Université Claude Bernard Lyon 1

RENDU DU PROJET

PARTIE : DETECTION DE TEXTE

AUTEUR : GODFREE AKAKPO

Le présent document constitue le rendu officiel du projet rassemblant les notions de détection d'objet, en occurrence dans ce projet précis la détection de texte élaboré dans le cours « Projet informatique » au sein de l'université Claude Bernard Lyon1. Ce projet a été conçu et réalisé par Godfree AKAKPO, étudiants en licence 3 informatique à l'université de Lyon 1.

Nb : Je rappelle que ce projet est l'une des parties d'un projet qui consiste à détecter et à reconnaitre du texte. Une partie qui peut être utiliser indépendamment. Donc dans cette partie nous aurons à parler que de la partie concernant la détection de texte.

OBJECTIF DU PROJET

L'objectif principal de ce projet était de concevoir, développer et mettre en œuvre un programme qui servira à vérifier s'il existe du texte dans un model ou pas si oui de nous ressortir les coordonnées associes aux textes. Ce programme est question n'est pas une révolution, ni une nouveauté puisque cela existe déjà et beaucoup plus performant.

TECHNOLOGIE UTILISEE

Nous avons opté pour Python pour le développement de notre programme en raison de leur adaptabilité, de leur robustesse et de leur adéquation aux besoins du projet. Comme ce projet est beaucoup plus tourné vers le Deep Learning alors Python se présente comme l'outils idéals car il est beaucoup plus adapté.

ORGANISATION DU DOCUMENT

Ce document est divisé en plusieurs sections pour faciliter la compréhension et l'évaluation du projet. Les sections incluent :

- Petit rappel sur les notions de base pour comprendre l'aspect technique du projet.
- Qu'est-ce qu'un Réseau de neurone et ses couches ?
- Fonction de Perte (MSEloss)
- Les classes et fonctions principales du programme
- Les différents résultats
- Versions améliorées du projet.

En résumé, ce projet représente notre engagement envers l'excellence dans la conception et la mise en œuvre d'une solution informatique répondant aux besoins spécifiques énoncés dans le cahier des charges. Nous sommes impatients de présenter chaque aspect du projet dans les sections suivantes.

Petit rappel sur les notions de base pour comprendre l'aspect technique du projet.

Tout d'abord il faut comprendre les notions mathématiques comme les matrices et tenseurs. Connaître et maitriser les listes en python.

Qu'est-ce qu'un Réseau de neurone et ses couches ?

Les réseaux de neurones, également connus sous le nom de réseaux de neurones artificiels (ANN) ou réseaux de neurones simulés (SNN) constituent un sous-ensemble de l'apprentissage automatique et sont au cœur des algorithmes de l'apprentissage en profondeur. Leur nom et leur structure sont inspirés du cerveau humain, imitant la manière dont les neurones biologiques s'envoient des signaux.
Source : https://www.ibm.com/fr-fr/topics/neural-networks

Généralement, chaque neurone d'une couche est lié avec tous les neurones de la couche en aval et celle-ci uniquement.

Fonction de Perte (MSEloss)

La fonction de perte ou de loss permet de calculer l'erreur entre les prédictions de ton modèle et les valeurs réelles.
Plus la loss baisse, plus le modèle est performant !

Pendant l'entraînement, cela permet au modèle de savoir s'il avance dans la bonne direction ou non.
L'objectif étant que loss soit le plus proche possible de zéro.
Cependant, il y a plusieurs manières de calculer l'erreur entre les prédictions et les valeurs réelles.

Dans notre programme spécialement on a utilisé la fonction MSEloss. Mais pourquoi avons-nous choisis la fonction MSEloss ? La Mean Squared Error (MSE), en français erreur quadratique moyenne, est une des fonctions de loss les plus utilisées dans les problèmes de régression.

Elle calcule la différence moyenne au carré entre les valeurs prédites et les valeurs réelles.

Donc pas de panique lorsque la valeur de retour est très éloignée 0. Cette valeur de retour est la valeur prédit par rapport à la valeur réelle.

