

**Ecole Polytechnique de Ouagadougou**

**………………………………………**

**Institut du génie informatique et des télécommunications**

**………………………………………**

**Réseaux de neurones et apprentissage profond**

**Projet de Deep Learning : Classification des**

**Fleurs avec un Réseau de Neurones Convolutif**

**(CNN)**

**Membres du groupe : Enseignant : Dr TIEMOUNOU**

THIOMBIANO K.R Eulalie

Table des matières

[Table des illustrations 3](#_Toc176213950)

[Introduction 4](#_Toc176213951)

[Téléchargement et exploration de la dataset 5](#_Toc176213952)

[Prétraitement des images 7](#_Toc176213953)

[Division les données 7](#_Toc176213954)

[Construction du modèle 8](#_Toc176213955)

[Pytorch 9](#_Toc176213956)

[Entrainement du modèle 9](#_Toc176213957)

[Evaluation des modèles 10](#_Toc176213958)

[Amélioration et optimisation 12](#_Toc176213959)

[Utilisation de technique avancé VGG16 13](#_Toc176213960)

[Récapitulatif Pytorch 15](#_Toc176213961)

[Tensorflow 15](#_Toc176213962)

[Resultats et discussion 18](#_Toc176213963)

[Conclusion 18](#_Toc176213964)

# Table des illustrations

[Figure 1: Echantillon de fleur par catégorie 5](#_Toc176213888)

[Figure 2: Répartition des fleurs par classe 6](#_Toc176213889)

[Figure 3: Distribution des largeurs et hauteurs des images 6](#_Toc176213890)

[Figure 4: Code normalisation et rédimensionnement 7](#_Toc176213891)

[Figure 5: Séparation des données 8](#_Toc176213892)

[Figure 6: Construction CNN simple 8](#_Toc176213893)

[Figure 7: couche convolutionnelle et couche de pooling supplémentaires 8](#_Toc176213894)

[Figure 8: CNN avec couches fully-connected 9](#_Toc176213895)

[Figure 9: CNN avec dropout 9](#_Toc176213896)

[Figure 10: code entrainement 10](#_Toc176213897)

[Figure 11: code de validation 10](#_Toc176213898)

[Figure 12: Hyperparamètres et configurations de modèles 10](#_Toc176213899)

[Figure 13: Evaluation des modèles 11](#_Toc176213900)

[Figure 14: Résultat par architecture 11](#_Toc176213901)

[Figure 15:Augmentation des données 12](#_Toc176213902)

[Figure 16: Echantillon augmenté 13](#_Toc176213903)

[Figure 17: Métriques après augmentation des données 13](#_Toc176213904)

[Figure 18: Métriques par catégories avec CNNWithMoreFC 13](#_Toc176213905)

[Figure 19: Code pour charger VGG16 14](#_Toc176213906)

[Figure 20: Téléchargement de VGG16 en cours 14](#_Toc176213907)

[Figure 21: VGG score 14](#_Toc176213908)

[Figure 22: CNN simple 15](#_Toc176213909)

[Figure 23: CNN avec dropout 16](#_Toc176213910)

[Figure 24: Avec augmentation de données 16](#_Toc176213911)

[Figure 25: Avec Resnet50 17](#_Toc176213912)

[Figure 26: Association classes entier 17](#_Toc176213913)

# Introduction

L'identification automatique des espèces de plantes à partir d'images est une application significative de la vision par ordinateur qui a des implications importantes dans des domaines tels que la biologie, l'écologie et la conservation de la biodiversité. Avec l'avènement des technologies de deep Learning, la classification d'images a vu des avancées notables, permettant des applications plus précises et efficaces dans le traitement automatique des images naturelles.

Ce projet vise à développer un modèle de deep Learning basé sur les réseaux de neurones convolutifs (CNN) pour classifier des images de différentes espèces de fleurs. Utilisant la dataset Flowers Recognition disponible sur Kaggle, ce travail explore la capacité des CNNs à apprendre des caractéristiques visuelles distinctes des fleurs pour automatiser leur classification. Cette dataset comprend des images de cinq catégories de fleurs : marguerites, pissenlits, roses, tournesols et tulipes, offrant un ensemble de données diversifié pour entraîner et tester notre modèle.

Dans la suite de ce rapport, nous commencerons par télécharger et explorer la dataset disponible sur Kaggle, ce qui nous permettra de comprendre la distribution et les caractéristiques des images de fleurs. Ce premier pas est crucial pour définir les méthodes de prétraitement nécessaires et adapter nos techniques de deep learning aux spécificités des données. Nous procéderons ensuite au prétraitement des images. Cette étape comprend le redimensionnement et la normalisation des images pour uniformiser les entrées du modèle. L'importance de cette phase réside dans sa capacité à influencer directement l'efficacité de l'apprentissage du modèle. La construction du modèle CNN sera détaillée, exposant les choix d'architecture des réseaux de neurones convolutifs utilisés. Nous explorerons différentes configurations pour déterminer la plus efficace en termes de précision et de généralisabilité. L'entraînement du modèle sera minutieusement expliqué, où nous ajusterons les hyperparamètres et utiliserons l'ensemble de validation pour éviter le surapprentissage. Cette phase est essentielle pour affiner les performances du modèle. Nous évaluerons ensuite les performances du modèle à travers diverses métriques comme la précision, le rappel, et le score F1. L'analyse des résultats nous aidera à comprendre les forces et les faiblesses de notre approche. En complément, nous intégrerons des techniques d'augmentation des données pour améliorer la robustesse du modèle. Cette méthode permettra d'augmenter artificiellement la variété des données d'entraînement sans collecter de nouvelles images, en appliquant des transformations comme la rotation, le zoom et le retournement horizontal des images. Nous explorerons également l'utilisation de VGG16, un modèle de réseau de neurones convolutif pré-entraîné. En adoptant une approche de transfert d'apprentissage, nous ajusterons VGG16 pour notre tâche spécifique de classification de fleurs, ce qui devrait permettre d'atteindre une précision accrue grâce à des caractéristiques visuelles déjà apprises sur de vastes datasets.

# Téléchargement et exploration de la dataset

La dataset des fleurs que nous apprennons à la machine à distinguer selon leurs caractéristiques visuelles se trouve sur kaggle nous avons télécharger le fichier zipper d’une taille de 236 Mo nous avons ensuite dézippé le fichier dans notre répertoire afin de pouvoir l’utiliser.

Afin de prendre connaissance et de comprendre notre dataset nous avons procédé aux étapes suivantes :

* Voir les sous dossiers qui se trouvent dans le dossier dézippé (leur nombre et leurs noms)

Après avoir chargé la base de données nous avons trouvé que dans le dossier nommé flower il y’avait 5 autres sous dossiers nommés daisy, dandelion, rose, sunflower et tulip.

* Voir le nombre de classes de fleurs et leurs noms

Il y’a 5 catégories de fleurs du même nom que les dossiers c’est-à-dire daisy, dandelion, rose, sunflower et tulip.

* Connaitre le nombre total de fleurs

Le nombre total de fleurs dont nous disposons est de 4317 fleurs.

* Afficher un échantillon de fleurs

Nous avons affiché 5 fleurs par catégories pour prendre connaissance du contenu de chaque classe.

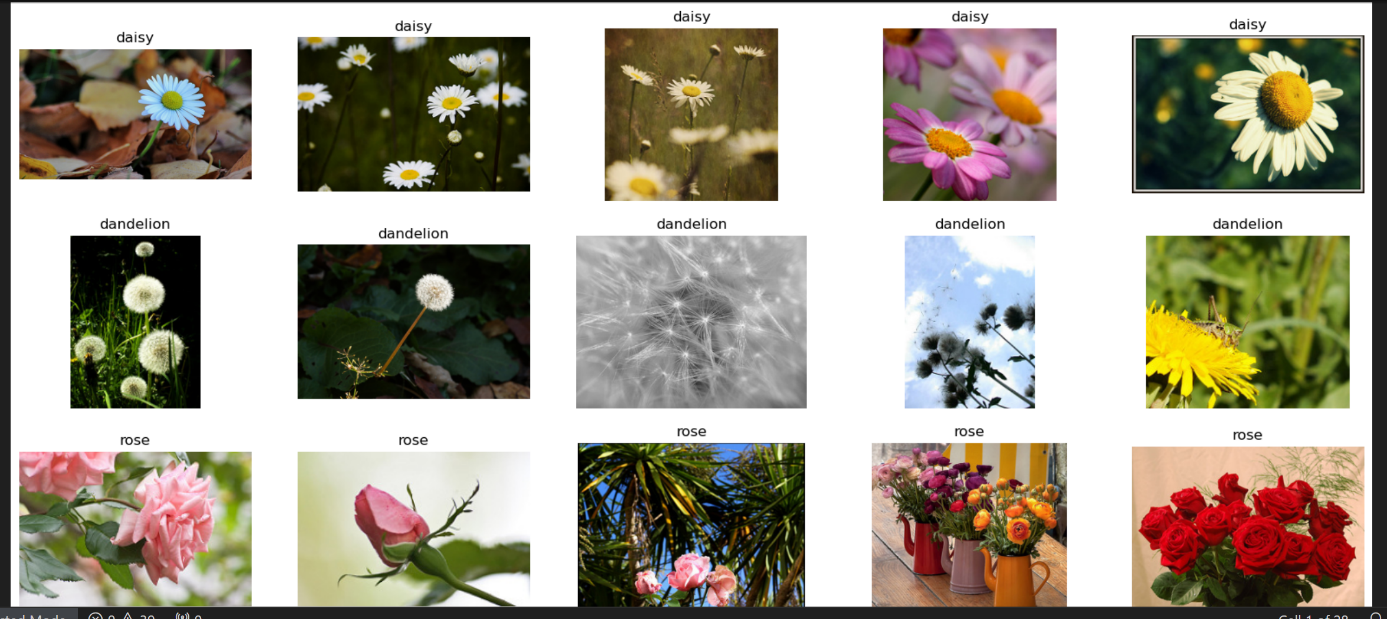


Figure 1: Echantillon de fleur par catégorie

* Connaitre pour chaque catégorie de fleurs le nombre

Les différences de nombre entre les classes ne sont pas extrêmement grandes, mais elles sont notables. Par exemple, il y a une différence de 319 spécimens entre la classe la plus représentée (dandelion) et la moins représentée (sunflower).

