images en 224x224

```

38 -         imagesc(app.ImageAxes,im);
39
40
41 -         % Ajoute le label
42 -         img = imresize(im, [227 227]);
43 -         label = classify(app.net, img);
44 -         app.Label.Text = label;
45 -     end
46 - end
47
48
49 - % Callbacks that handle
50 - methods (Access = priv
51
52 -         % Code that executes after component creation
53 -         function startupFcn(app)
54
55 -             % Configure image axes
56 -             app.ImageAxes.Visible = 'off';
57 -             app.ImageAxes.Colormap = gray(256);
58 -             axis(app.ImageAxes, 'image');
59
60 -             % Chargement du CNN
61 -             app.net = app.vigneMat.vignenette;

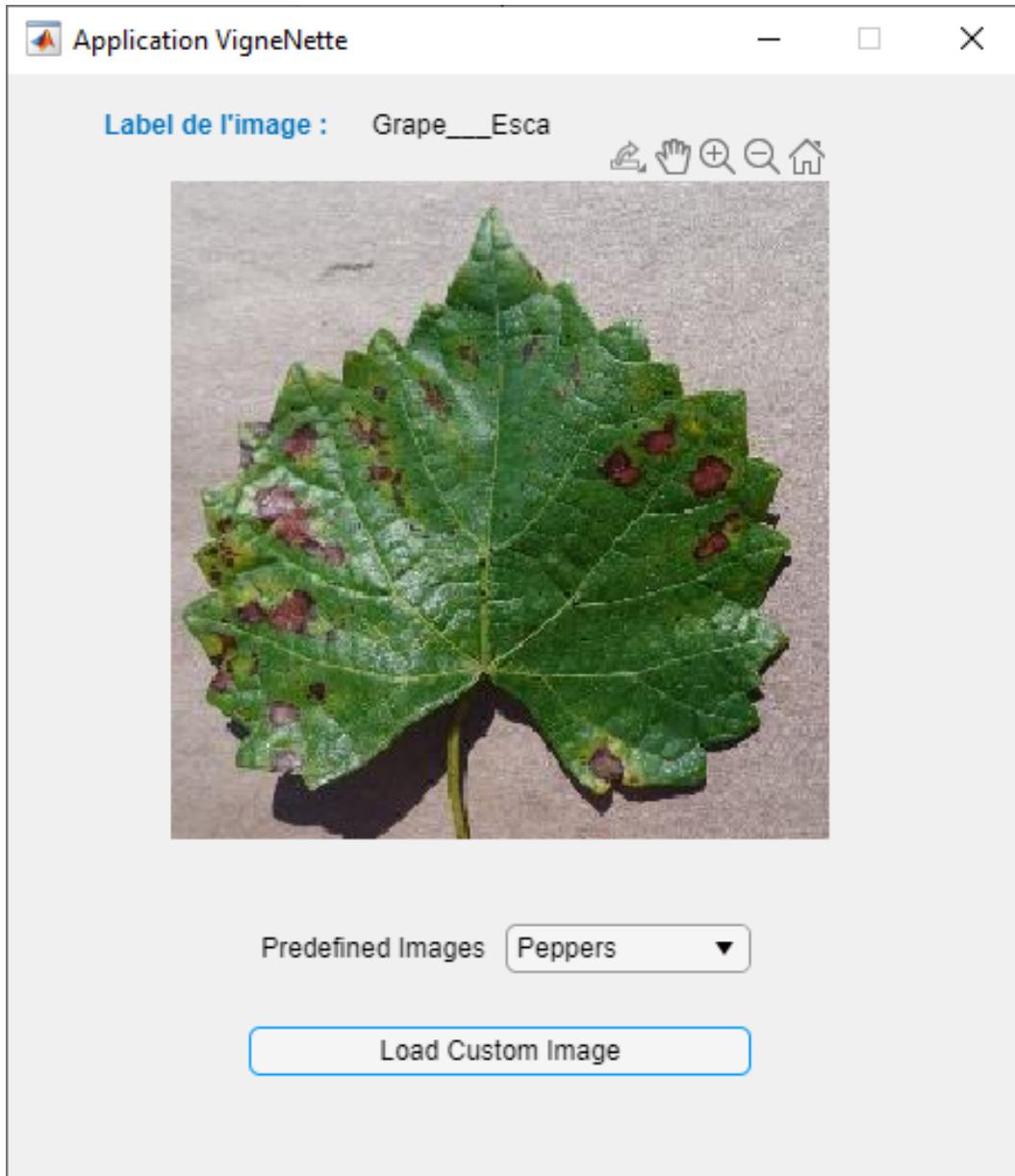
```

