Rapport de projet :

Classification d'images CIFAR avec CNN (Convolutional Neural Network)

Deep Learning

par Kévin TANG et Yann VINCENT

Université Claude Bernard Lyon 1

1. Objectif

Notre objectif dans ce projet est d'utiliser des architectures de réseaux de neurones convolutionnels (CNN) pour faire de la classification d'image. Nous avons choisi le dataset CIFAR10 pour commencer. Nous voulons construire un modèle simple par nous même que nous pourrons comparer avec des modèles connus, puis avec des modèles pré entraînés. Et comparer tous ces résultats avec un autre dataset plus complexe, CIFAR100 dans notre cas.

2. Qu'est-ce qu'un réseau de neurones convolutifs (CNN) ?

La conception des Convolutional Neural Networks (CNN) repose sur une architecture en couches sophistiquée qui permet l'extraction et la hiérarchisation automatique de caractéristiques à partir de données structurées en grille, principalement des images.
Cette section approfondit la compréhension des différentes couches qui composent un CNN et explore comment ces couches interagissent pour créer des représentations complexes et informatives.

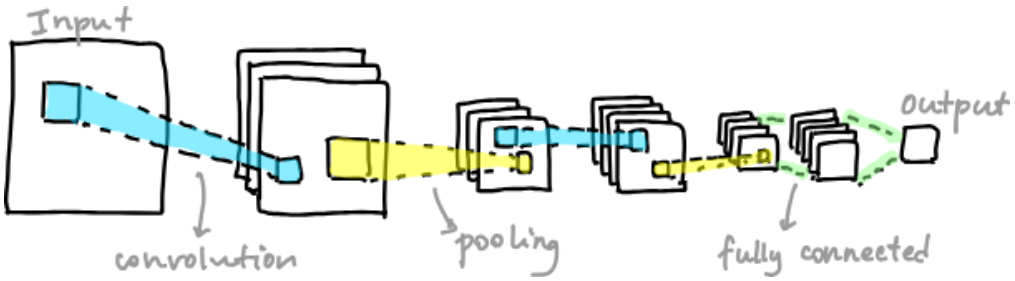

Figure 1 : Illustration de l'architecture d'un CNN

2.1 Couche d'entrée

La première étape d'un CNN est la couche d'entrée, chargée de recevoir les données brutes telles que des images. Chaque pixel est traité comme une caractéristique d'entrée. L'importance de cette couche réside dans sa capacité à capturer les détails initiaux des données, créant ainsi une base pour l'extraction ultérieure de caractéristiques.

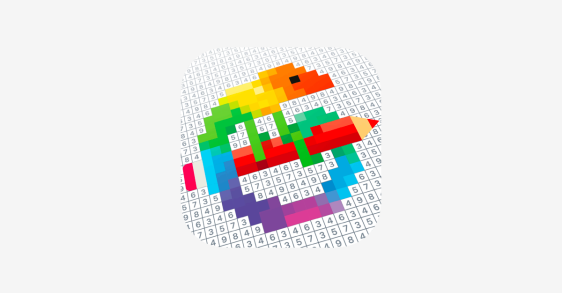

Figure 2 : Illustration des pixels d'une image

2.2 Couche convolutionnelle

Le concept central des CNN repose sur l'utilisation de couches convolutives. Ces couches effectuent des opérations de convolution en appliquant des filtres (kernels) à des parties locales de l'image d'entrée. Ces filtres agissent comme des détecteurs de motifs, permettant au réseau d'apprendre des caractéristiques bas-niveau, telles que des bords, des textures et des formes, au fur et à mesure qu'il progresse dans les différentes couches du réseau.
Les couches de convolution filtrent donc les entrées pour extraire des fonctionnalités spécifiques.

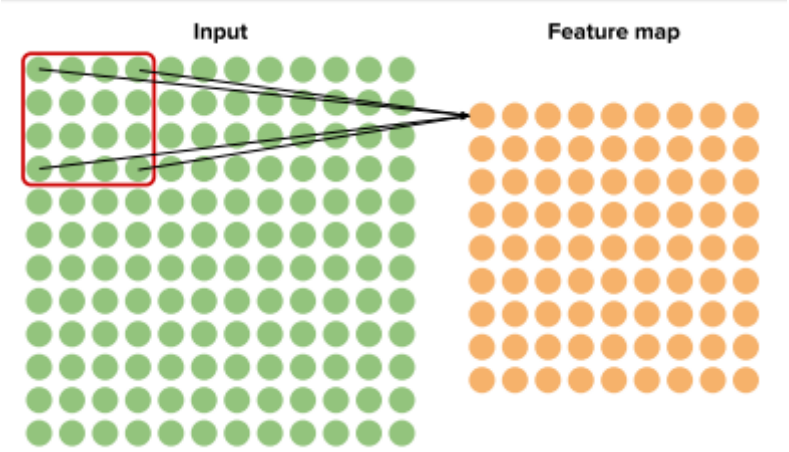

Figure 3 : Illustration du fonctionnement d'une couche convolutive

2.3 Fonction d'activation

Après la convolution, l'introduction d'une fonction d'activation, souvent la fonction ReLU, est essentielle pour insuffler une non-linéarité dans le modèle. Cela permet au réseau de modéliser des relations complexes entre les caractéristiques extraites, une dimension cruciale pour la capture d'informations abstraites.
La fonction ReLU est devenue la pierre angulaire des CNN en raison de sa simplicité et de son efficacité. Mathématiquement, elle est définie comme f(x)=max(0,x), où x représente la sortie de la couche précédente.

