Documentation Technique Complète

Projet Deep Learning - Classification CIFAR-10

Etudiant: Ahmed Belhareth **Module:** Deep Learning **Enseignant:** Haythem Ghazouani **Date:** Décembre 2024

Table des Matières

1. Introduction et Objectif
2. Architecture du Projet
3. Technologies Utilisées
4. Dataset CIFAR-10
5. Architecture du Modèle
6. Pipeline d'Entraînement
7. Prétraitement et Augmentation
8. API REST FastAPI
9. Interface Streamlit
10. MLOps avec MLflow
11. Conteneurisation Docker
12. Tests Unitaires
13. Guide d'Installation
14. Résultats et Métriques

---

1. Introduction et Objectif

1.1 Problématique

Classifier automatiquement des images en 10 catégories à partir du dataset CIFAR-10 contenant 60 000 images de 32x32 pixels.

1.2 Solution Proposée

Utilisation d'un modèle **ResNet-18 pré-entraîné sur ImageNet** avec fine-tuning sur CIFAR-10 pour atteindre une accuracy optimale.

1.3 Résultats Obtenus

Métrique	Résultat
Accuracy	**87.09%**
F1-Score	**0.8705**

---

2. Architecture du Projet

2.1 Structure des Répertoires

```
projet-deep-learning/
│
├── src/                          # Code source principal
│   ├── __init__.py
│   ├── api/                      # API REST FastAPI
│   │   ├── __init__.py
│   │   ├── main.py              # Point d'entrée de l'API
│   │   └── endpoints.py         # Définition des endpoints
│   │
│   ├── data/                     # Gestion des données
│   │   ├── __init__.py
│   │   ├── dataset.py           # Chargement CIFAR-10
│   │   └── preprocessing.py     # Prétraitement et augmentation
│   │
│   ├── models/                   # Modèles de deep learning
│   │   ├── __init__.py
│   │   ├── architecture.py      # Architecture ResNet
│   │   └── training.py          # Boucle d'entraînement
│   │
│   └── utils/                    # Utilitaires
│       ├── __init__.py
│       ├── config.py            # Configuration globale
│       └── metrics.py           # Métriques d'évaluation
│
├── streamlit_app/                # Interface utilisateur
│   ├── app.py                   # Application Streamlit principale
│   ├── components/              # Composants UI réutilisables
│   └── pages/                   # Pages supplémentaires
│
├── notebooks/                    # Jupyter Notebooks
│   ├── 01_exploration_donnees.ipynb
│   ├── 02_entrainement_modele.ipynb
│   └── 03_evaluation.ipynb
│
├── tests/                        # Tests unitaires
│   ├── __init__.py
│   ├── test_api.py
│   ├── test_data.py
│   └── test_models.py
│
├── docker/                       # Conteneurisation
│   ├── Dockerfile.api
│   ├── Dockerfile.streamlit
│   └── docker-compose.yml
│
├── models/                       # Modèles sauvegardés
│   └── best_model.pth           # Meilleur modèle entraîné
│
├── mlruns/                       # Tracking MLflow
│
├── test_images/                  # Images de test
```

```
│   ├── airplane.jpg
│   ├── bird.jpg
│   └── car.jpg
│
├── requirements.txt              # Dépendances Python
└── README.md                     # Documentation rapide
```

---

3. Technologies Utilisées

3.1 Stack Technique Complet

Catégorie	Technologie	Version	Rôle
Deep Learning	PyTorch	>=2.0.0	Framework principal
	torchvision	>=0.15.0	Datasets et transforms
	timm	>=0.9.0	Modèles pré-entraînés
Computer Vision	OpenCV	>=4.8.0	Traitement d'images
	Pillow	>=10.0.0	Manipulation d'images
	Albumentations	>=1.3.0	Augmentation avancée
Data Science	NumPy	>=1.24.0	Calcul numérique
	Pandas	>=2.0.0	Manipulation de données
	Matplotlib	>=3.7.0	Visualisation
	Seaborn	>=0.12.0	Graphiques statistiques
	Scikit-learn	>=1.3.0	Métriques ML
MLOps	MLflow	>=2.8.0	Tracking expériences
	Docker	-	Conteneurisation
API Backend	FastAPI	>=0.104.0	API REST
	Uvicorn	>=0.24.0	Serveur ASGI
Interface UI	Streamlit	>=1.28.0	Dashboard web
	Plotly	>=5.0.0	Graphiques interactifs
Tests	pytest	>=7.4.0	Tests unitaires
	pytest-cov	>=4.1.0	Couverture de code
	httpx	>=0.25.0	Tests API async

