Deep learning

RegNet

Date : 17/02/2025

Auteur : Nathan Verdier, Oussama Riahi
Programme : Master SETSIS
Université : Université Clermont-Ferrand

ÉCOLE UNIVERSITAIRE
DE PHYSIQUE ET D'INGÉNIERIE
Université Clermont Auvergne

Sommaire

Lien du projet

Vous pouvez trouver le projet sur GitHub en suivant ce lien github.

Résumé de l'article et contribution

L'article propose **RegNet**„ une nouvelle architecture pour la classification d'images inspirée de **ResNet**. Il introduit un **module régulateur** basé sur des réseaux récurrents **convolutionnels** (ConvRNN) pour extraire des caractéristiques complémentaires. Ce module agit comme une mémoire, permettant de capter la dépendance **spatio-temporelle** entre les blocs du réseau. L'approche vise à surmonter la limitation des connexions de raccourci qui favorisent la redondance des informations. Les auteurs explorent différentes variantes, notamment **ConvLSTM** et **ConvGRU**, pour implémenter le régulateur. Le module régulateur est facilement intégré en parallèle aux raccourcis existants dans diverses architectures basées sur ResNet. Des expérimentations sur **CIFAR-10, CIFAR-100 et ImageNet** montrent une amélioration significative de la précision de classification. **RegNet** permet également de réduire la profondeur du réseau tout en optimisant l'utilisation des paramètres. L'article présente une analyse détaillée de la fusion des caractéristiques et de la réutilisation de l'information entre les blocs.

Analyse de la méthode

Données utilisées

L'évaluation de RegNet a été réalisée sur trois jeux de données bien connus en classification d'images :
- **CIFAR-10** (10 classes, 50K images d'entraînement, 10K de test)
- **CIFAR-100** (100 classes, même structure que CIFAR-10)
- **ImageNet** (1.28M images d'entraînement, 50K de validation, 1000 classes)

Modèle et architecture

RegNet repose sur ResNet en y ajoutant un **module régulateur** sous la forme d'un **ConvRNN** inséré entre les blocs résiduels. Il existe deux variantes principales :
- **RegNet classique** : Ajout d'un ConvRNN dans chaque bloc résiduel.
- **Bottleneck RegNet** : Optimisation pour les modèles plus profonds, utilisant une architecture en goulot d'étranglement.

Optimisation et entraînement

Les modèles sont entraînés avec **SGD** (momentum 0.9, weight decay 1e-4), et une **réduction progressive du taux d'apprentissage**.
Des techniques classiques d'**augmentation de données** (recadrage, miroir, etc.) sont utilisées.

Résultats obtenus

- **CIFAR-10/100** : RegNet réduit significativement l'erreur par rapport à ResNet et SE-ResNet.
- **ImageNet** : RegNet-50 surpasse ResNet-50 et se rapproche des performances de ResNet-101 avec moins de calculs.
- **Meilleure efficacité paramétrique** : RegNet atteint de meilleures performances avec **moins de couches** qu'un ResNet classique.

Tests et évaluation

Objectif des tests

Évaluer la robustesse de RegNet sur la **classification d'images**, en comparant ses performances avec celles de ResNet et SE-ResNet.
Les critères testés :
- **Précision du modèle** (Top-1 et Top-5 accuracy)
- **Impact du module régulateur selon son positionnement**
- **Efficacité paramétrique** (gain en performance vs. coût en calculs)

Données utilisées

- **CIFAR-10/100** pour des tests sur de petites images
- **ImageNet** pour des images plus complexes et en haute résolution

Observations

- RegNet améliore la classification par rapport aux modèles classiques, grâce à une meilleure exploitation des dépendances spatiales et temporelles.
- L'ajout du module ConvRNN est **plus efficace sur les couches basses** du réseau.
- RegNet permet de **réduire la profondeur** nécessaire pour atteindre une précision donnée.

Problèmes rencontrés

- L'augmentation de la taille du modèle rend l'entraînement **plus coûteux en calcul**.
- Un ajustement fin des **hyperparamètres** (ex. taux d'apprentissage) est nécessaire pour éviter le surajustement.