Modifier la ligne :

```
1 img = imresize(im, [224 224]);
```

### Test de l'application

Lancement de l'application et essai avec une image de feuille de vigne



Lancement de l'application et test avec une image de feuille de vigne

### e. IHM : Créer une application autonome pour Windows

#### RÉSUMÉ :

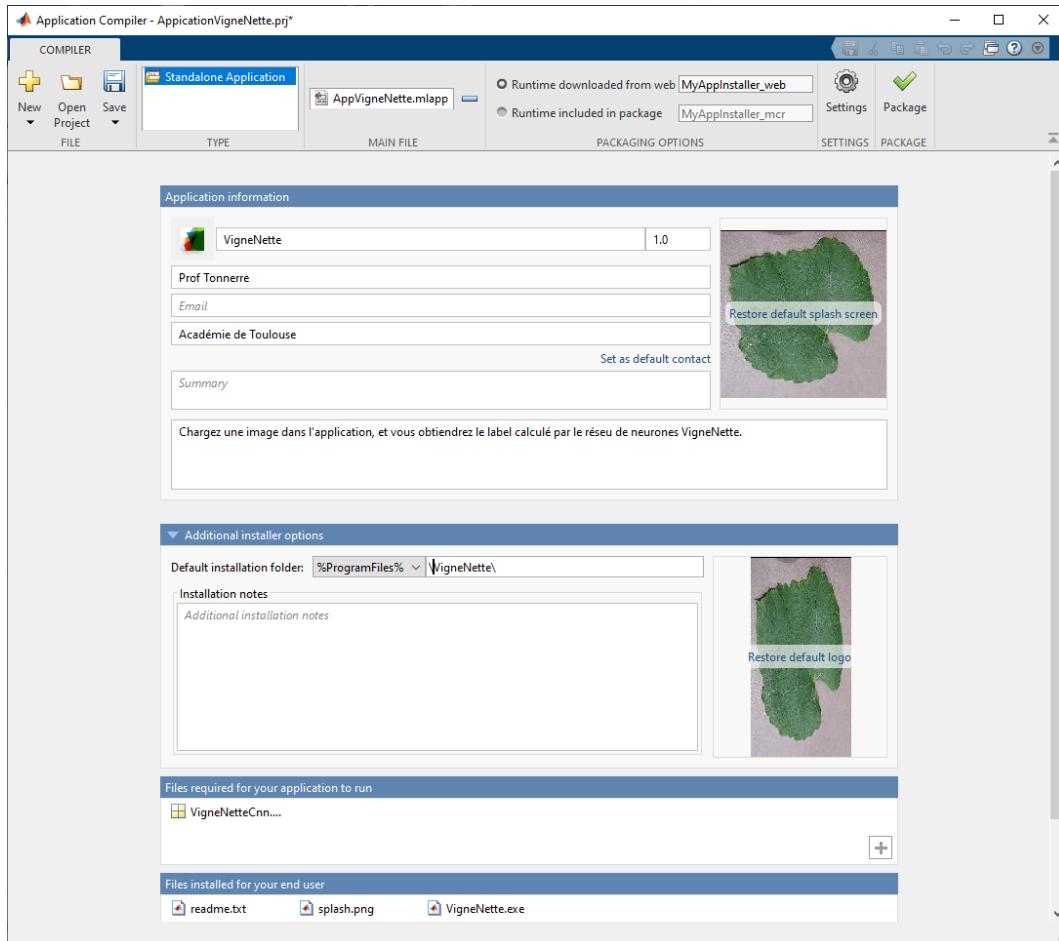
Il est possible de créer une application Windows qui pourra être installée et utilisée sur d'autres PC.

Tous les fichiers sont sur le [Github SI-IA-VigneNette](#)[p.]. (y compris l'executable qui permet d'installer l'application sous Windows).

### Déployer l'application en créant un installateur pour Windows

AppDesigner permet de déployer l'application et de créer une application autonome pour Windows.

Depuis AppDesigner : Share > Standalone Desktop App

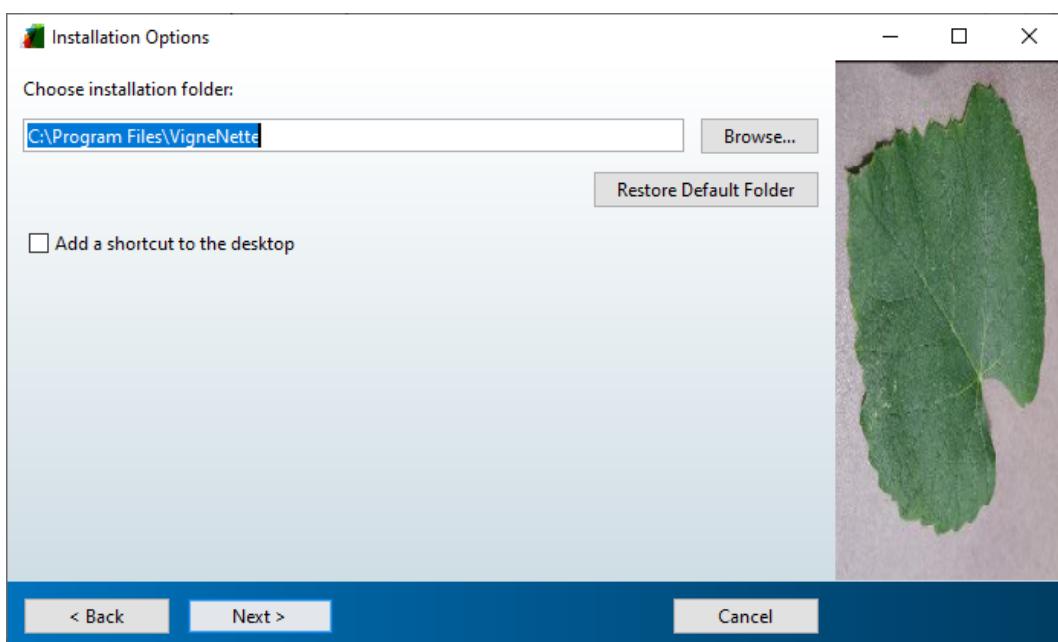
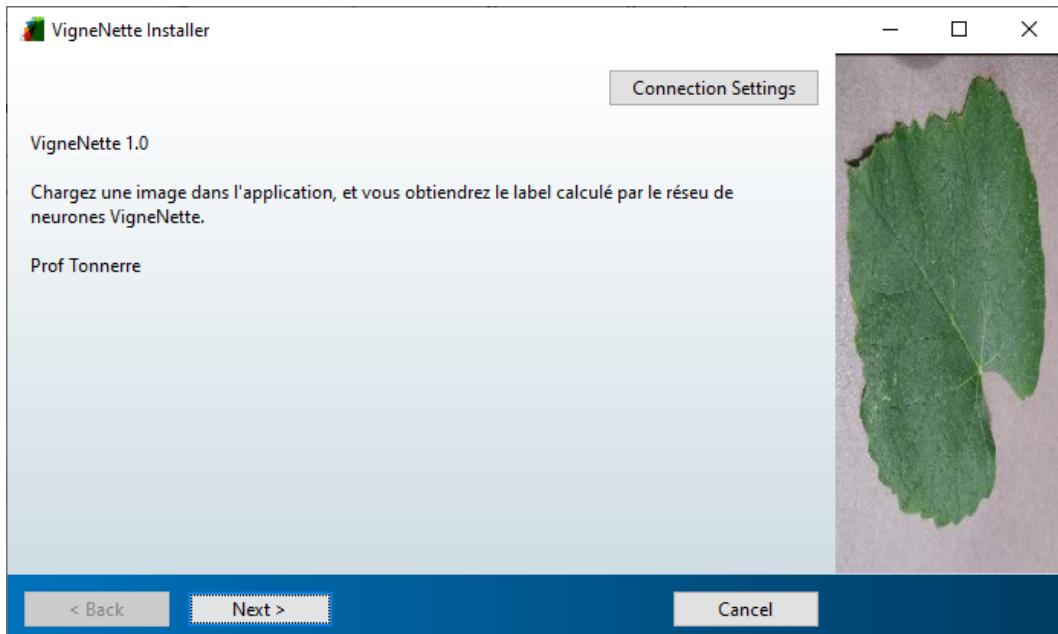


