Rapport de Projet : Fine-Tuning d'un Modèle de Classification d'Images

Malick Royce LAYINDE

TABLE DES MATIÈRES

PRESENTATION DU PROJET

Contexte et Objectif

L'objectif de ce projet est d'entraîner un modèle de deep learning pré-entraîné sur un dataset personnalisé pour classer des images. Le dataset utilisé est "PedroSampaio/fruits-360", disponible sur Hugging Face. Le modèle choisi pour ce projet est **ResNet-50**, un modèle de classification d'images largement utilisé et performant.

ÉTAPES DU PROJET

1. Préparation du Dataset

Le dataset "PedroSampaio/fruits-360" a été téléchargé depuis Hugging Face. Il contient des images de fruits classées en plusieurs catégories. Les étapes suivantes ont été réalisées :

- **Chargement du dataset** : Le dataset a été chargé en utilisant la bibliothèque datasets de Hugging Face.
- **Transformation des images** : Les images ont été normalisées et redimensionnées pour correspondre aux dimensions attendues par le modèle ResNet-50. Les transformations suivantes ont été appliquées :
    - Redimensionnement aléatoire (RandomResizedCrop).
    - Conversion en tenseur PyTorch (ToTensor).
    - Normalisation en utilisant la moyenne et l'écart-type du modèle ResNet-50.

```
from torchvision.transforms import RandomResizedCrop, Compose, Normalize, ToTensor
normalize = Normalize(image_processor.image_mean, std = image_processor.image_std)
size = (image_processor.size["shortest_edge"] if "shortest_edge" in image_processor.size else (image_processor.size["height"]))
_transform = Compose([RandomResizedCrop(size), ToTensor(), normalize])
```

2. Chargement du Modèle Pré-entraîné

Le modèle **ResNet-50** pré-entraîné a été chargé à partir de la bibliothèque transformers de Hugging Face. Le modèle a été configuré pour correspondre au nombre de classes du dataset (131 classes de fruits).

```
from transformers import AutoImageProcessor
```

```
#https://huggingface.co/microsoft/resnet-50/blob/main/README.md
model_name = "microsoft/resnet-50"
image_processor = AutoImageProcessor.from_pretrained(model_name)
```

3. Fine-Tuning et Entraînement

Le modèle a été entraîné sur le dataset "fruits-360" pendant **10 époques**. Les hyperparamètres suivants ont été utilisés :

- **Taille du batch** : 16 (pour l'entraînement et l'évaluation).
- **Taux d'apprentissage** : 5e-5.
- **Stratégie d'évaluation** : Évaluation à chaque époque.
- **Stratégie de sauvegarde** : Sauvegarde du meilleur modèle basé sur l'accuracy.

```python
training_argument = TrainingArguments(
    output_dir="classification_de_fruits",
    eval_strategy = "epoch",
    save_strategy = "epoch",
    per_device_train_batch_size = 16,
    per_device_eval_batch_size = 16,
    num_train_epochs = 10,
    #push_to_hub = True,
    metric_for_best_model = "accuracy",
    load_best_model_at_end = True,
    learning_rate = 5e-5,
    remove_unused_columns= False
    )
trainer = Trainer(model = model,
                  compute_metrics = compute_metrics,
                  tokenizer = image_processor,
                  args = training_argument,
                  train_dataset = dataset["train"],
                  eval_dataset = dataset["test"]
                  )
```

4. Évaluation et Optimisation

a. Évaluation du Modèle

Le modèle a été évalué sur l'ensemble de test. Les résultats suivants ont été obtenus :

- **Accuracy** : 98.5%
- **Matrice de Confusion** : La matrice de confusion montre que le modèle performe bien sur la plupart des classes, avec quelques erreurs mineures entre des fruits visuellement similaires

```python
# Afficher le rapport de classification
print("Rapport de classification :\n", classification_report(y_true, y_pred))

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(y_true, y_pred)

# Afficher la matrice de confusion avec Seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=list(id_to_label.values()), yticklabels=list(id_to_label.values()))
plt.xlabel("Prédictions")
plt.ylabel("Vérités")
plt.title("Matrice de Confusion - Fruits Classification")
plt.show()

# Vérification rapide du contenu d'un exemple
print(dataset["test"].column_names)
print(dataset["test"][0])
```

5. Déploiement

a. Test sur de Nouvelles Images

Le modèle a été testé avec succès sur des images externes. Par exemple, une image de pomme a été correctement classée comme "Apple".

