TP3 : Classification d'images par CNN

Raport

JIN Pin
Zhang Yuheng

Palaiseau, le 21 mars 2024

1 Structuration des données

1.1 Q1

En apprentissage automatique, la division d'un jeu de données en trois sous-ensembles distincts (training, validation, test) est une pratique très courante.
Training Set : Le sous-ensemble d'entraînement est utilisé pour entraîner le modèle, c'est-à-dire pour permettre au modèle d'apprendre à partir des données. C'est sur cet ensemble que le modèle ajuste ses paramètres pour pouvoir faire des prédictions.
Validation Set : Une fois que le modèle a été entraîné, il est testé sur le sous-ensemble de validation. Ce processus est utile pour régler les hyperparamètres du modèle et pour fournir une évaluation de la performance du modèle pendant ou après l'entraînement. Le but est de détecter le surapprentissage, c'est-à-dire lorsque le modèle apprend trop bien les détails et les bruits du jeu d'entraînement au détriment de sa capacité à généraliser à de nouvelles données.
Test Set : Le sous-ensemble de test est utilisé pour évaluer la performance finale du modèle. Il est crucial que le modèle n'ait jamais vu les données de test pendant l'entraînement, afin que ce sous-ensemble puisse fournir une estimation non biaisée de la performance du modèle sur de nouvelles données.
Diviser les données de cette manière aide à évaluer la capacité du modèle à généraliser au-delà de l'ensemble d'entraînement, à éviter le surapprentissage et à garantir que les mesures de performance sont représentatives de la capacité du modèle à traiter des données non vues.

1.2 Q2

La fonction **sandardize** réalise la standardisation de l'image. Plus précisément, la fonction calcule d'abord une moyenne par image et l'écart-type de chaque image, puis utilise les deux résultats calculés pour convertir les images initiales aux images standardisées selon la formule :

$$img_{standard} = \frac{img - img_{mean}}{img_{std}}$$

Les intérêts de la fonction 'standardize' :
1) **Réduction des biais** : En normalisant, on peut réduire les biais apportés par les différents éclairages et contrastes dans les images, qui peuvent introduire des biais dans le modèle d'apprentissage.

2) **Accélération de l'apprentissage** : La normalisation des caractéristiques peut aider l'algorithme de descente de gradient à converger plus rapidement vers le minimum.
3) **Amélioration de la performance** : La normalisation permet d'améliorer les performances du modèle en assurant que toutes les caractéristiques soient sur une échelle comparable, ce qui est particulièrement bénéfique pour les modèles sensibles à l'échelle des données d'entrée.

2 Architecture du réseau

2.1 Q1

"Trainable params" se réfère aux paramètres qui peuvent être ajustés pendant le processus d'entraînement, ce sont les poids et les biais que le réseau de neurones peut apprendre à partir des données d'entraînement. Leur nombre est généralement calculé en additionnant tous les paramètres des couches entraînables du réseau de neurones.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 8)         224

 dropout (Dropout)           (None, 32, 32, 8)         0

 max_pooling2d (MaxPooling2  (None, 16, 16, 8)         0
 D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 64)                131136

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 10)                650

 dropout_2 (Dropout)         (None, 10)                0

=================================================================
Total params: 132010 (515.66 KB)
Trainable params: 132010 (515.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

FIGURE 1 – Paramètres dans le modèle.

Dans le modèle CNN que l'on utilise (Fig. 1), le nombre total de paramètres pouvant être entraînés est de 132010.

3 Apprentissage

3.1 Q1

Epoch (Époque) : Une époque correspond à un passage complet de l'ensemble des données d'entraînement à travers le réseau neuronal une fois. Autrement dit, une époque est un cycle d'apprentissage complet où chaque échantillon a été utilisé pour la mise à jour des poids du réseau.
Step (Étape) : Une étape correspond à une passe en avant et une passe en arrière effectuées avec un lot de données. Lors d'une étape, les paramètres du réseau (poids et biais) sont mis à jour en fonction des données de ce lot.
Batch (Lot) : Un lot est un sous-ensemble des données d'entraînement utilisé pour une étape d'entraînement. La taille du lot (batch size) est le paramètre qui détermine combien d'échantillons d'entraînement sont inclus dans un lot.
Généralement, une époque contient généralement plusieurs étapes, dont le nombre dépend de la taille du lot et de la taille totale de l'ensemble des données d'entraînement.

