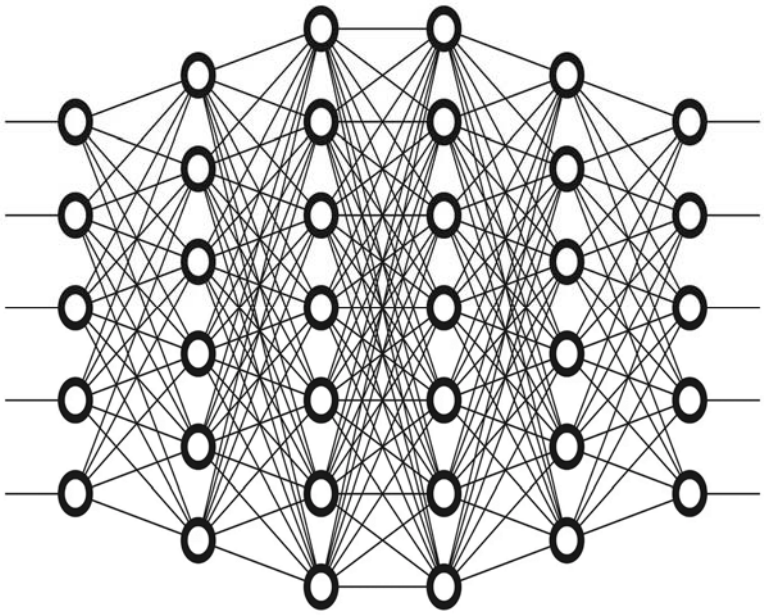




Déploiement d'une machine learning de classification d'image dans un environnement de cloud computing



Déploiement d'une machine learning de classification d'image dans un environnement de cloud computing

TorchServe est un serveur performant, flexible et facile à utiliser pour servir les modèles de machine learning PyTorch en production. Les modèles peuvent être diverses et variés du réseau de neurone convolutif au réseau de neurone LLM (langage large model). Le modèle utilisé est un réseau de neurones convolutifs (CNN) resnet18. Ce modèle sert à la classification d'objet sur une image .
Les réseaux de neurones convolutifs, également connus sous le nom de CNN (convolutif neural network), sont un type spécifique de réseaux neuronaux qui sont généralement composés des couches suivantes :

- ✓ Couche de convolution
- ✓ Couche de pooling
- ✓ Les couches d'activation
- ✓ Couche FC (fully connected ou chaque entrée est connectée à tous les neurones)

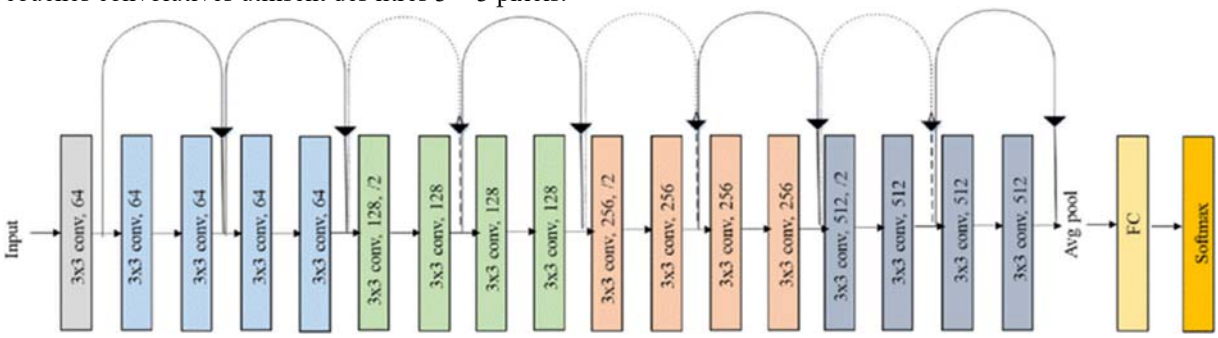
La couche de convolution et la couche de pooling peuvent être ajustées avec précision en fonction des hyperparamètres qui seront modifiés lors de l'entrainement :

La couche de convolution (CONV) utilise des filtres qui effectuent des opérations de convolution sur les images à l'entrée du convoluteur. Ses hyperparamètres incluent la nature , la taille du filtre et le déplacement du filtre sur l'image

La couche de mise en commun (POOL) est une opération de sous-échantillonnage, généralement appliquée après une couche de convolution, qui diminue la taille de l'image afin d'obtenir une certaine invariance spatiale. En particulier, le choix du maximum et ou le calcul de la moyenne sur plusieurs pixels sont des types particuliers de mise en commun où la valeur maximale et la valeur moyenne sont prises, respectivement.