Le code

Le code implémente les composants suivants :

- **ConvLSTMCell**
  Une cellule ConvLSTM qui traite des cartes de caractéristiques spatiales. La cellule concatène l'entrée courante avec l'état caché précédent, applique une convolution pour calculer les portes de l'LSTM, puis met à jour l'état de la cellule et l'état caché. Un contrôle d'erreur vérifie que les dimensions de l'entrée et de l'état caché correspondent.
- **RegNetBlock**
  Un bloc régulé combinant un bloc de convolution classique avec un module ConvLSTM. Ce bloc comprend :
  - Une première convolution avec gestion du stride, suivie d'une normalisation par lots et d'une activation ReLU.
  - Un module ConvLSTM (si activé) qui traite la sortie de la première convolution, dont la sortie est ensuite concaténée avec les caractéristiques convolutionnelles et fusionnée via une convolution 1×1 suivie d'une normalisation et d'une activation.
  - Une deuxième convolution avec normalisation par lots.

  - Une connexion résiduelle qui peut inclure un ajustement (downsampling) lorsque le nombre de canaux change ou lorsque le stride est différent de 1.

- **Architecture RegNet**
  Deux architectures sont proposées :

  - **RegNetCIFAR** : Conçue pour des images de 32×32 (ex. CIFAR-10). Elle comporte une couche d'entrée, trois groupes de RegNetBlocks, une couche de pooling adaptatif et une couche entièrement connectée pour la classification.

  - **RegNetImageNet** : Conçue pour des images de 224×224 (ex. ImageNet). Elle utilise une première convolution 7×7 suivie d'un max-pooling, puis quatre groupes de RegNetBlocks, un pooling adaptatif et une couche entièrement connectée.

- **Fonctions d'Entraînement et de Test**
  Les fonctions `train_model` et `test_model` gèrent la boucle d'entraînement et l'évaluation du modèle, en affichant la perte et la précision pendant l'entraînement.

- **Interface en Ligne de Commande**
  Le programme principal accepte des arguments pour sélectionner le jeu de données (CIFAR-10 ou ImageNet), spécifier le chemin des données, le nombre d'époques, la taille du batch et le taux d'apprentissage.

Prérequis

- Python 3.x
- PyTorch
- Torchvision
- argparse

Installez les packages requis avec pip :

```
pip install -r requirements.txt
```

Utilisation

Exécutez le programme principal avec les arguments désirés.

Execution :

```
python3 trainv2.py
```

Structure du Code

- **ConvLSTMCell**
  Implémente une cellule ConvLSTM qui concatène l'entrée avec l'état caché précédent, applique une convolution pour calculer les portes, et met à jour les états. Un contrôle d'erreur vérifie que les dimensions correspondent.
- **RegNetBlock**
  Ce bloc combine des couches convolutionnelles et un module ConvLSTM avec une connexion résiduelle. Il gère le changement de dimensions via le paramètre de stride et réalise la fusion des caractéristiques via une convolution 1×1.
- **RegNetCIFAR & RegNetImageNet**
  Ces classes construisent le réseau complet pour CIFAR-10 et ImageNet, respectivement, en empilant plusieurs RegNetBlocks. Dans RegNetCIFAR, pour chaque couche, les états caché et de la cellule sont réinitialisés pour éviter des conflits de dimensions.
- **Fonctions d'Entraînement et de Test**
  La fonction `train_model` exécute la boucle d'entraînement en mettant à jour les poids du modèle et en affichant la progression (perte et précision). La fonction `test_model` évalue la précision du modèle sur le jeu de test.
- **Programme Principal**
  Utilise `argparse` pour permettre la sélection du jeu de données et la configuration des hyper-paramètres tels que le nombre d'époques, la taille du batch et le taux d'apprentissage. Selon le jeu de données choisi, les transformations appropriées et les DataLoaders sont créés.

Remarques

- **Réinitialisation des États LSTM** :
  Pour chaque couche du réseau, les états caché et de la cellule sont réinitialisés afin d'éviter des conflits de dimensions lors du passage entre les couches.
- **Gestion des Erreurs** :
  La cellule ConvLSTM intègre une vérification des dimensions pour s'assurer que l'entrée et l'état caché sont compatibles, et renvoie une erreur descriptive si ce n'est pas le cas.

Conclusion générale

RegNet constitue une **amélioration significative** de ResNet en exploitant un module régulateur basé sur des **ConvRNNs**. Il permet d'**apprendre des caractéristiques complémentaires** et **d'améliorer la classification des images** tout en **réduisant le besoin de profondeur du réseau**. Ces résultats ouvrent la voie à son application dans d'autres architectures basées sur ResNet et d'autres tâches comme la détection d'objets et la super-résolution.