Rapport d'Expérimentation

Table des Matières

Introduction

Ce rapport présente les résultats des expérimentations réalisées sur différents réseaux de neurones. Pour chaque réseau testé, nous fournissons des détails sur l'architecture, les hyperparamètres utilisés, ainsi que les performances obtenues.

Attention a prendre en consideration que le dataset doit etre "TP2-images".

Réseau dynamique (TP2-train-docu)

Architecture du Réseau

Le modèle de réseau de neurones utilisé est conçu pour s'adapter aux limites de mémoire disponibles. Les couches suivantes sont utilisées dans l'ordre :

- **Input Layer:** `Input(shape=(input_shape))`
- **Convolutional Layers:** Plusieurs couches convolutives en fonction du niveau de complexité :
  - Chaque couche convolutive utilise des filtres de taille (3, 3) avec activation ReLU.
  - **BatchNormalization Layer:** `BatchNormalization()` est appliqué après chaque couche convolutive.
  - **MaxPooling Layer:** `MaxPooling2D((2, 2))` est appliqué après chaque convolution.
  - **Dropout Layer:** `Dropout(0.25)` est appliqué toutes les deux couches convolutives.
- **Flatten Layer:** `Flatten()`
- **Dense Layer 1:** `Dense(128, activation='relu')`
- **Dropout Layer:** `Dropout(0.5)`
- **Output Layer:** `Dense(num_classes, activation='softmax')`

2. Learning Rate

Le learning rate par défaut utilisé par l'optimiseur Adam est appliqué, c'est-à-dire `0.001`.

3. Optimiseur

L'optimiseur utilisé est `Adam`.

4. Taille des Batchs, Epochs et Itérations

- **Taille des batchs:** Les tailles sont ajustées en fonction de la configuration mémoire disponible, avec des valeurs possibles de 8 ou 16.
- **Nombre d'époques:** Les époques peuvent être ajustées entre 10 et 20.
- **Nombre d'itérations:** Calculé automatiquement en fonction de la taille du dataset.

5. Modifications Apportées sur le Dataset

- **Équilibrage des Classes:** Des techniques d'augmentation d'images peuvent être appliquées pour équilibrer les classes dans le dataset, incluant des rotations, des flips, des zooms, et des ajustements de luminosité.
- **Répartition Apprentissage/Test:** Le dataset peut être divisé avec un ratio de 80% pour l'apprentissage et 20% pour le test.

Analyse

- **Justification des paramètres :**

  1. **Taille des images (img_sizes) :**

     - **32x32 pixels** : Recommandé pour les modèles simples en raison de la rapidité d'entraînement et de l'efficacité mémoire, malgré la perte possible de détails.
     - **64x64 pixels** : Bon compromis entre détails et consommation de ressources, mais peut être excessif pour des tâches simples.
     - **128x128 pixels** : Recommandé pour des architectures plus complexes, apportant davantage de détails au prix d'une plus grande consommation de mémoire et du risque de surajustement.
     - **256x256 pixels** : Utilisé pour des tâches très complexes avec des modèles avancés, mais très gourmand en ressources.

2. **Nombre d'époques (epochs_list) :**
    - **10 époques** : Convient pour des tests rapides, mais peut entraîner un sous-apprentissage.
    - **20 époques** : Bon compromis pour la plupart des tâches, permettant une bonne convergence sans surapprentissage.
    - **30 époques** : Utile pour des tâches complexes, mais augmente le risque de surajustement et le temps d'entraînement.
3. **Taille des batchs (batch_sizes) :**
    - **8** : Offre une meilleure précision des gradients mais entraîne un apprentissage plus lent et des gradients plus bruités.
    - **16** : Bon équilibre entre la stabilité des gradients et la vitesse d'entraînement, mais peut encore être lent pour des tâches complexes.
    - **32** : Recommandé pour une bonne balance entre stabilité et vitesse, bien adapté aux GPU modernes.
    - **64** : Accélère l'entraînement mais exige beaucoup de mémoire et peut limiter les ajustements fins du modèle.

En résumé, nous avons utilisé ce fichier pour déterminer quel résultat serait le meilleur, afin de l'appliquer ensuite à un réseau plus complexe. Nous avons compris que la meilleure taille d'image (img_sizes) est de 32x32 pour les modèles simples, mais comme le modèle principal que nous avons choisi est le modèle 9, qui est plus complexe, la meilleure taille d'image est de 128x128. Nous n'avons pas prit 258x258 car plus la taille de l'image est grande, plus le risque de surajustement augmente, surtout si le dataset est limité en taille. Le meilleur nombre d'époques (epochs_list) est de 20, et les meilleures tailles de batch (batch_sizes) sont de 16 ou 32, selon les préférences pour équilibrer stabilité et vitesse.