Les couches d'activation après chaque convolution dans un CNN, nous appliquons une fonction d'activation non linéaire, telle que ReLU, ELU ou l'une des autres variantes .

L'architecture du ResNet-18 d'origine est illustrée ci-dessous. Il a un total de dix-huit couches dans le réseau (17 couches convolutives, une couche entièrement connectée et une couche softmax supplémentaire pour effectuer la tâche de classification). Les couches convolutives utilisent des iltres 3 × 3 pixels.



La couche FC (fully connected) est la dernière couche du réseau avant la couche d'activation du type softmax

Le TP se décompose en trois parties :

l'entrainement du réseau de neurone convolutif qui sera très peu abordé
La préparation des données afin de les déployer par la suite dans un conteneur
Le déploiement du conteneur sous kubernetes et le test du réseau de neurone pour la classification

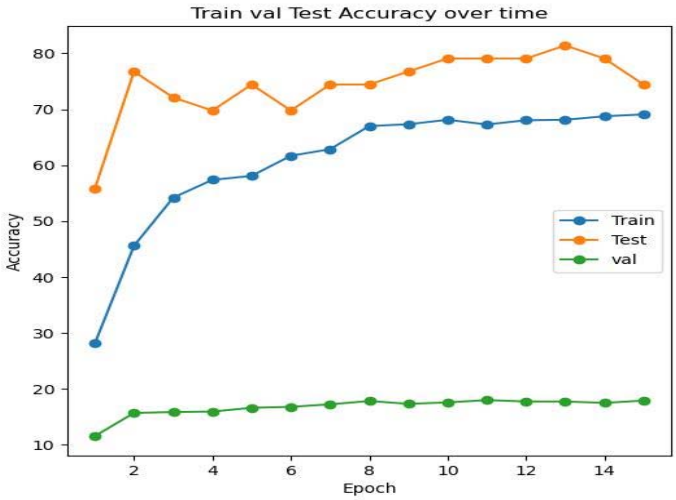
La première partie qui n'est pas traitée ici et l'entrainement par transfert learning du modèle avec un dataset d'image. Une fois obtenue les coefficients et biais du modèle par l'intermédiaire de pytorch , on va déployer celui -ci dans un cluster kubernetes sous Torchserve. La dernière partie est consacrée à vérifier le bon fonctionnement du classificateur.

Obtenir les poids et les biais de notre nouveau modèle
Le dataset d'entrainement se trouve sur le chemin suivant :
C:\python\pycharprojects\pythonproject\dataset\train
Une centaine d'image par classe pour l'entrainement
Le dataset de test se trouve sur le chemin suivant :
C:\python\pycharprojects\pythonproject\dataset\test
Une dizaine d'image par classe pou rle test
Lancez la compilation sous pycharm

