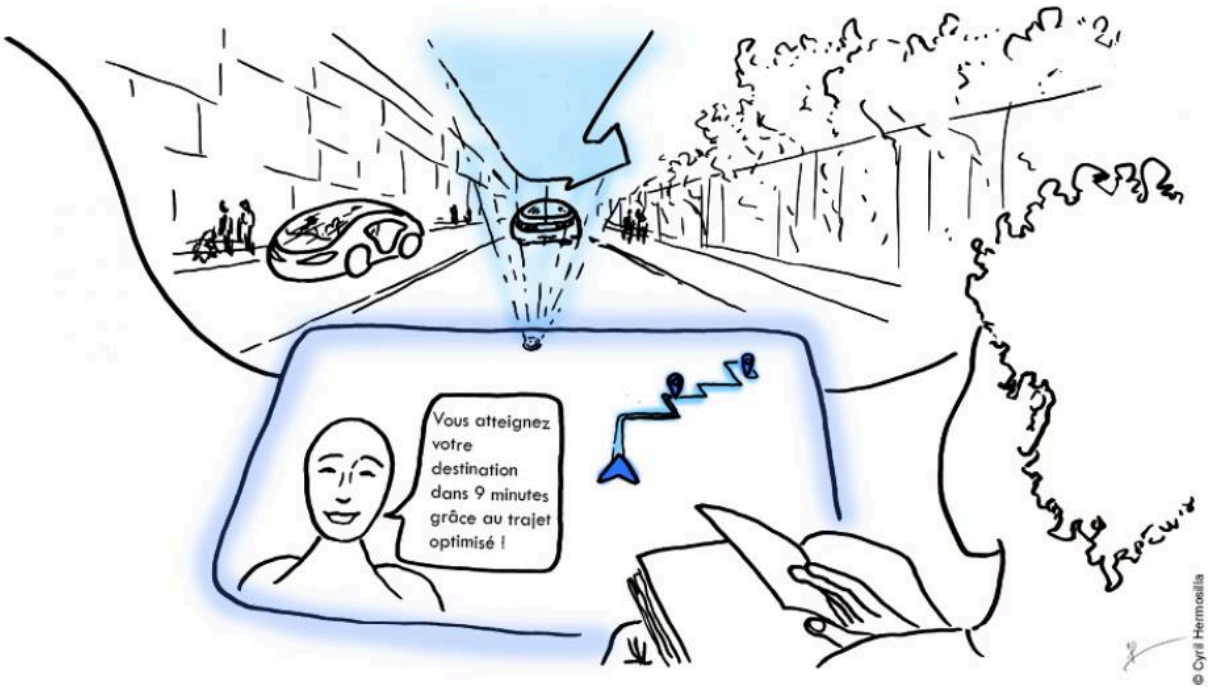
TP Jour 4 - Optimisation des modèles IA



Contexte metier

Vous travaillez pour une entreprise de maintenance industrielle qui souhaite automatiser la detection d'anomalies sur des pièces manufacturées (bouteilles, câbles, capsules, etc.).
Problematique : Le modele Vision Transformer actuel est trop lent et coûteux.
Vous devez l'optimiser pour réduire le temps d'inférence de 80% et les coûts de 70%.
Dataset : MVTec Anomaly Detection (images industrielles reelles avec anomalies)

Prerequis techniques

- Python 3.8+
- PyTorch 2.0+
- Transformers (Hugging Face)
- ONNX Runtime
- GPU recommandé mais non obligatoire (tout fonctionne sur CPU)
- 10 GB espace disque disponible
- Compte Hugging Face (gratuit)
- Compte Azure optionnel (pour Partie 5 uniquement)

Structure de départ fournie

```
TD_Jour4_Etudiant/
├── README.md                  Instructions installation
├── ENONCE.md                  Ce fichier
├── requirements.txt           Dependances Python
├── download_mvtec.py          Telechargement dataset
├── mvtec_dataset.py           Dataset PyTorch
├── load_model_hf.py           Chargement modele HF
├── train_utils.py             Fonctions utilitaires
├── benchmark_gpu.py           Script Partie 1
├── quantization_demo.py       Script Partie 2
└── onnx_optimization.py       Script Partie 3
```