---

1. Architecture du Réseau (TP2-train-test3.py)

Le modèle de réseau de neurones utilisé est basé sur une architecture de couches convolutives séquentielles. Les couches suivantes sont utilisées dans l'ordre :

Cette architecture utilise une profondeur modérée avec des couches de convolution alternées avec BatchNormalization et Dropout pour éviter le surapprentissage. La dernière couche Dense est suivie d'une couche de 512 unités pour une classification plus fine.

- **Input Layer:** `Input(shape=(32, 32, 3))`
- **Conv2D Layer 1:** `Conv2D(64, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 1:** `BatchNormalization()`
- **MaxPooling2D Layer 1:** `MaxPooling2D((2, 2))`
- **Dropout Layer 1:** `Dropout(0.3)`
- **Conv2D Layer 2:** `Conv2D(128, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 2:** `BatchNormalization()`
- **MaxPooling2D Layer 2:** `MaxPooling2D((2, 2))`
- **Dropout Layer 2:** `Dropout(0.4)`
- **Conv2D Layer 3:** `Conv2D(256, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 3:** `BatchNormalization()`
- **MaxPooling2D Layer 3:** `MaxPooling2D((2, 2))`
- **Dropout Layer 3:** `Dropout(0.5)`
- **Flatten Layer:** `Flatten()`
- **Dense Layer 1:** `Dense(512, activation='relu')`
- **Dropout Layer 4:** `Dropout(0.5)`
- **Output Layer:** `Dense(num_classes, activation='softmax')`

2. Learning Rate

Le learning rate par défaut utilisé par l'optimiseur Adam est appliqué, c'est-à-dire `0.001`.

3. Optimiseur

L'optimiseur utilisé est `Adam`.

4. Taille des Batchs, Epochs et Itérations

- **Taille de img_size:** 32
- **Taille des batchs:** 16
- **Nombre d'époques:** 20
- **Nombre d'itérations:** Calculé automatiquement en fonction de la taille du dataset.

5. Modifications Apportées sur le Dataset

- **Équilibrage des Classes:** Des techniques d'augmentation d'images ont été appliquées pour équilibrer les classes dans le dataset, incluant des rotations, des flips, des zooms, et des ajustements de luminosité.
- **Répartition Apprentissage/Test:** Le dataset a été divisé avec un ratio de 80% pour l'apprentissage et 20% pour le test.

6. Vitesse d'Apprentissage, Précision et Recall

Les résultats obtenus après l'entraînement du modèle sont les suivants :

Matrice de Confusion Précision par Époque Recall par Époque Perte par Époque

7. Analyse et Recommandations pour les Expérimentations Futures

Les résultats montrent que les classes d'animaux comme le chien et la poule ont un taux de réussite plus faible comparé à d'autres classes comme l'éléphant et le papillon. Cela pourrait être dû à des différences dans le nombre d'images ou à la complexité des caractéristiques visuelles de ces classes. Pour les prochaines expérimentations :

- **Augmentation des Données:** Il serait utile d'augmenter davantage les données pour les classes sous-représentées.
- **Complexité du Modèle:** Tester des modèles avec plus de couches ou des configurations de convolutions plus complexes pourrait améliorer la performance pour les classes ayant de faibles performances.
- **Optimisation des Hyperparamètres:** Tester différentes valeurs de learning rate et batch size pour trouver la combinaison optimale.

---

1. Architecture du Réseau TP2-train-test4.py

Le modèle de réseau de neurones utilisé est basé sur une architecture de couches convolutives avec un pooling global. Les couches suivantes sont utilisées dans l'ordre :

Cette architecture se concentre sur la réduction de la dimensionnalité avec GlobalAveragePooling après une série de convolutions profondes, suivie d'une couche Dense pour la classification.

- **Input Layer:** `Input(shape=(32, 32, 3))`
- **Conv2D Layer 1:** `Conv2D(64, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 1:** `BatchNormalization()`
- **MaxPooling2D Layer 1:** `MaxPooling2D((2, 2))`
- **Conv2D Layer 2:** `Conv2D(128, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 2:** `BatchNormalization()`
- **MaxPooling2D Layer 2:** `MaxPooling2D((2, 2))`
- **Conv2D Layer 3:** `Conv2D(256, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 3:** `BatchNormalization()`
- **MaxPooling2D Layer 3:** `MaxPooling2D((2, 2))`
- **Conv2D Layer 4:** `Conv2D(512, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 4:** `BatchNormalization()`
- **GlobalAveragePooling Layer:** `GlobalAveragePooling2D()`
- **Dropout Layer:** `Dropout(0.5)`
- **Output Layer:** `Dense(num_classes, activation='softmax')`

2. Learning Rate

Le learning rate par défaut utilisé par l'optimiseur Adam est appliqué, c'est-à-dire `0.001`.

3. Optimiseur

L'optimiseur utilisé est `Adam`.