---

4. Dataset CIFAR-10

4.1 Description

Le dataset CIFAR-10 est un benchmark standard pour la classification d'images :

- **60 000 images** couleur de 32x32 pixels
- **50 000 images** d'entraînement
- **10 000 images** de test
- **10 classes** équilibrées (6 000 images par classe)

4.2 Classes

ID	Français	Anglais
0	Avion	airplane
1	Automobile	automobile
2	Oiseau	bird
3	Chat	cat
4	Cerf	deer
5	Chien	dog
6	Grenouille	frog
7	Cheval	horse
8	Navire	ship
9	Camion	truck

4.3 Chargement du Dataset

Fichier: `src/data/dataset.py`

```
def load_cifar10_dataset(download: bool = True):
    """Télécharge et charge le dataset CIFAR-10."""
    train_dataset = torchvision.datasets.CIFAR10(
        root=str(DATA_DIR),
        train=True,
        download=download,
        transform=get_train_transforms()
    )
    test_dataset = torchvision.datasets.CIFAR10(
        root=str(DATA_DIR),
        train=False,
        download=download,
        transform=get_test_transforms()
    )
    return train_dataset, test_dataset
```

4.4 DataLoaders

- **Batch Size:** 128
- **Validation Split:** 10%
- **Num Workers:** 2
- **Pin Memory:** True (optimisation GPU)

---

5. Architecture du Modèle

5.1 ResNet-18 avec Transfer Learning

Fichier: `src/models/architecture.py`

```
Architecture: ResNet-18
```

```
├── Conv1 (7x7, 64 filtres)
├── BatchNorm + ReLU + MaxPool
├── Layer1 (2 BasicBlocks, 64 filtres)
├── Layer2 (2 BasicBlocks, 128 filtres)
├── Layer3 (2 BasicBlocks, 256 filtres)
├── Layer4 (2 BasicBlocks, 512 filtres)
├── AdaptiveAvgPool (1x1)
└── Classifier Custom
    ├── Dropout (p=0.2)
    └── Linear (512 → 10)
```

5.2 Classe CIFAR10ResNet

```
class CIFAR10ResNet(nn.Module):
    def __init__(
        self,
        model_name: str = "resnet18",
        num_classes: int = 10,
        pretrained: bool = True,
        dropout_rate: float = 0.2,
        freeze_backbone: bool = False
    ):
        # Charger ResNet pré-entraîné sur ImageNet
        weights = models.ResNet18_Weights.IMAGENET1K_V1
        self.backbone = models.resnet18(weights=weights)

        # Remplacer la tête de classification
        num_features = self.backbone.fc.in_features  # 512
        self.backbone.fc = nn.Sequential(
            nn.Dropout(p=dropout_rate),
            nn.Linear(num_features, num_classes)
        )
```

5.3 Paramètres du Modèle

Paramètre	Valeur
Paramètres totaux	~11.2M
Paramètres entraînables	~11.2M
Taille d'entrée	32x32x3
Taille de sortie	10

5.4 Fonctionnalités Avancées

- **Gel du backbone:** Option pour fine-tuning partiel
- **Extraction de features:** Méthode `get_features()` pour analyse
- **Support multi-architectures:** ResNet-18, ResNet-34, ResNet-50, EfficientNet-B0

---

6. Pipeline d'Entraînement

6.1 Configuration

Fichier: `src/utils/config.py`

```
# Hyperparamètres
BATCH_SIZE = 128
NUM_EPOCHS = 50
LEARNING_RATE = 0.001
WEIGHT_DECAY = 1e-4

# Early Stopping
EARLY_STOPPING_PATIENCE = 10
EARLY_STOPPING_MIN_DELTA = 0.001

# Optimiseur et Scheduler
OPTIMIZER = "adamw"
SCHEDULER = "cosine"
```