Objectifs pédagogiques

1. Comparer performances CPU vs GPU
2. Appliquer quantization pour reduire la taille
3. Optimiser l'inference avec ONNX Runtime
4. Analyser les compromis precision/performance/cout
5. Recommander une strategie de deploiement

Duree estimee

- Partie 1 : 2-3h (entrainement CPU peut être long)
- Partie 2 : 1h
- Partie 3 : 1h
- Partie 4 : 30-45 min
- Partie 5 : 1-2h (optionnelle)

Total : 4h30-6h (sans Partie 5 optionnelle)

Partie 1 : Benchmark CPU vs GPU (2-3h)

Objectif

Comparer les performances d'entraînement d'un Vision Transformer sur CPU et GPU.

Execution

Mode test rapide (pour vérifier que tout fonctionne) :

```bash
python benchmark_gpu.py --category bottle --epochs 1 --batch-size 8
```

Mode complet (pour resultats finaux) :

```bash
python benchmark_gpu.py --category bottle --epochs 3 --batch-size 16
```

Note importante : L'entraînement CPU peut prendre 1-2h selon votre machine. Lancez le script et laissez-le tourner. Si vous avez un GPU, le script vous entraînera automatiquement sur les deux devices.

Résultats attendus (ordres de grandeur)

- Speedup GPU : 5-15x selon configuration
- Accuracy : 75-90% apres 3 epochs
- F1-Score : 70-85%
- Temps CPU : 30-90 min
- Temps GPU : 3-10 min

Questions mi-temps

1. **Performance** : Quel speedup obtenez-vous avec le GPU ? Est-ce conforme aux attentes ?
2. **Precision** : L'accuracy est-elle identique entre CPU et GPU ? Pourquoi une petite différence est normale ?
3. **Cout Azure** : Calculez le cout theorique d'entrainement
   - CPU Standard DS2 v2 : $0.90/h

   - GPU NC6 avec T4 : $3.50/h
   - Formule : (temps_secondes / 3600) * tarif_horaire
4. **Rentabilite** A partir de quel speedup le GPU devient-il rentable ?
   - Indice : Comparez le ratio des coûts (3.50 / 0.90 = 3.9x)
5. **API Hugging Face** : Comparez avec l'API Hugging Face Inference
   - Gratuite : 30,000 requetes/mois
   - Payante : $0.06/1000 requetes
   - Dans quels scénarios l'API est-elle plus avantageuse ?

Checkpoint Partie 1

Avant de passer à la partie 2, vérifiez que vous avez :
- [ ] Les modeles entraines : `model_cpu_bottle.pth` (et `model_gpu_bottle.pth` si GPU)
- [ ] Le graphique : `benchmark_cpu_gpu.png`
- [ ] Les temps d'entraînement notes
- [ ] Les métriques (accuracy, F1-score) notées

Livrables
- Modèles entraînés (au moins CPU)
- Graphique de comparaison
- Tableau avec temps, accuracy, F1-score, coût
- Réponses aux questions de reflexion

Partie 2 : Quantization (1h)

Objectif

Réduire la taille du modèle pour déploiement sur edge devices (caméras industrielles).

Objectif metier : Taille < 30 MB et rétention precision > 95%
  Execution
```bash
python quantization_demo.py --category bottle --model-path model_cpu_bottle.pth
```
Note :
Utilisez le modèle CPU entraîné en Partie 1.
Le script charge le modèle, applique la quantization, et compare les performances.