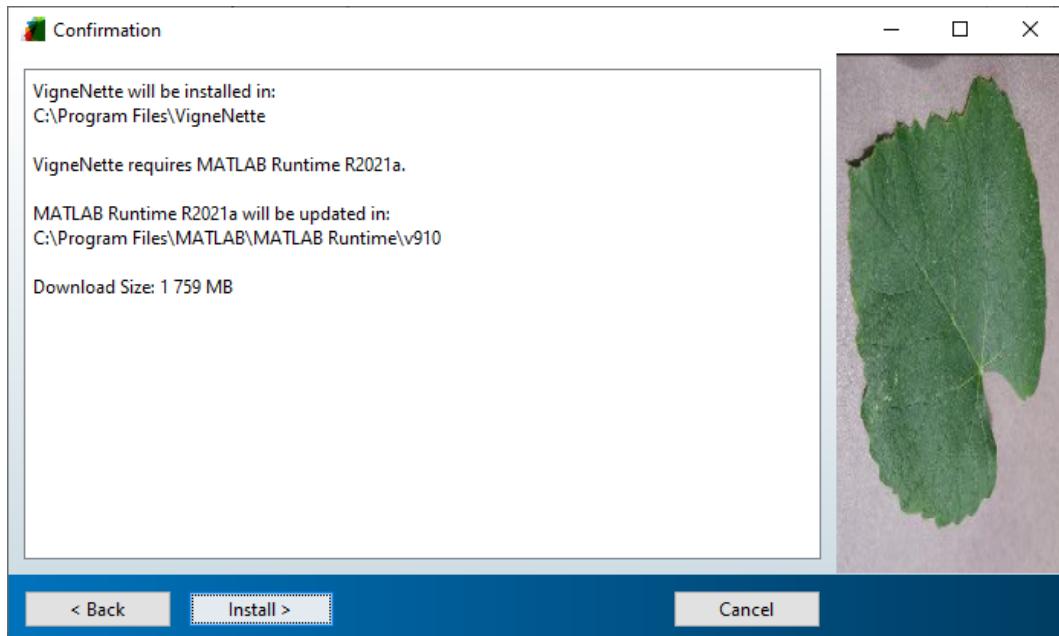
Création de l'application Windows (IHM de classification avec VigneNette)

Cliquer sur « Package »

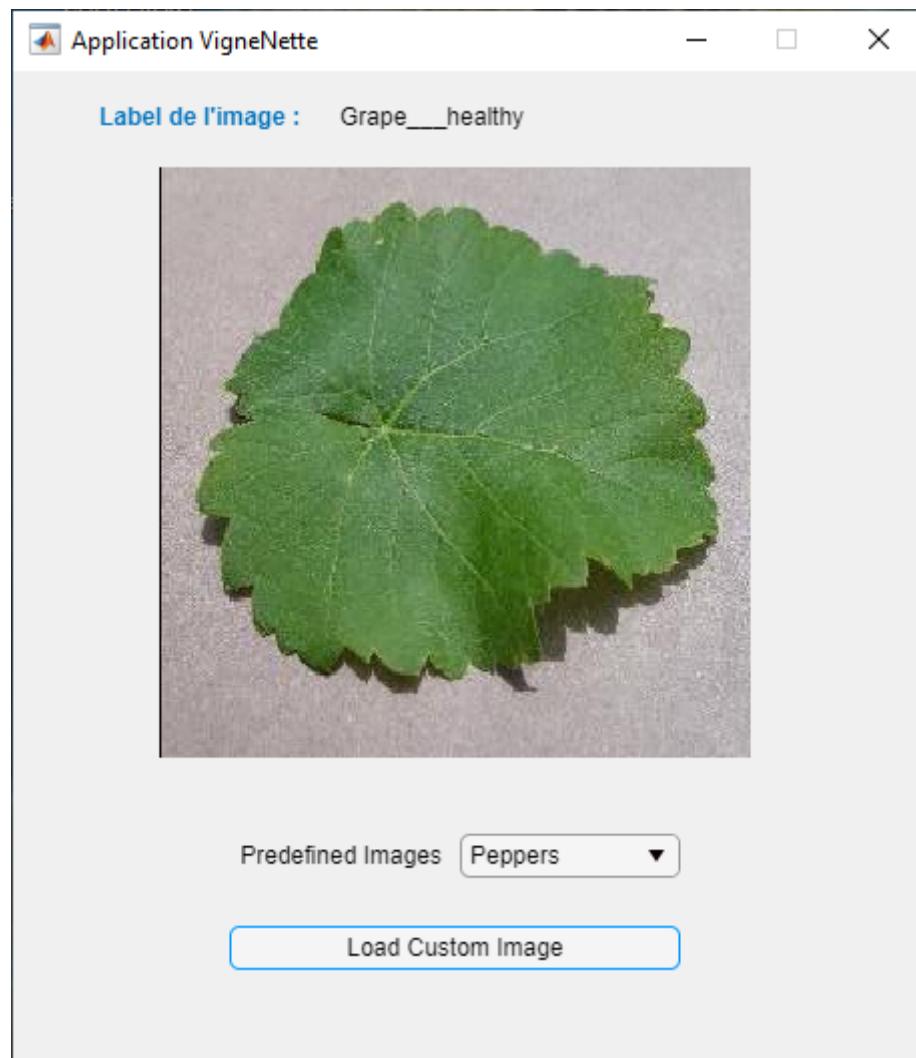
### Installer l'application

Matlab crée trois répertoires : l'installateur se trouve dans le répertoire « for\_redistribution ». Il suffit de lancer le fichier généré (ici MyAppInstallerWeb.exe et l'application s'installe comme n'importe quel logiciel sous Windows.