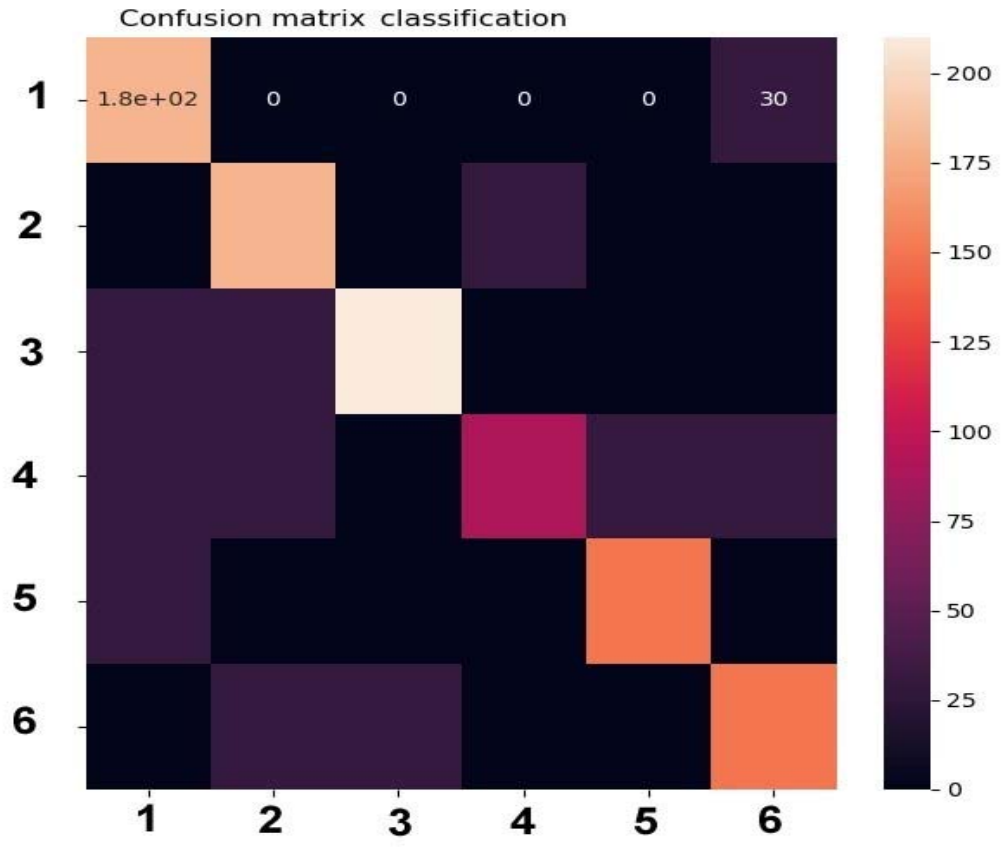
Une fois la compilation terminée on retrouve les coefficients et biais de notre modèle .pth et le fichier json des classes d'objet à classifier

- ✓ Index_to_name.json
- ✓ resnet18.pth

Les résultats de test de notre modèle de réseau de neurones sont fournis par la courbe suivante
Pour notre échantillon de test (Test) une vingtaine d'images nous obtenons après apprentissage du réseau de neurone une précision proche de 80%. Avec un apprentissage sur une vingtaine d'image par classe (val) le résultat reste médiocre 20% mais proportionnel au Train. Et pour un entrainement avec 200 images par classe Train (70%) de bon résultats



La matrice de confusion nous donne la proportion des vrais positifs comparés à la proportion des faux positifs. Si le modèle était parfait on obtiendrait un maximum sur la diagonale et 0 sur les autres cases hors diagonales. Prenons un exemple simple pour mieux comprendre sur la ligne 1 pour l'image 1 on obtient pour trente images un résultat de classification qui aurait dû être l'image 1 mais malheureusement le classificateur a choisi l'image 6.



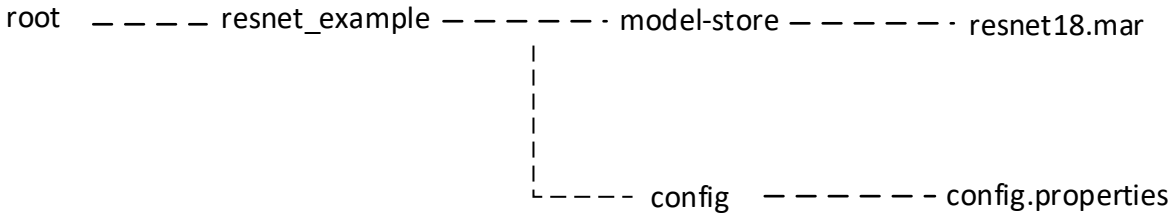
La partie suivante va nous permettre de mettre en place les outils de classification de notre réseau de neurones sur un cloud privé par l'intermédiaire d'un conteneur et de Torchserve

Quatre autres fichiers sont importants pour l'intégration du modèle :

Les fichiers handler.py , model.py, config.properties t resnet18.pth

On lance par la suite notre cluster Kubernetes worker1,worker2,worker3 et master ainsi que putty et winscp sur le master. Le nœud master contient préalablement les outils de compilation torchserve.