3.2 Q2

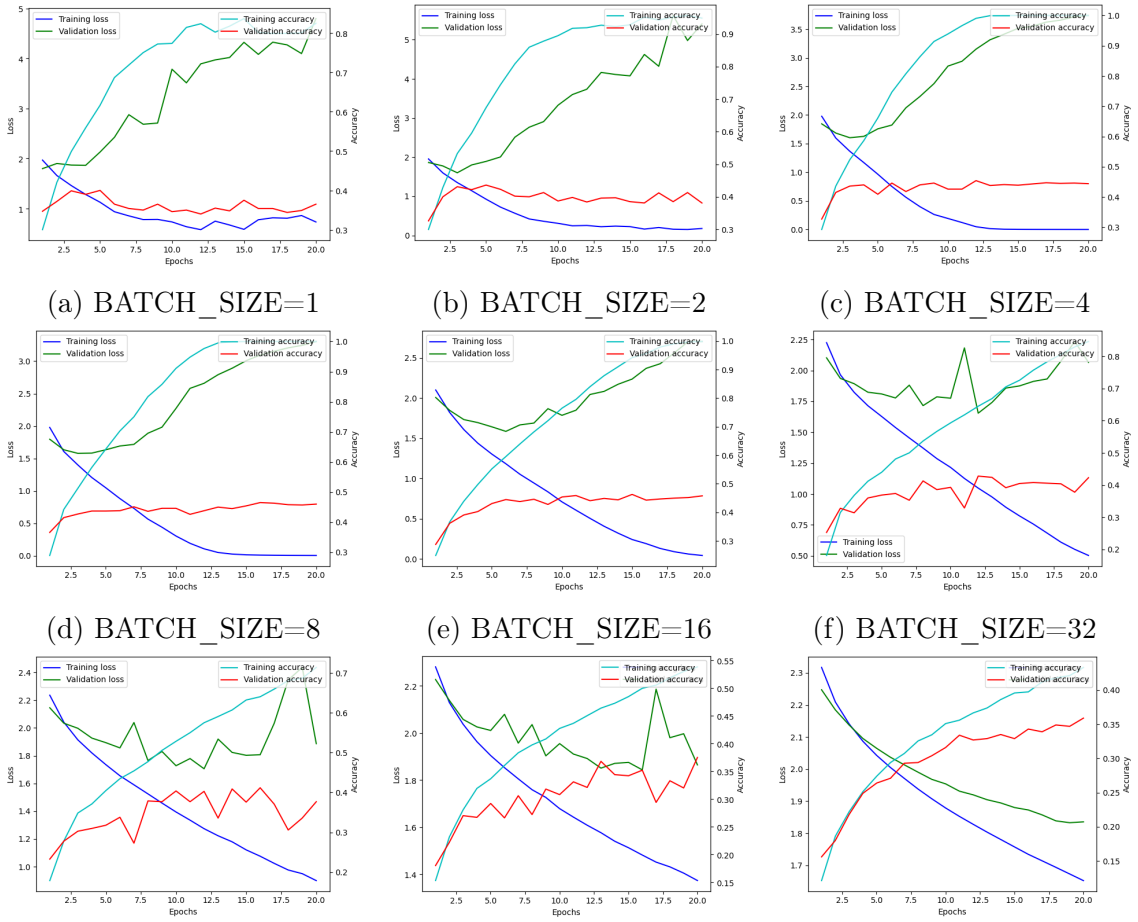
Nous choisissons BATCH_SIZE=[1, 2, 4, 8, 16, 32, 64, 128, 256] et analysons en fonction des résultats de formation du modèle correspondant. Voici l'effet de la taille de BATCH_SIZE sur le temps de calcul d'une étape et d'une époque.

Table 1 – Effet de la taille de BATCH_SIZE.

Taille de BATCH_SIZE	Temps d'une étape (ms)	Temps d'une époque (s)
1	2.65	13.26
2	3.42	8.54
4	4.59	5.74
8	5.77	3.61
16	8.67	2.71
32	13.55	2.13
64	22.69	1.79
128	36.91	1.48
256	69.29	1.39

En analysant les données dans le tableau, nous pouvons observer que le temps nécessaire pour terminer une étape augmente globalement avec l'augmentation de la

taille de BATCH_SIZE, tandis que le temps nécessaire pour terminer une époque diminue. Avec l'augmentation de la taille du lot, chaque étape traite plus de données et nécessite donc plus de temps pour effectuer les calculs. Cependant, bien que chaque étape prenne plus de temps, le nombre d'étapes nécessaires pour traiter tout l'ensemble de données diminue, réduisant ainsi le temps total par époque.
En même temps, nous analysons et comparons les changements dans la courbe d'entraînement du modèle sous ces tailles de BATCH_SIZE dans Fig. 2..