Résultats attendus (ordres de grandeur)
- Compression : 2-4x (50-75% reduction)
- Speedup inference : 1.5-3x sur CPU
- Perte accuracy : < 2%
- Perte F1-score : < 3%
- Taille finale : 20-40 MB

Questions de réflexion

1. **Compression** : Quelle réduction de taille obtenez-vous ? Est-ce suffisant pour l'objectif métier ?
2. **Performance** : Quel est le gain en vitesse d'inférence ? Sur quel hardware (CPU/GPU) ?
3. **Precision** : Quelle est la perte de précision (accuracy, F1-score) ? Est-elle acceptable ?
4. **Objectif metier** : L'objectif (< 30 MB, > 95% rétention) est-il atteint ? Si non, quelles alternatives ?
5. **Vision Transformers vs CNN** : Pourquoi la quantization est-elle moins efficace sur les Vision Transformers ?
   - Indice : Les Transformers ont beaucoup de couches Linear (attention) qui sont quantifiees, mais aussi des opérations non quantifiables (LayerNorm, Softmax)
   - Ressource : https://pytorch.org/docs/stable/quantization.html quantization-aware-training
6. **Dynamique vs Statique** : Dans quels scénarios privilegier quantization dynamique vs statique ?
   - Dynamique : Poids en int8, activations en float32
   - Statique : Poids ET activations en int8 (nécessite calibration)
   - Ressource : https://pytorch.org/blog/quantization-in-practice/

Checkpoint Partie 2

Avant de passer à la partie 3, vérifiez que vous avez :

- [ ] Le modele quantifie : `model_quantized_bottle.pth`
- [ ] Les métriques de comparaison (taille, temps, accuracy)
- [ ] La validation de l'objectif métier
- [ ] Votre recommandation pour déploiement edge

Livrables

- Modele quantifie
- Tableau comparatif (original vs quantifie)
- Validation objectif métier
- Recommandation déploiement edge (justifiée)

Partie 3 : ONNX Runtime (1h)

Objectif

Optimiser l'inférence avec ONNX Runtime pour déploiement cloud.

Execution

```bash
python onnx_optimization.py --category bottle --model-path model_cpu_bottle.pth --batch-size 32
```

Note : Le script exporte le modèle vers ONNX, puis compare les performances d'inférence PyTorch vs ONNX Runtime.

Résultats attendus (ordres de grandeur):

- Speedup ONNX vs PyTorch : 1.5-3x sur CPU
- Speedup ONNX vs PyTorch : 1.2-2x sur GPU
- Difference accuracy : < 0.5%
- Taille fichier ONNX : Similaire au .pth

Hypothèses pour calculs de coût
- Azure ML CPU (Standard DS2 v2) : $0.50/h
- Azure ML GPU (NC6 T4) : $3.50/h
- Temps inference moyen : Mesure dans votre benchmark
- Volume : 10,000 images/jour
- Batch size : 32 images
- Facteur utilisation : 24/7 (ou heures ouvrables uniquement)

Questions de réflexion
1. **Performance** : Quel speedup obtenez-vous avec ONNX Runtime vs PyTorch ?
   - Sur CPU ? Sur GPU ?
   - Est-ce conforme aux attentes ?
2. **Precision** : La précision est-elle préservée ?
   - Difference d'accuracy < 0.5% ?
   - Pourquoi une différence minimale est normale ?
3. **Avantages ONNX** Quels sont les avantages du format ONNX pour le déploiement ?
   - Interoperabilite (frameworks, langages)
   - Optimisations cross-platform
   - Pas besoin de PyTorch en production
   - Support hardware varie
4. **Cout mensuel** : Calculez le cout pour 10,000 images/jour sur Azure ML
   - Formule : (images/jour / batch_size) * temps_ms * (1/3600000) * tarif_horaire * 30 jours
   - Comparez PyTorch vs ONNX Runtime
5. **Azure ML vs HF API** : Comparez les deux approches
   - HF API : Gratuit (30k req/mois), puis $0.06/1000 req
   - Azure ML : Controle total, scalabilite, mais cout infrastructure
   - Quand privilegier l'un ou l'autre ?
6. **Optimisations ONNX** Quelles optimisations ONNX Runtime applique-t-il ?
   - Fusion d'operations (Conv + BN + ReLU)
   - Elimination d'operations inutiles
   - Réordonnancement pour localite mémoire
   - Utilisation de kernels optimises (BLAS, cuDNN)
   - Ressource : https://onnxruntime.ai/docs/performance/model-optimizations.html

