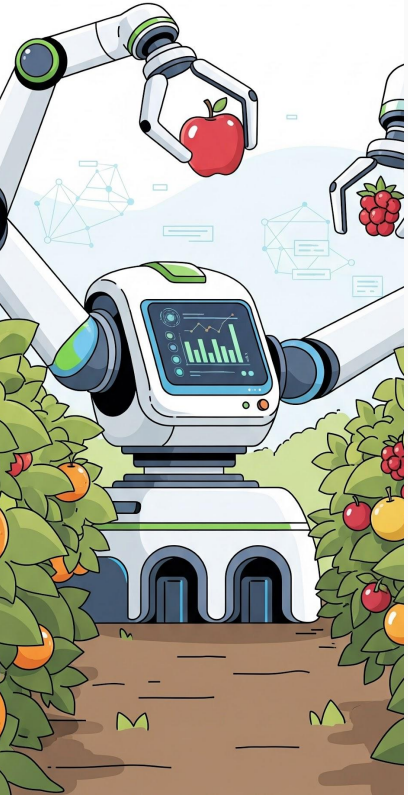
OPENCLASSROOMS



Projet 11

Réalisez un traitement dans un environnement Big Data sur le Cloud

Exécutez un traitement d'images dans un environnement Big Data

David Scanu | Novembre 2025

Parcours **AI Engineer**

📖 Étapes du projet

- 🌐 Contexte et problématique métier
- 🍉 Analyse du jeu de données
- 📓 Amélioration du notebook local de l'alternant
- 👷 Identification des services Big Data nécessaires
- 🏗️ Architecture du pipeline Big Data dans le Cloud
- 🐍 Développement d’un Script PySpark de traitement des données
- ☁️ Déploiement et exécution du script dans le Cloud
- 🔎 Inspection et téléchargement des résultats
- 🖥️ Démonstration
- 🚀 Applications & Perspectives
- 🎯 Mission Accomplie

📖 Étapes du projet

🌐 Contexte et problématique métier

Entreprise

Fruits! : **start-up AgriTech** qui développe des robots cueilleurs intelligents pour **préserver la biodiversité des fruits**.

Rôle

Data Scientist dans une start-up de l'AgriTech.

Objectif

> Exécuter un traitement d'images dans un environnement Big Data sur le Cloud

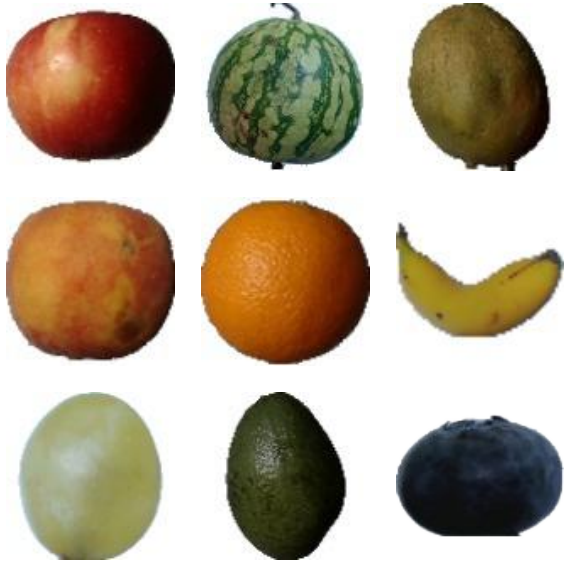
Étapes

- Mettre en place un **pipeline PySpark distribué**
- **Extraire des features d'images** avec MobileNetV2
- **Réduire les dimensions des features** avec PCA
- **Télécharger les résultats**

🍉 Jeu de données

🥝 Fruits-360 Dataset

- **Créateur** : Mihai Oltean (2017)
- **Taille** : 155,491 images
- **Classes** : 224 classes
- **Format** : JPG, 100x100 pixels (standardisé)
- **Contenu** : Fruits, légumes, noix et graines avec de multiples variétés
  - 29 types de pommes
  - 12 variétés de cerises
  - 19 types de tomates
  - Et bien d'autres...
- **Méthode de capture** : Images capturées par rotation (20s à 3 rpm) sur fond blanc
- **Licence** : CC BY-SA 4.0