4. Taille des Batchs, Epochs et Itérations

- **Taille de img_size:** 32
- **Taille des batchs:** 16
- **Nombre d'époques:** 20
- **Nombre d'itérations:** Calculé automatiquement en fonction de la taille du dataset.

5. Modifications Apportées sur le Dataset

- **Équilibrage des Classes:** Des techniques d'augmentation d'images ont été appliquées pour équilibrer les classes dans le dataset, incluant des rotations, des flips, des zooms, et des ajustements de luminosité.
- **Répartition Apprentissage/Test:** Le dataset a été divisé avec un ratio de 80% pour l'apprentissage et 20% pour le test.

6. Vitesse d'Apprentissage, Précision et Recall

Les résultats obtenus après l'entraînement du modèle sont les suivants :

Matrice de Confusion Précision par Époque Recall par Époque Perte par Époque Exactitude par Époque

7. Analyse et Recommandations pour les Expérimentations Futures

Les résultats montrent que les classes d'animaux comme le chat et le chien ont un taux de réussite plus faible comparé à d'autres classes comme l'araignée et l'éléphant. Cela pourrait être dû à des différences dans le nombre d'images ou à la complexité des caractéristiques visuelles de ces classes. Pour les prochaines expérimentations :

- **Augmentation des Données:** Il serait utile d'augmenter davantage les données pour les classes sous-représentées.
- **Complexité du Modèle:** Tester des modèles avec plus de couches ou des configurations de convolutions plus complexes pourrait améliorer la performance pour les classes ayant de faibles performances.
- **Optimisation des Hyperparamètres:** Tester différentes valeurs de learning rate et batch size pour trouver la combinaison optimale.

---

1. Architecture du Réseau TP2-train-test5.py

Le modèle de réseau de neurones utilisé est basé sur une architecture de couches convolutives avec l'ajout de kernels 1x1 pour augmenter la capacité d'extraction des caractéristiques. Les couches suivantes sont utilisées dans l'ordre :

Cette architecture utilise un bloc plus complexe avec des convolutions de petites tailles de noyaux (1x1 et 3x3) combinées avec des MaxPooling et Dropout. Cela permet une extraction des caractéristiques très fine.

- **Input Layer:** `Input(shape=(32, 32, 3))`
- **Conv2D Layer 1:** `Conv2D(64, (1, 1), padding='same', activation='relu')`
- **BatchNormalization Layer 1:** `BatchNormalization()`
- **Conv2D Layer 2:** `Conv2D(64, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 2:** `BatchNormalization()`
- **MaxPooling2D Layer 1:** `MaxPooling2D((2, 2))`
- **Dropout Layer 1:** `Dropout(0.3)`
- **Conv2D Layer 3:** `Conv2D(128, (1, 1), padding='same', activation='relu')`
- **BatchNormalization Layer 3:** `BatchNormalization()`
- **Conv2D Layer 4:** `Conv2D(128, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 4:** `BatchNormalization()`
- **MaxPooling2D Layer 2:** `MaxPooling2D((2, 2))`
- **Dropout Layer 2:** `Dropout(0.4)`
- **Conv2D Layer 5:** `Conv2D(256, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 5:** `BatchNormalization()`
- **MaxPooling2D Layer 3:** `MaxPooling2D((2, 2))`
- **Flatten Layer:** `Flatten()`
- **Dense Layer 1:** `Dense(512, activation='relu')`
- **Dropout Layer 3:** `Dropout(0.5)`
- **Output Layer:** `Dense(num_classes, activation='softmax')`

2. Learning Rate

Le learning rate par défaut utilisé par l'optimiseur Adam est appliqué, c'est-à-dire `0.001`.

3. Optimiseur

L'optimiseur utilisé est `Adam`.

4. Taille des Batchs, Epochs et Itérations

- **Taille de img_size:** 32
- **Taille des batchs:** 16
- **Nombre d'époques:** 20
- **Nombre d'itérations:** Calculé automatiquement en fonction de la taille du dataset.

5. Modifications Apportées sur le Dataset

- **Équilibrage des Classes:** Des techniques d'augmentation d'images ont été appliquées pour équilibrer les classes dans le dataset, incluant des rotations, des flips, des zooms, et des ajustements de luminosité.
- **Répartition Apprentissage/Test:** Le dataset a été divisé avec un ratio de 80% pour l'apprentissage et 20% pour le test.

6. Vitesse d'Apprentissage, Précision et Recall

Matrice de Confusion Précision par Époque Recall par Époque Perte par Époque Exactitude par Époque

7. Analyse et Recommandations pour les Expérimentations Futures

Les résultats montrent que certaines classes comme l'éléphant et l'araignée ont un taux de réussite élevé, tandis que d'autres comme le chat et le chien présentent des difficultés. Pour les prochaines expérimentations :

- **Augmentation des Données:** Continuer à augmenter les données pour les classes sous-représentées et difficiles.
- **Complexité du Modèle:** Tester des configurations avec davantage de couches et des combinaisons de kernels plus complexes.
- **Optimisation des Hyperparamètres:** Tester différentes combinaisons de learning rate et batch size pour trouver une configuration optimale.