2.4 Couche de Pooling

Outre les couches convolutives, les CNN incorporent également des couches de pooling, telles que le max pooling, qui réduisent la dimension spatiale des représentations, préservant les caractéristiques les plus importantes et rendant le modèle plus robuste aux variations. Cela permet ainsi de réduire la quantité d'informations à traiter dans les couches suivantes. Cependant, cet écrémage est effectué de manière à conserver les informations cruciales pour la tâche en cours.

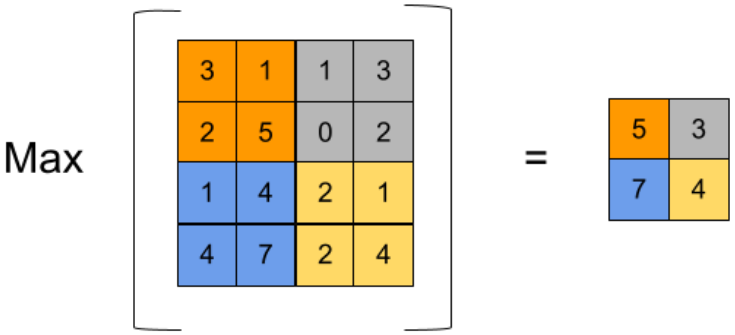

Figure 4 : Exemple du fonctionnement du max pooling

Bien que le max pooling soit largement utilisé, d'autres techniques de pooling, telles que le pooling moyen et le pooling global, offrent des alternatives. Ces méthodes varient dans la manière dont elles agrègent l'information, présentant des compromis entre la préservation de l'information et la réduction dimensionnelle.

2.5 Couches entièrement connectées

Les couches entièrement connectées marquent une étape cruciale dans l'architecture des Convolutional Neural Networks (CNN). Après que les couches convolutives ont extrait des caractéristiques spatiales, les couches entièrement connectées synthétisent ces informations pour des tâches plus complexes telles que la classification.

Les caractéristiques extraites par les couches convolutives sont spatialement dispersées. Les couches entièrement connectées, en revanche, rassemblent ces informations de manière globale. Chaque neurone dans une couche entièrement connectée est connecté à tous les neurones de la couche précédente, créant ainsi une représentation agrégée des caractéristiques spatiales.

Ces couches sont souvent utilisées pour des tâches de classification, où le modèle doit prendre une décision finale basée sur les caractéristiques extraites. Les connexions globales permettent au réseau d'apprendre des motifs complexes qui contribuent à la prise de décision, allant au-delà des informations locales.

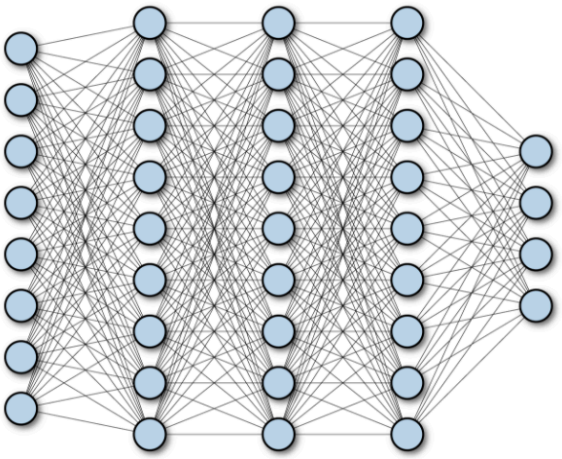

Figure 5 : Illustration de couches entièrement connectées

2.6 Entraînement et apprentissage

L'étape d'entraînement des Convolutional Neural Networks (CNN) constitue un processus crucial où le réseau apprend à ajuster automatiquement ses poids et ses biais pour optimiser sa performance. Cette phase est essentielle pour permettre au modèle de généraliser efficacement à des données non vues auparavant.

Au cœur de l'entraînement des CNN se trouve la technique de rétropropagation du gradient. Lors de cette phase, le modèle est exposé à un ensemble d'exemples étiquetés, et une fonction de coût (ou perte) est définie pour évaluer la différence entre les prédictions du modèle et les étiquettes réelles. La rétropropagation consiste à propager cette erreur en sens inverse à travers le réseau, ajustant ainsi les poids et les biais à l'aide de l'algorithme d'optimisation. Cela permet au modèle d'apprendre à représenter de manière plus précise les caractéristiques discriminantes des données.