Les classes et fonctions principales du programme

Class TextDetectionModel(nn.Module): cette classe définis l'objet Model. Elle permet d'instancier le model. Il formé de 3 couches de convolution et une couche linéaire

Class CustomDataset(Dataset)*:* cette classe permet d'instancier un objet Dataset qui a pour rode charger les données

DataLoader(custom_dataset, batch_size=128, shuffle=True) : cette fonction est une fonction prédéfinis de la bibliothèque torch.utils.data qui permet de charger les donnes par lots dans le but principale d'économiser de la mémoire.

optim.Adam(model.parameters(), lr=0.0001) : cette fonction est la fonction qui définis l'l'optimiseur qui sera utiliser pour l'entrainement du model.

Boucle d'apprentissage : dans cette boucle se passe l'apprentissage des données

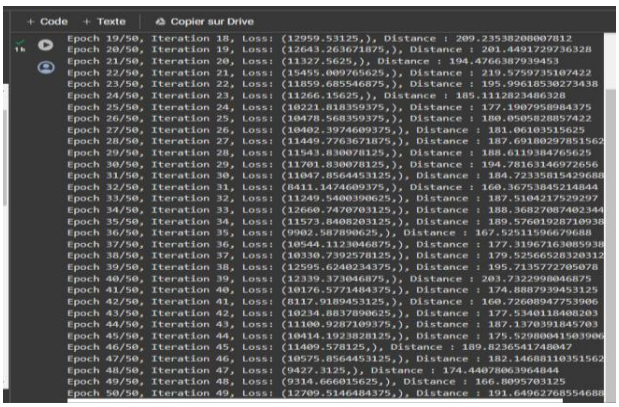
nn.MSELoss()(regression_output, coordinates) : comme définie plus haut cette fonction communément appelé la loss qui est une clé du bon fonctionnement de ce programme. Puisse qu'on ne parle pas d'une probabilité mais plutôt d'une valeur a prédire ici alors La MSELoss est l'idéale a utilisé. Elle calcule la différence moyenne au carré entre les valeurs prédites et les valeurs réelles.