Les valeurs pour chaque catégorie sont consignées dans l’histogramme de la figure ci-dessous :

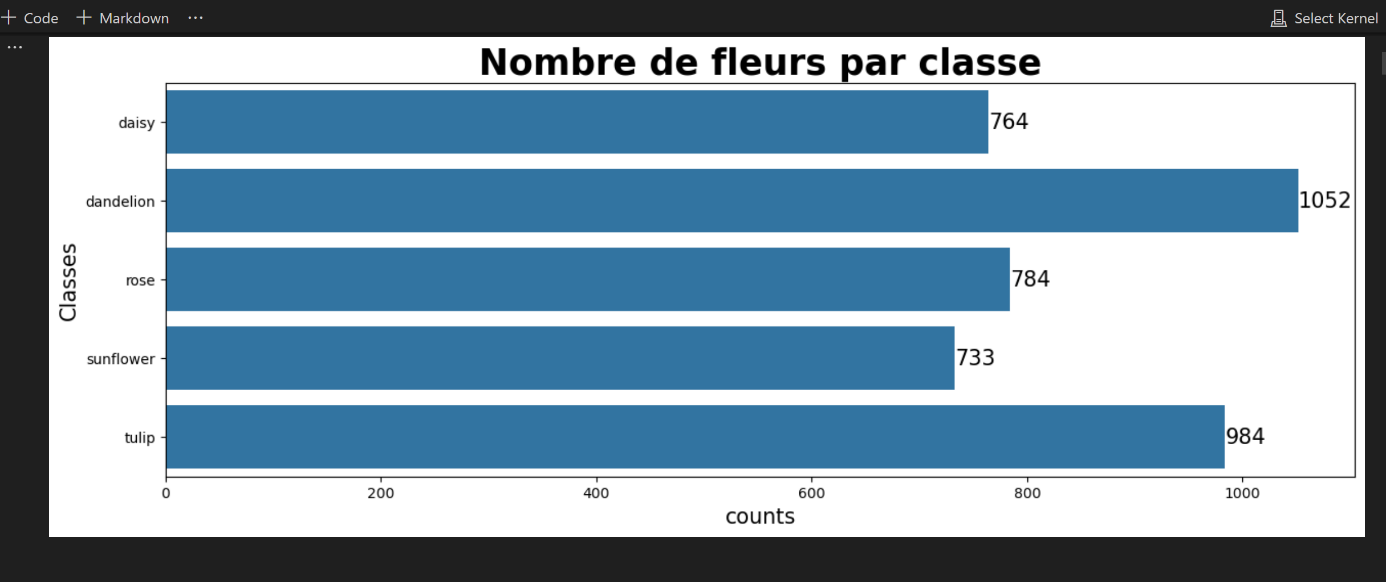


Figure 2: Répartition des fleurs par classe

* Connaitre la taille des fleurs dans la dataset

Pour entrainer un modèle CNN la taille des données est importante nous avons donc essayé de visualiser les longueurs et les largeurs des fleurs. C’est ce que montre la figure suivante.

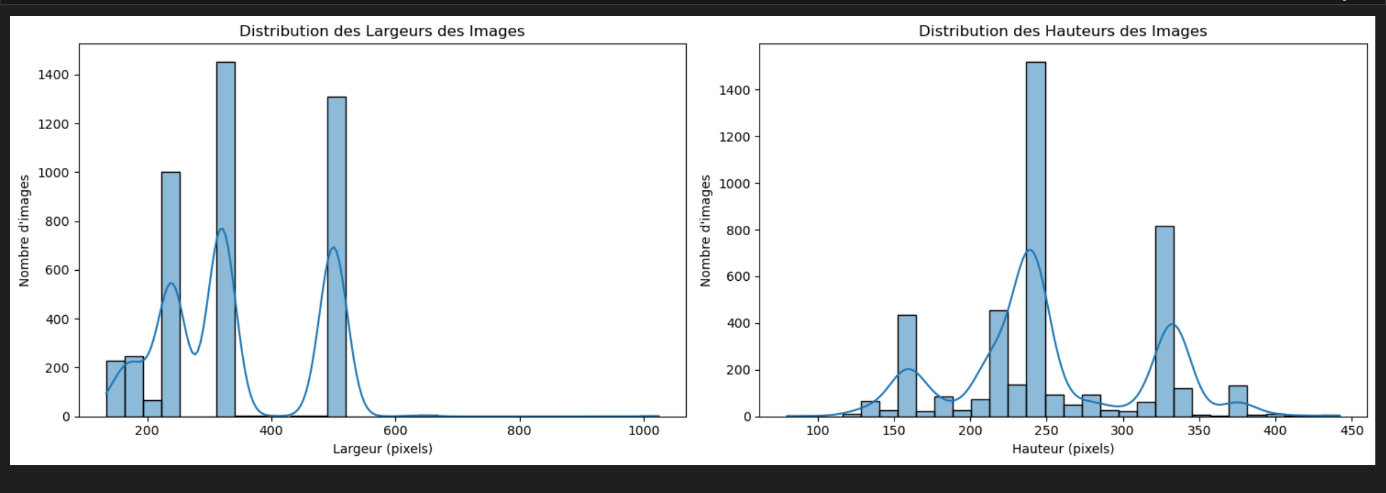


Figure 3: Distribution des largeurs et hauteurs des images

De cette figure on a que les largeurs des images sont principalement concentrées autour de quelques valeurs spécifiques, avec des pics notables aux alentours de 250, 450, et 600 pixels. La majorité des images ont des largeurs comprises entre 200 et 500 pixels.

Et que les hauteurs des images montrent une concentration similaire, avec des pics prononcés autour de 150, 250, et 350 pixels. La majorité des images ont des hauteurs comprises entre 100 et 300 pixels.

En résumé nous avons appris que la dataset contient 5 sous dossiers dont les noms correspondent au nom des classes de fleurs ; que les fleurs sont au nombre de 4317 avec une répartition inégale marqué par une différence de 319 éléments entre la catégorie la plus représentée(dandelion) et la catégorie la moins représentée (sunflower) ; que les fleurs ont des largeurs majoritairement comprise entre 200 et 500 pixels et des hauteurs comprises entre 100 et 300 pixels.

# Prétraitement des images

Pour prétraiter les images, nous avons effectué les opérations suivantes :

* Redimensionner les images à une taille commune.

En analysant les distributions des largeurs et hauteurs des images, on constate que la majorité des images ont une largeur comprise entre 200 et 500 pixels et une hauteur comprise entre 100 et 350 pixels. Pour simplifier l'entrée pour un CNN, il est courant de redimensionner toutes les images à une taille uniforme, souvent une taille carrée comme 128x128, 224x224, ou 256x256 pixels. Nous allons choisir 224x224 pixels pour cet exemple, car c'est une taille standard utilisée dans de nombreux modèles pré-entraînés.

* Normaliser les pixels des images

Normaliser les pixels des images pour que leurs valeurs soient comprises entre 0 et 1 présente plusieurs avantages importants lorsqu'on travaille avec des réseaux de neurones, notamment les réseaux de neurones convolutifs (CNN). La normalisation des pixels permet de réduire les disparités dans les valeurs d'entrée, ce qui aide à stabiliser et à accélérer l'entraînement des réseaux de neurones.

Pour le redimensionnement et la normalisation nous avons utilisé la fonction preprocess\_image comme illustré dans la figure ci-après.