L'objectif principal de la rétropropagation est de minimiser la fonction de perte. Les algorithmes d'optimisation, tels que la descente de gradient stochastique (SGD) ou des variantes plus avancées comme Adam, sont utilisés pour ajuster les paramètres du modèle de manière itérative. Cette minimisation permet d'obtenir des poids et des biais qui optimisent la performance du modèle sur les données d'entraînement.

L'entraînement se déroule souvent sur plusieurs passes à travers l'ensemble d'entraînement, chaque passe étant appelée une époque. Pendant chaque époque, le modèle ajuste ses paramètres pour minimiser la fonction de perte sur l'ensemble d'entraînement. Une itération se produit lorsqu'un lot (batch) de données est présenté au modèle, et la rétropropagation est effectuée. La combinaison d'époques et d'itérations permet au modèle de converger vers une représentation optimale des données d'entraînement.

2.7 Conclusion

En résumé, les Convolutional Neural Networks (CNN) sont des architectures sophistiquées qui révolutionnent le traitement des données en grille, en particulier les images. Chaque couche, de l'entrée aux couches entièrement connectées, contribue à l'extraction de caractéristiques essentielles.
L'entraînement des CNN, basé sur la rétropropagation du gradient, optimise automatiquement les poids et les biais pour une meilleure performance. Les algorithmes d'optimisation, comme la descente de gradient stochastique, jouent un rôle crucial dans ce processus.
En se déroulant sur plusieurs époques et itérations, l'entraînement permet au modèle de converger vers une représentation optimale des données. Les CNN ont transformé la vision par ordinateur, ouvrant des perspectives prometteuses pour l'avenir de l'intelligence artificielle, en particulier dans des domaines tels que la reconnaissance d'objets et la segmentation d'images.

3. Les modèles de CNN

Après avoir expérimenté la construction de nos propres modèles, notamment un CNN simple, LeNet, AlexNet et VGGNet, nous avons choisi d'explorer plusieurs architectures de réseaux neuronaux convolutifs renommées. Parmi celles-ci, DenseNet121, DenseNet201, InceptionV3 et ResNet sont des modèles pré-entraînés que nous avons intégrés à nos expérimentations. Notre objectif est d'évaluer et de comparer les performances de ces architectures sur nos jeux de données, apportant ainsi une perspective approfondie à notre analyse.

3.1 CNN Simple

Pour amorcer notre projet, nous avons débuté avec ce modèle de CNN. Cette architecture simple, mais performante, a constitué notre point de départ dans l'exploration des réseaux neuronaux convolutifs. Ce CNN a été choisi pour sa conception simple et sa capacité à fournir des résultats encourageants, le rendant idéal pour les premières phases de notre investigation.
Le modèle prend en entrée des images de dimensions 32x32 pixels avec une représentation en trois canaux de couleur (RGB). Son architecture débute par une couche de convolution comportant 32 filtres, suivie d'une couche de pooling. Ce motif est répété avec 64 filtres. Ensuite, les données sont aplaties pour être soumises à deux couches entièrement connectées (Dense) équipées d'activations ReLU. La dernière couche, comprenant 10 neurones avec une activation softmax, indique que le modèle est adapté à des problèmes de classification impliquant 10 classes distinctes. Ce CNN a été spécifiquement conçu pour la classification d'images de petite taille.

```
{
    'name': 'CNN simple',
    'model': Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        Flatten(),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
},
```

Figure 6 : Implémentation CNN Simple

3.2 LeNet

Proposé par Yann LeCun, Léon Bottou, Yoshua Bengio et Patrick Haffner dans les années 1990, LeNet est considéré comme l'un des premiers CNN réussis. À l'origine destiné à la reconnaissance de chiffres manuscrits, LeNet a introduit des concepts novateurs tels que les couches convolutives et de pooling, jetant ainsi les bases des architectures modernes.