(a) BATCH_SIZE=1 (b) BATCH_SIZE=2 (c) BATCH_SIZE=4

(d) BATCH_SIZE=8 (e) BATCH_SIZE=16 (f) BATCH_SIZE=32

(g) BATCH_SIZE=64 (h) BATCH_SIZE=128 (i) BATCH_SIZE=256

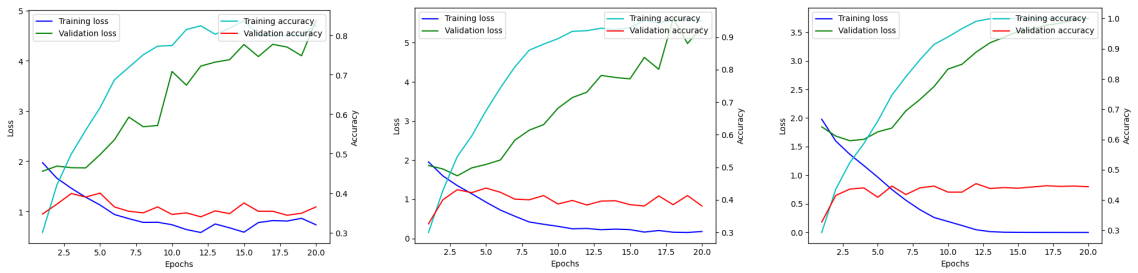
FIGURE 2 – Changements des courbes selon les tailles de BATCH_SIZE.

On trouve que la précision sur l'ensemble d'entraînement et sur l'ensemble de validation qui augmente puis diminue avec l'augmentation de la taille du lot.
Avec une petite taille de lot, le gradient calculé à chaque mise à jour est basé sur moins de données, ce qui entraîne une grande variance et du bruit dans l'estimation du gradient, pouvant conduire à des mises à jour des poids instables. En même temps, si le lot est trop petit, le modèle peut surajuster les quelques échantillons à

chaque mise à jour, ce qui peut conduire à une diminution de la précision au final. D'autre part, une plus grande taille de lot peut réduire la variance du gradient, rendant les mises à jour des poids plus stables et précises. Cette stabilité initiale peut entraîner une meilleure précision. Cependant, si la taille du lot est trop grande, le modèle peut ne pas avoir suffisamment de mises à jour pour apprendre complètement les données, et la grande taille de lot peut réduire la capacité du modèle à sortir des minima locaux, ce qui entraîne une convergence vers une solution suboptimale et donc une diminution de la précision.
En tout, choisir la taille de lot appropriée est une question d'équilibre entre performance et efficacité. Une taille de lot plus petite peut offrir des mises à jour du modèle plus fréquentes et une convergence potentiellement plus rapide, mais augmente le temps de calcul pour chaque époque ; tandis qu'une taille de lot plus grande peut traiter les données plus efficacement, à condition que la mémoire et les ressources matérielles le permettent, mais peut nécessiter davantage d'époques pour la convergence du modèle.

3.3 Q3

Nous avons changé le type d'optimiseur et tracé les courbes associées. Les images sont présentées ci-dessous (Fig. 3).



(a) 'SGD' (b) 'SGD avec Momentum' (c) 'Adam'

FIGURE 3 – Changements des courbes selon l'optimiseur.

Après comparaison, on trouve que les trois optimiseurs ont leurs caractéristiques.
- **SGD (Descente de Gradient Stochastique)** : C'est l'optimiseur le plus basique. Le SGD met à jour les poids en utilisant seulement un échantillon (ou un mini-lot) à la fois. Cela signifie que le modèle est mis à jour fréquemment, ce qui peut permettre un apprentissage rapide, mais peut aussi conduire à beaucoup de bruit et de fluctuations sur le chemin vers l'optimum global.

- **SGD avec Momentum** : Le terme de momentum aide l'optimiseur à maintenir la cohérence de la direction pendant la mise à jour, réduisant ainsi les fluctuations et pouvant accélérer la convergence de SGD.
- **Adam (Estimateur de Moment Adaptatif)** : Il combine les avantages de Momentum et RMSprop (Root Mean Square Propagation), en tenant compte non seulement de la moyenne décroissante exponentielle des gradients passés, comme le fait Momentum, mais aussi de la moyenne décroissante exponentielle des carrés des gradients passés. Cela permet de régler le taux d'apprentissage, calculant un taux d'apprentissage adaptatif indépendant pour chaque poids. Cette méthode converge souvent plus rapidement et est plus robuste par rapport au choix du taux d'apprentissage initial.

