

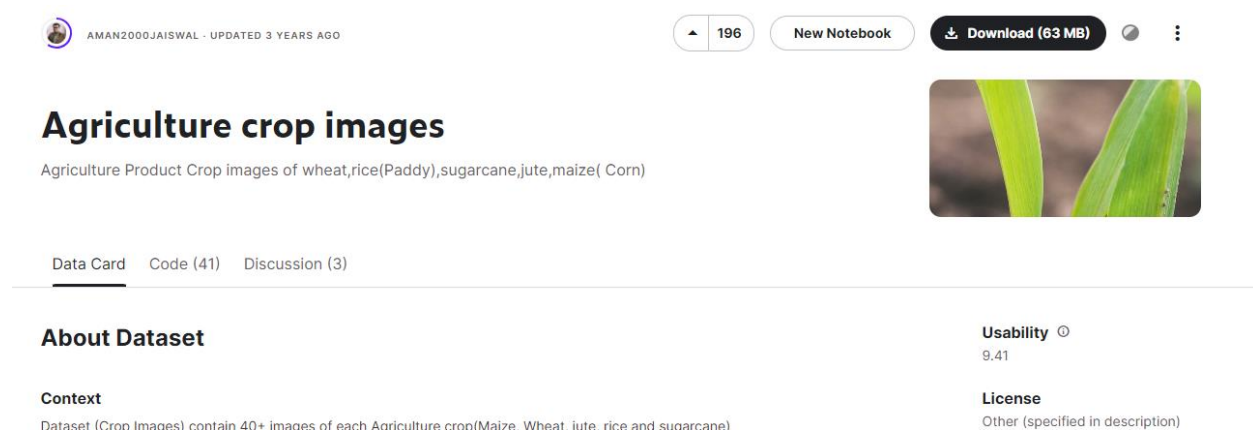
## Rapport TP3 8INF804 : Vision artificielle et traitement d'images

### Introduction

Lors de ce travail nous allons effectuer la comparaison des performances d'un CNN avec une architecture que nous allons entrainer de zéro et un CNN qui nous allons entrainer à l'aide du « transfer learning ». Nous allons prendre comme dataset, un dataset d'images de plantations que nous avons trouvées sur Kaggle. Ce dataset contient des images de cinq types de plantations dont : jute, maïs, riz, blé, et canne à sucre.

Lors de ce rapport nous allons présenter les différentes étapes de notre méthodologie, les choix et raisons de conception de notre CNN et une analyse des résultats obtenus. De plus, nous allons essayer de présenter les différentes difficultés rencontrées lors de ce TP et comment nous pourrions améliorer le travail.

### Dataset Kaggle



AMAN2000JAISWAL · UPDATED 3 YEARS AGO

196 New Notebook Download (63 MB)

### Agriculture crop images

Agriculture Product Crop images of wheat, rice(Paddy), sugarcane, jute, maize( Corn)

Data Card Code (41) Discussion (3)

#### About Dataset

**Context**  
Dataset (Crop Images) contain 40+ images of each Agriculture crop(Maize, Wheat, iute, rice and sugarcane)

**Usability** 9.41

**License**  
Other (specified in description)

<https://www.kaggle.com/datasets/aman2000jaiswal/agriculture-crop-images>

### Exécution de code

```
PROBLEMS 17 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\paul\OneDrive\Paul-Emmanuel\UQAC 1\8INF804 - Vision Artificielle\TP3_Project_Vision\TP3_Vision\ATTE_BERTRAND_EVORA_TOUGMA_TP3_VISION> python train.py
```

Exemples d'image de chaque classe du dataset :

Gauche à droite : Jute, Maize, Rice, Sugarcane, Wheat



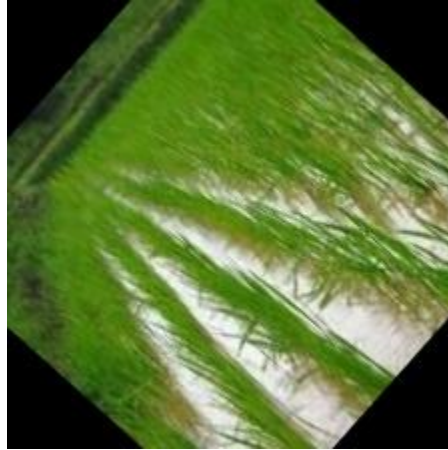
On constate que les différentes plantations ont des différences au niveau de leur texture, couleur, disposition et forme. On développera donc des CNN qui sont censés classifier ces plantations en fonction de ces caractéristiques.

