Rapport Final : Reproduction de SqueezeNet sur CIFAR-10

1. Présentation de l'Objectif et du Problème

Objectif de l'article :

L'article SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size a pour but de proposer une architecture de CNN (SqueezeNet) qui atteint une précision comparable à celle d'AlexNet sur la classification d'images tout en utilisant 50 fois moins de paramètres.

Problème adressé :

- Réduction de la taille du modèle : En minimisant le nombre de paramètres, l'architecture facilite l'entraînement distribué, la mise à jour « over-the-air » dans des systèmes embarqués (comme la conduite autonome) et le déploiement sur du matériel avec des contraintes en ressources (FPGA, ASIC, etc.).
- Maintien de la performance : Malgré la réduction importante de paramètres, le modèle cherche à conserver une performance de classification équivalente à celle des grands réseaux classiques.

2. Chargement et Prétraitement du Dataset

Chargement du dataset :
Pour ce projet, le dataset CIFAR-10 a été utilisé, car il est largement employé dans la communauté pour la classification d'images et permet d'adapter les architectures initialement conçues pour ImageNet à une échelle réduite.

Utilisation de la fonction tf.keras.datasets.cifar10.load_data() pour télécharger et charger les données d'entraînement et de test.

Transformations effectuées :

Normalisation des images :
Conversion des pixels en valeurs flottantes et division par 255 afin que les valeurs soient comprises entre 0 et 1.
Afin de faciliter la convergence lors de l'optimisation en s'assurant que l'échelle des données d'entrée est homogène.

Encodage des labels :
Passage au format one-hot via to_categorical pour être compatible avec la fonction de perte categorical cross-entropy.

Séparation Train/Validation/Test :
Le jeu CIFAR-10 initial contient 50 000 images pour l'entraînement et 10 000 pour le test.
Une fraction (environ 10 %) des données d'entraînement a été séparée pour constituer l'ensemble de validation.
Ces étapes sont la pour s'assurer que le modèle reçoit des données bien prétraitées de manière cohérente et optimisée pour l'entraînement.

3. Description de l'Architecture Proposée

L'architecture SqueezeNet s'appuie sur l'utilisation de plusieurs Fire Modules, qui permettent de réduire le nombre de paramètres en combinant deux étapes :

a. Le Fire Module

Couche Squeeze :
Utilise uniquement des filtres 1×1 (de type Conv2D) pour réduire la dimensionnalité en termes de canaux.

Exemple : Si le nombre de filtres dans l'expand est 64, alors on choisit souvent un squeeze de 16 (squeeze ratio ≈ 0.25 ou 0.125 selon la configuration d'origine).

Couche Expand :
Se décompose en deux branches :
Expand 1×1 : Convolution 1×1.
Expand 3×3 : Convolution 3×3 avec padding « same » pour conserver la dimension spatiale.
Les sorties de ces deux branches sont concaténées sur le canal.

Avantages :
La couche squeeze réduit le nombre de canaux en entrée aux couches 3×3, diminuant ainsi le coût en nombre de paramètres (stratégie 2).
L'utilisation majoritaire de 1×1 (stratégie 1) limite également le nombre de paramètres.

b. Organisation Globale du Réseau

Couche d'Entrée :
Une couche Conv2D de taille 3×3 pour traiter l'image d'entrée.
Exemple de taille d'entrée pour CIFAR-10 : (32, 32, 3)

Empilement des Fire Modules :
Plusieurs Fire Modules sont enchaînés.
Des couches de pooling (par exemple, MaxPooling2D) sont insérées après certains Fire Modules pour effectuer du downsampling.
Dans l'architecture originale, le downsampling est réalisé plus tard (stratégie 3) pour conserver des activations à haute résolution.

Sortie :
Après les Fire Modules, une dernière couche de convolution Conv2D réduit le nombre de canaux à celui du nombre de classes (pour CIFAR-10, 10).
Un GlobalAveragePooling2D est appliqué pour transformer les cartes d'activation en vecteur.
Une activation softmax fournit la distribution de probabilité finale.

c. Détails par couche
Voici un exemple récapitulatif simplifié pour notre version adaptée à CIFAR-10:

Couche	Type	Taille d'entrée	Taille de	Nbre de paramètres	Remarques
Conv1	Conv2D	(32, 32, 3)	(32, 32, 96)	≈ 3×3×3×96 = 2,592	Utilisation d'un
Pool1	MaxPooling	(32, 32, 96)	(16, 16, 96)	0	Pooling 2x2, stride
Fire Module 1	Fire Module	(16, 16, 96)	(16, 16, 128)	Variable (ex. ≈ 12,000)	Squeeze:16, Expand:64+64
Pool2	MaxPooling	(16, 16, 128)	(8, 8, 128)	0	
Fire Module 2	Fire Module	(8, 8, 128)	(8, 8, 256)	Variable	Augmentation progressive
...	...	...	...	...	Plusieurs Fire
ConvFinal	Conv2D	(taille spatiale, canaux)	(taille spatiale, 10)	Par exemple 1×1×NbCanaux×10	Convolution finale
GAP	GlobalAvgPool2D	(taille spatiale, 10)	(10)	0	Transformation finale
Softmax	Activation	(10)	(10)	0	Distribution de

4. Fonction de Perte et Optimiseur

Fonction de perte utilisée :
Categorical Cross-Entropy car ici, pour un problème de classification multi-classes comme CIFAR-10, la categorical cross-entropy mesure la différence entre la distribution prédite et la distribution réelle (one-hot), ce qui nous convient le mieux pour cette tâche.