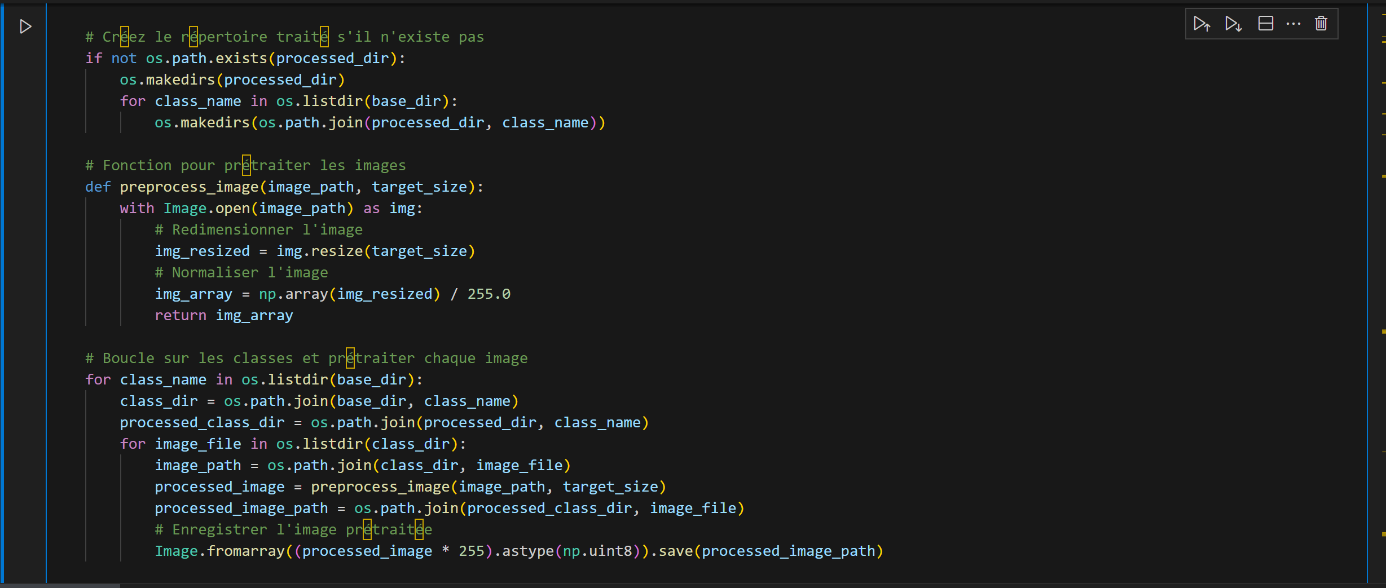


Figure 4: Code normalisation et rédimensionnement

# Division les données

Nous avons divisé les données redimensionnées et normalisées en données d’entrainement, de test et de validation dans les proportions respectives de 70% 15% 15% et nous avons ensuite enregistrées ces données séparées dans de nouveaux dossier comme le mentionne la figure ci-après.

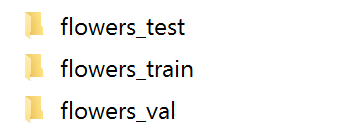


Figure 5: Séparation des données

# Construction du modèle

Pour la construction du modèle nous avons utilisé les bibliothèques de deep learning Pytorch et Tensorflow.

Nous avons installé les dépendances nécessaires, puis nous avons utilisé torchvision pour charger et transformer les images avant de construire le modèle CNN à proprement dit. Le code de construction du modèle est donné dans la figure ci-dessous.

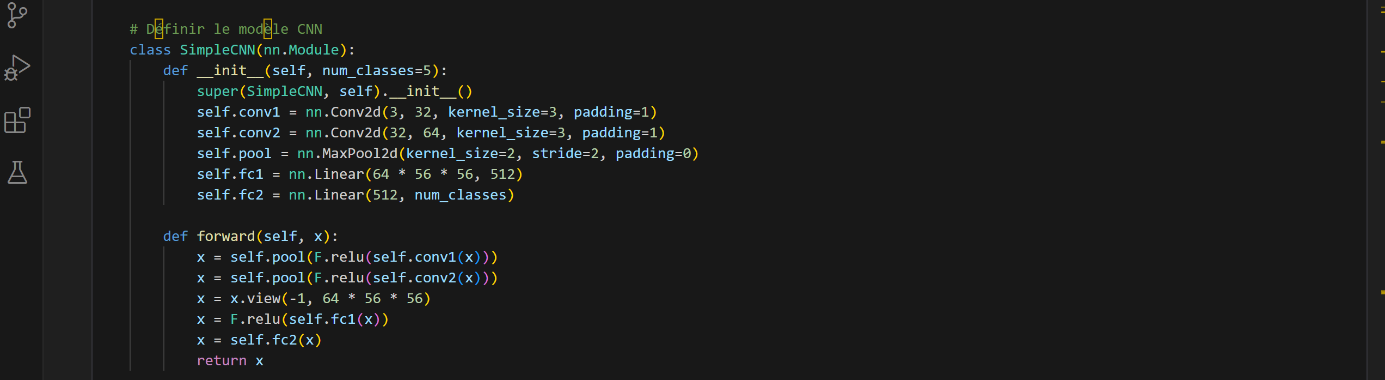


Figure 6: Construction CNN simple

Après la construction de ce modèle nous avons essayé d’autres architecture du CNN en ajoutant plus de couches convolutionnelles, des couches de pooling supplémentaires (figure 7), des couches fully-connected plus profondes (figure 8), et utiliser des techniques comme le dropout (figure 9) pour régulariser le modèle.

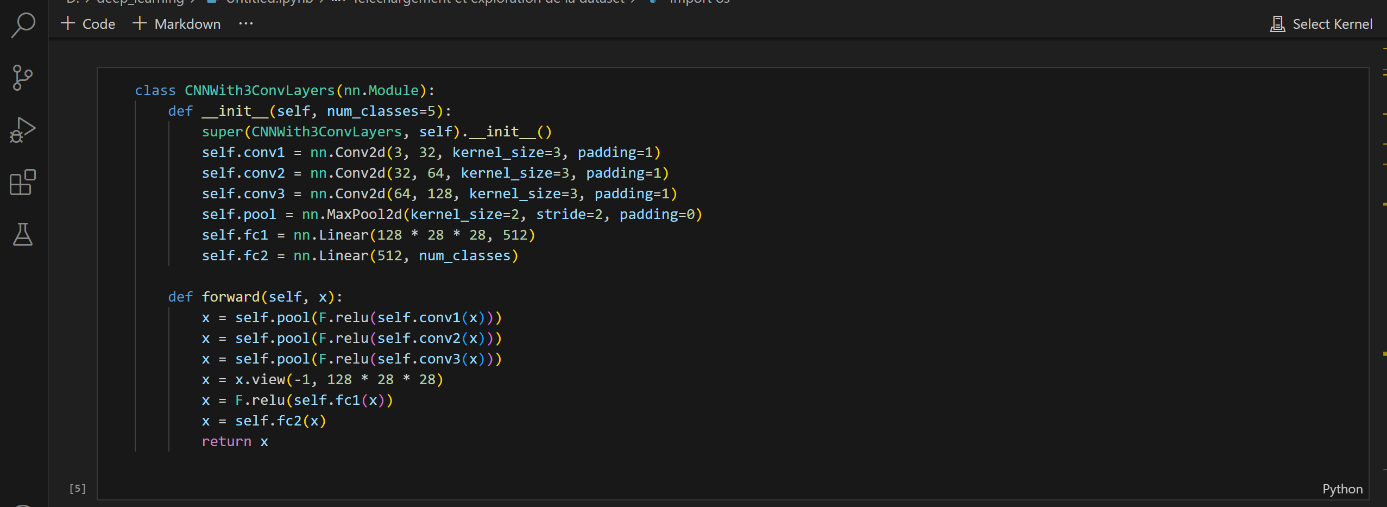


Figure 7: couche convolutionnelle et couche de pooling supplémentaires

Cette architecture comporte trois couches de convolution suivies de deux couches fully connected, augmentant la profondeur du réseau.

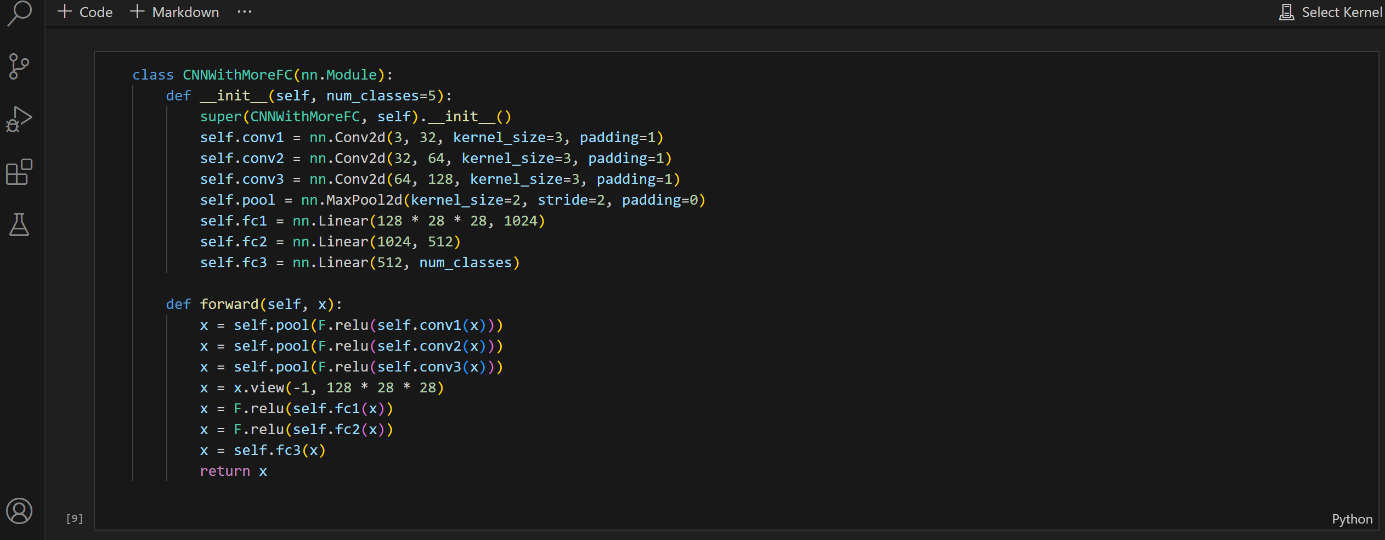


Figure 8: CNN avec couches fully-connected

Cette architecture utilise des couches de convolution pour extraire des caractéristiques des images, suivies de plusieurs couches fully connected pour effectuer la classification finale.