📓 Amélioration du notebook local de l'alternant

> Objectif : Comprendre et améliorer le code avant la migration cloud

💡 Étude du notebook PySpark existant

- `notebooks/alternant/P8_Notebook_Linux_EMR_PySpark_V1.0.ipynb`

🔍 Lacunes Critiques Détectées

❌ Absence de broadcast des poids TensorFlow

- Poids (~14 MB) transférés vers chaque partition
- Goulot d'étranglement réseau majeur

❌ PCA non implémentée

- Réduction dimensionnelle manquante
- Objectif du projet non atteint

Livrable

Notebook corrigé :
`p11-david-scanu-local-development.ipynb`

- **Broadcast TensorFlow** → Réduction -90% des transferts réseau
- **PCA MLlib** (1280D → 50D) → Conservation 92.93% variance
- **Tests locaux complets** → Validation avant déploiement cloud

Impact

- Validation locale avant coûts cloud
- Pipeline production-ready
- Scalabilité garantie

👷 Infrastructure Cloud identifiées : AWS Big Data

AWS EMR (Elastic MapReduce)

Utilisé pour la mise en place du **cluster PySpark** distribué pour exécuter le traitement d'images.

AWS S3 (Simple Storage Service)

Utilisé comme espace de stockage scalable pour :

- **Données d'entrée** (le *dataset* Fruits-360)
- **Scripts PySpark** et dépendances
- **Stockage des outputs** du traitement :
  - Features
  - Résultats PCA
  - Métadonnées
  - Rapports d'erreurs

AWS IAM (Identity and Access Management)

- Utilisé pour **gérer l'accès et les autorisations** au cluster EMR et aux *buckets* S3.

Région AWS :

- Région **eu-west-1**
- Conformité **RGPD** : Stockage et traitement des données sur le territoire européen.

AWS S3 : Simple Storage Service

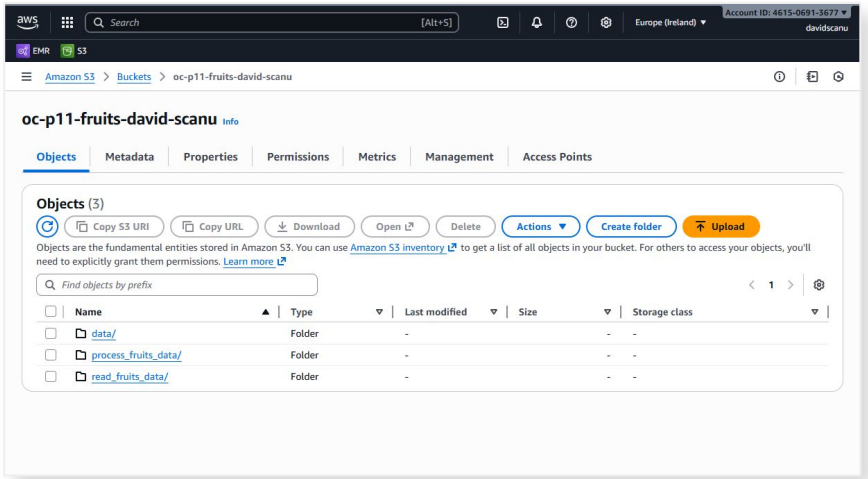
> Espace de stockage scalable pour le Big Data

Rôle

- **Stockage des données d'entrée** :
  - Dataset Fruits-360
- **Stockage des scripts et dépendances** :
  - **Scripts PySpark** : `process_fruits_data.py`
  - **Dépendances** : `install_dependencies.sh`
- **Stockage des outputs du pipeline :**
  - **Features** extraites par MobileNetV2 (1280D)
  - **Résultats de la réduction de dimension** PCA
  - **Métadonnées**
  - Rapports d'erreurs et informations du modèle