### Augmentation de données

Le dataset Kaggle sélectionné contient aussi des images qui sont issues des images de bases auxquelles des transformations ont été appliquées. Ainsi cette augmentation de données permet d'avoir un dataset plus complet et d'entraîner des modèles plus robustes à différents types d'images.

Exemple de photo originelle et un exemple de transformation :

a. Rotation



b. Reflet selon l'axe verticale



## Explication de l'architecture de notre CNN

Afin de développer notre architecture de notre CNN nous avons effectué une phase de recherche pendant laquelle nous avons étudié les « best practice » de l'industrie de Deep Learning. Ces différentes conventions qui sont généralement utilisées nous ont permis de constituer une architecture qui nous semble efficace.

Ainsi nous avons notre architecture de CNN :

```
def create_cnn_model():
    model = Sequential()

    model.add(Conv2D(filters=32, kernel_size=(3, 3), padding='Same',
                     activation='relu', input_shape=(48, 48, 3)))
    model.add(Conv2D(filters=32, kernel_size=(3, 3), padding='Same',
                     activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='Same',
                     activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='Same',
                     activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(128, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(5, activation="softmax"))

    optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])

    return model
```

- Nous avons utilisé plusieurs couches de convolutions afin de permettre au CNN de reconnaître avec différents niveaux de complexités des tendances dans les images.
- Suites à plusieurs essais, les valeurs des paramètres ont été choisis tel qu'ils maximisent la précision du modèle
- Une convention qui est souvent utilisé en DeepLearning est de prendre comme paramètre des puissances de 2 afin de mieux correspondre à l'architecture matérielles des processeurs par exemple 32 filtres
- On utilise des couches « Maxpooling » pour réduire la dimension spatiale et réduire la charge de calcul de l'entraînement
- Des couches « dropout » ont été ajouté pour éviter le « overfitting » du modèle en supprimant de façon aléatoire des neurones du modèle
- On utilise « padding » pour préserver la taille des images lorsqu'elles sont traitées par le CNN

- Enfin notre architecture se termine par une couche « fully connected » qui est relié à notre couche de sortie avec une activation de « softmax » qui permet de calculer cinq valeurs respectivement la probabilité que l'image soit dans chacune des 5 classes

Résultat de l'architecture :

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	896
conv2d_1 (Conv2D)	(None, 48, 48, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 24, 24, 32)	0
dropout (Dropout)	(None, 24, 24, 32)	0
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
conv2d_3 (Conv2D)	(None, 24, 24, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 128)	1179776
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 5)	645

```

=====
Total params: 1,245,989
Trainable params: 1,245,989
Non-trainable params: 0

```

## Explication de l'architecture du modèle de Transfert

Nous utilisons le modèle de classification d'images déjà entraîné VGG16 disponible dans les applications de Keras. Nous gelons les « hidden layers » afin de pouvoir se servir des paramètres et poids déjà entraînés du modèle.

A la place de la couche de sortie, nous allons ajouter à ce modèle une couche dense et une de sortie qui pourront mieux reconnaître les images des plantations d'agriculture.