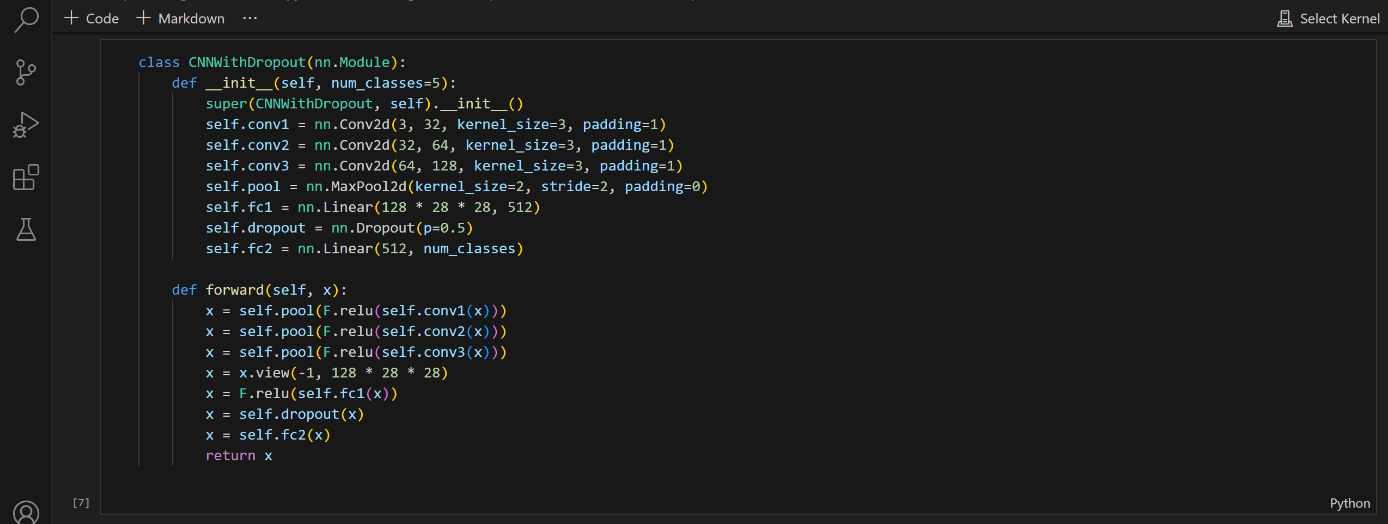


Figure 9: CNN avec dropout

Avec cette architecture, 50% des neurones sont ignorés pendant l'entraînement pour prévenir le sur-apprentissage.

# Pytorch

## Entrainement du modèle

Nous avons entrainé les différents modèles ci-dessus construits sur notre ensemble d’entrainement qui représente 70% de nos données prétraitées (figure 10) et nous avons utilisé l’ensemble de validation qui représente 15% de nos données pour ajuster les hyperparamètres (figure 11).

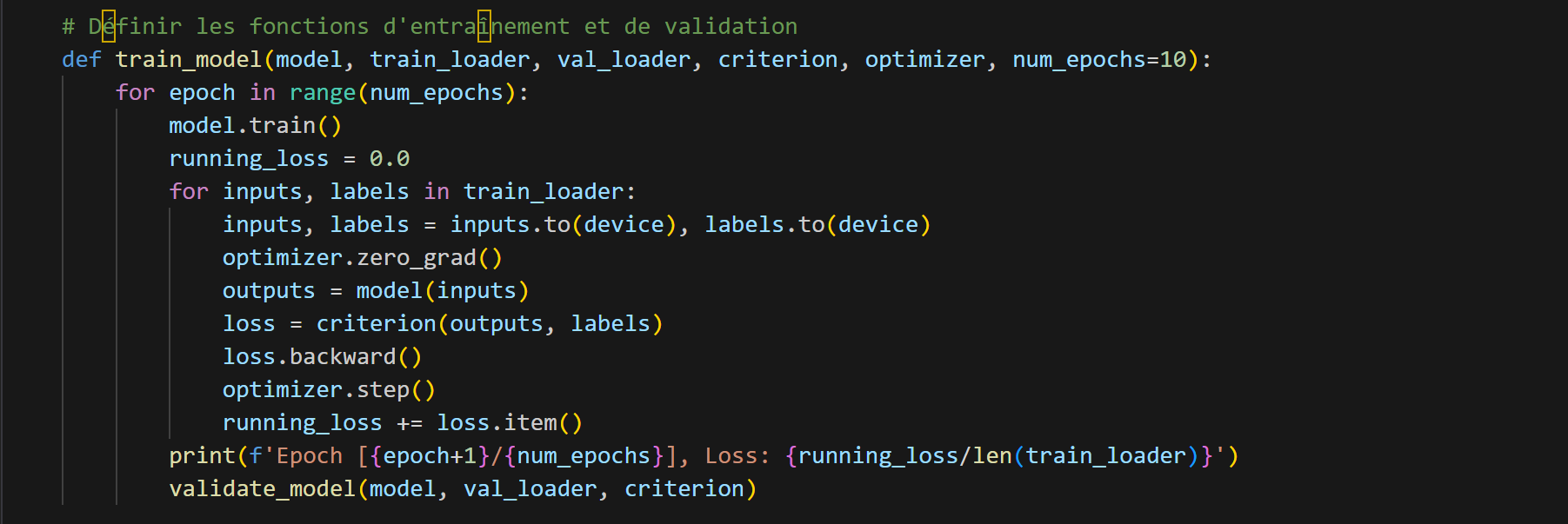


Figure 10: code entrainement

Cette fonction assure l'entraînement du modèle en minimisant la perte et en ajustant les poids du modèle au fil des époques, tout en fournissant un retour sur les performances grâce à la validation.

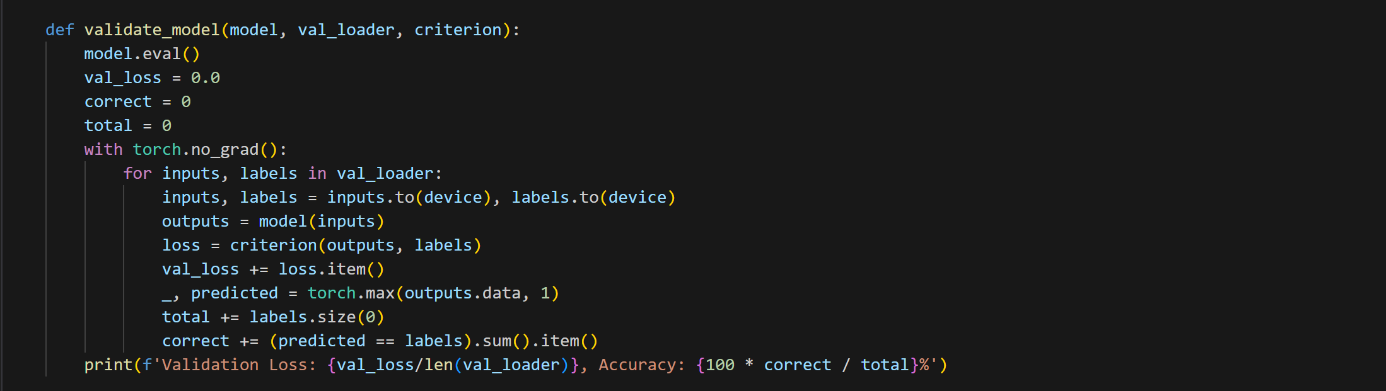


Figure 11: code de validation

Ce code est une fonction de validation pour évaluer les performances du modèle sur les données de validation après chaque époque d'entraînement.

## Evaluation des modèles

Nous avons entrainé notre modèle initial et ses variantes (figure 12), nous allons à présent évaluer les performances (figure 13) de chaque modèle sur l’ensemble de test. Pour cela nous avons écrit une fonction test afin de renvoyer sous forme de tableau les métriques accuracy, precision, rappel et f1-score de chaque architecture (figure 14).

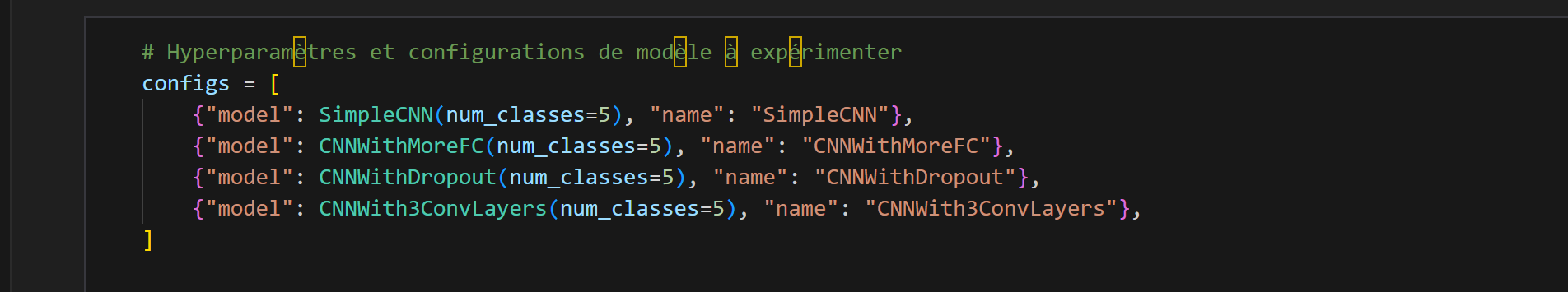


Figure 12: Hyperparamètres et configurations de modèles

Ce code prépare différentes configurations de modèles que nous allons utiliser pour expérimenter et comparer leurs performances. Chaque configuration est associée à un modèle spécifique avec un certain nombre de classes de sortie.

