CNN pour Classifier CIFAR-10

Dhleima Mohamed

December 2021

1 Introduction

1.1 Présentation du Ensemble de données de classification des photos CIFAR-10

L'ensemble de données de classification des photos CIFAR-10 est un sous-ensembles étiquetés de l'ensemble de données de 80 millions de minuscules images . Ils ont été collectés par Alex Krizhevsky, Vinod Nair et Geoffrey Hinton.

L'ensemble de données CIFAR-10 se compose de 60 000 images couleur 32x32 dans 10 classes, avec 6 000 images par classe. Il y a 50 000 images d'entraînement et 10 000 images de test. L'ensemble de données est divisé en cinq lots d'apprentissage et un lot de test, chacun avec 10 000 images. Le lot de test contient exactement 1000 images sélectionnées au hasard dans chaque classe. Les lots de formation contiennent les images restantes dans un ordre aléatoire, mais certains lots de formation peuvent contenir plus d'images d'une classe que d'une autre. Entre eux, les lots de formation contiennent exactement 5000 images de chaque classe.

1.2 Versions

. . .Ils ya toris versions de CIFAR-10 . . .

— Version python CIFAR-10 163 Mo c58f30108f718f92721af3b95e74349a ;
— Version Matlab CIFAR-10 175 Mo 70270af85842c9e89bb428ec9976c926 ;
— Version binaire CIFAR-10 (convient pour C) 162 Mo c32a1d4ab5d03f1284b67883e8d87530.

1.3 Versions Python3

— • data – un tableau numpy 10000x3072 de uint8 s. Chaque ligne du tableau stocke une image couleur 32x32. Les 1024 premières entrées contiennent les valeurs de canal rouges, les 1024 suivantes le vert et les 1024 finales le bleu. L'image est stockée dans l'ordre des lignes principales, de sorte que les 32 premières entrées du tableau sont les valeurs de canal rouge de la première ligne de l'image ;
— • labels – une liste de 10 000 numéros dans la plage 0-9. Le nombre à l'index i indique l'étiquette de la i ème image dans les données du tableau.

2 Probléme de classification des petites photos

Le problème de classification de petites photos CIFAR-10 est un ensemble de données standard utilisé en vision par ordinateur et en apprentissage en profondeur. Bien que l'ensemble de données soit efficacement résolu, il peut être utilisé comme base pour apprendre et pratiquer comment développer, évaluer et utiliser des réseaux de neurones d'apprentissage en profondeur convolutifs pour la classification d'images à partir de zéro.

Cela comprend comment développer un harnais de test robuste pour estimer les performances du modèle, comment explorer les améliorations du modèle et comment enregistrer le modèle et le charger plus tard pour faire des prédictions sur de nouvelles données.

3 Résolution des problèmes à l'aide CNN

Au lieu de passer en revue la littérature sur les modèles performants sur l'ensemble de données, nous pouvons développer un nouveau modèle à partir de zéro.

L'ensemble de données a déjà un ensemble de données d'entraînement et de test bien défini que nous utiliserons. Une alternative pourrait être d'effectuer une validation croisée k-fold avec ak=5 ou k=10. Ceci est souhaitable si les ressources sont suffisantes. Dans ce cas, et dans l'intérêt de garantir que les exemples de ce tutoriel s'exécutent dans un délai raisonnable, nous n'utiliserons pas la validation croisée k-fold

3.1 Charger l'ensemble de données

(trainX, trainY), (testX, testY) = cifar10.load$_data$()

3.2 Encodage des lables

trainY = to$_categorical(trainY)$
testY = to$_categorical(testY)$

3.3 Préparer les données de pixels

Nous ne connaissons pas la meilleure façon de mettre à l'échelle les valeurs de pixels pour la modélisation, mais nous savons qu'une certaine mise à l'échelle sera nécessaire.

Un bon point de départ est de normaliser les valeurs de pixels, par exemple de les remettre à l'échelle dans la plage [0,1].