L'application peut être lancée et utilisée sur Windows



### 3. 3-Simulation - Déploiement - Alertes

DURÉE : 1h30

RÉSUMÉ :

Simulation de l'application sur Simulink et déploiement sur Raspberry Pi.  
Alerte (via MQTT)

#### a. Simulation / déploiement

##### Simulink

Il est possible de l'exemple proposé par Matlab : [Classify Objects Using Deep Learning Algorithm on Raspberry Pi Hardware](#)<sup>[p.]</sup>

[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](#)<sup>[p.]</sup>
- [Simulink Support Package for Raspberry Pi Hardware](#)<sup>[p.][p.]</sup>

Ainsi que les modules complémentaires. (Cf. [Installation](#)<sup>[p.49]</sup>)

##### 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 câble 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 :

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

##### Installation de la caméra

Connecter la caméra et valider avec raspistill et raspivid.

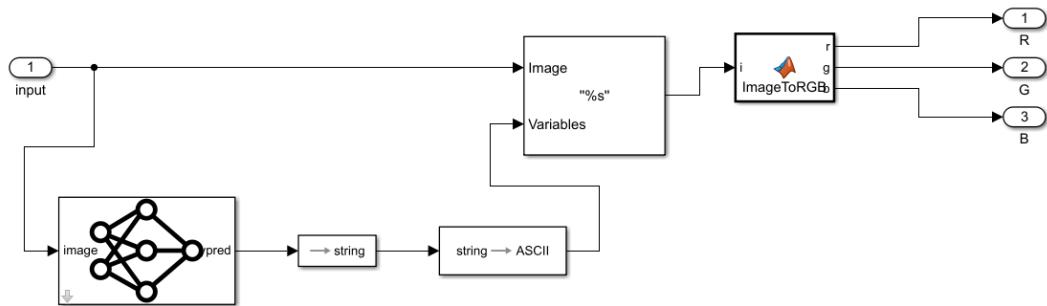
##### Simulation / Déploiement

##### Adaptation de l'exemple

Ouvrir l'exemple avec la commande

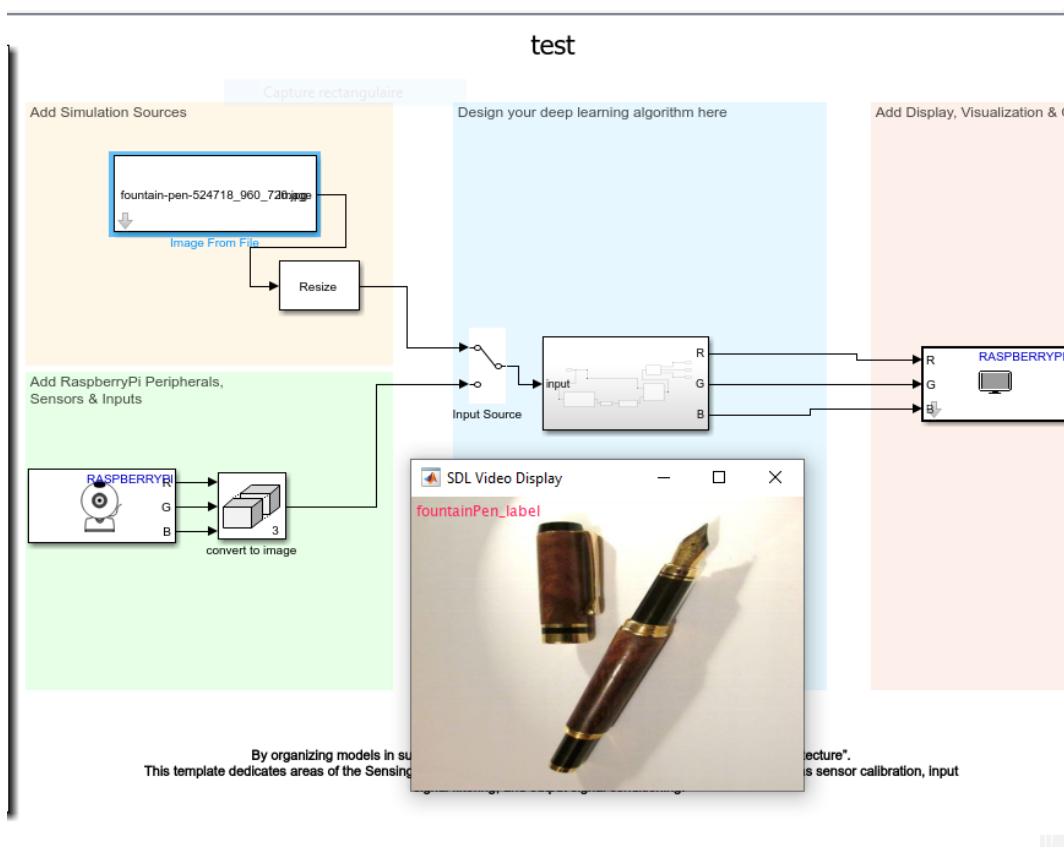
```
1 open_system('raspberrypi_object_classification');
```