Bucket utilisé :

- `oc-p11-fruits-david-scanu`



Exemples de chemins

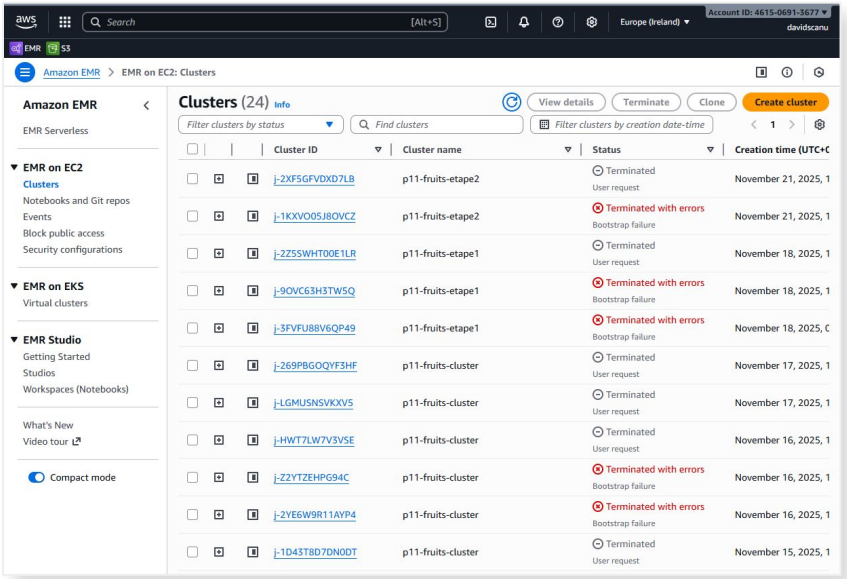
- **Image :**
  - `s3://…/data/raw/Training/Apple Braeburn/0_100.jpg`
- **Features :**
  - `s3://…/process_fruits_data/output/features/`
- **PCA :**
  - `s3://…/process_fruits_data/output/pca/`

Elastic MapReduce (EMR)

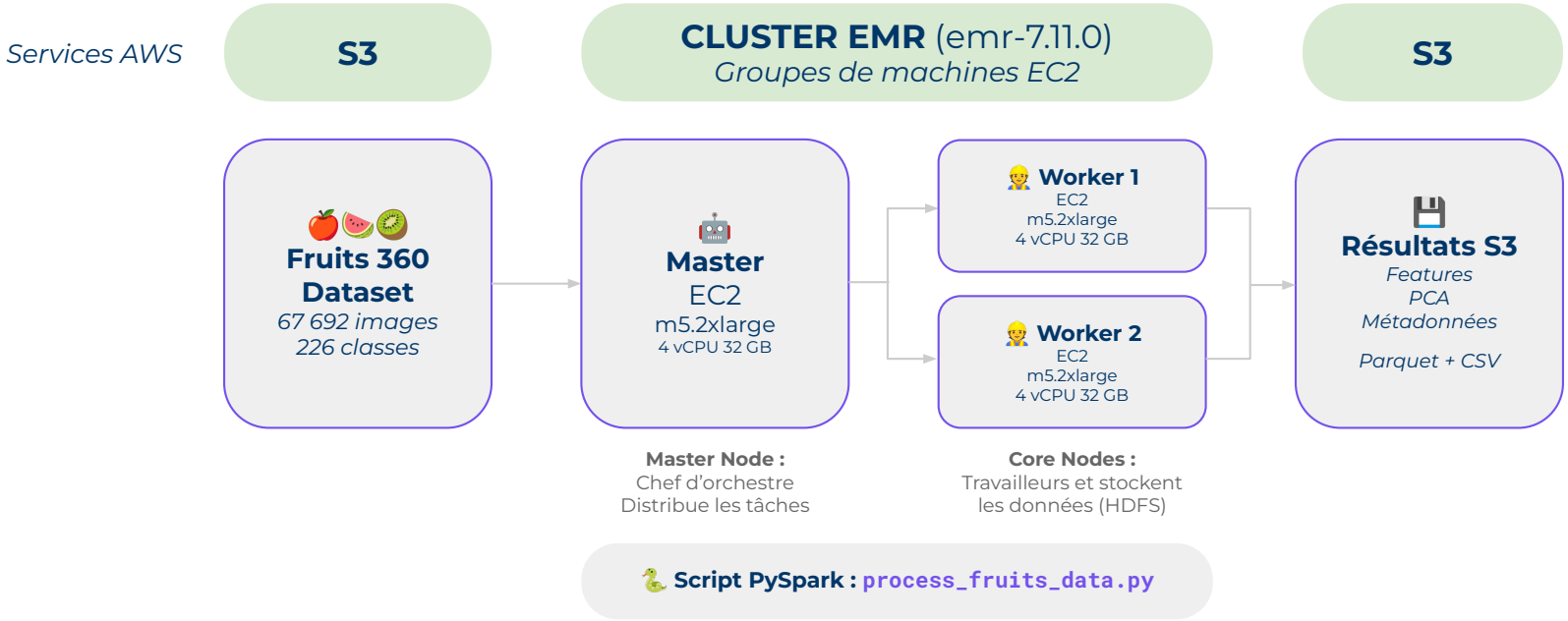
- Service cloud d'**Amazon Web Services (AWS)**
- Permet de **créer et de gérer des *clusters*** (groupes de serveurs)
- Exécution de ***frameworks* de Big Data**, tels qu'Apache Spark et Hadoop