---

1. Architecture du Réseau TP2-train-test6.py

Le modèle de réseau de neurones utilisé est basé sur une architecture de blocs résiduels pour améliorer l'apprentissage en profondeur. Les couches suivantes sont utilisées dans l'ordre :

Cette architecture utilise une approche de réseau profond avec des blocs résiduels simples pour améliorer l'apprentissage, en combinant la profondeur et des techniques avancées comme BatchNormalization et Dropout.

- **Input Layer:** `Input(shape=(32, 32, 3))`
- **Conv2D Layer 1:** `Conv2D(64, (7, 7), padding='same', strides=2, activation='relu')`
- **BatchNormalization Layer 1:** `BatchNormalization()`
- **MaxPooling2D Layer 1:** `MaxPooling2D((2, 2))`
- **Residual Block 1:** `residual_block(x, 64)`
- **MaxPooling2D Layer 2:** `MaxPooling2D((2, 2))`
- **Residual Block 2:** `residual_block(x, 128)`
- **MaxPooling2D Layer 3:** `MaxPooling2D((2, 2))`
- **Residual Block 3:** `residual_block(x, 256)`
- **MaxPooling2D Layer 4:** `MaxPooling2D((2, 2))`
- **Flatten Layer:** `Flatten()`
- **Dense Layer 1:** `Dense(512, activation='relu')`
- **Dropout Layer:** `Dropout(0.5)`
- **Output Layer:** `Dense(num_classes, activation='softmax')`

2. Learning Rate

Le learning rate par défaut utilisé par l'optimiseur Adam est appliqué, c'est-à-dire `0.001`.

3. Optimiseur

L'optimiseur utilisé est `Adam`.

4. Taille des Batchs, Epochs et Itérations

- **Taille de img_size:** 32
- **Taille des batchs:** 16
- **Nombre d'époques:** 20
- **Nombre d'itérations:** Calculé automatiquement en fonction de la taille du dataset.

5. Modifications Apportées sur le Dataset

- **Équilibrage des Classes:** Des techniques d'augmentation d'images ont été appliquées pour équilibrer les classes dans le dataset, incluant des rotations, des flips, des zooms, et des ajustements de luminosité.
- **Répartition Apprentissage/Test:** Le dataset a été divisé avec un ratio de 80% pour l'apprentissage et 20% pour le test.

6. Vitesse d'Apprentissage, Précision et Recall

Les résultats obtenus après l'entraînement du modèle sont les suivants :

Matrice de Confusion Précision par Époque Recall par Époque Perte par Époque Exactitude par Époque

7. Analyse et Recommandations pour les Expérimentations Futures

L'intégration de blocs résiduels a permis d'améliorer la capacité du modèle à apprendre des caractéristiques complexes, bien que certains problèmes persistent dans les classes comme le chien et le chat. Pour les prochaines expérimentations :

- **Augmentation des Données:** Augmenter davantage les données pour les classes sous-représentées.
- **Complexité du Modèle:** Tester des variantes avec plus de blocs résiduels ou d'autres architectures avancées comme ResNet.
- **Optimisation des Hyperparamètres:** Tester différentes configurations de learning rate et de taille des batches pour maximiser la performance du modèle.

---

1. Architecture du Réseau TP2-train-test7.py

Le modèle de réseau de neurones utilisé est basé sur une architecture de couches convolutives avec des kernels variés pour capturer différentes échelles de caractéristiques. Les couches suivantes sont utilisées dans l'ordre :

Cette architecture combine des convolutions de petite taille (1x1) avec des convolutions standard (3x3 et 5x5) pour capturer des caractéristiques à différentes échelles, suivie de GlobalAveragePooling pour une réduction de la dimensionnalité avant la classification.

- **Input Layer:** `Input(shape=(32, 32, 3))`
- **Conv2D Layer 1:** `Conv2D(64, (1, 1), padding='same', activation='relu')`
- **BatchNormalization Layer 1:** `BatchNormalization()`
- **Conv2D Layer 2:** `Conv2D(64, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 2:** `BatchNormalization()`
- **Conv2D Layer 3:** `Conv2D(128, (5, 5), padding='same', activation='relu')`
- **BatchNormalization Layer 3:** `BatchNormalization()`
- **MaxPooling2D Layer 1:** `MaxPooling2D((2, 2))`
- **Conv2D Layer 4:** `Conv2D(256, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 4:** `BatchNormalization()`
- **MaxPooling2D Layer 2:** `MaxPooling2D((2, 2))`
- **Conv2D Layer 5:** `Conv2D(512, (3, 3), padding='same', activation='relu')`

- **BatchNormalization Layer 5:** `BatchNormalization()`
- **GlobalAveragePooling Layer:** `GlobalAveragePooling2D()`
- **Dropout Layer:** `Dropout(0.5)`
- **Output Layer:** `Dense(num_classes, activation='softmax')`

2. Learning Rate

Le learning rate par défaut utilisé par l'optimiseur Adam est appliqué, c'est-à-dire `0.001`.