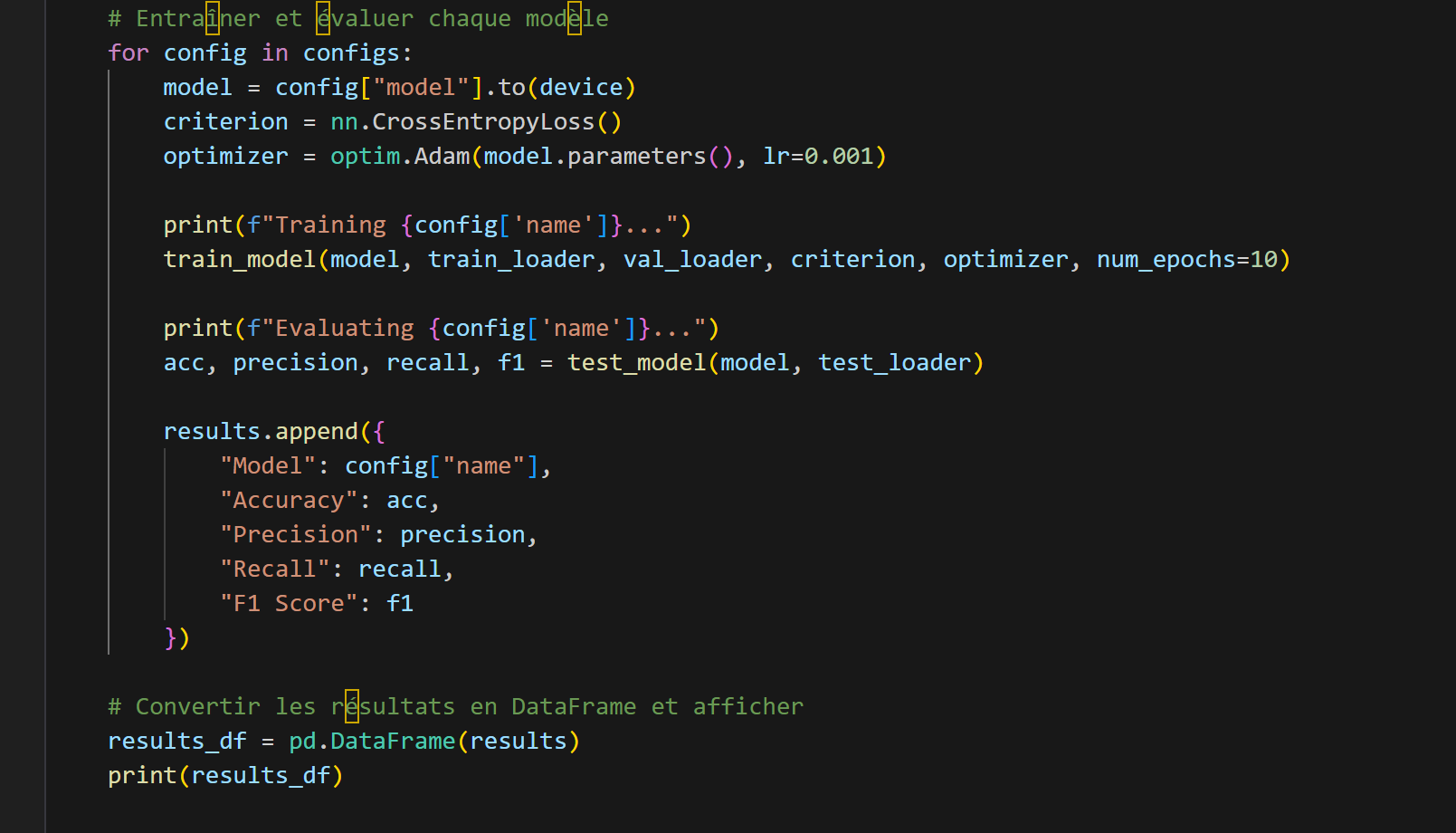


Figure 13: Evaluation des modèles

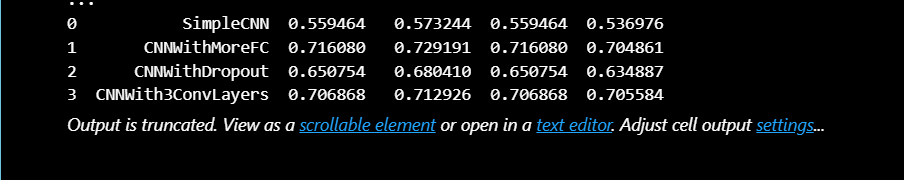


Figure 14: Résultat par architecture

Le modèle SimpleCNN a une performance correcte avec une précision, un rappel et un score F1 d'environ 0.559. C'est le modèle de base avec des performances inférieures par rapport aux autres modèles plus complexes.

Le modèle CNNWithMoreFC avec plus de couches fully connected, a la meilleure performance globale avec une précision, un rappel et un score F1 d'environ 0.716. Cela suggère que l'ajout de couches fully connected améliore la capacité du modèle à capturer des caractéristiques complexes.

Le modèle utilise le dropout pour régularisation et a une performance légèrement inférieure à CNNWithMoreFC mais toujours meilleure que SimpleCNN. Cela montre que le dropout aide à prévenir le sur-apprentissage tout en maintenant de bonnes performances.

Le modèle avec trois couches de convolution a de bonnes performances, supérieures à SimpleCNN mais légèrement inférieures aux modèles avec plus de couches fully connected ou dropout. Cela montre que l'augmentation de la profondeur des couches convolutives aide à extraire des caractéristiques plus détaillées mais pourrait nécessiter des techniques de régularisation supplémentaires pour atteindre les meilleures performances.

## Amélioration et optimisation

Nous avons appliqué la technique d’augmentation des données avec le code de la figure 15 nous avons ensuite affichés quelques fleurs issues de l’augmentation comme dans la figure 16 enfin nous avons réentraîner nos modèles précédents avec ces nouvelles données et les performances des modèles sont consignés dans la figure 17.

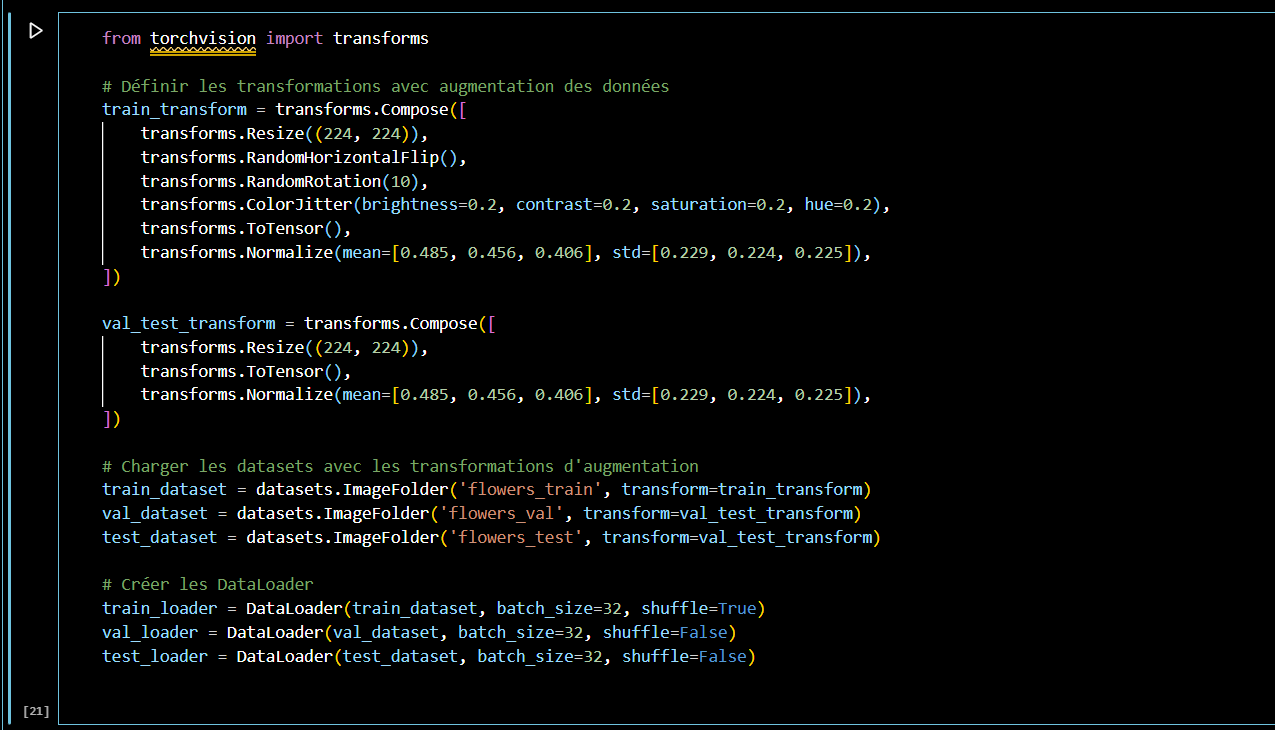


Figure 15:Augmentation des données

Cette image montre l’utilisation de torchvision pour définir des transformations d'augmentation des données sur des images, charger des ensembles de données (`datasets.ImageFolder`), et créer des DataLoaders pour l'entraînement, la validation et le test. Les transformations incluent la redimension des images, la rotation aléatoire, le flip horizontal, la modification de la couleur, et la normalisation. Les DataLoaders sont configurés avec une taille de lot de 32 et sont mélangés pour l'ensemble d'entraînement.