Ainsi nous avons le code suivant :

```
# Function to create a transfer learning model with VGG16
def create_transfer_model():
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(48, 48, 3))
    model = Sequential()
    model.add(base_model)
    model.add(Flatten())
    model.add(Dense(256, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(5, activation="softmax"))

    for layer in base_model.layers:
        layer.trainable = False

    optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])

    return model
```

Nous obtenons donc l'architecture suivante :

```
None
Model: "sequential_1"
-----
Layer (type)                Output Shape              Param #
-----
vgg16 (Functional)          (None, 1, 1, 512)        14714688
flatten_1 (Flatten)         (None, 512)              0
dense_2 (Dense)              (None, 256)              131328
dropout_3 (Dropout)         (None, 256)              0
dense_3 (Dense)              (None, 5)                1285
-----
Total params: 14,847,301
Trainable params: 132,613
Non-trainable params: 14,714,688
```

## Entraînement du modèle

Le dataset Kaggle est répartie sous différents dossiers ainsi lors du premier entraînement nous choisissons de prendre comme split :

- train\_set: `kag2` contenant les images de bases et les données augmentées qui contient 800 images
- test\_set : `test_crop_image` qui contient 50 images
- validation\_set : `some more images` qui contient 50 images

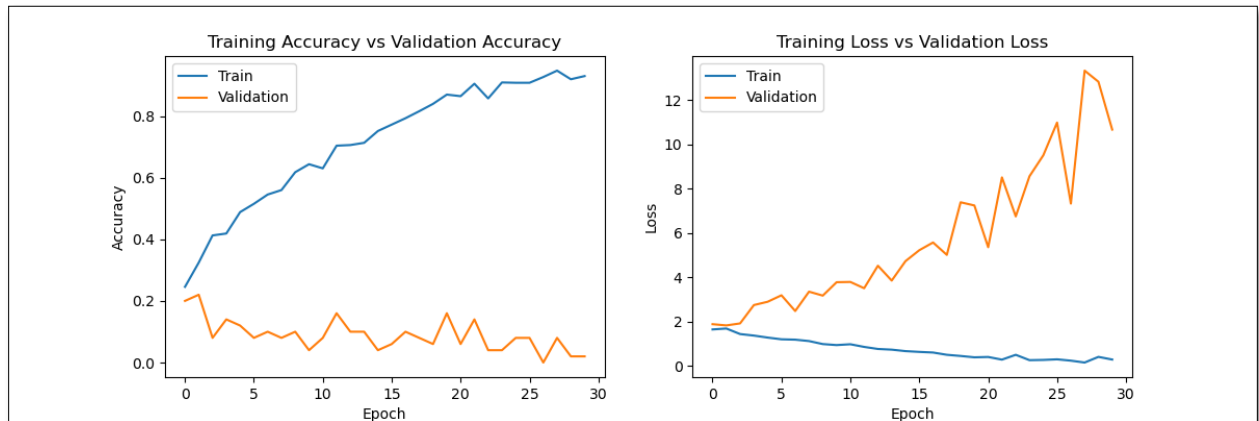
On affiche les courbes de précision du train\_set et du validation\_set, ces courbes nous permettent d'analyser les performances des modèles.

Enfin, on évalue la capacité des modèles à classifier les images qui se trouvent dans le test\_set.