3. Optimiseur

L'optimiseur utilisé est `Adam`.

4. Taille des Batchs, Epochs et Itérations

- **Taille de img_size:** 32
- **Taille des batchs:** 16
- **Nombre d'époques:** 20
- **Nombre d'itérations:** Calculé automatiquement en fonction de la taille du dataset.

5. Modifications Apportées sur le Dataset

- **Équilibrage des Classes:** Des techniques d'augmentation d'images ont été appliquées pour équilibrer les classes dans le dataset, incluant des rotations, des flips, des zooms, et des ajustements de luminosité.
- **Répartition Apprentissage/Test:** Le dataset a été divisé avec un ratio de 80% pour l'apprentissage et 20% pour le test.

6. Vitesse d'Apprentissage, Précision et Recall

Les résultats obtenus après l'entraînement du modèle sont les suivants :

Matrice de Confusion Précision par Époque Recall par Époque Perte par Époque Exactitude par Époque

7. Analyse et Recommandations pour les Expérimentations Futures

Le modèle a montré une amélioration significative avec des taux de réussite élevés pour plusieurs classes. Cependant, certaines classes continuent de poser des problèmes. Pour les prochaines expérimentations :

- **Augmentation des Données:** Continuer à explorer des techniques d'augmentation pour améliorer les performances sur les classes moins performantes.
- **Complexité du Modèle:** Tester des configurations plus complexes ou des architectures alternatives pour capturer plus de détails dans les images.
- **Optimisation des Hyperparamètres:** Essayer différentes valeurs de learning rate et de batch size pour raffiner les performances du modèle.

---

1. Architecture du Réseau TP2-train-test8.py

Le modèle de réseau de neurones utilisé est basé sur une architecture de modules Inception pour capturer différentes échelles de caractéristiques dans les images. Les couches suivantes sont utilisées dans l'ordre :

Cette architecture adopte une approche similaire à celle des architectures Inception, où des convolutions parallèles de différentes tailles sont utilisées pour extraire des caractéristiques de manière plus robuste.

- **Input Layer:** `Input(shape=(32, 32, 3))`
- **Inception Module 1:** `inception_module(inputs, 64)`
- **MaxPooling2D Layer 1:** `MaxPooling2D((2, 2))`
- **Inception Module 2:** `inception_module(x, 128)`
- **MaxPooling2D Layer 2:** `MaxPooling2D((2, 2))`
- **Inception Module 3:** `inception_module(x, 256)`
- **MaxPooling2D Layer 3:** `MaxPooling2D((2, 2))`
- **Flatten Layer:** `Flatten()`
- **Dense Layer 1:** `Dense(512, activation='relu')`
- **Dropout Layer:** `Dropout(0.5)`
- **Output Layer:** `Dense(num_classes, activation='softmax')`

2. Learning Rate

Le learning rate par défaut utilisé par l'optimiseur Adam est appliqué, c'est-à-dire `0.001`.

3. Optimiseur

L'optimiseur utilisé est `Adam`.

4. Taille des Batchs, Epochs et Itérations

- **Taille de img_size:** 32
- **Taille des batchs:** 16
- **Nombre d'époques:** 20
- **Nombre d'itérations:** Calculé automatiquement en fonction de la taille du dataset.

5. Modifications Apportées sur le Dataset

- **Équilibrage des Classes:** Des techniques d'augmentation d'images ont été appliquées pour équilibrer les classes dans le dataset, incluant des rotations, des flips, des zooms, et des ajustements de luminosité.
- **Répartition Apprentissage/Test:** Le dataset a été divisé avec un ratio de 80% pour l'apprentissage et 20% pour le test.

6. Vitesse d'Apprentissage, Précision et Recall

Les résultats obtenus pendant l'entraînement du modèle sont afficher dans l'image ci-dessous et explique la raison pour laquel nous avons mit fin a se reseau.

Précision par Époque

7. Analyse et Recommandations pour les Expérimentations Futures

Le modèle basé sur les modules Inception montre des résultats prometteurs, mais il y a des défis en termes de généralisation sur le jeu de données de test. Pour les prochaines expérimentations : Une taille d'image de 32x32 est très petite pour capturer des détails complexes nécessaires pour différencier les classes dans un problème de classification d'images avec plusieurs catégories. 20 époques n'est pas suffisant pour permettre à ce modèle d'apprendre efficacement des données lorsque la tâche est complexe et que les images sont de petite taille.