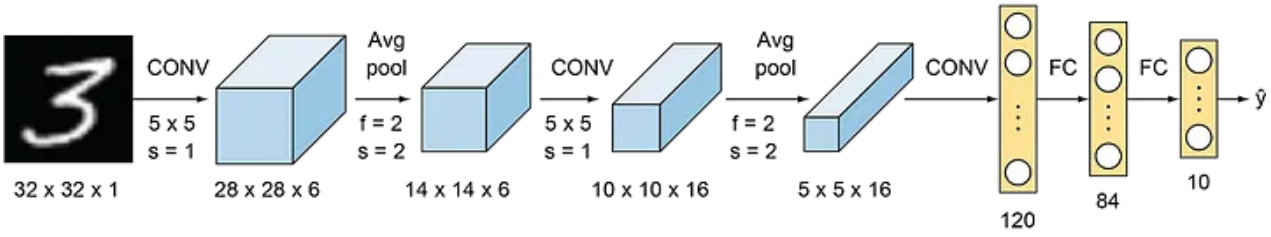

Figure 7 : Architecture LeNet

Le modèle LeNet, conçu initialement pour la reconnaissance de caractères manuscrits, a été adapté à notre projet en modifiant quelques aspects spécifiques. L'implémentation de LeNet (voir le code ci-dessous) comprend des couches convolutives et de pooling successives, suivies de couches entièrement connectées. Les ajustements ont été apportés à la taille des filtres et aux couches finales pour répondre à notre tâche de détection et classification d'images.

```
{
    'name': 'LeNet',
    'model': Sequential([
        Conv2D(6, (5, 5), activation='relu', input_shape=(32, 32, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(16, (5, 5), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(120, activation='relu'),
        Dense(84, activation='relu'),
        Dense(10, activation='softmax')
    ])
},
```

Figure 8 : Implémentation de LeNet

3.3 AlexNet

AlexNet, présenté en 2012 par Alex Krizhevsky, a marqué un tournant majeur en remportant le défi ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Cette architecture plus profonde que LeNet introduit une stratégie de réduction du surajustement via la technique de dropout. AlexNet a contribué à populariser l'utilisation des CNN pour des tâches complexes telles que la classification d'images à grande échelle.

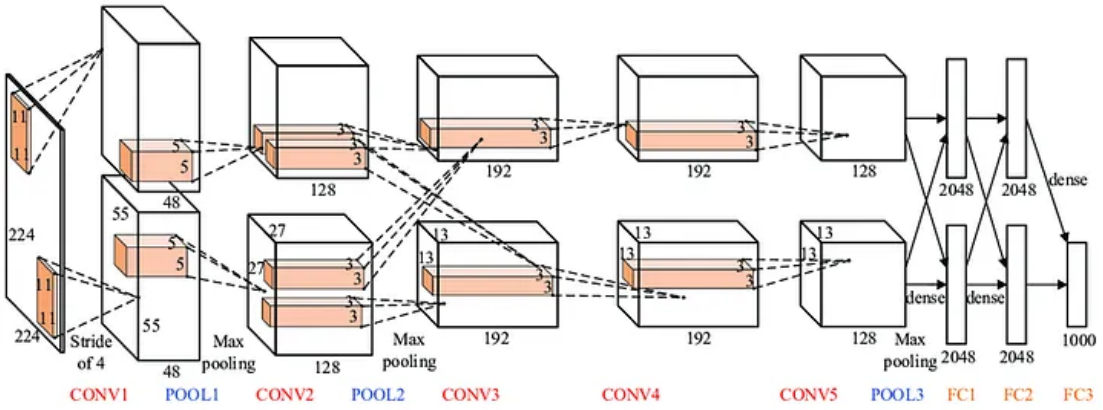

Figure 9 : Architecture AlexNet

AlexNet a été intégré dans notre projet avec des modifications spécifiques. L'implémentation d'AlexNet (voir le code ci-dessous) inclut des concepts tels que la normalisation par lots pour stabiliser l'entraînement et le dropout pour prévenir le surajustement.

```
{
    'name': 'AlexNet',
    'model': Sequential([
        # Première couche de convolution
        Conv2D(96, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        MaxPooling2D((2, 2)),
        BatchNormalization(),

        # Deuxième couche de convolution
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        BatchNormalization(),

        # Troisième couche de convolution
        Conv2D(384, (3, 3), activation='relu', padding='same'),

        # Quatrième couche de convolution
        Conv2D(384, (3, 3), activation='relu', padding='same'),

        # Cinquième couche de convolution
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        BatchNormalization(),

        # Couche complètement connectée
        Flatten(),
        Dense(4096, activation='relu'),
        Dropout(0.5),

        # Deuxième couche complètement connectée
        Dense(4096, activation='relu'),
        Dropout(0.5),

        # Couche de sortie
        Dense(10, activation='softmax')
    ])
},
```

Figure 10 : Implémentation de VGGNet

3.4 VGGNet

Proposé par le Visual Graphics Group (VGG) à l'Université d'Oxford, VGGNet s'est démarqué par sa profondeur remarquable. Avec une architecture uniforme de couches convolutives de petite taille (3x3), VGGNet a démontré que la profondeur du réseau améliore significativement les performances. Cependant, cette profondeur accrue nécessite plus de ressources computationnelles.