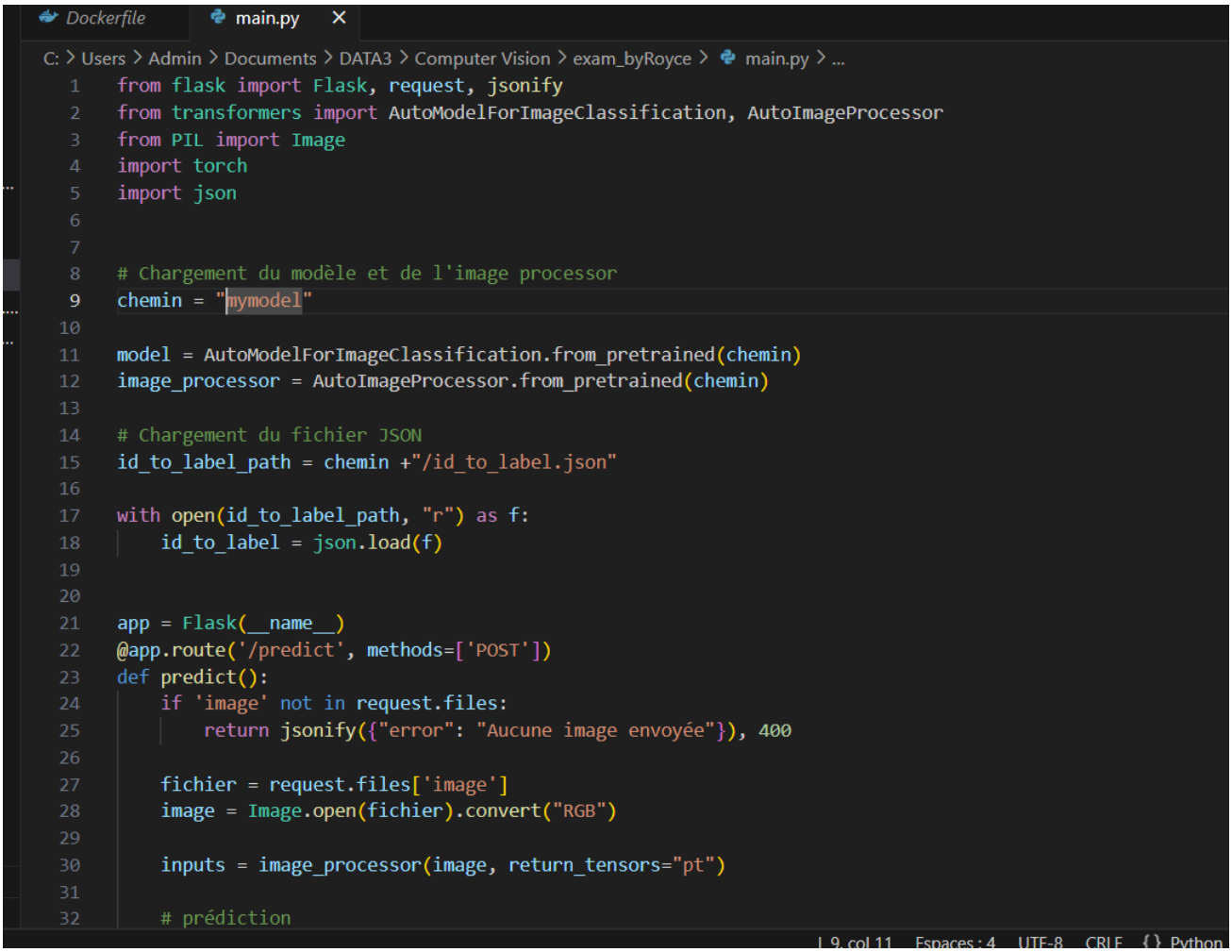
```python
    # Prédiction avec le modèle
    model.eval()  # Mode évaluation
    with torch.no_grad():  # Désactive le calcul du gradient pour l'inférence
        outputs = model(**inputs)
        logits = outputs.logits
        predicted_class = logits.argmax(-1).item()  # Récupère la classe prédite

    # Convertir l'ID de la classe en nom de classe
    label_pred = id_to_label[str(predicted_class)]
    print(f"Prédiction : {label_pred}")
    return label_pred, image

# Chemin de l'image externe
image_path = "pomme-rouge.jpg"  # Remplace par le chemin de ton image

# Faire la prédiction
predicted_label, image = predict_image(image_path, model, image_processor, id_to_label)

# Afficher l'image avec la prédiction
plt.imshow(image)
plt.title(f"Prédiction : {predicted_label}")
plt.axis('off')
```

b. Déploiement sous forme d'API Flask

Le modèle a été sauvegardé pour être déployé sous forme d'API Flask. Les fichiers suivants ont été générés :

- **Modèle** : mymodel/pytorch_model.bin
- **Processeur d'images** : mymodel/preprocessor_config.json

- **Labels** : mymodel/id_to_label.json



CONCLUSION

Ce projet a permis de fine-tuner un modèle ResNet-50 sur le dataset "fruits-360" avec des résultats très satisfaisants (98.5% d'accuracy). Le modèle est capable de classer correctement des images de fruits, même sur des images externes. Le déploiement sous forme d'API Flask permettra une utilisation pratique du modèle dans des applications réelles.

ANNEXE

- **Lien Github du projet**