Le sous-système

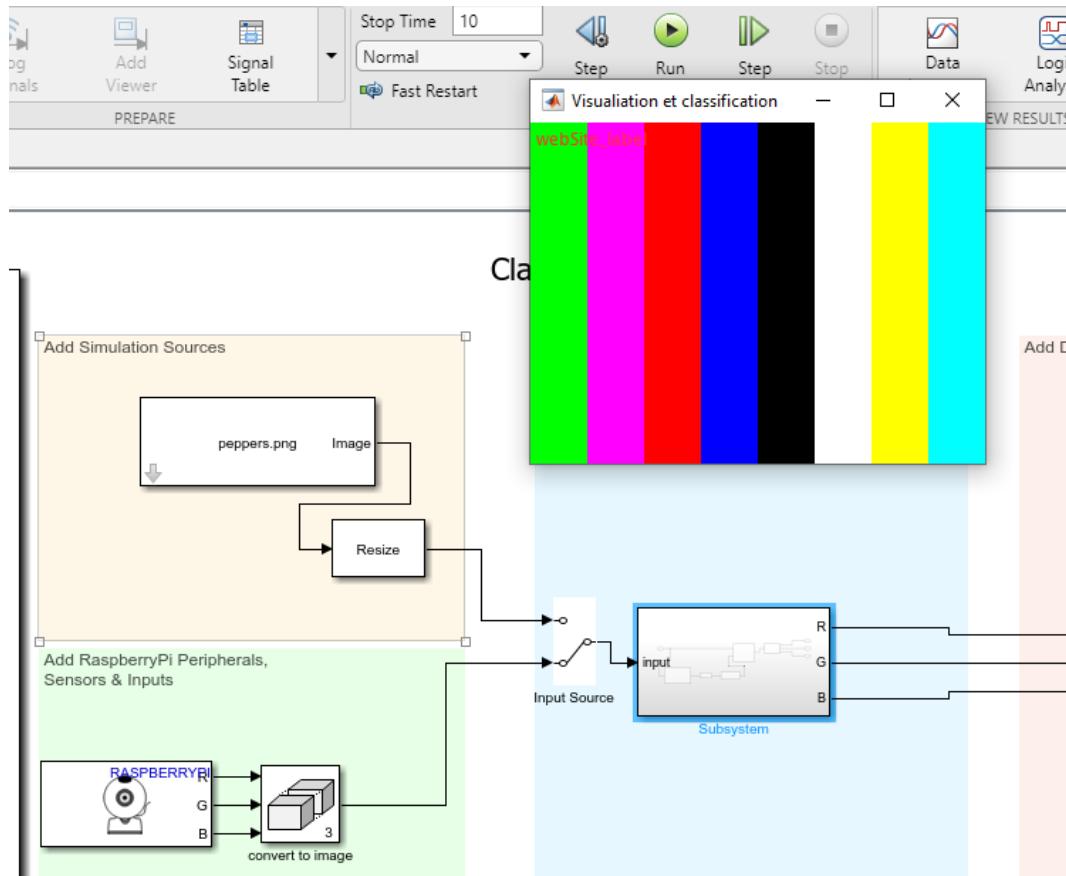


Mettre en œuvre l'exemple (ici avec squeezenet)

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 (Hardware - Build and Deploy). Attention cela prend quelques minutes le temps que le programme soit compilé puis 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'échantillonnage de la caméra est à 0.1. Ce qui fait rapidement planter la Raspberry car sa puissance de calcul ne suffit 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.

Il est important **d'alimenter suffisamment la Raspberry** sinon on risque d'avoir des redémarrages intempestifs.

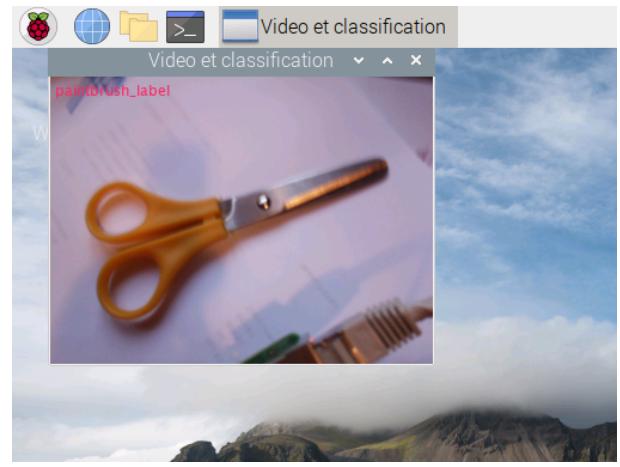
## Déploiement

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.

Pour que Simulink accepte de prendre notre cnn, il est impératif que le fichier matlab .mat contienne uniquement le cnn. (voir à ce sujet la rubrique « [Enregistrer et réutiliser un réseau de neurones](#)<sup>[p.37]</sup> »)

## ANNEXES

### 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](#)<sup>[p.]</sup>

#### Réseau ResNet-18

[Deep Learning Toolbox Model for ResNet-18 Network](#)<sup>[p.][p.]</sup>

#### Réseau AlexNet

[Deep Learning Toolbox Model for AlexNet Network](#)<sup>[p.]</sup>

#### Réseau MobileNet-v2

[Deep Learning Toolbox Model for MobileNet-v2 Network](#)<sup>[p.]</sup>

### 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](#)<sup>[p.50]</sup>

#### Support pour Raspberry Pi

Il faut ajouter deux add-ons :

[MATLAB Support Package for Raspberry Pi Hardware](#)<sup>[p.]</sup>

[Simulink Support Package for Raspberry Pi Hardware](#)<sup>[p.][p.]</sup>

#### Simulink coder

Cet add-on permet de générer du code C/C++ à partir de Simulink.