Dans le contexte de ce projet, utilisé pour :

- Mettre en place un **cluster PySpark distribué** pour réaliser des calculs sur un **grand volume de données**
- **Exécuter le traitement d'images de manière parallèle sur plusieurs machines**
  a. **Extraction de *features*** avec MobileNetV2
  b. **Réduction de dimension** avec PCA
- Plateforme utilisée : **AWS EMR 7.11.0** avec Spark 3.5.x



🏗️ Architecture du pipeline Big Data dans le Cloud



🐍 Script PySpark : `process_fruits_data.py`

> Pipeline PySpark pour l'extraction de features avec MobileNetV2 et PCA

- **Script :** `process_fruits_data.py`

Architecture

1. **Chargement des images depuis S3**
2. **Broadcast des poids MobileNetV2** (optimisation distribuée)
3. **Pandas UDF** pour extraction de features (1280D)
4. **Extraction de features** avec TensorFlow MobileNetV2 (broadcast des poids)
5. **Réduction de dimension** avec PCA (PySpark MLlib)
6. **Sauvegarde des résultats sur S3** (Parquet + CSV)

```python
# 1. Charger les images
df_images = load_images(spark, input_path, mode)

# 2. Extraire les labels
df_with_labels = extract_labels(df_images)

# 3. Broadcast du modèle
broadcast_weights, output_dim = broadcast_model_weights(spark)

# 4. Créer l'UDF d'extraction
extract_features_udf = create_feature_extraction_udf(broadcast_weights,
output_dim)

# 5. Extraire les features
df_features, df_errors = extract_features(df_with_labels,
extract_features_udf)

# 6. Appliquer la PCA
df_pca, pca_model = apply_pca(df_features, pca_components)

# 7. Sauvegarder les résultats
save_results(df_pca, df_errors, pca_model, output_path, pca_components)
```