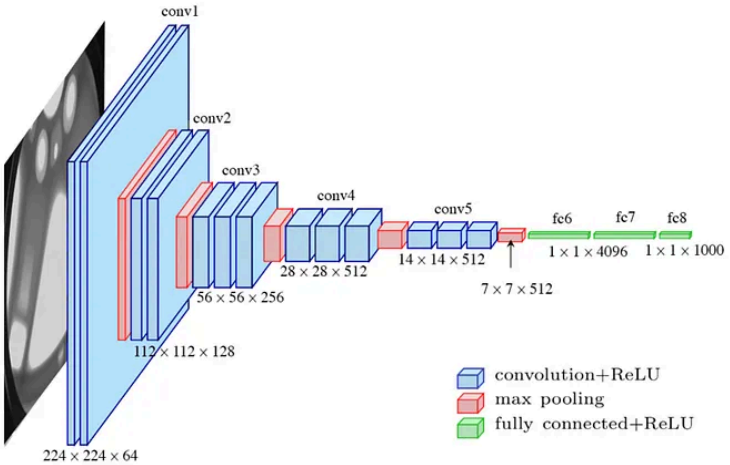

Figure 11 : Architecture VGGNet

VGGNet se caractérise par des couches convolutives profondes avec des filtres de petite taille, facilitant l'entraînement et le transfert de connaissances.

```
{
    'name': 'VGGNet',
    'model': Sequential([
        # Bloc 1
        Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3), padding='same'),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2), strides=(2, 2)),

        # Bloc 2
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2), strides=(2, 2)),

        # Bloc 3
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2), strides=(2, 2)),

        # Bloc 4
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2), strides=(2, 2)),

        # Bloc 5
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2), strides=(2, 2)),

        Flatten(),
        Dense(4096, activation='relu'),
        Dense(4096, activation='relu'),
        Dense(10, activation='softmax')
    ])
},
```

Figure 12 : Implémentation de VGGNet

3.5 DenseNet

Dans le cadre de notre exploration des architectures de réseaux neuronaux convolutifs (CNN), nous avons décidé d'intégrer DenseNet121 à nos expérimentations. Cette architecture, basée sur le concept innovant de connexions denses, promet des avantages significatifs en termes de performances et de généralisation du modèle.

DenseNet121 représente une architecture de réseau neuronal convolutif (CNN) faisant partie de la famille des DenseNets, également connus sous le nom de Densely Connected Convolutional Networks, et possède une architecture de 121 couches.
Ces réseaux ont été conçus pour surmonter les limitations des architectures classiques, telles que la disparition du gradient, en introduisant une connexion dense entre toutes les couches. Concrètement, chaque couche est connectée à toutes les couches précédentes. Contrairement aux architectures traditionnelles où les données circulent de manière linéaire, les DenseNets favorisent un échange direct d'informations à tous les niveaux, facilitant ainsi la propagation du gradient.

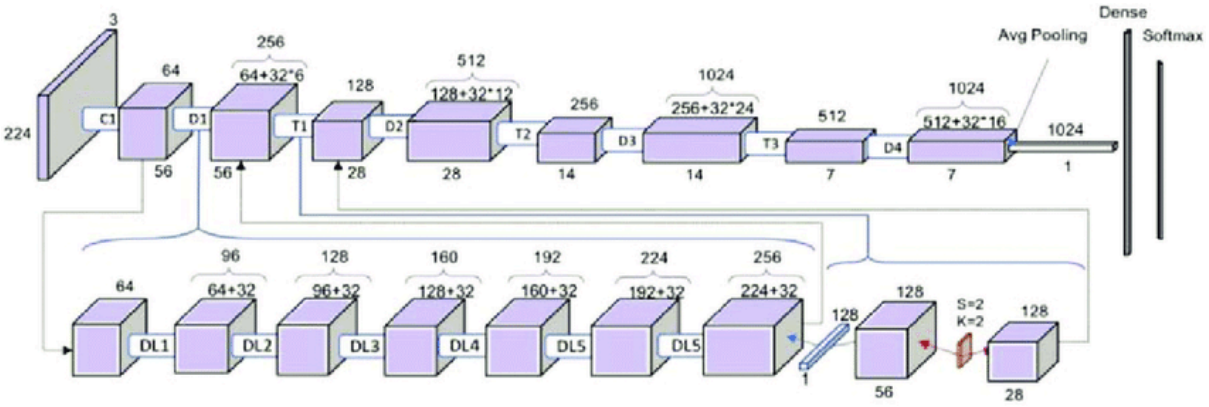

Figure 13 : Architecture DenseNet121

Pour s'adapter à notre ensemble de données spécifique, nous avons ajouté des couches personnalisées, assurant ainsi une correspondance optimale entre DenseNet121 et notre jeu de données.

```
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(100, activation='softmax')(x)
```

Figure 14 : Implémentation de DenseNet121

Dans nos expérimentations, nous avons également évalué l'architecture DenseNet201 en plus de DenseNet121. Cependant, les résultats obtenus n'ont pas présenté de différences significatives, ce qui a conduit à la sélection de DenseNet121 pour notre tâche spécifique de détection et classification d'images.

3.6 ResNet

Dans notre quête d'exploration des architectures de réseaux neuronaux convolutifs (CNN), nous avons pris la décision d'inclure ResNet (Residual Network) à nos expérimentations. ResNet, introduit pour résoudre le problème de la disparition du gradient, propose une structure novatrice basée sur des connexions résiduelles.