6.2 Boucle d'Entraînement

Fichier: `src/models/training.py`

```
def train_model(model, train_loader, val_loader, device, num_epochs=50)
    # Critère: Cross Entropy Loss
    criterion = nn.CrossEntropyLoss()

    # Optimiseur: AdamW avec weight decay
    optimizer = optim.AdamW(
        model.parameters(),
        lr=LEARNING_RATE,
        weight_decay=WEIGHT_DECAY
    )

    # Scheduler: Cosine Annealing
    scheduler = CosineAnnealingLR(optimizer, T_max=num_epochs)

    # Early Stopping
    early_stopping = EarlyStopping(patience=10, mode='min')

    for epoch in range(num_epochs):
        # Training
        train_loss, train_acc = train_one_epoch(...)

        # Validation
        val_loss, val_acc, metrics = validate(...)

        # Scheduler step
        scheduler.step()

        # Sauvegarder le meilleur modèle
        if val_acc > best_val_acc:
            torch.save(model.state_dict(), 'best_model.pth')
```

```
        # Early stopping check
        if early_stopping(val_loss, epoch):
            break
```

6.3 Early Stopping

```
class EarlyStopping:
    def __init__(self, patience=10, min_delta=0.001, mode='min'):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.best_score = None

    def __call__(self, score, epoch):
        # Arrête si pas d'amélioration pendant 'patience' époques
        ...
```

---

7. Prétraitement et Augmentation

7.1 Transformations d'Entraînement

Fichier: `src/data/preprocessing.py`

```
def get_train_transforms():
    return transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.ColorJitter(
            brightness=0.2,
            contrast=0.2,
            saturation=0.2,
            hue=0.1
        ),
        transforms.ToTensor(),
        transforms.Normalize(mean=CIFAR10_MEAN, std=CIFAR10_STD),
        transforms.RandomErasing(p=0.1)  # Cutout
    ])
```

7.2 Augmentations Albumentations

```
def get_albumentations_train_transform():
    return A.Compose([
        A.HorizontalFlip(p=0.5),
        A.ShiftScaleRotate(
            shift_limit=0.1,
            scale_limit=0.1,
            rotate_limit=15,
            p=0.5
        ),
        A.ColorJitter(...),
        A.CoarseDropout(  # Cutout avancé
```

```
            max_holes=1,
            max_height=8,
            max_width=8,
            p=0.3
        ),
        A.Normalize(mean=CIFAR10_MEAN, std=CIFAR10_STD),
        ToTensorV2()
    ])
```

7.3 Techniques d'Augmentation Avancées

MixUp

```
def mixup_data(x, y, alpha=0.2):
    """Mélange deux images et leurs labels."""
    lam = np.random.beta(alpha, alpha)
    index = torch.randperm(batch_size)
    mixed_x = lam * x + (1 - lam) * x[index]
    return mixed_x, y, y[index], lam
```

CutMix

```
def cutmix_data(x, y, alpha=1.0):
    """Découpe une région et la remplace."""
    # Calcul coordonnées du rectangle
    # Remplacement de la région
    return x_cutmix, y_a, y_b, lam
```

7.4 Normalisation

```
# Statistiques CIFAR-10
CIFAR10_MEAN = [0.4914, 0.4822, 0.4465]
CIFAR10_STD = [0.2470, 0.2435, 0.2616]
```

---

8. API REST FastAPI

8.1 Configuration

Fichier: `src/api/main.py`

```
app = FastAPI(
    title="API Classification CIFAR-10",
    description="API de classification d'images utilisant ResNet",
    version="1.0.0",
    docs_url="/docs",
    redoc_url="/redoc"
)

# CORS Middleware
app.add_middleware(
    CORSMiddleware,
```

```
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"]
)
```

8.2 Endpoints Disponibles

Endpoint	Méthode	Description
`/`	GET	Message de bienvenue
`/api/v1/health`	GET	État de santé de l'API
`/api/v1/classes`	GET	Liste des classes CIFAR-10
`/api/v1/predict`	POST	Prédiction sur une image
`/api/v1/batch_predict`	POST	Prédiction sur plusieurs images
`/api/v1/predict_base64`	POST	Prédiction avec image base64
`/api/v1/model/info`	GET	Informations sur le modèle
`/docs`	GET	Documentation Swagger UI
`/redoc`	GET	Documentation ReDoc

8.3 Schémas de Réponse

```
class PredictionResponse(BaseModel):
    class_id: int           # ID de la classe (0-9)
    class_name: str         # Nom de la classe
    confidence: float       # Confiance (0-1)
    probabilities: Dict[str, float]  # Probabilités par classe

class ModelInfoResponse(BaseModel):
    model_name: str
    num_classes: int
    input_size: str
    total_parameters: int
    trainable_parameters: int
    device: str
```