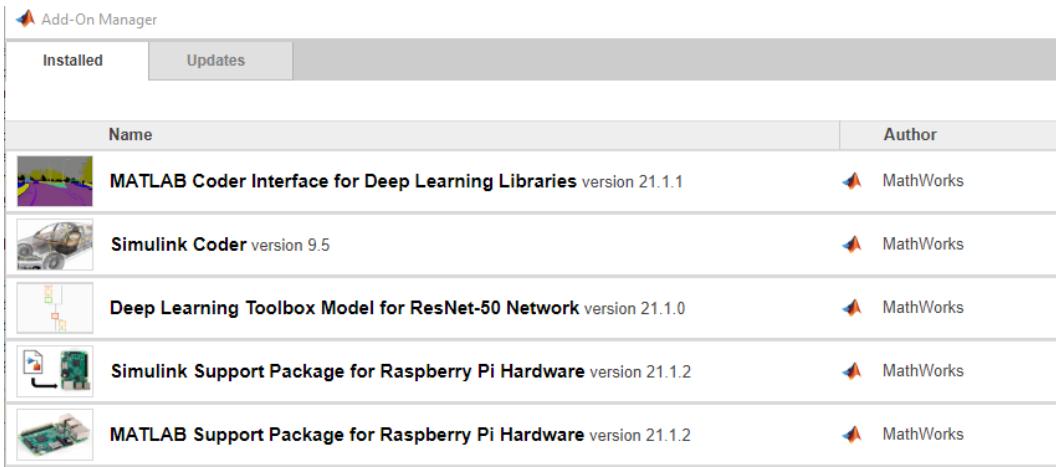
Utile pour déployer le modèle Simulink sur une Raspberry Pi.

Pour fonctionner, il faut un [compilateur compatible comme MinGW](#)<sup>[p.]</sup> pour Windows.

#### Matlab Coder interface for Deep Learning Libraries

Nécessaires pour la compilation via Simulink Coder

#### Résumé



## 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](#)<sup>[p.]</sup>

[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](#)<sup>[p.]</sup>
- [Simulink Support Package for Raspberry Pi Hardware](#)<sup>[p.][p.]</sup>

Ainsi que les modules complémentaires. (Cf. [Installation](#)<sup>[p.49]</sup>)