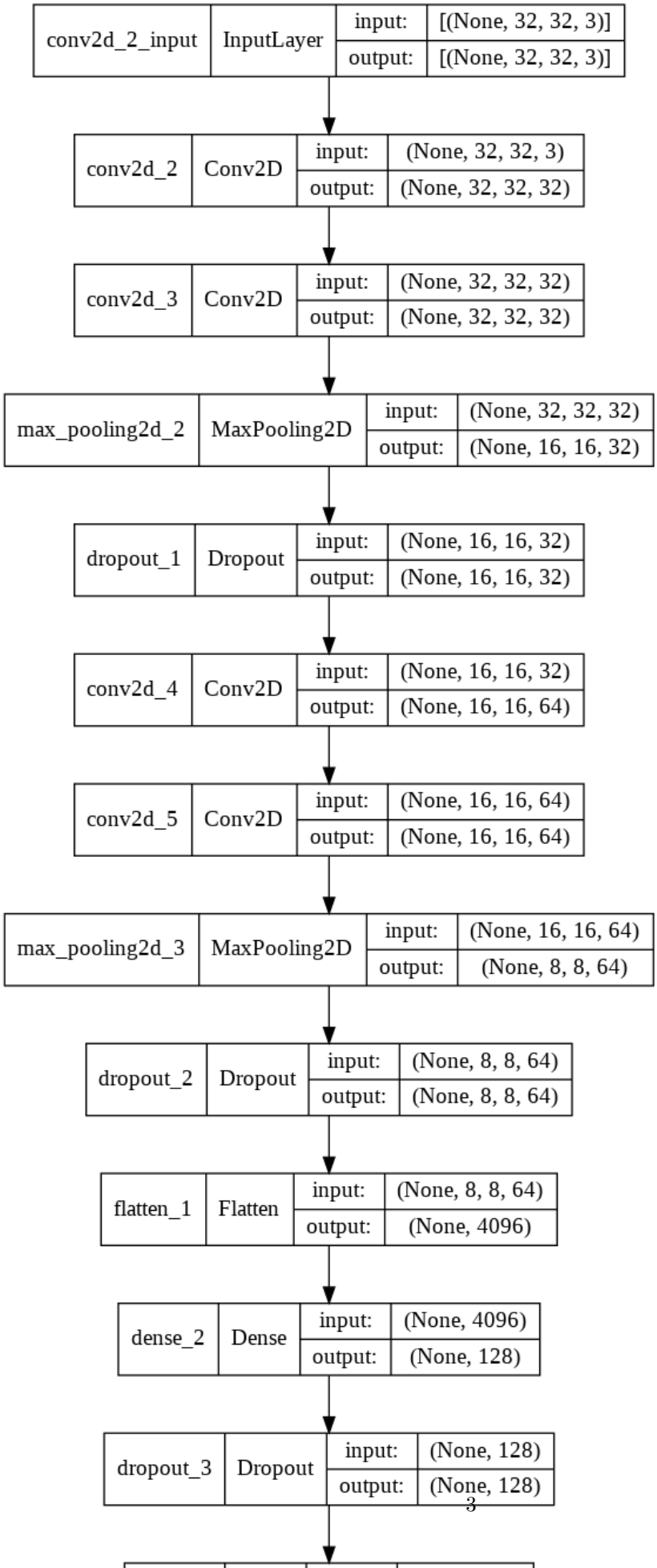
Cela implique d'abord de convertir le type de données d'entiers non signés en nombres flottants, puis de diviser les valeurs de pixels par la valeur maximale.

train$_norm = train.astype('float32')$
test$_norm = test.astype('float32')$
normalize to range 0-1
train$_norm = train_norm/255.0$
test$_norm = test_norm/255.0$

4 Développer un modèle Sequential

Nous avons besoin d'un moyen de créer un modèle de réseau neuronal. L'instriction ci-dessous définira ce modèle et peut être remplie ou remplacée pour une configuration de modèle donnée que nous souhaitons évaluer ultérieurement.

from keras.models import Sequential
model = Sequential()
. . ..