4 Hyperparamètres

4.1 Q1

Nous montrons ci-dessous tout le code lié à la définition du modèle.

```
######################### Code de Modèle ##########################
model = Sequential()
model.add(Conv2D(filters=8,
                 kernel_size = (3, 3),
                 activation = 'relu',
                 padding = 'same',
                 input_shape = (32, 32, 3),
                 kernel_regularizer = l2(0.00)))
model.add(Dropout(0.0))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(64, activation='relu',
                    kernel_regularizer = l2(0.00)))
model.add(Dropout(0.0))
model.add(Dense(10, activation='softmax',
                    kernel_regularizer = l2(0.00)))
model.add(Dropout(0.0))
weights_init = model.get_weights()
```

```
opt = SGD(lr=0.01,momentum=0.0)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['acc'])
print(model.summary())
class TimeHistory(Callback):
    def on_train_begin(self, logs={}):
        self.times = []
    def on_epoch_begin(self, batch, logs={}):
        self.epoch_time_start = time.time()
    def on_epoch_end(self, batch, logs={}):
        self.times.append(time.time() - self.epoch_time_start)
time_callback = TimeHistory()
filepath = "my_model.h5"
checkpoint = ModelCheckpoint(filepath,
                             monitor='val_acc',
                             verbose=1,
                             save_best_only=True,
                             mode='max',
                             period=2)
callbacks = [time_callback, checkpoint]
history = model.fit(x_train, y_train,
                    batch_size=32,
                    epochs=20,
                    verbose=1,
                    validation_data=(x_val, y_val),
                    callbacks=callbacks)
```

À partir du code, nous pouvons voir les hyperparamètres pertinents :
- **filters=8** : Dans la couche Conv2D, il s'agit du nombre de filtres de convolution, qui détermine la profondeur des cartes de caractéristiques. Un plus grand nombre de filtres peut capturer plus de caractéristiques, mais cela augmente également la complexité des calculs et le nombre de paramètres du modèle.
- **kernel_size = (3, 3)** : Également dans la couche Conv2D, il s'agit de la taille des noyaux de convolution, qui affecte la taille de la zone de capture des caractéristiques. Un noyau de convolution plus grand pourrait capturer des caractéristiques plus

larges, mais pourrait aussi entraîner plus de calculs.
- **padding = 'same'** : Il détermine si des zéros doivent être ajoutés autour des données pendant la convolution, de sorte que la dimension de la carte de caractéristiques de sortie soit la même que celle de l'entrée. Cela aide à préserver la dimension des cartes de caractéristiques, permettant des structures de réseau plus profondes.
- **kernel_regularizer = l2(0.00)** : Paramètre de régularisation L2, utilisé pour prévenir le surajustement. Si une valeur non nulle était définie, cela pourrait aider le modèle à généraliser sur des données non vues.
- **Dropout(0.0)** : Le taux dans la couche Dropout, utilisé pour réduire le surajustement. Pendant l'entraînement, il "abandonne" aléatoirement une partie des sorties des neurones, comme un moyen de prévenir le surajustement.
- **Dense(64, ...)** : Le nombre de neurones dans la première couche Dense, qui définit la largeur de la couche. Plus de neurones peuvent fournir des fonctions plus complexes, mais pourraient aussi conduire à un surajustement du modèle.
- **Dense(10, ...)** : Le nombre de neurones dans la couche de sortie, qui correspond généralement au nombre de classes dans une tâche de classification. Ce n'est pas un hyperparamètre réglable.
- **SGD(lr=0.01, momentum=0.0)** : SGD est l'optimiseur de descente de gradient stochastique, où lr est le taux d'apprentissage qui contrôle l'amplitude de la mise à jour des poids. Le momentum aide à accélérer SGD dans la bonne direction et à éviter les oscillations excessives.
- **batch_size=32** : Le nombre d'échantillons par lot pendant l'entraînement. Un plus grand lot peut améliorer l'utilisation de la mémoire et l'efficacité des calculs, mais pourrait également nécessiter plus de temps pour converger et affecter potentiellement la performance finale du modèle.
- **epochs=20** : Le nombre de fois que le jeu de données complet est parcouru pendant l'entraînement. Trop peu d'époques peuvent conduire à un entraînement insuffisant du modèle, tandis que trop peuvent conduire à un surajustement.
- **activation='relu'** et **activation='softmax'** : La fonction d'activation détermine la transformation non linéaire dans le réseau neuronal. relu est généralement utilisé pour les couches cachées, tandis que softmax est utilisé pour la couche de sortie dans les problèmes de classification multiple.
- **loss='categorical_crossentropy'** : La fonction de perte que le processus d'optimisation cherche à minimiser, elle quantifie l'écart entre les prédictions du modèle et les étiquettes réelles.

- **metrics=['acc']** : Les métriques utilisées pour évaluer la performance du modèle, elles reflètent directement l'efficacité du modèle pour une tâche spécifique.

5 Approfondissement du modèle

5.1 Q1

On essaye d'améliorer les performances du modèle en approfondissant le réseau à plusieurs couches de convolution. Bien que l'augmentation du nombre de couches ne semble pas avoir un grand impact sur la précision du modèle sur l'ensemble de validation, nous pensons que le modèle fonctionne relativement mieux lorsque nous ajoutons trois couches convolutives au modèle. La figure ci-dessous est la courbe d'entraînement du modèle à ce moment.