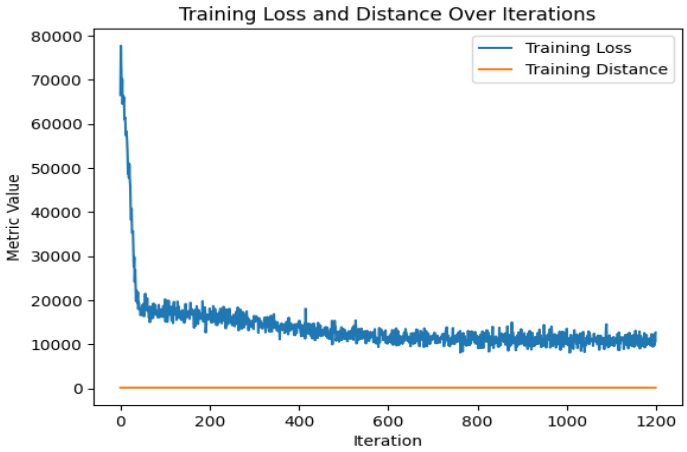
```
class TextDetectionModel(nn.Module):
    def __init__(self):
        super(TextDetectionModel, self).__init__()

        # Couches de convolution pour l'extraction des caractéristiques
        self.conv1 = nn.Conv2d(3, 6, kernel_size=3, stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        self.conv2 = nn.Conv2d(6, 6, kernel_size=3, stride=1, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        self.conv3 = nn.Conv2d(6, 10, kernel_size=3, stride=1, padding=1)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        # Couches entièrement connectées pour la régression
        self.fc1 = nn.Linear(38440, 512)
        self.relu3 = nn.ReLU()
        self.fc3 = nn.Linear(512, 4)  # 4 valeurs pour les coordonnées (x, y,
width, height)

    def forward(self, x):
        # Propagation avant à travers les couches de convolution
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.pool1(x)
        x = self.conv3(x)
        x = self.relu3(x)
        x = self.pool3(x)

        # Aplatir les caractéristiques pour les couches entièrement connectées
        x = x.view(x.size(0), -1)

        # Propagation avant à travers les couches entièrement connectées
        x = self.fc1(x)

        # Sorties pour la classification (2 classes) et la régression (4 valeurs)
        x = self.fc3(x)
        regression_output = x

        return  regression_output


model = TextDetectionModel()

class CustomDataset(Dataset):
    def __init__(self, csv_file, image_folder):
        self.data = pd.read_csv(csv_file)
        self.image_folder = image_folder
        self.transform = transforms.Compose([
            transforms.Resize((250, 250)),
            transforms.ToTensor(),
        ])

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        image_name = self.data.iloc[idx, 0]
        image_path = os.path.join(self.image_folder, image_name)
```

```
csv_path = '/content/drive/MyDrive/Classroom/csv_file/coordinates.csv'
images_path = '/content/drive/MyDrive/Classroom/img/'

# Créer une instance de votre Dataset personnalisé
custom_dataset = CustomDataset(csv_path, images_path)

dataloader = DataLoader(custom_dataset, batch_size=128, shuffle=True)


optimizer = optim.Adam(model.parameters(), lr=0.0001)

#-------------------------------------------------
checkpoint_path =
'/content/drive/MyDrive/Classroom/train_ckeck/model.pth'
if os.path.exists(checkpoint_path):
    checkpoint = torch.load(checkpoint_path)
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    epoch = checkpoint['epoch']
    loss = checkpoint['loss']
    distance = checkpoint['distance moyenne']
    # Ajoutez d'autres éléments que vous avez sauvegardés

    print(f"Chargement du point de contrôle à l'époque {epoch} avec une
perte de {loss}")
else:
    print("Aucun point de contrôle trouvé, l'entraînement commencera
depuis le début.")

#-------------------------------------------------
train_losses = []
iteration = 0
train_distances = []
n = 50
for epoch in range(n):
    model.train()

    for batch in dataloader:

        images = batch['image']
        coordinates = batch['coordonnees']
        """ print(f'Type of images: {type(images)}')
        print(batch)
        print(type(batch))
        print("images = ", type(images))
        print("coordinates = ", type(coordinates))"""

        optimizer.zero_grad()

        # Obtenez les sorties du modèle
        regression_output = model(images)

        # Calcul de la perte
        #print(type(regression_output))
        #print(regression_output)

        coordinates = torch.stack([coord.float() for coord in coordinates],
dim=1)
        #print(coordinates.shape)
        #print(coordinates)
        loss =  nn.MSELoss()(regression_output, coordinates)


        distance = torch.mean(torch.sqrt(torch.sum((regression_output -
coordinates)**2, dim=1)))
        train_distances.append(distance.item())

        # Rétropropagation et mise à jour des poids
        loss.backward()
        optimizer.step()
        train_losses.append(loss)

    print(f'Epoch {epoch+1}/{n}, Iteration {iteration}, Loss: {loss.item(), },
Distance : {distance.item()}')

    iteration += 1
# À la fin de votre boucle d'entraînement
checkpoint_path =
'/content/drive/MyDrive/Classroom/train_ckeck/model.pth'
torch.save({
    'epoch': epoch,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'loss': loss,
    'distance moyenne': distance,
    # Ajoutez d'autres informations que vous souhaitez sauvegarder
}, checkpoint_path)
plt.plot([loss.item() for loss in train_losses], label='Training Loss')
plt.plot([distance.item() for dis in train_distances], label='Training
Distance')
plt.xlabel('Iteration')
plt.ylabel('Metric Value')
plt.title('Training Loss and Distance Over Iterations')
plt.legend()
plt.show()
```