On doit créer l'arborescence suivante sur le nœud master :

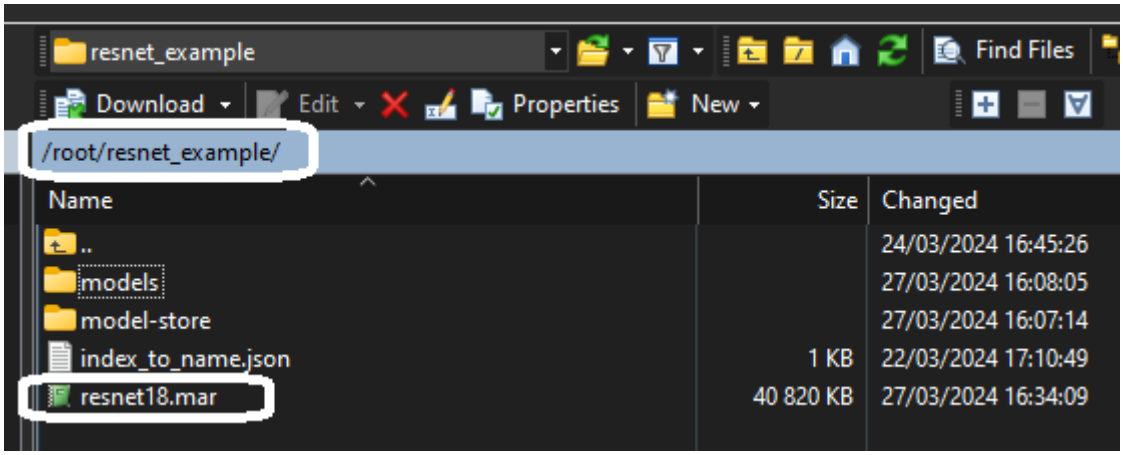


Dans le répertoire resnet_example lancez la commande suivante :

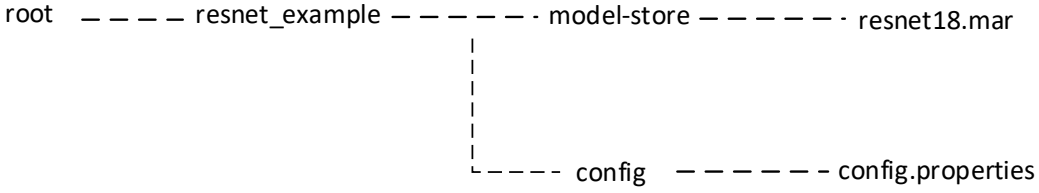
```
torch-model-archiver --model-name resnet18 --version 1.0 --model-file models/model.py --serialized-file models/ resnet18.pth  --extra-files index_to_name.json --handler models/handler.py
```

qui nous permettra d'obtenir le fichier resnet18.mar comprenant la structure de notre modèle CNN préalablement entrainée.

- ✓ **handler.py** : le fichier Python qui définit le prétraitement, l'inférence et le post-traitement des données.
- ✓ **model.py** : fichier dans lequel l'architecture du modèle est définie.
- ✓ **index_to_name.json** : le dictionnaire/json contenant les relations entre les ID (cible du modèle) et les étiquettes/noms
- ✓ **resnet18.pth** : la state_dict dumpée des poids du modèle entraîné.



Pour la création de notre pod on doit respecter une certaine arborescence de fichier pour monter un volume du type hostpath
Sur le nœud worker1 ou sera créé le pod on va mettre en place l'arborescence suivante



Torchserve est un serveur de modèle de machine learning développé conjointement avec le framework pytorch
Sur le nœud master1 /root/pytorchserve créons notre déploiement ainsi que les deux services nodePort associés à torchserve
On souhaite créer notre fichier manifest yaml qui doit répondre aux exigences suivantes :