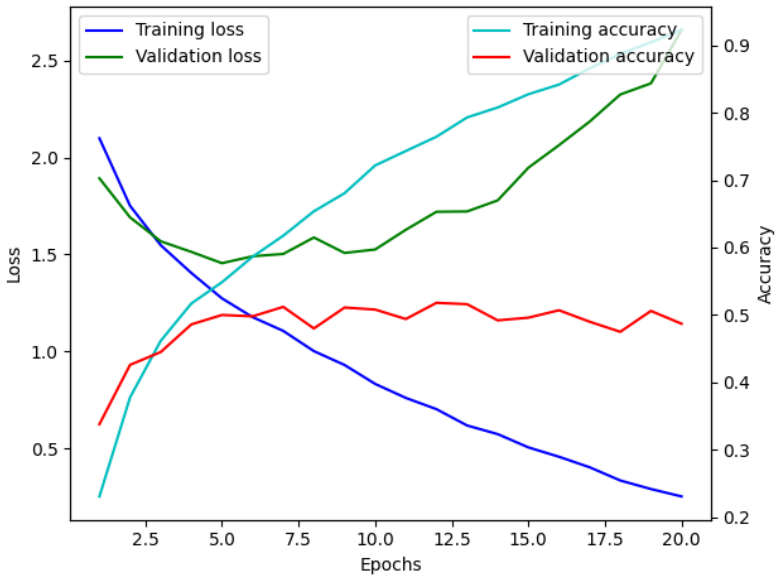


FIGURE 4 – Résultat de couches de convolution.

5.2 Q2

Nous avons utilisé la méthode de recherche en grille pour les hyperparamètres afin d'obtenir le modèle avec la plus haute précision sur l'ensemble de validation. Voici les contenus relatifs au meilleur modèle que nous avons obtenu.
Nous précisons les hyperparamètres utilisés dans le modèle.

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 32, 32, 8)	224
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 8)	0
conv2d_4 (Conv2D)	(None, 16, 16, 16)	1168
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 16)	0
conv2d_5 (Conv2D)	(None, 8, 8, 32)	4640
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 32)	0
flatten_1 (Flatten)	(None, 512)	0
dense_2 (Dense)	(None, 64)	32832
dropout_1 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 10)	650

Total params: 39514 (154.35 KB)
Trainable params: 39514 (154.35 KB)
Non-trainable params: 0 (0.00 Byte)

FIGURE 5 – Structure du modèle.

- Les hyperparamètres de la couche de base volumique à trois couches sont : filters=8, 16 et 32, kernel_size = (3, 3), padding = 'same', activation='relu', et kernel_regularizer = l2(0.00).
- Autres hyperparamètres : Dropout(0.0), Dense(64, ...), Adam(lr=0.001), batch_size=32, epochs=20, activation='softmax', loss='categorical_crossentropy', metrics=['acc'].

Nous dessinons les courbes d'entraînement du modèle optimal et les matrices de confusion correspondantes.

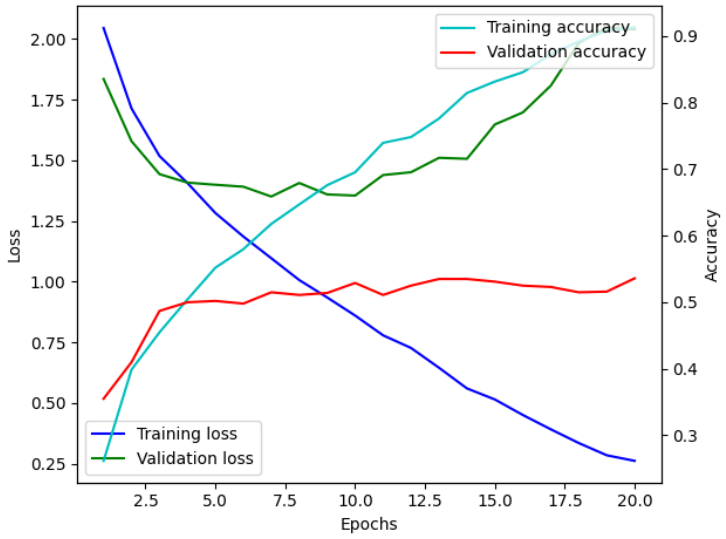


FIGURE 6 – les courbes d'entraînement.