ResNet se caractérise par l'utilisation de blocs résiduels qui permettent le saut d'une ou plusieurs couches, facilitant ainsi le flux du gradient pendant la rétropropagation. Cette approche unique a démontré son efficacité en permettant la formation de réseaux beaucoup plus profonds, avec une amélioration significative des performances.

L'intégration de ResNet à nos expérimentations vise à élargir notre compréhension des architectures de CNN et à évaluer ses performances par rapport aux autres modèles que nous avons étudiés. Nous avons également adapté ces architectures en ajoutant des couches personnalisées pour assurer une adéquation optimale avec notre ensemble de données spécifique.

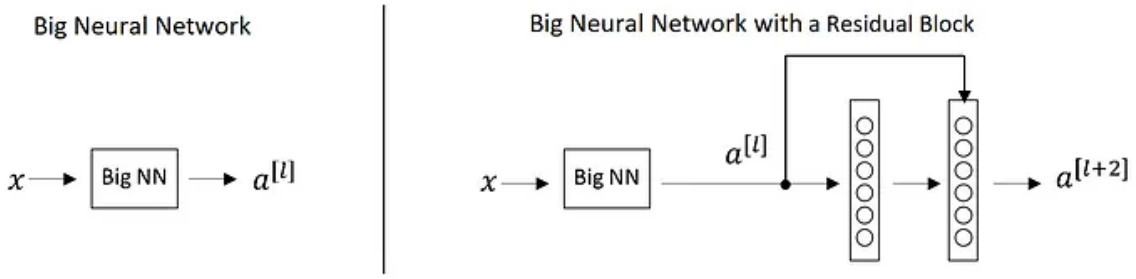

Figure 15 : Architecture ResNet

De même que pour l'architecture précédente, nous avons ajouté des couches personnalisées pour assurer une correspondance optimale entre ResNet et notre jeu de données.

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(100, activation='softmax')(x)
```

Figure 16 : Implémentation ResNet

3.7 InceptionNet

InceptionNet, également connu sous le nom de GoogleNet, est une architecture de réseau neuronal convolutif (CNN) qui se distingue par son approche multibranches innovante. Proposé par l'équipe Google Research en 2014, InceptionNet a remporté le défi ImageNet en introduisant des modules Inception.

L'idée clé derrière InceptionNet est d'exploiter des filtres de convolution de différentes tailles simultanément, permettant ainsi au réseau de capturer des motifs à différentes échelles. Les modules Inception regroupent des convolutions 1x1, 3x3, 5x5, et même des opérations de pooling, puis concatènent les résultats. Cette diversité d'opérations parallèles favorise une extraction de caractéristiques riche et complexe, ce qui en fait un choix intéressant pour des tâches de détection d'objets et de classification d'images.

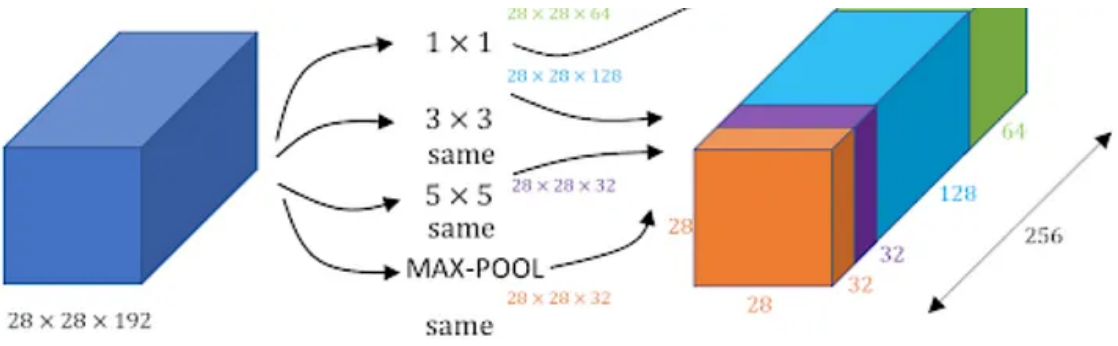

Figure 17 : Architecture InceptionNet

L'implémentation d'InceptionNet pour notre projet consiste en l'utilisation de modules inception au sein de notre architecture. Ces modules ont été adaptés pour s'aligner avec la résolution de nos images et les caractéristiques spécifiques de notre tâche.