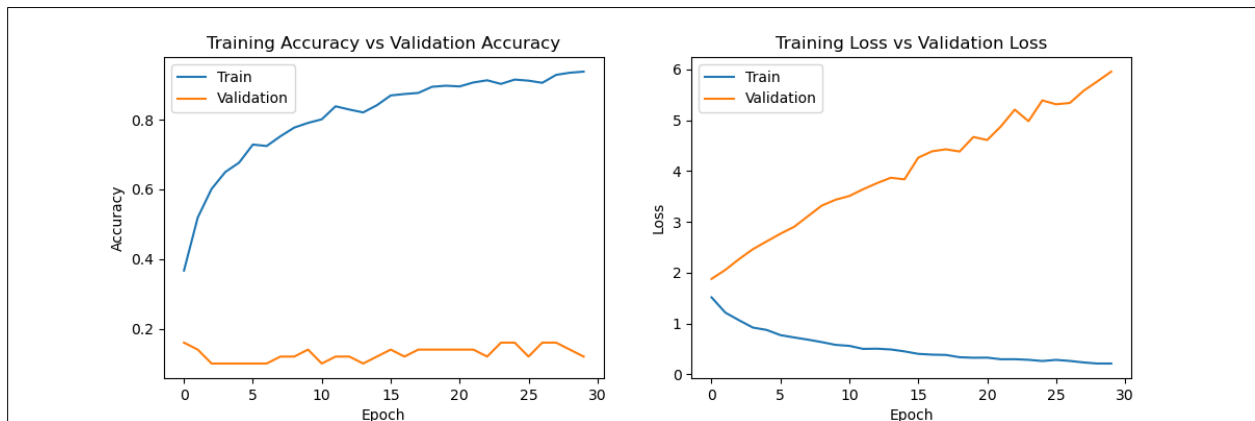
En présence du phénomène stochastique lié à l'entraînement des poids et biais nous lançons plusieurs fois l'apprentissage et nous analysons les résultats obtenus.

## Résultats obtenus

### CNN



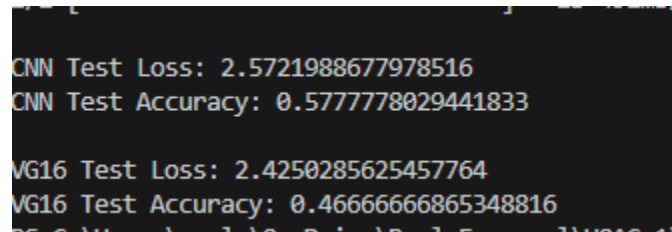
### VGG16





On constate que les deux modèles performant mal avec l'ensemble de validation, cependant ils arrivent à bien classifier les images qui se trouvent dans l'ensemble d'entraînement.

Evaluation de l'ensemble de test



```
CNN Test Loss: 2.5721988677978516
CNN Test Accuracy: 0.5777778029441833

VGG16 Test Loss: 2.4250285625457764
VGG16 Test Accuracy: 0.46666666865348816
```

La grosse différence entre les performances des modèles pour les trois ensembles (entraînement, validation et test) nous fait déjà penser que nous rencontrons un problème de « overfitting ».

Ainsi nous appliquons différentes méthodes **uniquement** au modèle CNN entraîné de zéro pour résoudre ce problème :

- L2 Regularization
- Batch normalization

Ces modifications impactent peu les résultats. De plus le fait que le modèle VGG16 performe aussi mal sur les ensembles de données nous fait penser que le problème vient d'ailleurs, ainsi nous faisons une nouvelle répartition de l'ensemble des données.

### Solution appliquée

Principalement notre hypothèse est que les données des trois différents dossiers sont assez différentes. Autrement dit, un entraînement uniquement sur les données du dossier « kag2 » **ne suffit pas** pour pouvoir bien généraliser aux images des dossiers « test\_crop\_image » et « some\_more\_images ».

Cette hypothèse est renforcée par la précision du « validation\_set » qui semble être bornée entre 0 et 20% peu importe les changements effectués aux modèles.

Ainsi nous prenons **toutes les images** du dataset, et nous les répartissons de façon **aléatoire** dans 3 nouveaux dossiers Shuffle/Test, Shuffle/Train et Shuffle/Validation.

Nous effectuons de nouveau l'entraînement en espérant que :

- Les modèles entraînés seront plus robustes à différents types d'images donc devraient être plus performant sur l'ensemble de test et l'ensemble de validation
- L'ensemble de validation sera plus représentatif du dataset donc **on ne devrait plus avoir** une précision de validation bornée entre 0 et 20%