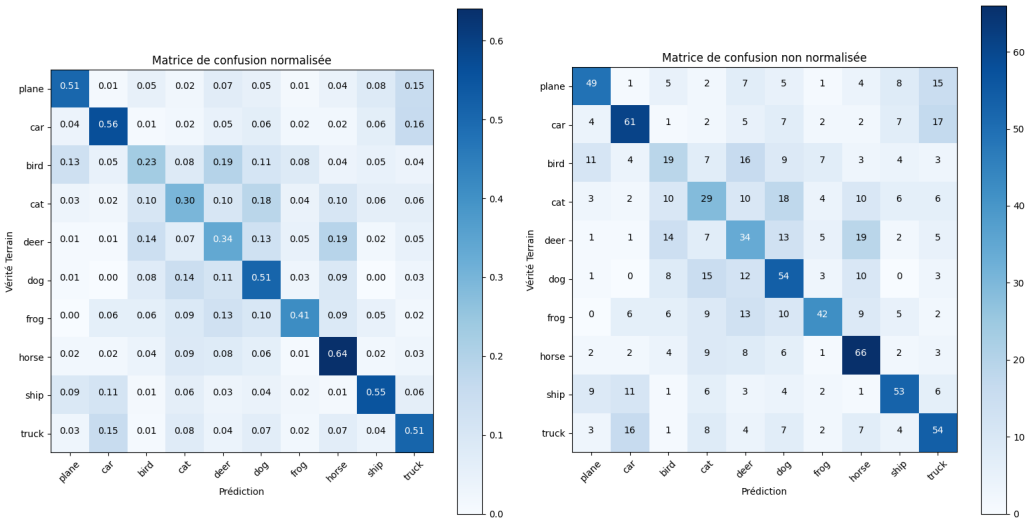


FIGURE 7 – Matrices de confusion correspondantes.

5.3 Q3

Après analyse, nous pensons que le modèle présente les avantages et inconvénients suivants.

Avantages :

1. Le modèle est assez complexe et complet, comprenant des couches de convolution, de pooling et des couches entièrement connectées, ce qui indique qu'il est capable de capter des caractéristiques de bas niveau à haut niveau.

2. Les courbes d'apprentissage montrent que la perte d'entraînement diminue continuellement avec le temps, tandis que la précision d'entraînement et la précision de validation augmentent, ce qui indique que le modèle apprend efficacement sur l'ensemble d'entraînement.

3. À partir de la matrice de confusion, on peut observer que le modèle performe bien dans la classification de la plupart des images, en particulier pour l'identification des images de chevaux.

Limitations :

1. Le phénomène de surajustement est apparu. Dans les dernières phases de l'entraînement, la fonction de perte sur l'ensemble de validation a commencé à augmenter. Bien que la précision d'entraînement soit élevée, la précision de validation, en comparaison, croît plus lentement. Néanmoins, la précision de classification des images sur l'ensemble de validation de ce modèle est toujours la plus élevée.

2. À partir de la matrice de confusion, on peut voir qu'il y a une confusion entre

certaines catégories, notamment les oiseaux, les chats et les chiens, ce qui indique que le modèle a toujours des défis à distinguer certaines catégories similaires.

6 Sur-apprentissage

6.1 Q1

Le sur-apprentissage est généralement indiqué par une diminution du coût d'entraînement au fil du temps, tandis que la courbe de coût de validation diminue initialement, puis augmente. Cela signifie que le modèle s'adapte trop spécifiquement aux données d'entraînement, perdant ainsi sa capacité à généraliser à de nouvelles données.
La figure ci-dessous est la courbe d'un modèle de sur-apprentissage évident.

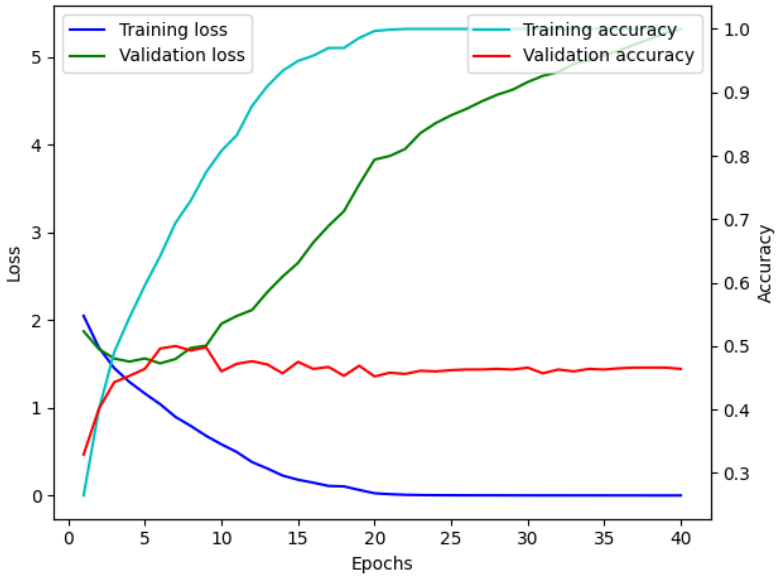


FIGURE 8 – Exemple de sur-apprentissage.