8.4 Exemple de Requête

```
# Prédiction simple
curl -X POST "http://localhost:8000/api/v1/predict" \
  -H "accept: application/json" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@image.jpg"

# Réponse
{
  "class_id": 0,
  "class_name": "Avion",
  "confidence": 0.95,
  "probabilities": {
    "Avion": 0.95,
    "Automobile": 0.02,
```

```
    ...
  }
}
```

---

9. Interface Streamlit

9.1 Configuration

Fichier: `streamlit_app/app.py`

```
st.set_page_config(
    page_title="Classification d'Images CIFAR-10",
    page_icon="🖼 ",
    layout="wide",
    initial_sidebar_state="expanded"
)
```

9.2 Fonctionnalités

1. **Upload d'images:** JPG, PNG, WEBP
2. **Prédiction en temps réel**
3. **Graphique des probabilités** (Plotly)
4. **Top 3 prédictions**
5. **Informations techniques**
6. **Sidebar avec classes**

9.3 Workflow de Prédiction

```
def predict_image(model, image, device):
    # 1. Prétraitement
    input_tensor = preprocess_single_image(image)
    input_tensor = input_tensor.to(device)

    # 2. Inférence
    with torch.no_grad():
        outputs = model(input_tensor)
        probabilities = torch.softmax(outputs, dim=1)
        confidence, predicted = probabilities.max(1)

    # 3. Résultats
    class_name = CIFAR10_CLASSES[predicted.item()]
    return class_name, confidence.item() * 100, probs
```

9.4 Cache du Modèle

```
@st.cache_resource
def load_classification_model():
    """Cache le modèle pour éviter rechargements."""
    device = get_device()
    model = load_model(model_path, device=str(device))
    return model, device
```

10. MLOps avec MLflow

10.1 Configuration

Fichier: `src/utils/config.py`

```
MLFLOW_TRACKING_URI = str(PROJECT_ROOT / "mlruns")
EXPERIMENT_NAME = "CIFAR10_Classification"
RUN_NAME_PREFIX = "resnet18_run"
```

10.2 Tracking des Expériences

```
# Initialisation
mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)
mlflow.set_experiment(EXPERIMENT_NAME)

# Démarrage d'un run
mlflow.start_run()

# Logger les hyperparamètres
mlflow.log_params({
    'model_name': MODEL_NAME,
    'num_epochs': num_epochs,
    'learning_rate': learning_rate,
    'batch_size': batch_size,
    'optimizer': 'AdamW',
    'scheduler': 'CosineAnnealingLR'
})

# Logger les métriques à chaque époque
mlflow.log_metrics({
    'train_loss': train_loss,
    'train_acc': train_acc,
    'val_loss': val_loss,
    'val_acc': val_acc,
    'f1_macro': f1_score
}, step=epoch)

# Sauvegarder le modèle
mlflow.pytorch.log_model(model, "model")

mlflow.end_run()
```

10.3 Interface MLflow UI

```
mlflow ui --port 5000 --backend-store-uri ./mlruns
```

Accès: http://localhost:5000

11. Conteneurisation Docker

11.1 Dockerfile API

Fichier: `docker/Dockerfile.api`

```
FROM python:3.10-slim

ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

WORKDIR /app

# Dépendances système
RUN apt-get update && apt-get install -y \
    build-essential curl libglib2.0-0 \
    libsm6 libxext6 libxrender-dev libgl1-mesa-glx

# Dépendances Python
COPY requirements.txt .
RUN pip install -r requirements.txt

# Code source
COPY src/ ./src/
COPY models/ ./models/

EXPOSE 8000

CMD ["uvicorn", "src.api.main:app", "--host", "0.0.0.0", "--port", "800
```

11.2 Docker Compose

Fichier: `docker/docker-compose.yml`

```
version: '3.8'

services:
  api:
    build:
      context: ..
      dockerfile: docker/Dockerfile.api
    container_name: cifar10_api
    ports:
      - "8000:8000"
    volumes:
      - ../models:/app/models:ro
      - ../mlruns:/app/mlruns
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/api/v1/health"
      interval: 30s
    networks:
      - cifar10_network

  streamlit:
    build:
```

```
      context: ..
      dockerfile: docker/Dockerfile.streamlit
    container_name: cifar10_streamlit
    ports:
      - "8501:8501"
    depends_on:
      - api
    networks:
      - cifar10_network

  mlflow:
    image: ghcr.io/mlflow/mlflow:v2.8.0
    container_name: cifar10_mlflow
    ports:
      - "5000:5000"
    command: mlflow server --host 0.0.0.0 --port 5000

networks:
  cifar10_network:
    driver: bridge
```