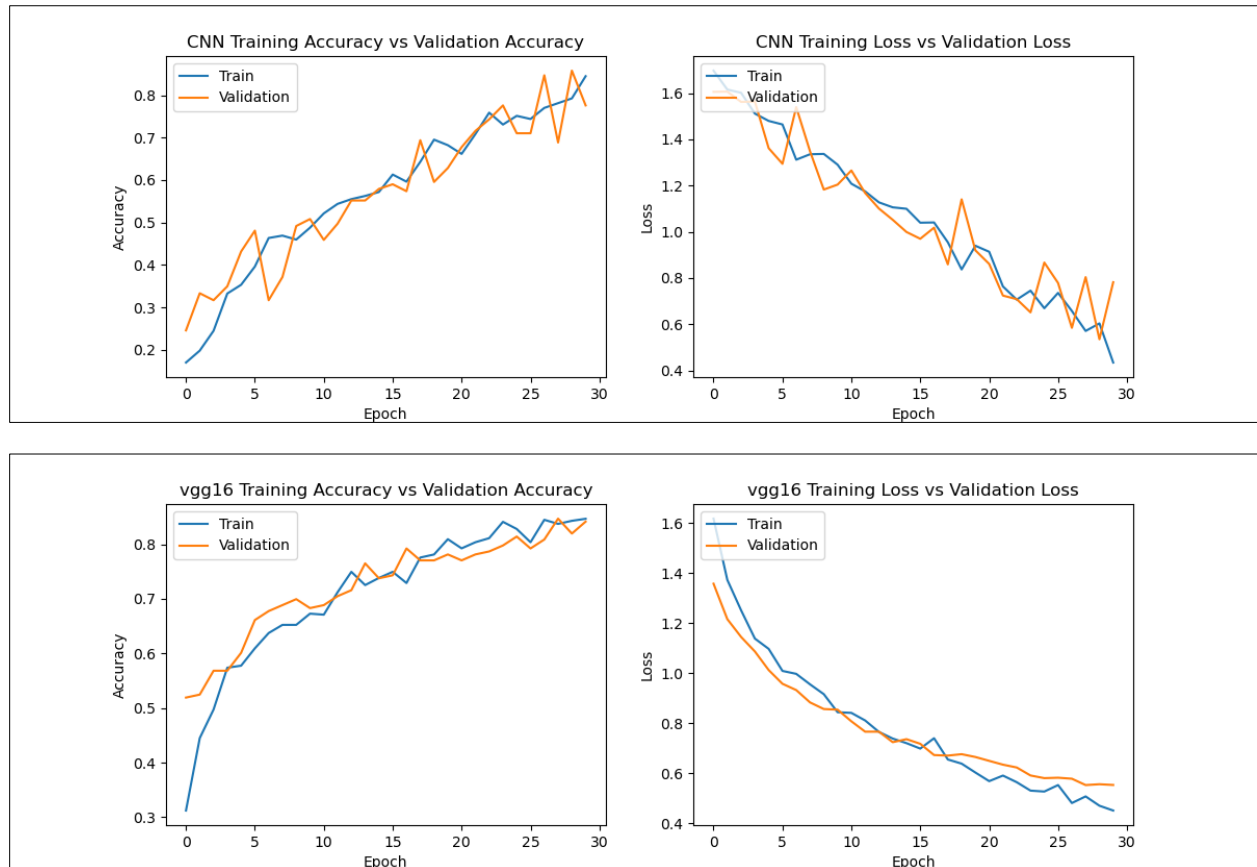


## Nouveau split

En premier nous avons l'ensemble d'entraînement, ensuite l'ensemble de test et en fin l'ensemble de validation.

```
Found 535 images belonging to 5 classes.  
Found 183 images belonging to 5 classes.  
Found 181 images belonging to 5 classes.
```

## Nouveaux résultats obtenus



On constate une forte amélioration des précisions et de pertes sur l'ensemble de validation.

## Comparaison des deux modèles

Les deux modèles ont des résultats similaires en termes de précision d'entraînement et de validation. Le modèle VGG16 semble commencer l'entraînement avec une meilleure précision pour les ensembles d'entraînement et de validation. La courbe d'apprentissage du modèle VGG16, issu du « transfer learning », est plus stable tandis que celle du modèle CNN entraîné de zéro a des variations assez brutes de précision.