On souhaite un replica de 1 pour les tests
L'image se trouve sur le registre privée docker-registry:5000/torchserver:v1
Un argument sera passé lors de la mise en place du conteneur :
args: ["torchserve --start --ts-config /home/model-server/config/config.properties --model-store model-store --models resnet18=resnet18.mar"]
Un node selector sera utilisé sur le nœud worker1
Deux montages de volume de stockage :
Un premier montage pour le téléchargement du réseau de neurone resnet18.mar
/home/model-server/model-store pour le conteneur qui sera l'image du dossier /root/resnet_example/model-store de l'hote
Un deuxième montage pour le téléchargement du ficher de configuration de Torchserve
/home/model-server/config pour le conteneur qui sera l'image du dossier /root/resnet_example/config de l'hote
Torchserve dispose de deux services un service sur le port 8080 pour l'inférence et un service sur le port 8081 pour la configuration à distance de Torchserve. Ces services utilises les requêtes https post restfull. Mettez en place un service Nodeport de façon à joindre ces deux services respectivement sur les ports 30080 et 30081
Un troisième service non exploité ici permet de faire de la métrique sur torchserve port 8082

Kubectl create -f torch.yaml

```
[root@master ~]# kubectl get all -A -o wide
NAMESPACE     NAME                                       READY   STATUS     RESTARTS        AGE     IP              NODE      NOMINATED NODE   READINESS GATES
default       pod/docker-registry-pod-66478b8ccb-n4kbk   1/1     Running    24 (113s ago)   36d     10.142.0.62     worker3   <none>           <none>
default       pod/docker-registry-pod-66478b8ccb-sww9f   1/1     Running    9 (2m2s ago)    11d     10.142.0.105    worker1   <none>           <none>
default       pod/docker-registry-pod-66478b8ccb-tvz9x   0/1     Unknown    23              36d     <none>          worker2   <none>           <none>
default       pod/torch-deployment-5467bd966-j8vqd       1/1     Running    1 (2m2s ago)    2d23h   10.142.0.104    worker1   <none>           <none>
kube-system   pod/calico-kube-controllers-57b57c56f-xr5th   1/1     Running    50 (113s ago)   37d     10.142.0.61     worker3   <none>           <none>
kube-system   pod/calico-node-2k472                      0/1     Init:1/3   0               37d     172.42.42.102   worker2   <none>           <none>
kube-system   pod/calico-node-b8pqb                      1/1     Running    37 (78s ago)    37d     172.42.42.100   master    <none>           <none>
kube-system   pod/calico-node-bvh97                      0/1     Running    17 (2m2s ago)   37d     172.42.42.101   worker1   <none>           <none>
kube-system   pod/calico-node-jjmm6                      1/1     Running    25 (113s ago)   37d     172.42.42.103   worker3   <none>           <none>
```

Notre pod qui met en place Torchserve et bien présent.

On peut vérifier les logs de lancement de torchserve sur notre pod

kubectl logs torch-deployment-768b945456-7gkxk

```
[root@master pytorchserve]# kubectl log torch-deployment-768b945456-7gkxk
error: unknown command "log" for "kubectl"

Did you mean this?
        top
        logs
[root@master pytorchserve]# kubectl logs torch-deployment-768b945456-7gkxk
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.
2024-03-27T15:53:33,357 [WARN ] main org.pytorch.serve.util.ConfigManager - Your torchserve instance can access any URL to load
2024-03-27T15:53:33,360 [INFO ] main org.pytorch.serve.servingsdk.impl.PluginsManager - Initializing plugins manager...
2024-03-27T15:53:33,424 [INFO ] main org.pytorch.serve.metrics.configuration.MetricConfiguration - Successfully loaded metrics c
2024-03-27T15:53:33,513 [INFO ] main org.pytorch.serve.ModelServer -
Torchserve version: 0.9.0
TS Home: /home/venv/lib/python3.9/site-packages
Current directory: /home/model-server
Temp directory: /home/model-server/tmp
Metrics config path: /home/venv/lib/python3.9/site-packages/ts/configs/metrics.yaml
Number of GPUs: 0
Number of CPUs: 4
Max heap size: 984 M
Python executable: /home/venv/bin/python
Config file: /home/model-server/config/config.properties
Inference address: http://0.0.0.0:8080
Management address: http://0.0.0.0:8081
Metrics address: http://0.0.0.0:8082
Model Store: /home/model-server/model-store
Initial Models: resnet18=resnet18.mar
Log dir: /home/model-server/logs
Metrics dir: /home/model-server/logs
Netty threads: 0
```

Vérifions si la machine learning de classification apporte de bon résultat en inférence

Lancez notre programme de test :