Checkpoint Partie 3
Avant de passer a la partie 4, vérifiez que vous avez :
- [ ] Le modele ONNX : `model_bottle_optimized.onnx`
- [ ] Les temps d'inference PyTorch et ONNX notes
- [ ] Le calcul du cout mensuel Azure ML
- [ ] Votre comparaison Azure ML vs HF API

Livrables
- Modele ONNX exporte
- Tableau comparatif PyTorch vs ONNX (temps, accuracy, coût)
- Calcul coût mensuel détaillé
- Recommandation déploiement cloud (justifiée)

Partie 4 : Synthese et recommandations (30-45 min)

Objectif

Comparez toutes les approches et recommandez une stratégie globale.

*Pour 10,000 images/jour
**Gratuit jusqu'à 30,000 req/mois, puis $0.06/1000 req
Questions de synthese
1. **Strategie par use case** : Quelle est la meilleure stratégie pour :

a) **Déploiement edge (caméras industrielles)** ?
- Contraintes : Taille < 30 MB, pas de connexion permanente, CPU faible
- Votre recommandation : ?
- Justification : ?

b) **Deployment cloud (Azure ML)** ?
- Contraintes : Cout optimise, scalabilité, latence < 100ms
- Votre recommandation : ?
- Justification : ?

Partie 5 : Azure ML Deployment (OPTIONNELLE - BONUS, 1-2h)

Note importante : Cette partie est optionnelle et rapporte des points bonus. Elle nécessite un compte Azure avec crédits disponibles.

Prerequis

- Compte Azure (Azure for Students : $100 credits, ou Free Trial : $200 credits)
- Azure CLI installe
- Crédits disponibles (coût estimé : $5-10 pour le TP)

Alternative sans Azure : Si vous n'avez pas accès à Azure, vous pouvez :

- Déployer localement avec FastAPI + Docker
- Utiliser l'API Hugging Face Inference
- Documenter la procedure theorique de deploiement Azure ML

Objectif

Déployer le modèle ONNX optimise sur Azure ML avec un endpoint manage.

Etapes detaillees

1. **Configurer Azure ML workspace**
   ```bash
   az login
   az ml workspace create --name tp-jour4-ws --resource-group tp-jour4-rg
   ```
2. **Enregistrer le modele ONNX**
   - Uploader `model_bottle_optimized.onnx`
   - Tagger avec version et metadata
3. **Creer un script de scoring** (`score.py`)
   - Fonction `init()` : Charger le modele
   - Fonction `run()` : Inference sur requete
4. **Creer un endpoint manage**
   - Type : Managed Online Endpoint
   - Instance : Standard_DS2_v2 (2 vCPU, 7 GB RAM)
5. **Deployer le modele**
   - Blue/Green deployment
   - Allocation trafic : 100% sur blue
6. **Tester l'endpoint**
   - Envoyer une image de test
   - Mesurer latence
   - Verifier predictions
7. **Configurer auto-scaling**
   - Min instances : 1
   - Max instances : 3
   - Seuil CPU : 70%

Questions
1. **Auto-scaling** : Comment Azure ML gere-t-il l'auto-scaling ?
   - Metriques utilisees (CPU, memoire, requetes/s)
   - Temps de scale-up/scale-down
   - Strategie de scaling (reactive vs predictive)
   - Ressource : https://learn.microsoft.com/azure/machine-learning/how-to-autoscale-endpoints
2. **Monitoring** : Quelles metriques monitorer ?
   - Latence (p50, p95, p99)
   - Throughput (requetes/s)
   - Taux d'erreurs (4xx, 5xx)
   - Utilisation ressources (CPU, memoire)
   - Drift du modele (distribution predictions)
3. **Blue/Green deployment** : Comment l'implementer ?
   - Déployer nouvelle version (green) a cote de l'ancienne (blue)
   - Tester green avec 10% du trafic
   - Si OK, basculer progressivement (50%, 100%)
   - Rollback instantane si probleme
4. **Securite** : Quelles mesures appliquer ?
   - Authentification : Key-based ou Token-based
   - Chiffrement : HTTPS/TLS obligatoire
   - Network isolation : VNet, Private Endpoint
   - RBAC : Role-Based Access Control
   - Audit : Logs d'acces et d'utilisation