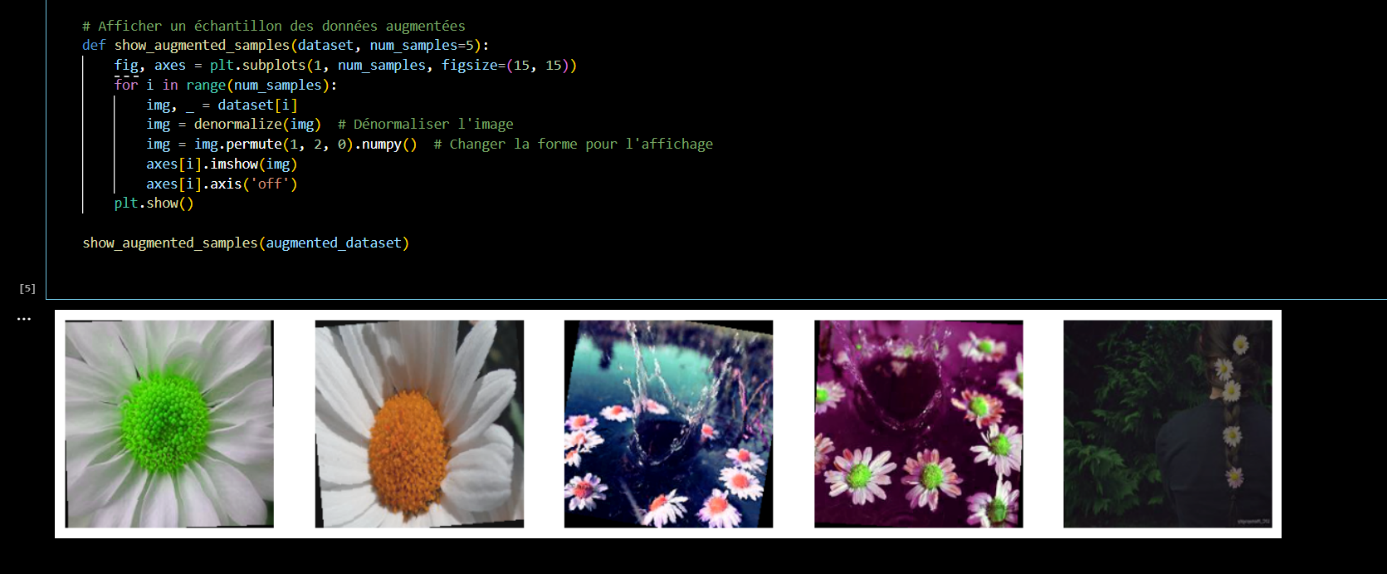


Figure 16: Echantillon augmenté

Les images dans cette figure montrent différentes transformations appliquées, comme des modifications de couleur, des rotations et d'autres ajustements visuels, illustrant ainsi l'augmentation des données pour améliorer l'entraînement d'un modèle.

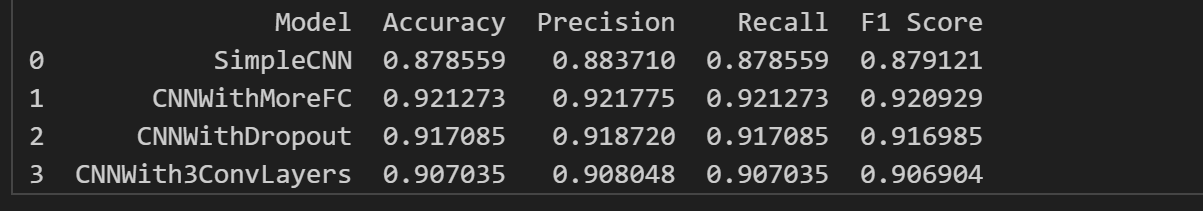


Figure 17: Métriques après augmentation des données

A l’issu de l’augmentation on constate que CNNWithMoreFC a toujours le meilleur accuracy comparé aux autres nous avons donc afficher les performances des fleurs selon le CNNWithMoreFC. Les résultats sont dans la figure 18.

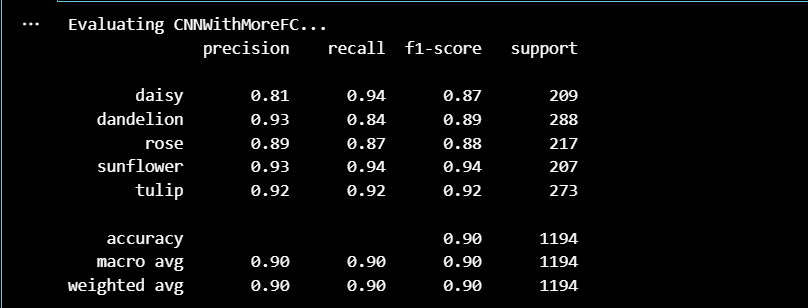


Figure 18: Métriques par catégories avec CNNWithMoreFC

Les résultats montrent des performances relativement élevées, avec des scores F1 variant de 0.87 à 0.94, indiquant une bonne capacité du modèle à classifier correctement les images de fleurs. La classe la mieux classifiée, en se basant sur le score F1, est la "sunflower" (tournesol) avec un score F1 de 0.94. Ce score indique que le modèle a à la fois une haute précision et un rappel élevé pour cette classe, ce qui signifie qu'il est très efficace pour identifier correctement les images de tournesols tout en minimisant les faux positifs et les faux négatifs.

## Utilisation de technique avancé VGG16

Nous constatons qu’avec l’augmentation des données les résultats se sont améliorés. Nous explorerons également l'utilisation de VGG16, un modèle de réseau de neurones convolutif pré-entraîné. Les figures 19 et 20 abordent cela.

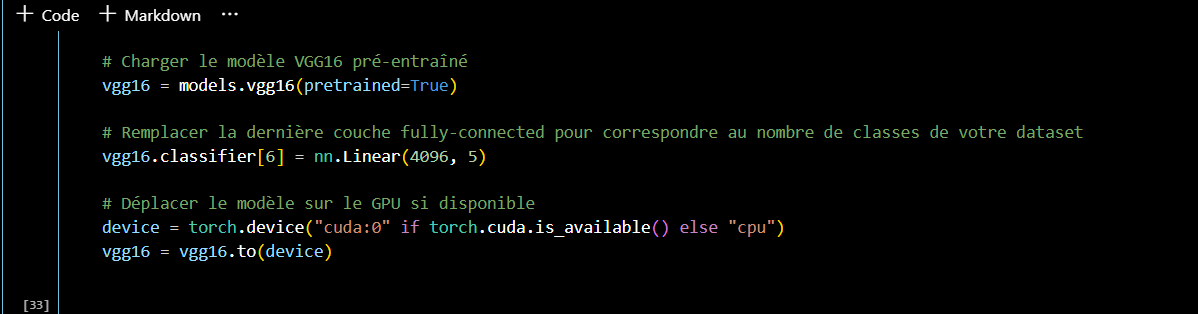
****

Figure 19: Code pour charger VGG16

Dans cette capture de code, nous voyons les étapes pour configurer le modèle VGG16 pré-entraîné afin de l'adapter à un nouveau dataset contenant cinq classes. Le script modifie la dernière couche fully-connected du modèle (vgg16.classifier[6]) pour qu'elle produise des sorties pour cinq classes au lieu des mille classes par défaut du modèle. Le modèle est également configuré pour fonctionner sur un GPU car nous avons installé tensorflow sur le GPU, ce qui permet une accélération significative de l'entraînement et de l'inférence.

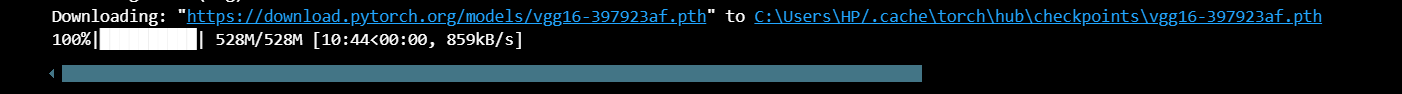


Figure 20: Téléchargement de VGG16 en cours

Cette capture montre le téléchargement du modèle VGG16 pré-entraîné depuis la plateforme PyTorch. Le fichier du modèle, spécifié par son URL unique, est en cours de téléchargement avec une barre de progression indiquant que 528MB sur un total de 528MB ont été téléchargés à une vitesse de 859kB/s. Cela montre comment PyTorch permet d'accéder facilement à des modèles complexes pré-entraînés, ce qui est crucial pour le transfert d'apprentissage.

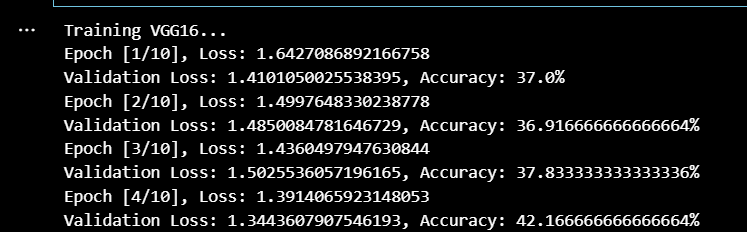


Figure 21: VGG score

Ces résultats montrent une tendance à la baisse de la perte à travers les époques, tandis que la précision s'améliore, ce qui est un signe positif indiquant que le modèle apprend et s'adapte progressivement aux données. Cependant, la précision reste relativement faible comparée à celle obtenue avec l’augmentation des données.

# Récapitulatif Pytorch

En résumé avec pytorch nous obtenons les meilleures performances avec CNNWithMoreFC après l’augmentation des données ce score est de 0.92%. Avec le modèle VGG16 les performances sont encore moins satisfaisantes qu’avec toutes les autres configurations sans augmentation ni VGG16.