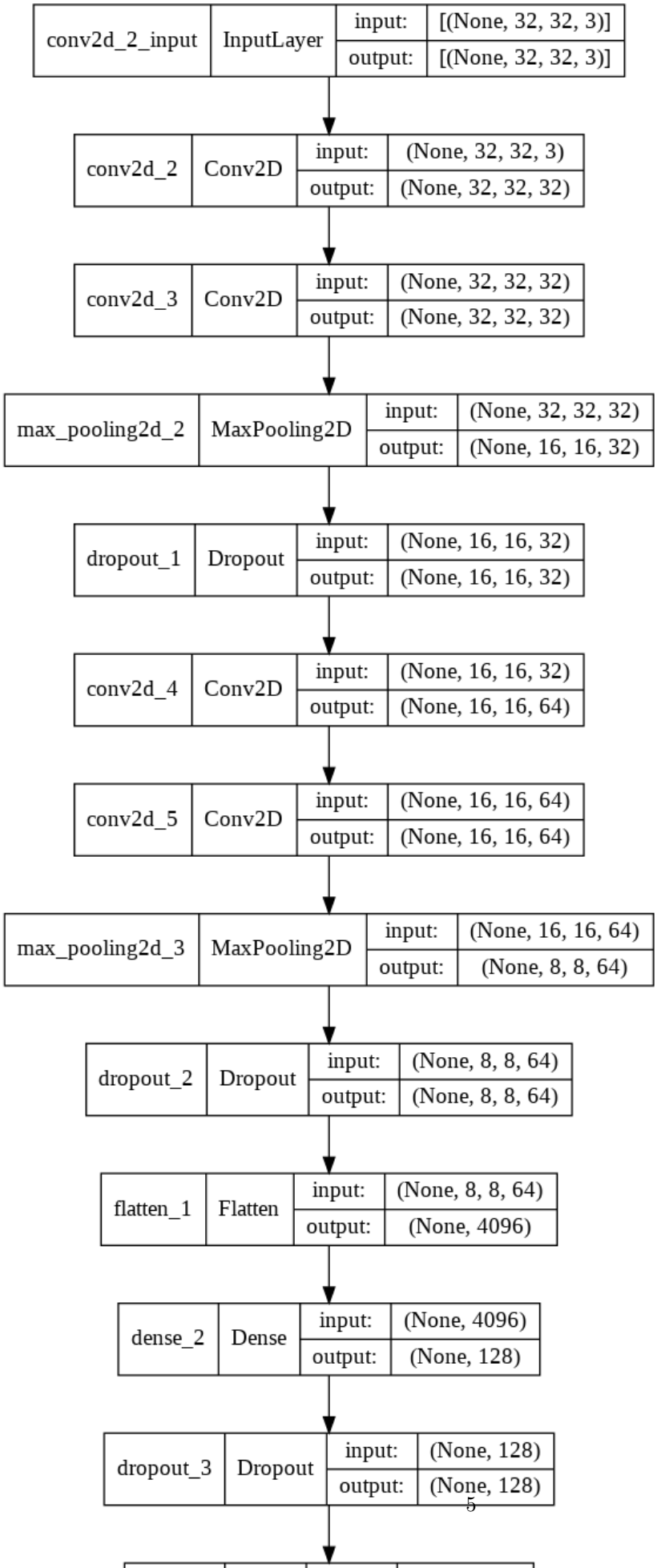

conv2d_2_input
InputLayer
input: [(None, 32, 32, 3)]
output: [(None, 32, 32, 3)]
conv2d_2
Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 32)
conv2d_3
Conv2D
input: (None, 32, 32, 32)
output: (None, 32, 32, 32)
max_pooling2d_2
MaxPooling2D
input: (None, 32, 32, 32)
output: (None, 16, 16, 32)
dropout_1
Dropout
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
conv2d_4
Conv2D
input: (None, 16, 16, 32)
output: (None, 16, 16, 64)
conv2d_5
Conv2D
input: (None, 16, 16, 64)
output: (None, 16, 16, 64)
max_pooling2d_3
MaxPooling2D
input: (None, 16, 16, 64)
output: (None, 8, 8, 64)
dropout_2
Dropout
input: (None, 8, 8, 64)
output: (None, 8, 8, 64)
flatten_1
Flatten
input: (None, 8, 8, 64)
output: (None, 4096)
dense_2
Dense
input: (None, 4096)
output: (None, 128)
dropout_3
Dropout
input: (None, 128)
output: (None, 128)

5 Évaluer le modèle

Une fois le modèle défini, nous devons l'adapter et l'évaluer. L'ajustement du modèle nécessitera que le nombre d'époques d'apprentissage et la taille du lot soient spécifiés.

je utiliser pour chaque execution une époche different aux ces qu'il precedent et batch$_size$64.

Dans la premier fois je testai un seul couche convolutive puit ajoute en autre et estimer une bonnes precision, finnalement j'ajoute des couches Dropout après chaque couche de max pooling et après la couche full connected, et utiliser un taux Dropout fixe de 20


conv2d_2_input
InputLayer
input: [(None, 32, 32, 3)]
output: [(None, 32, 32, 3)]
conv2d_2
Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 32)
conv2d_3
Conv2D
input: (None, 32, 32, 32)
output: (None, 32, 32, 32)
max_pooling2d_2
MaxPooling2D
input: (None, 32, 32, 32)
output: (None, 16, 16, 32)
dropout_1
Dropout
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
conv2d_4
Conv2D
input: (None, 16, 16, 32)
output: (None, 16, 16, 64)
conv2d_5
Conv2D
input: (None, 16, 16, 64)
output: (None, 16, 16, 64)
max_pooling2d_3
MaxPooling2D
input: (None, 16, 16, 64)
output: (None, 8, 8, 64)
dropout_2
Dropout
input: (None, 8, 8, 64)
output: (None, 8, 8, 64)
flatten_1
Flatten
input: (None, 8, 8, 64)
output: (None, 4096)
dense_2
Dense
input: (None, 4096)
output: (None, 128)
dropout_3
Dropout
input: (None, 128)
output: (None, 128)

6 Présentation des resultat

nous pouvons constater une augmentation de la précision de la classification d'environ 13 En examinant la courbe d'apprentissage du modèle, nous pouvons voir que le surapprentissage a été résolu. Le modèle converge bien pendant environ 40 ou 50 époques, auquel cas il n'y a plus d'amélioration sur l'ensemble de données de test.

Accuracy : ¿ 82.650

```
 # fit model
history = model.fit(trainX, trainY, epochs = Epochs, batch_size = 64, validation_data = (testX, testY), verbose=0)
  # evaluate model
_, acc = model.evaluate(testX, testY, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 82.650
```