Livrables (BONUS)
- Configuration Azure ML (fichiers YAML)
- Script de scoring (`score.py`)
- Endpoint deploye et teste
- Documentation procedure de deploiement
- Analyse coût réel du déploiement

Livrables finaux
  Structure de l'archive
Archive ZIP nommee : `nom_prenom_tp_jour4.zip`
```
nom_prenom_tp_jour4.zip
├── code/                    Si vous avez modifié les scripts
│   ├── benchmark_gpu.py
│   ├── quantization_demo.py
│   └── onnx_optimization.py
│
├── resultats/               OBLIGATOIRE
│   ├── benchmark_cpu_gpu.png
│   ├── model_cpu_bottle.pth
│   ├── model_quantized_bottle.pth
│   ├── model_bottle_optimized.onnx
│   └── logs/                Logs d'exécution (optionnel)
│
├── rapport.md     OBLIGATOIRE - Votre analyse complete + config d'installation + difficultées
│
└── bonus/                   Optionnel (Partie 5)
    ├── azure_ml_config.yaml
    ├── score.py
    └── deployment_doc.md
```

Ressources

Documentation officielle

- PyTorch Quantization : https://pytorch.org/docs/stable/quantization.html
- PyTorch Quantization in Practice : https://pytorch.org/blog/quantization-in-practice/
- ONNX Runtime : https://onnxruntime.ai/docs/

- ONNX Runtime Performance : https://onnxruntime.ai/docs/performance/model-optimizations.html
- Hugging Face Transformers : https://huggingface.co/docs/transformers
- Azure ML : https://learn.microsoft.com/azure/machine-learning/

Tutoriels recommandés
- Vision Transformers : https://huggingface.co/docs/transformers/model_doc/vit
- ONNX Export : https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html
- Azure ML Deployment : https://learn.microsoft.com/azure/machine-learning/how-to-deploy-online-endpoints

Dataset
- MVTec AD : **https://www.mvtec.com/company/research/datasets/mvtec-ad**
- Paper : https://www.mvtec.com/fileadmin/Redaktion/mvtec.com/company/research/datasets/mvtec_ad.pdf

Conseils pratiques

Avant de commencer
1. Lisez l'énoncé en entier
2. Vérifiez les prérequis (Python, GPU, espace disque)
3. Installez les dependances (`pip install -r requirements.txt`)
4. Testez le téléchargement du dataset

Pendant le TP
1. Commencez par le mode test rapide (1 epoch) pour vérifier que tout fonctionne
2. Lancez l'entraînement complet (3 epochs) et laissez tourner
3. Documentez vos observations au fur et à mesure
4. Notez tous les chiffres (temps, accuracy, taille, coût)
5. Faites des checkpoints entre chaque partie

Pour l'analyse
1. Pensez toujours au compromis precision/performance/coût
2. Justifiez vos recommandations avec des chiffres concrets
3. Contextualiser par rapport au cas d'usage industriel
4. Comparez avec les ordres de grandeur attendus

Gestion du temps
- Si l'entraînement CPU est trop long (>2h), utilisez --epochs 2
- Si vous n'avez pas de GPU, concentrez-vous sur CPU + ONNX
- La Partie 5 (Azure ML) est optionnelle, ne la faites que si vous avez le temps

En cas de probleme
1. Vérifiez les logs d'erreur
2. Consultez la documentation
3. Testez avec un batch size plus petit

Vous pouvez documenter la procédure théorique pour le bonus.

Bon courage !