11.3 Commandes Docker

```
# Build et lancement
cd docker
docker-compose up --build

# Lancement en arrière-plan
docker-compose up -d

# Arrêt
docker-compose down

# Logs
docker-compose logs -f api
```

---

12. Tests Unitaires

12.1 Structure des Tests

Répertoire: `tests/`

```
tests/
├── __init__.py
├── test_api.py       # Tests des endpoints API
├── test_data.py      # Tests du chargement des données
└── test_models.py    # Tests de l'architecture du modèle
```

12.2 Tests du Modèle

Fichier: `tests/test_models.py`

```
class TestModelArchitecture:
    def test_create_model(self):
        """Teste la création du modèle."""
        model = create_model()
        assert isinstance(model, CIFAR10ResNet)
        assert model.num_classes == 10

    def test_forward_pass(self):
        """Teste la passe avant."""
        model = create_model()
        x = torch.randn(4, 3, 32, 32)
        output = model(x)
        assert output.shape == (4, 10)

    def test_backward_pass(self):
        """Teste la rétropropagation."""
        model = create_model()
        x = torch.randn(4, 3, 32, 32)
        y = torch.randint(0, 10, (4,))
        loss = criterion(model(x), y)
        loss.backward()
        assert any(p.grad is not None for p in model.parameters())
```

12.3 Exécution des Tests

```
# Tous les tests
pytest tests/ -v

# Avec couverture
pytest tests/ -v --cov=src --cov-report=html

# Tests spécifiques
pytest tests/test_models.py -v
```

12.4 Résultats des Tests

- **29/29 tests passent**
- Couverture: ~85%

---

13. Guide d'Installation

13.1 Prérequis

- Python 3.10+
- Git
- GPU (optionnel mais recommandé)

13.2 Installation

```
# 1. Cloner le dépôt
git clone https://github.com/ahmedbelhareth/DeepLerningProjectTekup.git
```

```
cd projet-deep-learning

# 2. Créer environnement virtuel
python -m venv venv

# 3. Activer l'environnement
# Windows:
venv\Scripts\activate
# Linux/Mac:
source venv/bin/activate

# 4. Installer les dépendances
pip install -r requirements.txt
```

13.3 Lancement des Services

```
# API FastAPI (http://localhost:8000/docs)
uvicorn src.api.main:app --reload --port 8000

# Interface Streamlit (http://localhost:8501)
streamlit run streamlit_app/app.py

# MLflow UI (http://localhost:5000)
mlflow ui --port 5000 --backend-store-uri ./mlruns
```

13.4 Entraînement du Modèle

```
python -m src.models.training
```

---

14. Résultats et Métriques

14.1 Performance du Modèle

Métrique	Valeur
Accuracy	87.09%
F1-Score (macro)	0.8705
Precision (macro)	0.87
Recall (macro)	0.87
Top-5 Accuracy	99.2%

14.2 Performance par Classe

Classe	Precision	Recall	F1-Score
Avion	0.89	0.88	0.88
Automobile	0.92	0.93	0.92
Oiseau	0.79	0.81	0.80
Chat	0.75	0.73	0.74

Cerf	0.86	0.87	0.86
Chien	0.80	0.79	0.79
Grenouille	0.91	0.92	0.91
Cheval	0.90	0.89	0.89
Navire	0.92	0.93	0.92
Camion	0.91	0.90	0.90

14.3 Temps d'Entraînement

- **GPU (NVIDIA):** ~15-20 minutes
- **CPU:** ~2-3 heures

14.4 Temps d'Inférence

- **GPU:** ~5ms par image
- **CPU:** ~50ms par image

Annexes

A. Variables d'Environnement

```
PYTHONUNBUFFERED=1
CUDA_VISIBLE_DEVICES=0
```

B. Ports Utilisés

Service	Port
FastAPI	8000
Streamlit	8501
MLflow	5000

C. Références

- CIFAR-10 Dataset
- PyTorch Documentation
- FastAPI Documentation
- Streamlit Documentation
- MLflow Documentation

Document rédigé par Ahmed Belhareth Projet Deep Learning - TEKUP 2024