---

1. Architecture du Réseau TP2-train-test9.py

Le modèle de réseau de neurones utilisé est basé sur une architecture classique de couches convolutives suivies de couches denses pour la classification binaire. Les couches suivantes sont utilisées dans l'ordre :

Cette architecture utilise la fonction d'activation sigmoid dans la couche de sortie, adaptée pour un problème de classification binaire.

- **Input Layer:** `Input(shape=(128, 128, 3))`
- **Conv2D Layer 1:** `Conv2D(64, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 1:** `BatchNormalization()`
- **MaxPooling2D Layer 1:** `MaxPooling2D((2, 2))`
- **Dropout Layer 1:** `Dropout(0.3)`
- **Conv2D Layer 2:** `Conv2D(128, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 2:** `BatchNormalization()`
- **MaxPooling2D Layer 2:** `MaxPooling2D((2, 2))`
- **Dropout Layer 2:** `Dropout(0.4)`
- **Conv2D Layer 3:** `Conv2D(256, (3, 3), padding='same', activation='relu')`
- **BatchNormalization Layer 3:** `BatchNormalization()`
- **MaxPooling2D Layer 3:** `MaxPooling2D((2, 2))`
- **Dropout Layer 3:** `Dropout(0.5)`
- **Flatten Layer:** `Flatten()`
- **Dense Layer 1:** `Dense(512, activation='relu')`
- **Dropout Layer 4:** `Dropout(0.5)`
- **Output Layer:** `Dense(num_classes, activation='sigmoid')`

2. Learning Rate

Le learning rate par défaut utilisé par l'optimiseur Adam est appliqué, c'est-à-dire `0.001`.

3. Optimiseur

L'optimiseur utilisé est `Adam`.

4. Taille des Batchs, Epochs et Itérations

- **Taille de img_size:** 32
- **Taille des batchs:** 16
- **Nombre d'époques:** 20
- **Nombre d'itérations:** Calculé automatiquement en fonction de la taille du dataset.

5. Modifications Apportées sur le Dataset

- **Équilibrage des Classes:** Des techniques d'augmentation d'images ont été appliquées pour équilibrer les classes dans le dataset, incluant des rotations, des

flips, des zooms, et des ajustements de luminosité.
- **Répartition Apprentissage/Test:** Le dataset a été divisé avec un ratio de 80% pour l'apprentissage et 20% pour le test.

6. Vitesse d'Apprentissage, Précision et Recall

Les résultats obtenus après l'entraînement du modèle sont les suivants :

Matrice de Confusion Précision par Époque Recall par Époque Perte par Époque Exactitude par Époque

7. Analyse et Recommandations pour les Expérimentations Futures

Le modèle avec cette configuration montre une bonne performance générale, mais il y a encore des marges d'amélioration pour mieux généraliser sur les données de test. Pour les prochaines expérimentations :

- **Augmentation des Données:** Continuer à expérimenter avec des techniques d'augmentation pour améliorer la robustesse du modèle.
- **Complexité du Modèle:** Tester des architectures plus complexes pour capturer davantage de nuances dans les images.
- **Optimisation des Hyperparamètres:** Ajuster les paramètres comme le learning rate et la taille des batchs pour maximiser la performance.

Réseau Dynamique 9 (TP2-train-test9)

l'architecture est la meme on a simplement modifier image size, batchs et epoque afin de verifier la difference des resultat entre l'ancien qui etait : img_size=32, epochs=20, batch_size=16 et celui si qui est : img_size=128, epochs=20, batch_size=32

Matrice de Confusion Précision par Époque Recall par Époque Perte par Époque Exactitude par Époque

Analyse des Résultats

1. **Performance du Modèle :**
    - **Accuracy et Val_Accuracy** : Le modèle montre une progression régulière de l'accuracy, mais les courbes de validation montrent une certaine stagnation vers la fin de l'entraînement, ce qui suggère un début de sur-apprentissage.
    - **Precision et Val_Precision** : La précision augmente, mais avec des fluctuations, indiquant une instabilité dans la capacité du modèle à bien classer les exemples de validation.
    - **Recall et Val_Recall** : Le rappel suit une tendance similaire à celle de la précision, montrant une amélioration générale mais avec des oscillations.
    - **Loss et Val_Loss** : La perte diminue globalement, mais la perte de validation stagne, ce qui peut être un signe de sur-apprentissage.
2. **Complexité du Modèle (complexity_level=1) :**
    - Le modèle utilisé ici est assez simple, ce qui peut limiter sa capacité à capturer des détails complexes, malgré l'augmentation de la taille des images à 128x128. Le modèle semble sous-optimisé pour la tâche, avec un potentiel d'amélioration si la complexité est augmentée.
3. **Taille des Lots (batch_size=32) :**
    - Une taille de lot de 32 offre un bon compromis entre stabilité et vitesse d'entraînement. Cependant, les résultats montrent que même avec cette configuration, le modèle n'atteint pas une convergence optimale.
4. **Taille d'Image (img_size=128) :**
    - L'augmentation de la taille d'image à 128x128 aide le modèle à capturer plus de détails, mais les fluctuations dans les courbes de validation suggèrent que d'autres ajustements sont nécessaires, comme augmenter la complexité du modèle.