#### 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 câble 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 :

```
1 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](#)<sup>[p.]</sup>

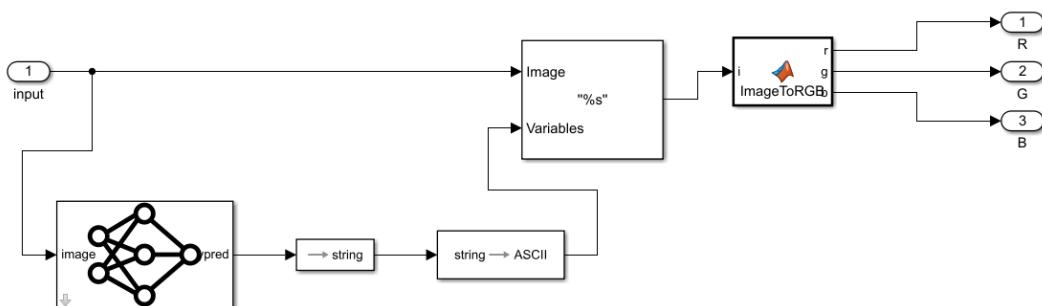
## Installation de la caméra

Connecter la caméra et valider avec raspistill et raspivid.

## Simulation / Déploiement

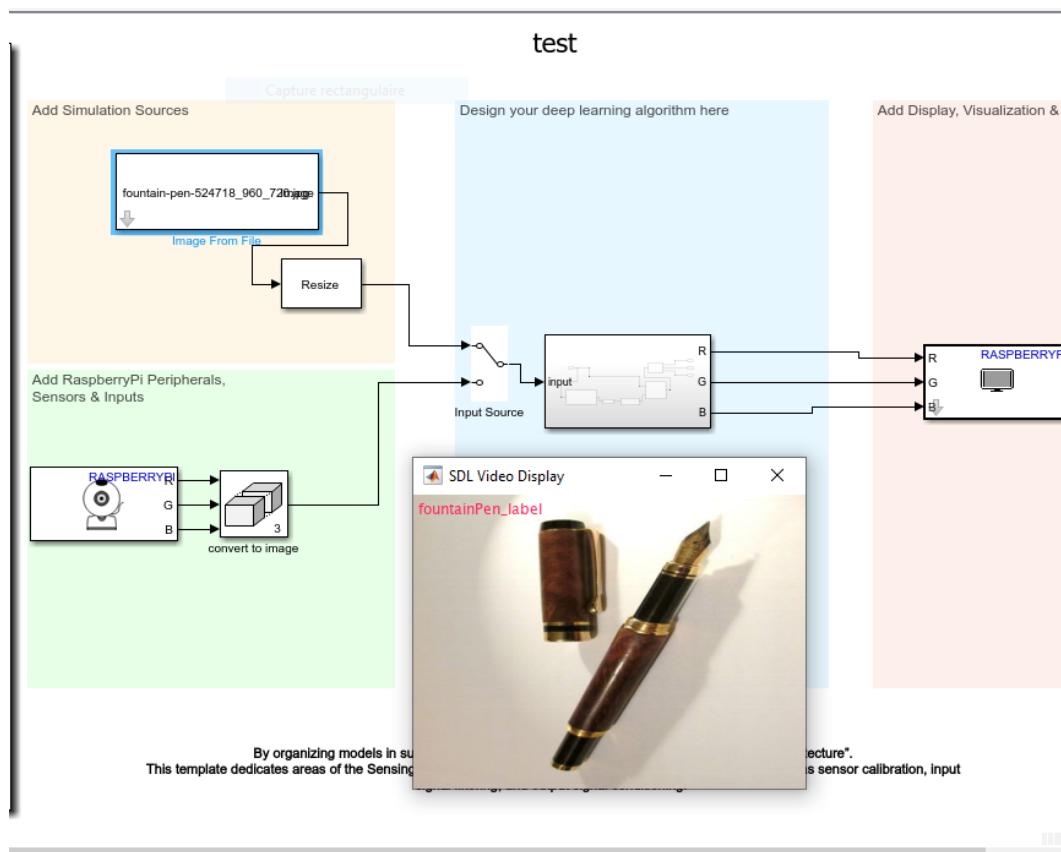
### Adaptation de l'exemple

Le sous-système

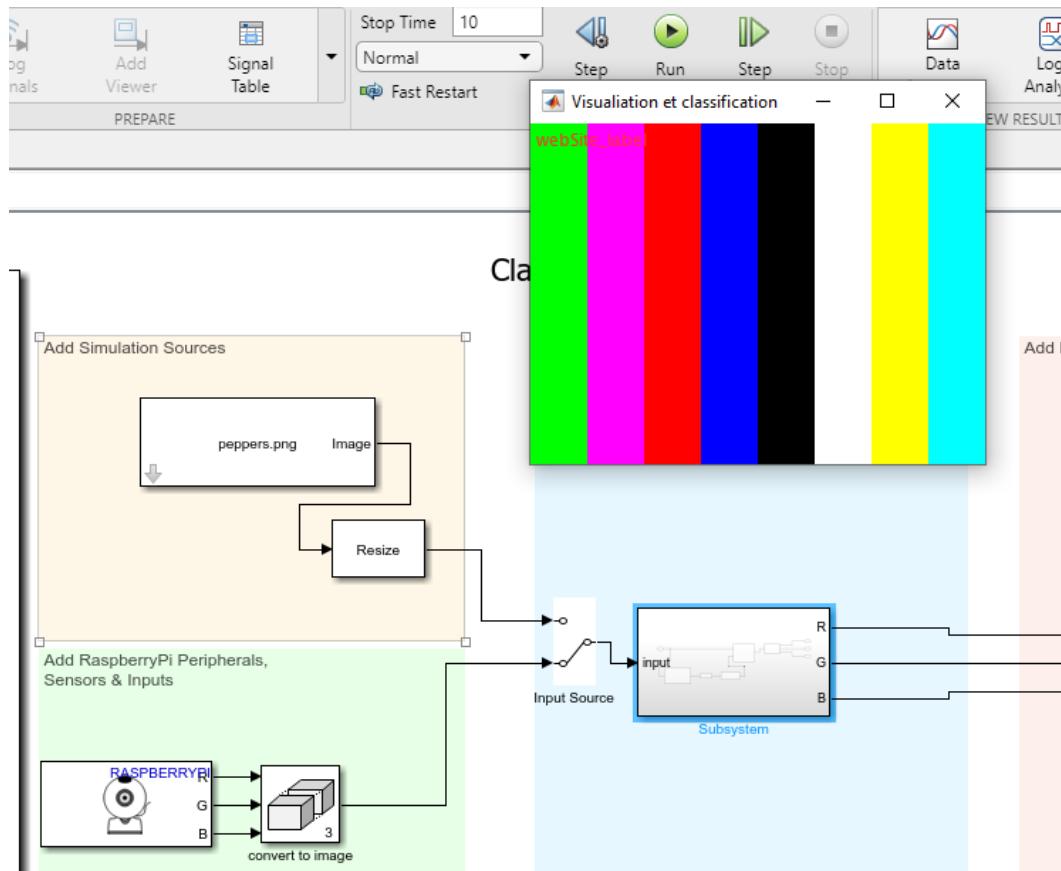


Mettre en œuvre l'exemple (ici avec squeezenet)

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 (Hardware - 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 suffit 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.

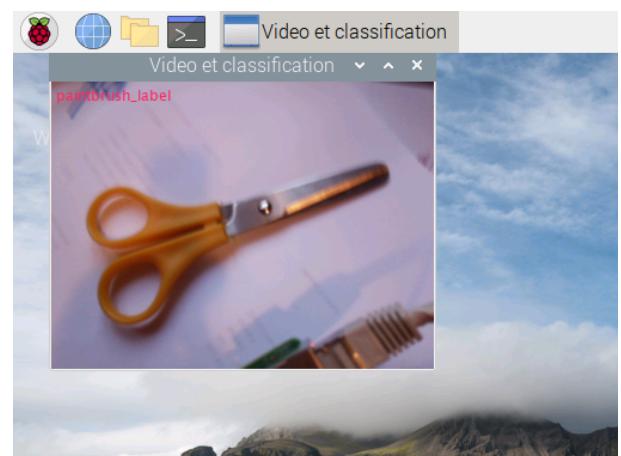
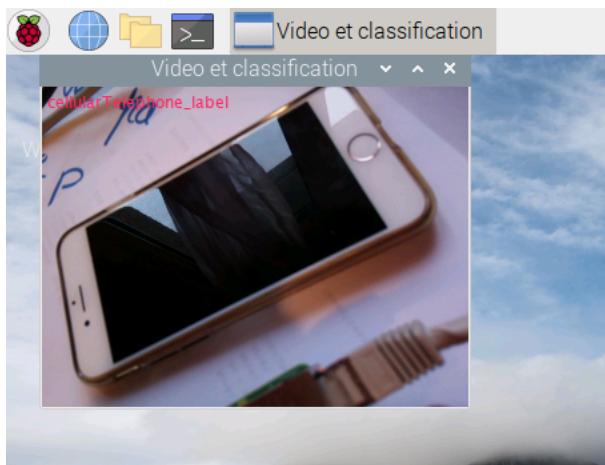
## Déploiement

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.



# SOLUTIONS

## Solution n°1

Exercice p. 13

```

1 net = alexnet; % Charge le réseau neuronal
2
3 img = imread('plage.jpg'); %lit l'image la charge en mémoire
4 img = imresize(img,[227,227]); % mise à la taille input du réseau
5 label = classify(net, img); % on récupère le label
6 figure %crée la fenête d'affichage
7
8 imshow(img, 'InitialMagnification', 'fit')% affiche l'image
9 title(string(label)); % met le label en titre

```

## Solution n°2

Exercice p. 15

```

1 % Charge le réseau neuronal
2 net = alexnet;
3 % récupération de toutes les classes
4 classNames = net.Layers(end).ClassNames;
5
6 %lit l'image la charge en mémoire en tableau de pixels
7 img = imread('panda.jpg');
8 % mise à la taille input du réseau
9 img = imresize(img,[227,227]);
10 % on récupère le label
11 [label, scores] = classify(net, img);
12
13 % Affichage de l'image
14 figure(1) %crée la fenête d'affichage
15 imshow(img, 'InitialMagnification', 'fit')% affiche l'image
16 title(string(label) + ", " + num2str(100*scores(classNames == label), '%.2f') + "%");
17
18

```

## Solution n°3

Exercice p. 17