# Tensorflow

Avec Tensorflow nous avons procéder aux mêmes étapes avec les mêmes constructions de modèles et nous avons mentionné les résultats du CNN simple et du CNN avec dropout dans les figures 22 et 23.

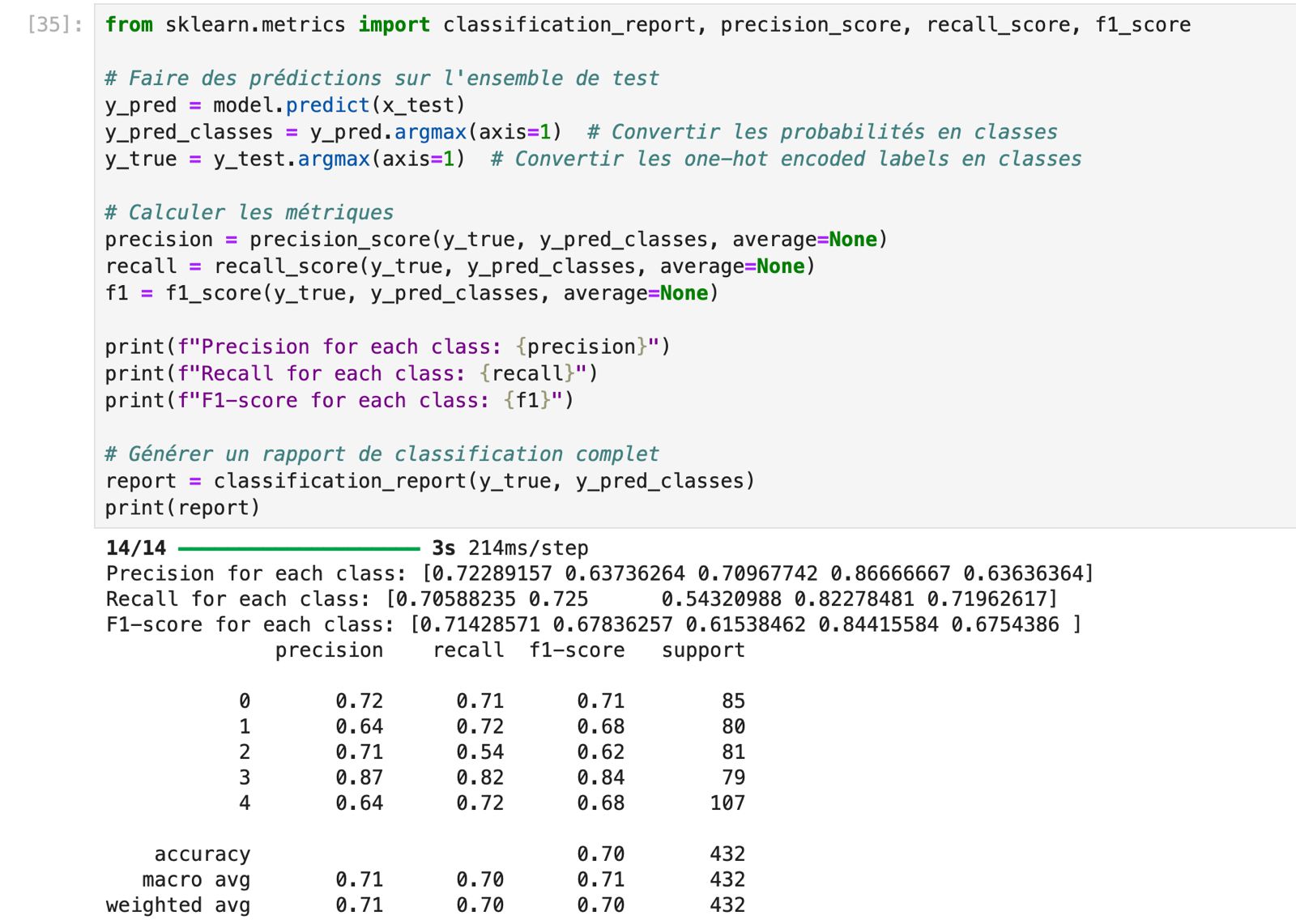


Figure 22: CNN simple

Le code effectue des prédictions sur un ensemble de test, convertit les probabilités en classes discrètes, et calcule les métriques de performance telles que la précision, le rappel, et le score F1 pour chaque classe. Les résultats sont ensuite détaillés dans un rapport de classification complet, affichant les métriques pour chaque classe ainsi que des moyennes globales. Les données montrent des variations dans la performance du modèle par classe, avec une précision globale moyenne autour de 71%.

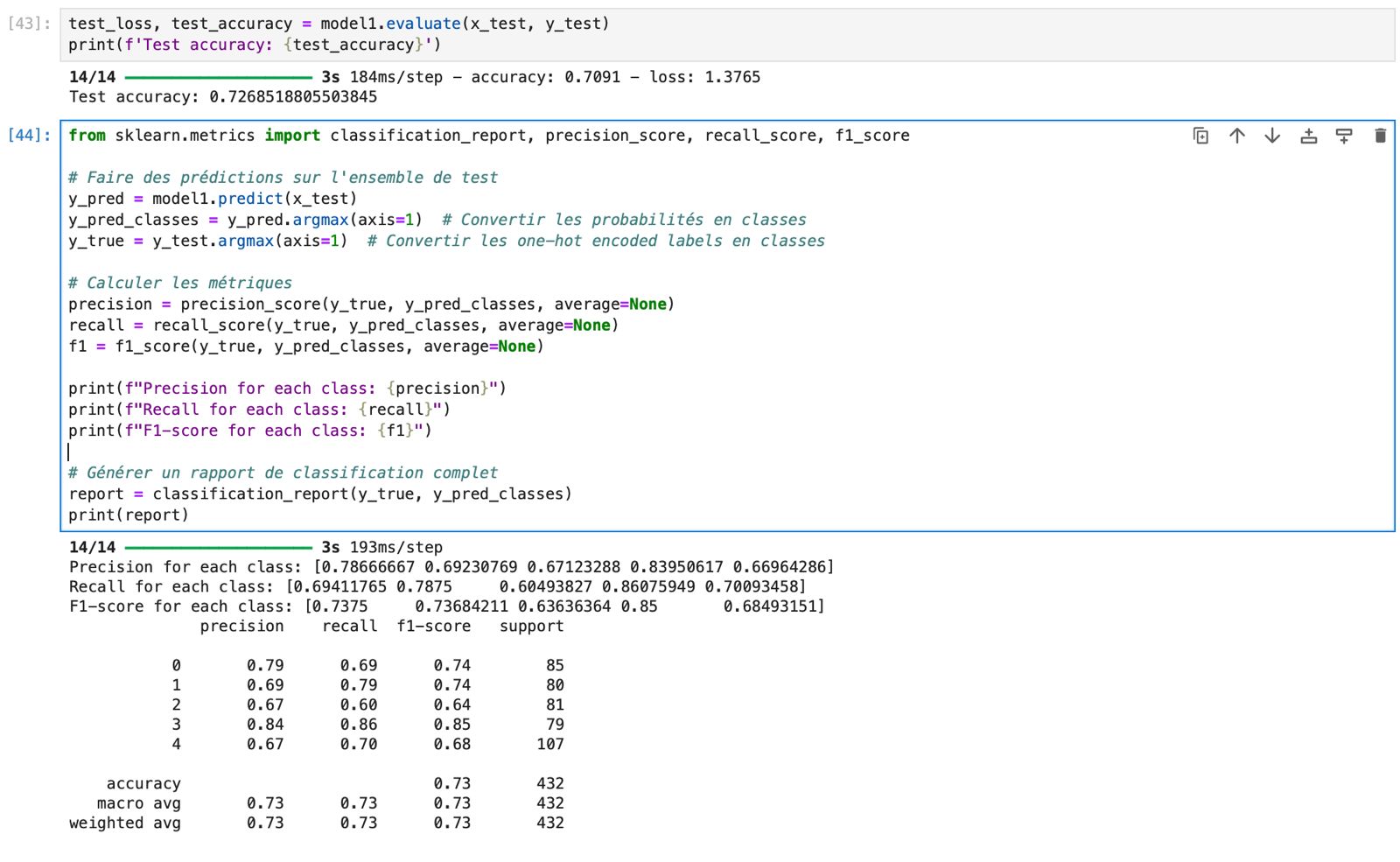


Figure 23: CNN avec dropout

Cette figure montre les métriques de précision, rappel, et score F1 pour chaque classe, ainsi qu'une précision globale de test de 72.68%. Un rapport de classification complet est aussi généré, détaillant les performances pour chaque classe et fournissant des moyennes globales, permettant une évaluation détaillée de la performance du modèle.

Avec Tensorflow nous avons de meilleurs résultats avec le dropout nous avons appliqué l’augmentation des données puis le Resnet50 et les résultats sont consignés dans les figures 24 et 25.