Conclusion

Bien que l'augmentation de la taille des images à 128x128 ait permis au modèle de mieux capturer les détails, la faible complexité du modèle limite ses performances globales. Pour améliorer les résultats, il est recommandé d'augmenter la complexité du modèle (par exemple, en passant à complexity_level=3 ou 5) tout en conservant une taille d'image élevée et une taille de lot de 32 ou 64. Cela devrait aider à améliorer la capacité du modèle à généraliser sur des données non vues et à stabiliser les performances sur les ensembles de validation.

Conclusion

Résumé des résultats globaux

Dans cette étude, plusieurs architectures de réseaux de neurones ont été testées avec différentes configurations de paramètres pour évaluer leur performance sur un ensemble de données de classification d'images. Les modèles testés incluent des architectures basées sur des couches convolutives simples, des blocs résiduels, des modules Inception, et d'autres variations.

Comparaison des différents réseaux testés

1. **Réseau Dynamique (TP2-train-docu) :**
    - Ce réseau a été conçu pour s'adapter aux limites de mémoire disponibles tout en maximisant l'extraction des caractéristiques. Les résultats montrent que le modèle fonctionne bien avec des images de petite taille (32x32) mais présente des risques de surapprentissage avec des tailles d'image plus grandes (128x128).
2. **TP2-train-test3 :**

    - Ce modèle simple basé sur des convolutions a montré des performances acceptables, mais il a des difficultés avec certaines classes comme les chiens et les chats. Les résultats suggèrent que l'ajout de couches et une augmentation de la complexité pourraient améliorer la précision.

3. **TP2-train-test4 :**
    - L'ajout de pooling global dans cette architecture a permis d'améliorer la capture des caractéristiques globales des images. Cependant, des classes comme le chat et le chien restent problématiques, suggérant que d'autres ajustements sont nécessaires.

4. **TP2-train-test5 :**
    - Ce modèle a intégré des kernels 1x1 pour augmenter la capacité d'extraction de caractéristiques, ce qui a amélioré la performance sur certaines classes. Cependant, l'architecture reste limitée pour les classes complexes, nécessitant une meilleure optimisation des hyperparamètres.

5. **TP2-train-test6 :**
    - L'architecture avec blocs résiduels a montré des améliorations significatives dans l'apprentissage, mais certaines classes comme le chien et le chat continuent de poser problème. Une augmentation de la complexité du modèle pourrait aider à surmonter ces défis.

6. **TP2-train-test7 :**
    - Ce réseau utilise des kernels variés pour capturer différentes échelles de caractéristiques. Les résultats montrent une amélioration générale, mais des classes comme le chat et le chien nécessitent toujours une attention particulière.

7. **TP2-train-test8 :**
    - Le modèle basé sur les modules Inception a montré des difficultés à généraliser sur le jeu de test, en raison de la petite taille d'image (32x32) utilisée. Les résultats suggèrent que des tailles d'image plus grandes et plus d'époques sont nécessaires pour améliorer les performances.

8. **TP2-train-test9 :**
    - Ce modèle utilise des images de taille 128x128 et a montré une meilleure performance générale, bien que le surapprentissage soit un risque. L'augmentation de la complexité du modèle pourrait aider à mieux capturer les détails tout en évitant le surapprentissage.

Nous avons aussi realiser dautre reseau qui non pas ete retester depuis les derniere amelioration mais etait tous de meme vraiment interessent a voir soit :

- TP2-train-test10.py (Classification Multiclasse avec Tanh) Cette architecture utilise la fonction d'activation tanh dans la couche de sortie. Cette fonction est moins courante pour la classification multiclasse mais peut être utilisée dans des situations spécifiques où des valeurs de sortie dans une gamme symétrique sont nécessaires.
- TP2-train-test11.py (Régression avec Linear) Cette architecture est adaptée pour un problème de régression, utilisant la fonction d'activation linear (pas d'activation) dans la couche de sortie pour prédire des valeurs continues.
- old-test1.py Le modèle utilise des couches de convolution suivies de couches de pooling et de couches denses pour prédire la classe des images parmi plusieurs catégories. Il est adapté à une tâche de classification d'images où les classes sont bien définies, comme la reconnaissance d'animaux. La fonction d'activation softmax est utilisée dans la couche de sortie pour générer des probabilités pour chaque classe.
- old-test2.py traite à la fois des problèmes de classification binaire (détection de fursuits) et de classification multi-classes (classification d'espèces). Deux modèles distincts sont construits pour ces tâches, avec une gestion spécifique des données et des sorties adaptées à chaque cas. Pour la classification binaire, la fonction d'activation sigmoid est utilisée dans la couche de sortie, tandis que pour la classification multi-classes, la fonction softmax est utilisée.
- old-test3.py Ce fichier propose deux architectures CNN distinctes et explore la combinaison des prédictions de ces modèles pour améliorer la performance globale. Cette approche permet d'augmenter la robustesse des prédictions en moyennant les sorties de plusieurs réseaux. Chaque modèle est optimisé pour une classification multi-classes avec une fonction softmax en sortie.
- TP2-train-test2.py Le modèle utilise plusieurs couches de convolution et de pooling pour extraire les caractéristiques des images et les classifier en différentes catégories. Une fonction softmax est utilisée dans la couche de sortie pour prédire les probabilités associées à chaque classe. Des visualisations des performances, telles que des courbes de précision et de rappel, ainsi qu'une matrice de confusion, sont générées pour évaluer le modèle.