```

1 % Charge le réseau neuronal
2 net = alexnet;
3 % récupération de toutes les classes
4 classNames = net.Layers(end).ClassNames;
5
6 %lit l'image la charge en mémoire en tableau de pixels
7 img = imread('panda.jpg');
8 % mise à la taille input du réseau
9 img = imresize(img,[227,227]);
10 % on récupère le label
11 [label, scores] = classify(net, img);
12
13 % Affichage de l'image
14 figure(1) %crée la fenête d'affichage
15 imshow(img, 'InitialMagnification', 'fit')% affiche l'image
16 title(string(label) + ", " + num2str(100*scores(classNames == label), '%.2f') + "%");
17
18 % affiche les 5 plus fortes probabilités
19 % Trie par score décroissant et trouve les indices pour les 5 plus forts
20 [~,idx] = sort(scores,'descend');
21 idx = idx(5:-1:1);
22 % récupère les classes et les scores pour ces indices
23 classNamesTop = net.Layers(end).ClassNames(idx);

```

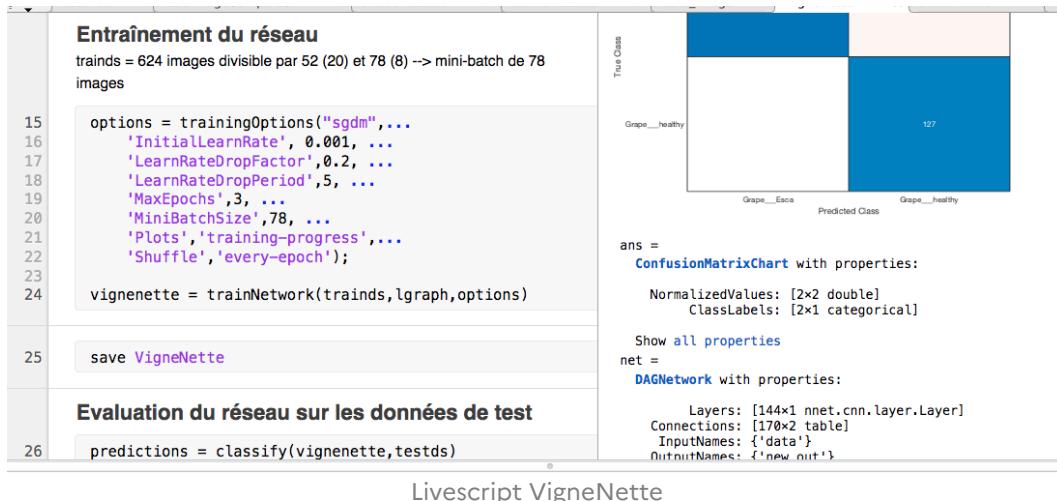
```

24 scoresTop = scores(idx);
25 % affiche une nouvelle figure avec un graphique en barres
26 figure(2)
27 barh(scoresTop)
28 xlim([0 1])
29 title('Top 5 des prédictions')
30 xlabel('Probabilité')
31 yticklabels(classNamesTop)
32
33

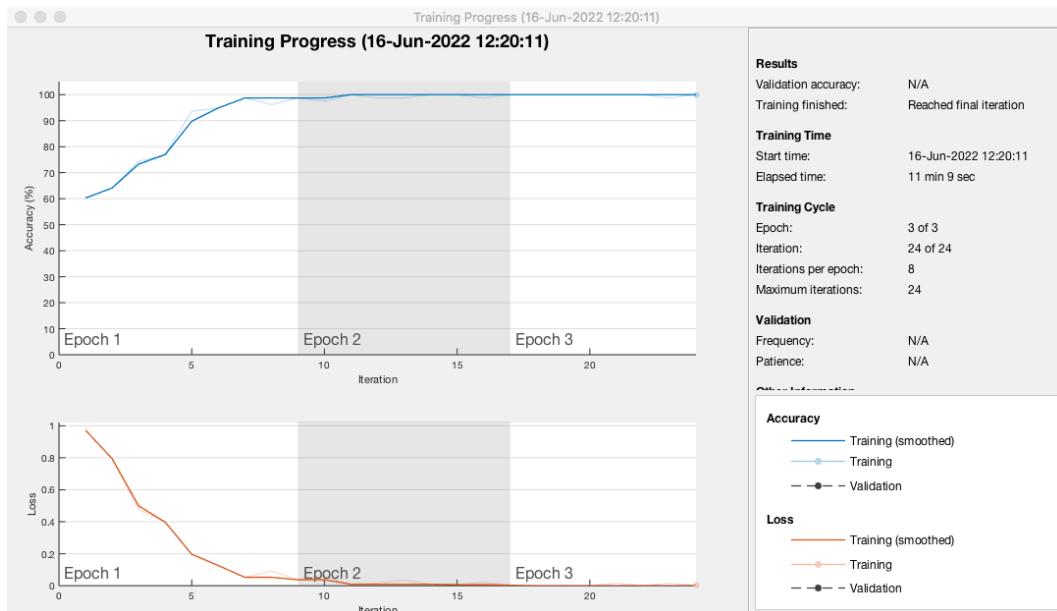
```

**Solution n°4**

Exercice p. 32

Voir sur [Github SI-IA-VigneNette](#)<sup>[p.]</sup> le fichier vigneNette mlx

Livescript VigneNette



Affichage du « training progress » pendant l'entraînement du réseau sur les images

## GLOSSAIRE

### CNN

CNN = Convolutionnal Neural Network (réseau neuronal convolutif). Ce sont des réseaux de neurones qui peuvent comprendre plusieurs dizaines voire plusieurs centaines de couches. Ils intègrent plusieurs couches convolutives qui appliquent des filtres sur des images pour en extraire des caractéristiques

### Deep Learning

Un sous-ensemble de l'intelligence artificielle et du machine learning, où la machine apprend sur de gros volumes de données, en utilisant des réseaux de neurones profonds (sur un grand nombre de couches)