```
base_model = InceptionV3(weights = 'imagenet')
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(100, activation='softmax')(x)
```

Figure 18 : Implémentation InceptionNet

3.8 Influence de ces architectures

Le domaine des réseaux neuronaux convolutifs (CNN) a connu une évolution significative, marquée par l'émergence de plusieurs architectures influentes. LeNet, en tant que précurseur, a jeté les bases avec ses couches convolutives et de pooling. AlexNet, par la suite, a introduit la profondeur et des techniques de régularisation telles que la normalisation par lots et le dropout. VGGNet a poussé encore plus loin en démontrant les avantages d'une profondeur accrue.
Les architectures mentionnées ci-dessus ont joué un rôle crucial dans l'évolution des CNN, servant de point de départ à de nombreuses variantes ultérieures. Leurs choix de conception, notamment la profondeur, la taille des filtres, et les techniques de régularisation, ont ouvert la voie à des réseaux plus spécialisés, adaptés à des tâches spécifiques et à différentes ressources disponibles.
Cependant, l'exploration ne s'arrête pas là. Dans le cadre de notre projet, nous avons intégré des architectures modernes telles que DenseNet121, InceptionNet, et ResNet50. Ces ajouts représentent l'évolution continue des CNN, mettant en lumière des approches innovantes pour capturer des caractéristiques complexes dans des images.
En somme, cette diversité d'architectures offre un panorama complet des avancées qui façonnent constamment le domaine de la vision par ordinateur.

4. Application pratique

4.1 Les données

Pour notre projet nous avons utilisé les ensembles de données CIFAR-10 et CIFAR-100.

Le CIFAR-10 comprend 60 000 images couleur de 32x32 pixels réparties en 10 classes, avec 6 000 images par classe. Ces classes comprennent des objets courants tels que des avions, des voitures, des oiseaux, des chats, et d'autres catégories. La diversité des classes et la résolution relativement faible des images rendent le CIFAR-10 un ensemble de

données de référence pour l'apprentissage de la vision par ordinateur sur des images de petite taille.

Le CIFAR-100 est une extension du CIFAR-10, avec 100 classes différentes, chacune comprenant 600 images. Les classes du CIFAR-100 sont plus spécifiques, englobant une gamme plus large de catégories, y compris des sous-catégories artistiques, des insectes, des véhicules inhabituels, etc. L'ajout de classes rend le CIFAR-100 plus complexe, offrant ainsi un défi supplémentaire pour l'évaluation des modèles de réseaux de neurones.

La familiarité des objets dans le CIFAR-10 et la diversité accrue des classes dans le CIFAR-100 en font des jeux de données polyvalents pour évaluer la capacité des modèles à généraliser à différentes tâches de classification d'images.

Nous avons fait le choix de ne pas faire un prétraitement lourd car les données de CIFAR sont déjà propres. C'est aussi un choix de notre part de prendre un dataset couramment utilisé pour perdre le moins de temps possible et se concentrer sur le model.

4.2 Entraînement

Comme dit précédemment nous avons commencé avec un modèle simple que nous avons amélioré au fil du temps avant de tester des modèles pré-faits. Nous avons fait un fonction d'entraînement de model qui évalue au fur et mesure des epoch l'accuracy. Nous avons choisi l'accuracy comme métrique car elle évalue simplement la capacité du modèle à attribuer correctement des classes aux images, ce qui est l'objectif principal de la classification. C'est une métrique claire et facile à comprendre. Pour l'optimiseur nous avons décidé garder tout au long de nos expérimentations l'optimiseur Adam car il combine les avantages de l'optimisation stochastique et de l'adaptabilité des taux d'apprentissage, ce qui permet de converger plus rapidement vers des solutions de meilleure qualité sur ce jeu de données. Nous avons donc fait varier le taux d'apprentissage, mais le taux par défaut 0.001 a été dans notre cas le meilleur choix. Plus petit l'apprentissage était beaucoup trop long, et plus grand overfitting était trop présent. Pour la loss nous avons pris la "categorical_crossentropy" car elle est adaptée à CIFAR-10 car il s'agit d'un problème de classification multiclasse, elle mesure efficacement l'écart entre les prédictions du modèle et les véritables labels de classe.
Les premier résultats étaient déjà satisfaisant car nous avions 70% d'accuracy sur notre model. Nous avons testé des modèles pré-faits de plus en plus complexes. AlexNet nous a permis d'augmenter légèrement notre résultat. VGGNet était trop complexe pour notre modèle et a complètement overfiter sur une classe (toutes les prédictions vers la classe oiseau). Les modèles pré-entrainés que nous avons légèrement adapté à nos données nous ont donné les meilleurs résultats comme DenseNet121.
Nous avons ensuite décidé de prendre un dataset plus complexe pour faire un comparatif. Nous avons réalisé les mêmes tests sur CIFAR-100.
Les résultats sur notre modèle de base étaient de 34% d'accuracy. Nous avons ensuite utilisé tous les autres modèles et nous remarquons la même sorte d'amélioration. En effet chaque modèle qui améliore les résultats sur le CIFAR 10 les améliore aussi sur le CIFAR