Détails du script sur GitHub : process_fruits_data.py

☁️ Déploiement et exécution du script dans le Cloud

⚙️ Configuration

- **Configuration :** `config.sh`
- **Dependencies :** `install_dependencies.sh`

📊 Spécifications techniques

- 🌍 Région AWS: **eu-west-1**
- 💾 Bucket S3: **oc-p11-fruits-david-scanu**
- 📂 Input data:
  - `s3://oc-p11-fruits-david-scanu/data/raw/`
- 📂 Output:
  - `s3://oc-p11-fruits-david-scanu/process_fruits_data/output/`
- 🖥️ Cluster: **p11-fruits-etape2**
- 📦 EMR Release: **emr-7.11.0**
- 💻 Master: **m5.2xlarge**
- 💻 Core: **2 x m5.2xlarge**
- 🧠 Spark Executor Memory: **8g**
- 🧠 Spark Driver Memory: **8g**

```bash
# Création du Cluster EMR
aws emr create-cluster \
  --name "P11-Fruits-PCA" \
  --release-label "emr-7.1.0" \
  --applications Name=Spark Name=Hadoop \
  --instance-groups Master=m5.2xlarge Core=m5.2xlarge×2 \
  --bootstrap-actions s3://.../scripts/install_dependencies.sh

# Soumission du Job PySpark
aws emr add-steps \
  --cluster-id j-XXXXX \
  --region eu-west-1 \
  --steps Type=Spark,Args=[
    s3://.../process_fruits_data/scripts/process_fruits_data.py,
    s3://oc-p11-fruits-david-scanu/data/raw//Training,
    s3://oc-p11-fruits-david-scanu/process_fruits_data/output/,
    full,
    50
  ]

# Terminaison du Cluster
aws emr terminate-clusters --cluster-ids j-XXXXX
```

💡 **Automatisation** : 11 scripts bash pour orchestration complète
⚙️ **3 modes** : mini (3 min) | apples (20 min) | full (83 min)

👋 Récapitulatif de l'exécution du job

ℹ️ Information du job

- **Date d'exécution** : 25 novembre 2025
- **Mode** : FULL (67,692 images)
- **Cluster EMR** : j-3Q36EOOGGHSE8
- **Step ID** : s-08453052BV9925LTFVFN
- **Job Status** : ✅ COMPLETED (exit code 0)

☁️ Infrastructure déployée

- **Plateforme** : AWS EMR 7.11.0
- **Spark : 3.5.x**
- **Cluster** : 3× m5.2xlarge
- **vCPU total** : 24 cores
- **RAM totale** : 96 GB
- **Région** : eu-west-1 (GDPR)
- **S3 Bucket** : `oc-p11-fruits-david-scanu`

⏳ Temps d'exécution

- **Création + Bootstrap** : ~15 min
- **Exécution job** : 83 min
- **Terminaison** : ~2 min

Métrique	MINI (300)	APPLES (6,404)	FULL (67,692)	Évolution
Images	300	6,404	**67,692**	×226 vs MINI
Temps	3min 34s	~20-25 min	**83 min**	×23 vs MINI
Débit	84 img/min	~260-320 img/min	**814 img/min**	×9.7 vs MINI
Variance PCA	92.93%	83.40%	**71.88%**	-21.05 pp vs MINI
Classes	~3-5	~29	**226**	Toutes les classes
Coût estimé	~0.50€	~0.90€	**1.60€**	Beaucoup plus économique

👀 Observations

- **Meilleur parallélisme** : débit a augmenté de façon spectaculaire avec le volume de données
- **Variance totale est plus faible** car nous avons toutes les classes (diversité maximale)
- **Scalabilité excellente** : 226× plus d'images mais seulement 23× plus de temps
- **Beaucoup plus efficace** à très grande échelle

🔎 Inspection et téléchargement des résultats

📦 Structure des outputs générés sur S3

- `s3://oc-p11-fruits-david-scanu/process_fruits_data/outputs/output-full/`
- Poids de stockage : ~1.7-2.0 GB