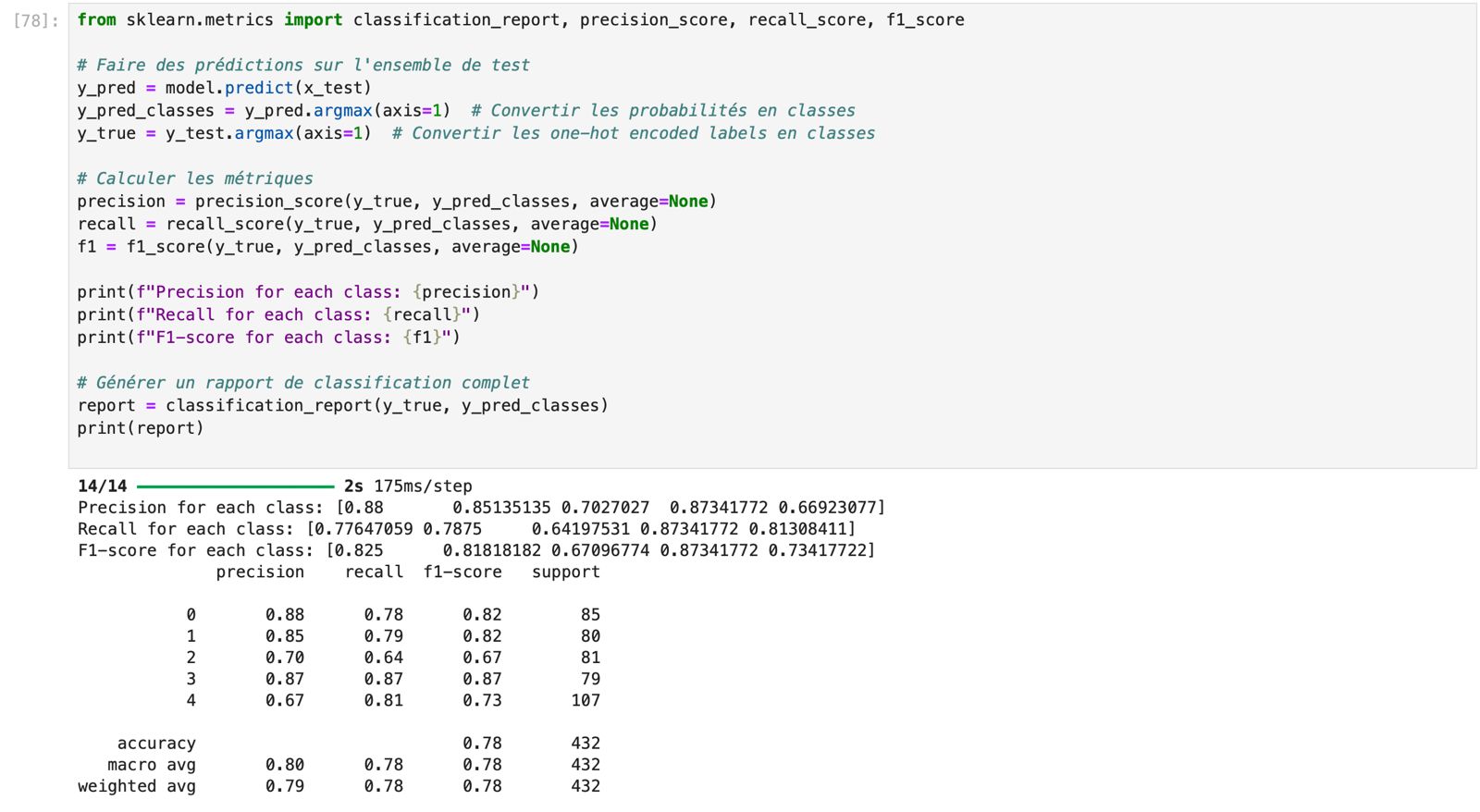


Figure 24: Avec augmentation de données

Le code affiche les métriques de précision, rappel et score F1 pour chaque classe, ainsi qu'un rapport de classification complet incluant ces métriques pour l'ensemble du dataset. Les résultats montrent une précision globale de 80%, un rappel de 78%, et un score F1 de 78%, indiquant une performance relativement équilibrée du modèle sur les différentes classes.

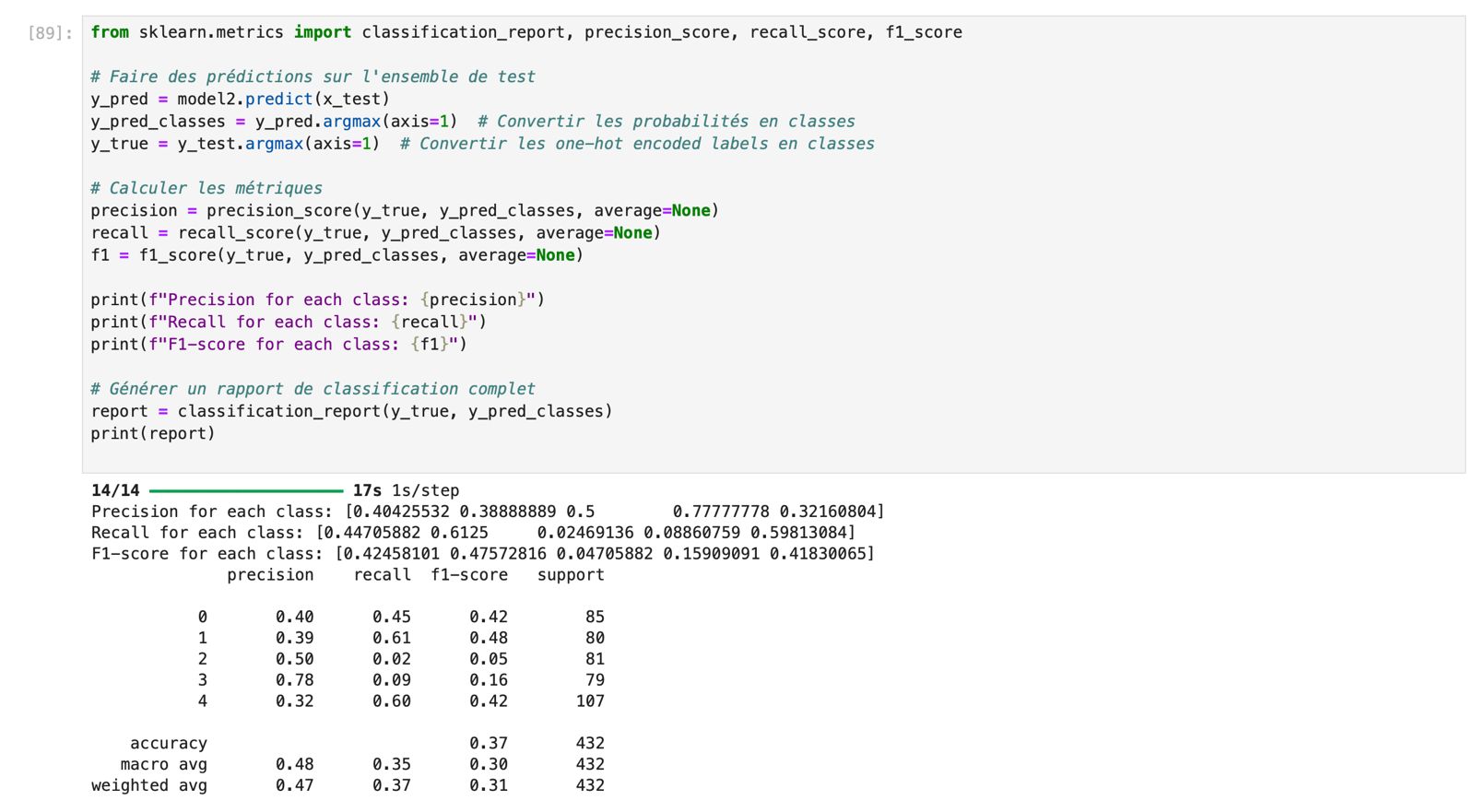


Figure 25: Avec Resnet50

Cette capture d'écran montre l'évaluation d'un modèle de classification où les métriques de précision, rappel et score F1 sont calculées pour chaque classe. Les résultats indiquent une variation significative dans les performances entre les classes, avec une précision générale relativement faible de 48%, un rappel de 35% et un score F1 de 37%.



Figure 26: Association classes entier

# Resultats et discussion

Notre étude a révélé que l'augmentation des données conduit à une amélioration significative des performances des modèles de classification des images de fleurs. En utilisant les frameworks TensorFlow et PyTorch, nous avons constaté que les modèles VGG16 et ResNet50, sans augmentation des données, affichent des performances relativement faibles, avec une précision moyenne d'environ 40%. En revanche, l'introduction de l'augmentation des données a permis d'atteindre des précisions bien supérieures, avec 80% pour TensorFlow et 92% pour PyTorch.

Parmi les différentes catégories de fleurs, la fleur de type "sunflower" (tournesol) a été la mieux classifiée dans toutes les configurations testées, que ce soit avec ou sans augmentation des données, et indépendamment du framework ou du modèle spécifique utilisé. Ceci souligne la robustesse des modèles à reconnaître et classer correctement cette catégorie spécifique.

Le modèle qui a démontré les meilleures performances globales est le CNN avec des couches fully connected supplémentaires (CNNWithMoreFC), particulièrement lorsqu'il est utilisé avec PyTorch et l'augmentation des données. Ce modèle a non seulement excellé dans la classification générale, mais il a aussi obtenu des scores F1 particulièrement élevés pour les fleurs de type "sunflower" et "tulip", avec des scores respectifs de 0.94 et 0.92.

# Conclusion

Ces résultats mettent en évidence l'efficacité de l'augmentation des données comme technique pour améliorer la performance des modèles de deep learning, spécialement dans des applications de vision par ordinateur où la variabilité des données peut être un défi majeur. Ils suggèrent également que des approches ciblées, telles que l'optimisation de l'architecture du réseau, peuvent être particulièrement bénéfiques pour des catégories spécifiques d'objets, comme démontré par la haute précision obtenue pour les tournesols et les tulipes.

Ces découvertes offrent des perspectives intéressantes pour des recherches futures, notamment l'exploration d'autres techniques d'augmentation ou l'ajustement des architectures de réseau pour d'autres catégories moins performantes, afin d'améliorer encore la généralisabilité et l'efficacité des modèles de classification d'images.