Recommandations pour les futurs travaux

1. **Augmenter la Complexité du Modèle :**
    - Pour des tâches complexes, il est recommandé d'augmenter la complexité du modèle en ajoutant plus de couches convolutives ou en utilisant des architectures plus avancées comme ResNet ou des modules Inception plus profonds.

2. **Ajuster les Paramètres d'Apprentissage :**
    - Tester différentes tailles d'image, tailles de batch, et taux d'apprentissage pour trouver la configuration optimale pour chaque tâche. Les tailles d'image plus grandes peuvent offrir de meilleurs résultats pour des tâches complexes, tandis que des tailles de batch plus petites peuvent améliorer la précision des gradients.

3. **Augmenter les Données :**
    - Continuer à augmenter les données pour les classes sous-représentées, en utilisant des techniques avancées d'augmentation d'images pour améliorer la robustesse du modèle.

4. **Suivi du Surapprentissage :**
    - Surveiller les courbes de validation pour éviter le surapprentissage, en ajustant les paramètres comme le nombre d'époques et en utilisant des techniques de régularisation comme le dropout ou l'arrêt précoce.

En conclusion, les modèles testés montrent une progression dans la compréhension des architectures adaptées à cette tâche de classification d'images. Toutefois, des ajustements supplémentaires dans la complexité des modèles, l'augmentation des données et l'optimisation des hyperparamètres sont nécessaires pour maximiser les performances sur ce jeu de données. Notre meilleur résultat, et celui que l'on perçoit comme notre réseau principal, est le TP2-train-test9.

Data-warehouse

Structure du Dataset sans normalisation

Le dataset est composé à l'origine de la façon suivante :

- **TP2-images**
    - **cane** : Contient des images de chien

- **cavallo** : Contient des images de cheval
- **elefante** : Contient des images de elephant
- **farfalla** : Contient des images de papillon
- **fursuit** : Contient des images de fursuit
- **gallina** : Contient des images de poule
- **gatto** : Contient des images de chat
- **mucca** : Contient des images de vache
- **pecora** : Contient des images de mouton
- **ragno** : Contient des images de araignee
- **scoiattolo** : Contient des images de ecureuil

Structure du Dataset Normaliser

Le nouveau dataset est composé de la façon suivante :

- **TP2-images**
  - **fursuit** : Contient des images de fursuit
  - **especes** : Contient des dossiers de chaque espèce
    - **cane** : Contient des images de chien
    - **cavallo** : Contient des images de cheval
    - **elefante** : Contient des images d'éléphant
    - **farfalla** : Contient des images de papillon
    - **gallina** : Contient des images de poule
    - **gatto** : Contient des images de chat
    - **mucca** : Contient des images de vache
    - **pecora** : Contient des images de mouton
    - **ragno** : Contient des images d'araignée
    - **scoiattolo** : Contient des images d'écureuil

Qu'est-ce qui a été normalisé ?

- **Suppression des images nuisibles** à l'apprentissage du réseau CNN.
- **Duplication des images** pour équilibrer les proportions de chaque dossier.
- **L'ajout d'images** n'a pas été mis en place car cela détruisait l'apprentissage du réseau même s'il réglait le problème des proportions.
- **Mise en place d'un poids pour chaque image** afin d'équilibrer les proportions a été pris en compte et nous avons réalisé des manipulations(orientation de l'image, angle, miroir, bruit, luminausiter) sur les images dupliquer afin d'equilibrer les dossier sans faire de l'overfitting.

Documentations

- Multi-Label Classification with Deep Learning
- Activation Functions
- tensorflow

Dépendances

- time
- os
- sys
- random
- gc
- importlib
- collections
- NumPy
- TensorFlow
- Scikit-learn
- Tabulate
- Pandas
- io
- contextlib
- psutil

Auteurs

Nom : Kevin Da Silva CodeMS : DASK30049905

Nom : Samuel Dextraze CodeMS : DEXS03039604

Nom : Saïd Ryan Joseph CodeMS : JOSS92030104