Nous avons constaté qu'après la 7ème époque, la fonction de perte du modèle sur l'ensemble d'apprentissage a continué à maintenir une tendance à la baisse, tandis que la fonction de perte sur l'ensemble de validation a commencé à croître. Dans le même temps, bien que la précision des prédictions du modèle sur l'ensemble d'apprentissage ait augmenté, la précision sur l'ensemble de validation a commencé à diminuer.
On peut donc penser qu'à partir de la septième époque, un surapprentissage se produit.

6.2 Q2

Plusieurs mécanismes peuvent aider à atténuer le phénomène de sur-apprentissage, tels que :
- **Régularisation** : Comme les régularisations L1 et L2 qui pénalisent la complexité du modèle pour réduire sa tendance à surajuster.
- **Early Stopping** : Surveiller la performance sur l'ensemble de validation pendant l'entraînement et arrêter l'entraînement lorsque la performance ne s'améliore plus ou commence à décliner, afin d'empêcher le modèle d'apprendre excessivement des données d'entraînement.
- **Dropout** : L'élimination aléatoire (mise à zéro) d'une partie des neurones pendant l'entraînement peut réduire les adaptations complexes et coadaptatives entre les couches, forçant le réseau à apprendre des caractéristiques plus robustes.
- **Batch Normalization** : La normalisation des entrées de chaque couche peut accélérer le processus d'entraînement, améliorer la stabilité du modèle et contribuer à prévenir le surapprentissage.

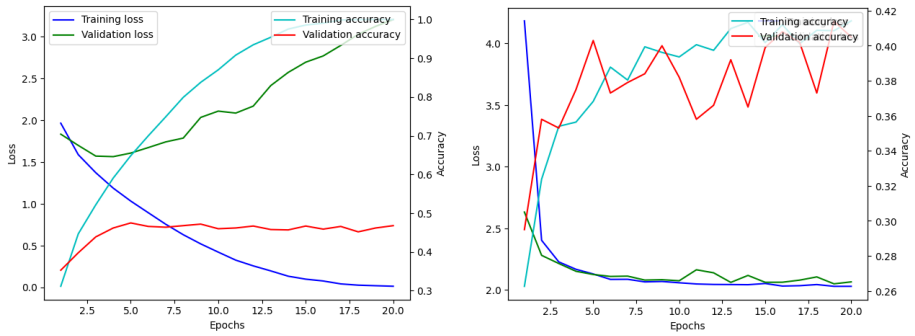
Voici quelques exemples de photos comparatifs montrant le fonctionnement de ces technologies. Nous montrons des images sans aucun traitement, en ajoutant respectivement un mécanisme de régularisation, un mécanisme d'abandon et un mécanisme de BatchNormalization.

À partir de la figure, nous pouvons constater que par rapport à la courbe originale : 1) la fonction de perte du modèle ajoutant le mécanisme de régularisation et le mécanisme d'abandon sur l'ensemble de vérification continue de diminuer ; 2) le modèle ajoutant la couche Batch Normalization ne parvient pas à empêcher complètement le phénomène de surajustement, mais atténuant également la croissance de la fonction de perte.

7 Cartes d'activation

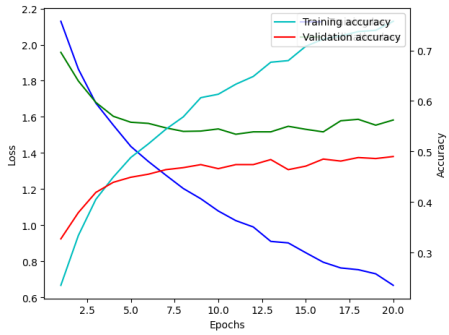
7.1 Q1

Les cartes d'activation représentent la réponse d'une certaine couche d'un réseau neuronal aux données d'entrée, montrant comment les caractéristiques apprises par cette couche sont activées dans des régions locales de l'image d'entrée. Dans une réseau de neurones convolutif (CNN), la première couche est généralement responsable de l'extraction des caractéristiques de bas niveau, telles que les bords, les coins, les

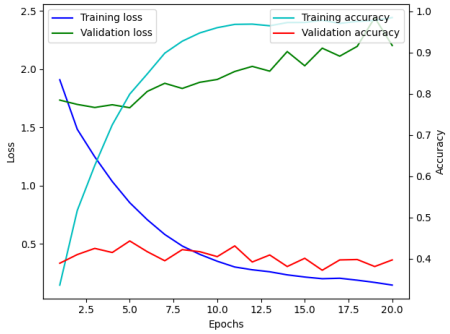


(a) sans mécanisme

(b) avec régularisation



(c) avec dropout



(d) avec batch normalization

FIGURE 9 – Amélioration par les mécanisme pour empêcher le surapprentissage.

couleurs et les textures, et ces cartes d'activation peuvent fournir des informations intuitives sur la manière dont le réseau extrait ces caractéristiques primaires des images.