## Evaluation de l'ensemble de test

```
Summary of training
6/6 [=====] - 2s 269ms/step
6/6 [=====] - 3s 538ms/step

CNN Cohen's Kappa: 0.72
VG16 Cohen's Kappa: 0.76

CNN Classification Report:
      precision    recall  f1-score   support

     0       0.75      0.67      0.71        36
     1       0.73      0.69      0.71        35
     2       0.83      0.69      0.76        36
     3       0.75      0.83      0.79        36
     4       0.80      0.97      0.88        38

 accuracy          0.77
 macro avg         0.77
weighted avg         0.77

VG16 Classification Report:
      precision    recall  f1-score   support

     0       0.71      0.94      0.81        36
     1       1.00      0.74      0.85        35
     2       0.83      0.69      0.76        36
     3       0.71      0.69      0.70        36
     4       0.86      0.95      0.90        38

 accuracy          0.81
 macro avg         0.82
weighted avg         0.82
```

L'analyse du rapport des modèles montrent que les deux présentent des résultats assez similaires.

Le rapport est intéressant car il donne la précision de chaque modèle pour chaque classe du dataset.

Pour les deux modèles la classe « 4 », Wheat, a la meilleure précision ce qui nous semble logique car cette plantation a une coloration assez différente des autres.

Les deux modèles ont des bons coefficients de Kappa.

## Difficultés rencontrées

Les difficultés principales rencontrées lors de ce travail sont à deux niveaux :

### 1. Elaboration de l'architecture CNN

Sans expérience au préalable avec les CNN, constituer une architecture CNN semblait être une tâche assez mystérieuse. La phase de recherche sur les architectures CNN et les bonnes pratiques à utiliser a permis d'avoir une bonne base pour une première idée d'architecture.

Ainsi nous avons pu mettre en place une architecture qui correspondait bien à ce qui est utilisé actuellement en industrie. Ensuite une phase d'essai a suivi pour mieux ajuster les différents paramètres.

### 2. Utilisation du dataset

La difficulté principale de ce travail était au niveau de la répartition de l'ensemble de données.

- Le fait que les deux modèles entraînés n'arrivaient pas à généraliser les prédictions au niveau de l'ensemble de test
- Les performances faibles des deux modèles sur l'ensemble de validation

L'utilisation des dossiers qui se trouvaient déjà dans le dataset Kaggle pour le « train, validation et test » split semblait être une décision logique qui finalement impactait largement les performances des modèles.

Pour résoudre ce problème nous avons développé différents scripts qui nous permettait de repartir de façon **aléatoire** et **mesuré** l'ensemble des données. Ainsi nous avons développé deux modèles qui **généralise bien les prédictions** sur l'ensemble de test et qui avait des bonnes performances sur l'ensemble de validation.

## Conclusion

Ce travail nous a permis d'acquérir de l'expérience dans l'utilisation des outils tels que Keras pour faire de l'apprentissage profond. Nous avons exploré différentes méthodes dont l'entraînement d'un CNN de zéro et l'ajustement d'un modèle de DeepLearning déjà entraîné. Ces deux méthodes présentent chacune des avantages et des inconvénients principalement l'apprentissage par transfert permet d'exploiter des outils déjà disponibles et de profiter des modèles qui sont entraînés sur des ensembles de données plus vastes. Tandis qu'entraîner un CNN de zéro nous permet d'avoir plus de liberté dans l'architecture du modèle et de comment nous voulons l'ajuster en fonction du besoin.

Des améliorations possibles au travail réalisé sont les suivants :

- Utilisation de ressources matérielles plus puissantes (tels que GPU, TPU) afin d'accélérer l'entraînement des modèles
- Améliorer le mécanisme de répartition des données afin que le « train, test, validation split » soit effectué de façon automatique à chaque fois que l'entraînement est lancé
- Effectuer un meilleur prétraitement des données afin de s'assurer qu'on a pu éviter des fuites entre les différents ensembles : validation, test et entraînement