100. On pourrait donc conclure que le modèle est plus dur à prédire mais similaire dans la façon dont les réseaux de neurones réagissent avec lui.

4.3 Résultats

Les résultats expérimentaux de nos modèles sur les ensembles de données CIFAR-10 et CIFAR-100 sont résumés dans le tableau ci-dessous. Les mesures d'accuracy obtenues démontrent l'efficacité de chaque architecture dans la tâche de classification d'images, offrant ainsi un aperçu comparatif de leurs performances respectives sur ces deux datasets.

	CIFAR-10	CIFAR-100
CNN simple	70.09 %	34.09 %
LeNet	59.07 %	27.48 %
AlexNet	73.20 %	46.34 %
VGGNet	10.00 %	-
DenseNet121	82.35 %	56.17 %
ResNet	75.61 %	48.24 %
InceptionNet	-	-

Figure 19 : Tableau récapitulatif des mesures d'accuracy

Toutes les mesures ont été effectuées approximativement dans des conditions équivalentes, avec des paramètres uniformes tels que le taux d'apprentissage, le nombre d'époques, calcul de la loss et d'autres configurations pour garantir une comparaison juste et précise entre les différentes architectures.

Nous avons aussi afficher dans le code a la fin du CIFAR-10 les prédiction pour 10 images de chaque classe avec sa prédiction et si elle est juste ou fausse, ainsi que l'accuracy et quelque autre metrics pour chaques classes différentes. Nous pouvons par exemple nous rendre compte qu'en moyenne la classe "chat" est plus dure à prédire que les autres. Et la classe "Automobile" est plus simple à prédire que les autres.
Cet affichage des images est aussi disponible pour CIFAR-100 mais moins lisible.

4.4 Conclusion

La conclusion de notre exploration des architectures de réseaux neuronaux convolutifs (CNN) sur les jeux de données CIFAR-10 et CIFAR-100 met en évidence la diversité des performances des modèles testés.

Le modèle CNN simple a montré une efficacité décente sur CIFAR-10, mais des performances moins convaincantes sur CIFAR-100. De même pour les architectures classiques telles que LeNet et AlexNet.
Les modèles plus avancés, tels que DenseNet121 et ResNet, ont démontré de solides performances sur les deux jeux de données, soulignant l'importance des architectures modernes dans la résolution de tâches de classification complexes. On constate que DenseNet s'est révélé être le meilleur modèle dans notre expérimentation.

Les résultats obtenus révèlent que des améliorations sont encore possibles, notamment sur CIFAR-100, suggérant qu'il existe des opportunités pour perfectionner davantage les performances des modèles dans le domaine de la classification d'images. Cette constatation souligne la persistance des défis et des opportunités d'amélioration dans ce domaine en constante évolution, incitant à poursuivre la recherche et le développement de modèles plus efficaces et adaptés à des tâches de classification plus complexes.

4.5 Problèmes rencontrés

Au cours de la réalisation de notre projet, plusieurs défis ont émergé, impactant notre démarche expérimentale.

L'une des contraintes majeures a été le manque de temps dû au temps d'exécution très longs, limitant ainsi la portée de nos explorations et de nos entraînements. De plus, certains modèles, tels qu'InceptionV3, ont démontré une forte exigence en termes de ressources computationnelles, au point de provoquer des dysfonctionnements sur nos machines, nous empêchant ainsi d'exécuter ces modèles.
Face à ces contraintes, nous avons pris la décision de limiter le nombre d'époques à 5 lors de l'entraînement de nos modèles. Cette approche visait à réduire significativement le temps d'exécution. Cependant, cette concession pourrait avoir influé sur la qualité de l'entraînement, les modèles n'atteignant peut-être pas une convergence optimale avec un nombre d'époques limité.

Lors de nos expérimentations, un problème notable est survenu avec VGGNet, où le modèle a manifesté des signes d'overfitting. Une particularité à noter est que le modèle a catégorisé la plupart des images dans la même classe, indépendamment de leurs caractéristiques distinctes. Cette tendance peut résulter d'une complexité excessive du modèle par rapport à la taille de l'ensemble de données, soulignant ainsi l'importance cruciale de la sélection appropriée du modèle en fonction des caractéristiques spécifiques de l'ensemble de données.

Enfin, nous avons enregistré nos modèles pour assurer la reproductibilité de nos expériences. Cependant, en raison de leur taille importante, nous avons rencontré des soucis pour les partager sur GitHub. Malgré cela, nous sommes prêts à fournir les modèles sur demande.

4.6 Organisation du travail

Tout au long du projet, nous avons travaillé en étroite collaboration, échangeant idées et codes sur Discord. Chacun a joué un rôle essentiel de manière égale, contribuant à la conception, à l'implémentation et à l'évaluation des modèles.

5. Table des illustrations

6. Bibliographie

Thomas Ranvier. Convolutional Neural Networks.

Yann LeCun, Léon Bottou, Yoshua Bengio et Patrick Haner. GradientBased Learning Applied to Document Recognition.
http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

Bhavesh Singh Bisht. Types of Convolutional Neural Networks: LeNet, AlexNet, VGG-16 Net, ResNet and Inception Net.
https://medium.com/analytics-vidhya/types-of-convolutional-neural-networks-lenet-alexnet-vgg-16-net-resnet-and-inception-net-759e5f197580