Dans notre code, les résultats de la prédiction du modèle réduit, après avoir été traités par la fonction d'activation ReLU, sont sommés sur la dimension des canaux pour obtenir l'intensité totale d'activation de chaque carte. En normalisant ces intensités, une masque est généré qui peut être superposé sur l'image d'entrée originale pour révéler les régions d'entrée auxquelles la première couche répond le plus fortement.

7.2 Q2

Puisque notre modèle utilise 3 couches convolutives à la fin, nous redessinons les cartes d'activation correspondant aux deuxième et troisième couches.

Nous constatons qu'à mesure que le nombre de couches augmente, la résolution de chaque carte d'activation diminue et devient de plus en plus floue.

Après plusieurs couches de convolution et de pooling, la résolution spatiale de l'image peut diminuer. Les couches de pooling sont spécifiquement conçues pour réduire les

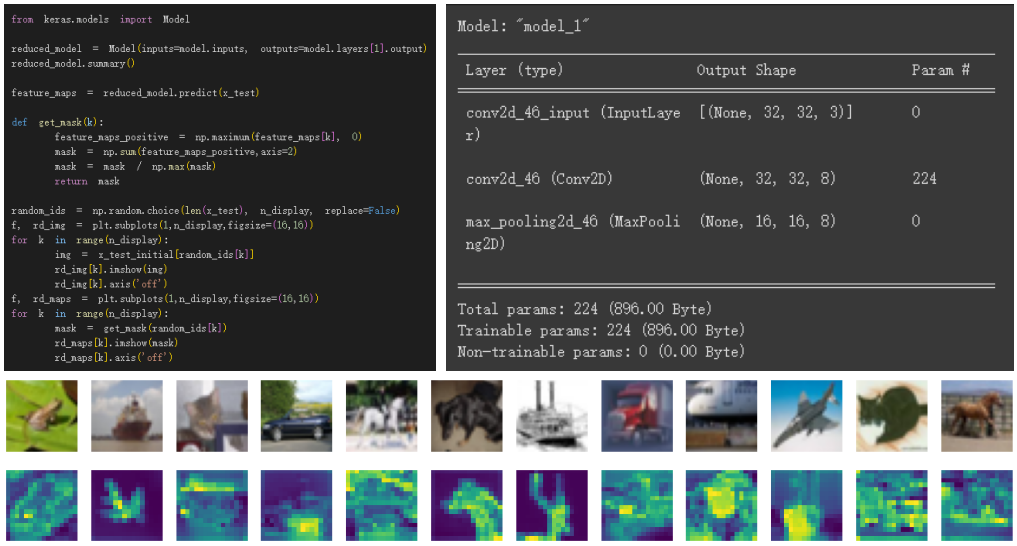


FIGURE 10 – Carte d'activation de la première couche.

dimensions des cartes de caractéristiques, et elles réduisent généralement la taille de ces cartes en effectuant un sous-échantillonnage des pixels adjacents.
Dans les couches plus profondes du réseau, le champ récepteur de chaque neurone est plus grand. Par conséquent, chaque activation correspondra probablement à une région plus grande de l'image d'entrée, plutôt qu'à un seul pixel.
Les réseaux profonds transforment l'entrée originale en représentations abstraites de plus haut niveau à chaque couche. Cela signifie qu'avec l'augmentation du nombre de couches, chaque couche tente de capturer des représentations de données de plus en plus complexes, rendant les détails au niveau des pixels initiaux moins évidents.

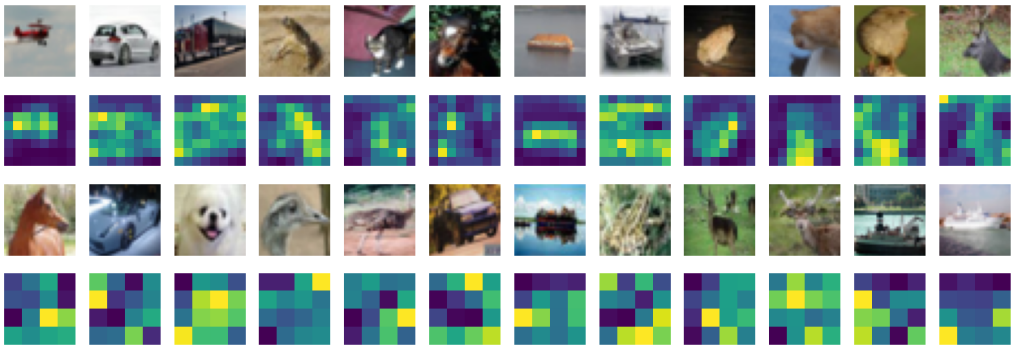


FIGURE 11 – Carte d'activation de la deuxième et troisième couche.