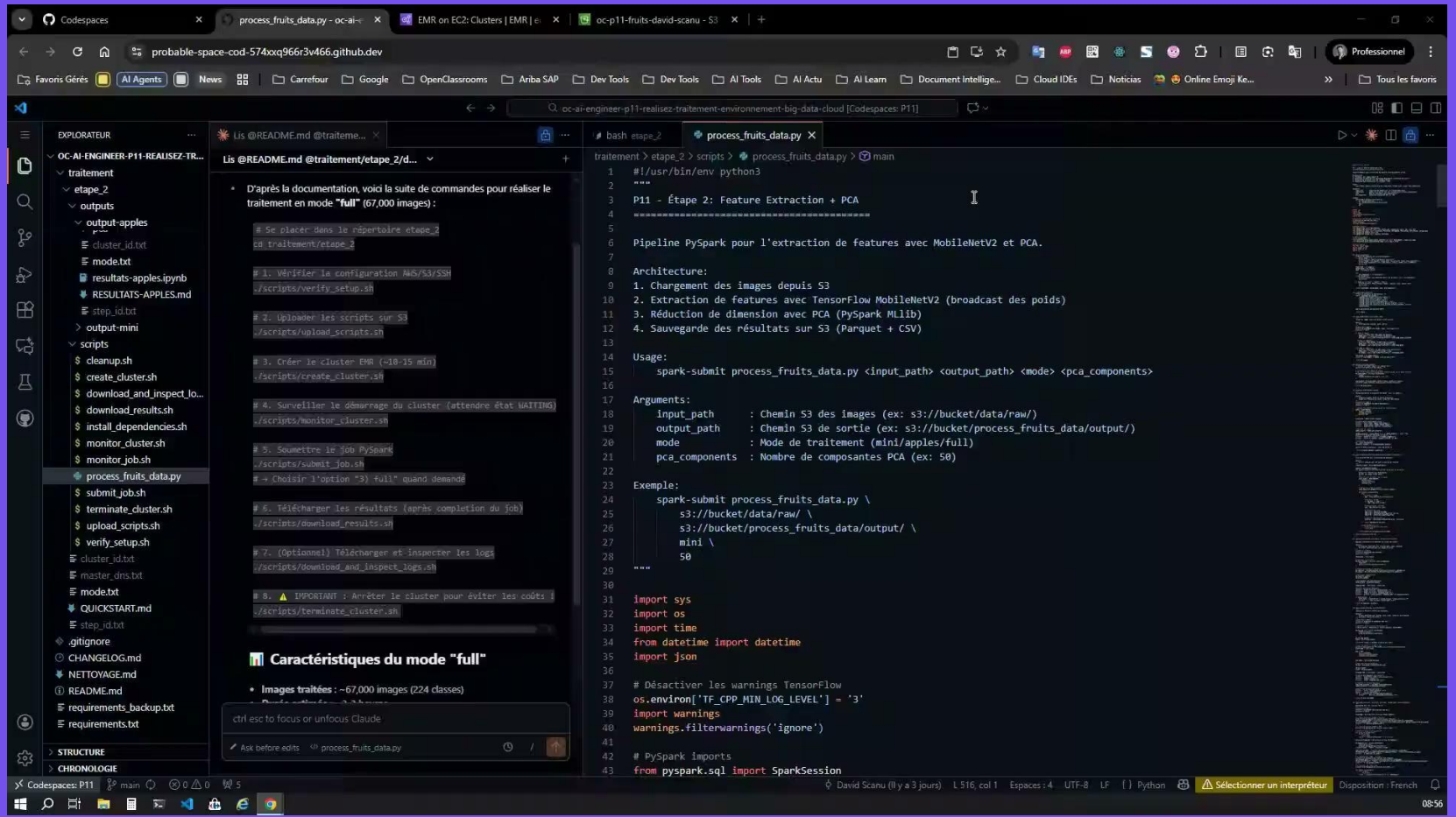
Téléchargement Local

- **Script de téléchargement complet** :
  - `/scripts/download_results.sh full`
- **Emplacement stockage local :**
  - `traitement/etape_2/outputs/output-full/`
- **Formats disponibles** :
  - Parquet (optimisé Spark) + CSV (lisible