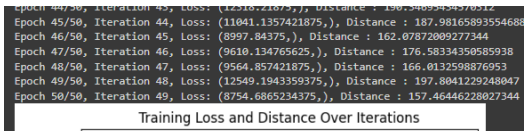
Les différents résultats

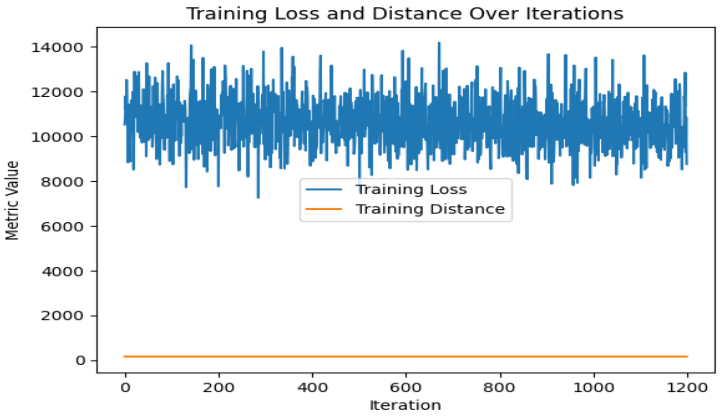
50 premières epoch de notre apprentissage en temps réel





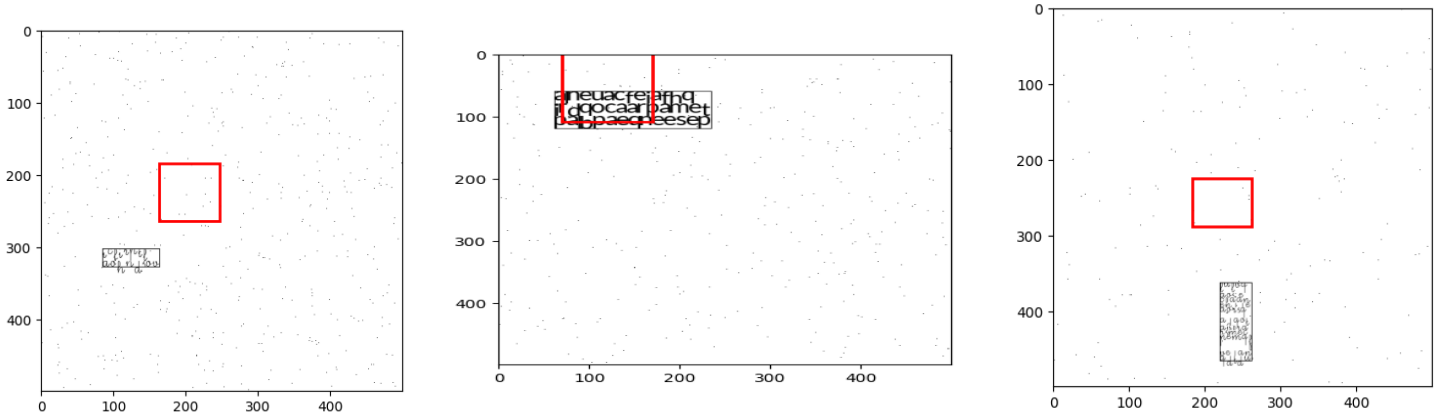
En route vers la 100 -ème epoch de notre apprentissage en temps réel





Malheureusement le model peine à apprendre plus et on a dû l'arrêter mais cela affecte les résultats attendus.

Place aux résultats de nos test



Ça se rapproche quand même de la zone surtout pour le deuxième teste mais cela reste moins précise.

Le cadre en noir est le résultat attendu et le carde en rouge est le résultat prédit.
Pour être plus précis dans les chiffres :
Loss : 8754.6865234375 (a interpreter selon le contexte)
Distance moyenne entre la valeur predit et la valeur reelle : 157.46446228027344

.

En conclusion, le projet « Détection de text » a représenté un défi stimulant qui a mis en lumière notre capacité à concevoir, développer et déployer une solution informatique robuste. À travers une analyse approfondie des besoins, une conception réfléchie basée sur cette architecture, et une implémentation méticuleuse avec les technologies basées sur l'intelligence artificielle, nous avons réussi à répondre aux objectifs initiaux. Les phases de test ont confirmé la fiabilité de notre programme, tout en soulignant des opportunités d'amélioration. Au-delà des aspects techniques, ce projet a renforcé notre collaboration en équipe et a été une source d'apprentissage continu. Nous sommes reconnaissants envers les enseignants pour leur soutien, et nous sommes confiants que les compétences acquises seront des atouts précieux dans nos futures entreprises