Optimiseur choisi :
Adam (avec un learning rate initial de 0.001) car comme vu en cours, Adam offre souvent une convergence plus rapide grâce à son adaptation du learning rate pour chaque paramètre.
Il permet de commencer la recherche d'hyperparamètres avec une convergence stable, en particulier sur des datasets comme CIFAR-10.
En phase expérimentale, Adam a montré une diminution rapide de la perte, ce qui a facilité l'analyse comparative.

5. Stratégie de Partitionnement Train/Validation/Test

Données d'entraînement : 80 % du dataset CIFAR-10 (issu des 50 000 images initiales après séparation).
Validation : 10 % des images d'entraînement pour ajuster les hyperparamètres et surveiller la surapprentissage (overfitting).
Test : 10 000 images fournies par le dataset pour l'évaluation finale.
Cette répartition permet d'avoir suffisamment de données pour l'entraînement tout en conservant un ensemble de validation pour la recherche d'hyperparamètres et l'early stopping.

6. Détails Expérimentaux

Paramètres fixes et réglages initiaux :

Taille du dataset :
Entraînement : ≈ 45 000 images
Validation : ≈ 5 000 images
Test : 10 000 images
Nombre d'époques : 100 (avec early stopping, l'entraînement peut s'arrêter avant)
Batch size : 64
Temps d'entraînement : Variable selon le matériel (quelques heures sur GPU de l'école)
Expériences réalisées :

Recherche de learning rate :
Des tests variés (ex. 0.0001, 0.001, 0.01) sont effectués pour identifier le learning rate qui provoque une diminution stable de la loss.
Recherche en grille d'hyperparamètres :
Variation du squeeze ratio, de la proportion de filtres 3×3 dans les Fire Modules, du dropout, etc.
Un tableau récapitulatif des hyperparamètres et des performances (ex. précision sur validation) est dressé.

Mécanismes implémentés :

Early Stopping :
Arrêt de l'entraînement si la perte de validation ne s'améliore pas pendant 10 époques consécutives, avec restauration des meilleurs poids.
Learning Rate Scheduling :
Utilisation de ReduceLROnPlateau qui baisse le learning rate d'un facteur (par exemple 0.5) lorsque la loss de validation stagne.

Effets observés :
La diminution du learning rate permet d'affiner la convergence et d'obtenir une stabilité dans la baisse de la loss.
L'early stopping permet d'éviter un surentraînement, notamment lorsque la courbe de validation commence à augmenter tandis que la courbe d'entraînement continue de décroître.

7. Analyse des Étapes de Recherche d'Hyperparamètres

Perte initiale cohérente :
La loss initiale, mesurée avant tout apprentissage, se situe autour de 2.3 pour 10 classes, indiquant un comportement conforme aux attentes pour une prédiction aléatoire.
Recherche du learning rate optimal :
Par tests itératifs, un learning rate initial de 0.001 a montré une diminution stable de la loss. Des variations trop faibles (ex. 0.0001) ralentissent l'apprentissage, tandis que des valeurs trop élevées (ex. 0.01) induisent une oscillation ou divergence.
Recherche en grille :
En ajustant les hyperparamètres tels que le squeeze ratio (entre 0.125 et 0.5) et la proportion de filtres 3×3 (entre 30 % et 70 %), les configurations optimales semblent être celles qui minimisent la perte de validation tout en gardant un modèle compact.

Observations sur TensorBoard :
Les courbes d'entraînement et de validation de la loss convergent progressivement, avec éventuellement un léger écart qui signale l'overfitting si le gap s'élargit trop.
Lorsque l'early stopping est activé, on observe une stabilisation ou une légère augmentation de la loss de validation.
Un ajustement du learning rate via le scheduler se voit sous la forme d'un ralentissement progressif de la descente de la loss en fin d'entraînement.
Hyperparamètres optimaux (exemple) :

Dropout : 0.5, afin de réduire le surapprentissage.
Learning rate initial : 0.001 avec une réduction d'un facteur de 0.5 lorsque la loss stagne.
Squeeze ratio : 0.125 à 0.25 pour obtenir un bon compromis entre nombre de paramètres et performance.

8. Apprentissages et Perspectives

Ce que j'ai appris de ce projet :

L'importance de concevoir des architectures compactes et efficaces pour des applications embarquées.
La nécessité d'une recherche minutieuse d'hyperparamètres et d'une observation attentive des courbes d'apprentissage (via TensorBoard).
Les compromis entre réduction du nombre de paramètres et maintien de la performance.
La mise en place de mécanismes d'early stopping et de scheduling du learning rate est essentielle pour optimiser l'entraînement.

Ce que je ferais différemment en recommençant :

Explorer de manière plus systématique la recherche en grille sur le squeeze ratio, la répartition des filtres (1×1 vs 3×3), le taux de dropout, etc.
Ajouter éventuellement des techniques supplémentaires de régularisation (comme la normalisation par batch) pour mieux contrôler le flux de gradients.
Utiliser la validation croisée pour obtenir une estimation plus robuste des performances.
Approfondir l'analyse de l'impact de la taille des Fire Modules sur la performance finale, en comparant plusieurs configurations.