☁️ Commandes AWS

```
# Télécharger tous les résultats du mode FULL
aws s3 cp
s3://oc-p11-fruits-david-scanu/process_fruits_data/outputs/
output-full/ ./traitement/etape_2/outputs/output-full/
```

```
s3://oc-p11-fruits-david-scanu/process_fruits_data/outputs/output-full/
├── features/
│   ├── parquet/features_20251125_092304/
│   │   ├── _SUCCESS
│   │   └── part-*.parquet (~1.5-1.8 GB estimé)
│   └── csv/features_20251125_092304/
│       ├── _SUCCESS
│       └── part-*.csv (2116 fichiers)
├── pca/
│   ├── parquet/pca_20251125_092304/
│   │   ├── _SUCCESS
│   │   └── part-*.parquet (~150-200 MB estimé)
│   └── csv/pca_20251125_092304/
│       ├── _SUCCESS
│       └── part-*.csv
├── metadata/metadata_20251125_092304/
│   ├── _SUCCESS
│   └── part-*.csv (~7 MB, 69,808 lignes dont headers)
├── model_info/
│   ├── model_info_20251125_092304/
│   │   ├── _SUCCESS
│   │   └── part-*.txt (JSON: ~3 KB)
│   └── variance_20251125_092304/
│       ├── _SUCCESS
│       └── part-*.csv (variance par composante)
└── errors/ # Absent car 100% succès ✅
```


Codespaces
process_fruits_data.py - oc-ai-e
EMR on EC2: Clusters | EMR | e
oc-p11-fruits-david-scanu - S3
probable-space-cod-574xxq966r3v466.github.dev
Professionnel
Favoris Gérés
AI Agents
News
Carrefour
Google
OpenClassrooms
Ariba SAP
Dev Tools
Dev Tools
AI Tools
AI Actu
AI Learn
Document Intellige...
Cloud IDEs
Noticias
Online Emoji Ke...
Tous les favoris
oc-ai-engineer-p11-realisez-traitement-environnement-big-data-cloud [Codespaces: P11]
EXPLORATEUR
OC-AI-ENGINEER-P11-REALISEZ-TR...
traitement
etape_2
outputs
output-apples
cluster_id.txt
mode.txt
resultats-apples.ipynb
RESULTATS-APPLES.md
step_id.txt
output-mini
scripts
cleanup.sh
create_cluster.sh
download_and_inspect_lo...
download_results.sh
install_dependencies.sh
monitor_cluster.sh
monitor_job.sh
process_fruits_data.py
submit_job.sh
terminate_cluster.sh
upload_scripts.sh
verify_setup.sh
cluster_id.txt
master_dns.txt
mode.txt
QUICKSTART.md
step_id.txt
.gitignore
CHANGELOG.md
NETTOYAGE.md
README.md
requirements_backup.txt
requirements.txt
STRUCTURE
CHRONOLOGIE
Lis @README.md @traiteme...
Lis @README.md @traitement/etape_2/d...
D'après la documentation, voici la suite de commandes pour réaliser le traitement en mode "full" (67,000 images) :
Se placer dans le répertoire etape_2
cd traitement/etape_2
1. Vérifier la configuration AWS/S3/SSH
./scripts/verify_setup.sh
2. Uploader les scripts sur S3
./scripts/upload_scripts.sh
3. Créer le cluster EMR (~10-15 min)
./scripts/create_cluster.sh
4. Surveiller le démarrage du cluster (attendre état WAITING)
./scripts/monitor_cluster.sh
5. Soumettre le job PySpark
./scripts/submit_job.sh
→ Choisir l'option "3) full" quand demandé
6. Télécharger les résultats (après completion du job)
./scripts/download_results.sh
7. (Optionnel) Télécharger et inspecter les logs
./scripts/download_and_inspect_logs.sh
8. ⚠ IMPORTANT : Arrêter le cluster pour éviter les coûts !
./scripts/terminate_cluster.sh
Caractéristiques du mode "full"
Images traitées : ~67,000 images (224 classes)
ctrl esc to focus or unfocus Claude
Ask before edits
process_fruits_data.py
bash etape_2
process_fruits_data.py
traitement > etape_2 > scripts > process_fruits_data.py > main
#!/usr/bin/env python3
"""
P11 - Étape 2: Feature Extraction + PCA
Pipeline PySpark pour l'extraction de features avec MobileNetV2 et PCA.
Architecture:
1. Chargement des images depuis S3
2. Extraction de features avec TensorFlow MobileNetV2 (broadcast des poids)
3. Réduction de dimension avec PCA (PySpark MLlib)
4. Sauvegarde des résultats sur S3 (Parquet + CSV)
Usage:
spark-submit process_fruits_data.py <input_path> <output_path> <mode> <pca_components>
Arguments:
input_path : Chemin S3 des images (ex: s3://bucket/data/raw/)
output_path : Chemin S3 de sortie (ex: s3://bucket/process_fruits_data/output/)
mode : Mode de traitement (mini/apples/full)
pca_components : Nombre de composantes PCA (ex: 50)
Exemple:
spark-submit process_fruits_data.py \
s3://bucket/data/raw/ \
s3://bucket/process_fruits_data/output/ \
mini \
50
"""
import sys
import os
import time
from datetime import datetime
import json
# Désactiver les warnings TensorFlow
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import warnings
warnings.filterwarnings('ignore')
# PySpark imports
from pyspark.sql import SparkSession
Codespaces: P11
main
David Scanu (Il y a 3 jours)
L 516, col 1
Espaces : 4
UTF-8
LF
Python
Sélectionner un interpréteur
Disposition : French
08:56

🚀 Applications & Perspectives

🌟 Applications immédiates

- Classification multi-classe
- Recherche de similarité entre fruits
- Clustering pour découverte de patterns

💡 Évolutions possibles

- **Scale-out** :
  - Plus de nœuds pour millions d'images
- **Autres modèles** :
  - ResNet, EfficientNet, ViT

🔬 Cas d'Usage Métier

Pour "Fruits!" (AgriTech) :

- **Reconnaissance en temps réel** :
  - Robots cueilleurs intelligents
- **Optimisation récolte** :
  - Détection maturité par features visuelles
- **Traçabilité** :
  - Identification variétés pour biodiversité

Extensions possibles

- **Détection anomalies** : Fruits abîmés, maladies
- **Prédiction qualité** : Score basé sur features extraites
- **Multi-modal** : Fusion images + capteurs (poids, température)

🎯 Mission Accomplie

> ✅ Pipeline Big Data Production-Ready validé à grande échelle

📊 Résultats

- **67,692 images traitées** en 83 minutes (814 img/min)
- **Extraction de features** avec ImageNetV2 (1280 dims)
- **Réduction de dimensions** (PCA) : 71.88% variance
- **Erreurs** : zéro

💼 Valeur Ajoutée pour l'Entreprise

- **Avant** : Traitement local, limité, non scalable
- **Après** : Cloud distribué, Scalable à millions d'images, Production-ready

🏆 Accomplissements

- **Architecture production-ready**
- **Pipeline PySpark distribué** (AWS EMR + S3)
- **Optimisations Big Data** (Broadcast des poids TensorFlow TF, Pandas UDF, PCA MLlib)
- **Migration cloud AWS** (EMR + S3)
- **Conformité GDPR** (région eu-west-1)
- **Coûts maîtrisés** (< 3€ total projet)

🚀 Ce projet démontre

- Maîtrise Big Data
- Architecture Cloud
- Deep Learning à échelle
- Optimisations Big Data avancées
- Gestion projet complète

Merci 🙏

Avez-vous des questions ?

🗂️ Livrables

- [GitHub] Projet 11 : Réalisez un traitement dans un environnement Big Data sur le Cloud
- Script PySpark | GitHub
- Résultats - Étape 2 : Feature Extraction + PCA (Mode FULL) | Google Docs

📺 Vidéo de démonstration de l'application

- https://drive.google.com/file/d/1mNIPZiiiolxgunTkLZ-HbXhzl8H_wmaB/view?usp=sharing

👋 A propos

- David Scanu - Développeur en intelligence artificielle - Carrefour | LinkedIn
- Formation AI Engineer - OpenClassrooms

👋 Documentation

- Tutorial: Getting started with Amazon EMR