Deep Learning & MLOps

Lab 1 - Convolutional Neural Networks & Object Detectors

Setup & Tools

Se puede utilizar cualquier librería o tecnología para realizar los entrenamientos y análisis siempre y cuando se puedan lograr los objetivos del laboratorio.

Disclaimer: El código en este ejercicio no fue 100 % corroborado por lo que queda de la mano del desarrollador debugearlo y completar acorde sea el caso

Disclaimer2: Al realizar todo el ejercicio correctamente con conclusiones consistentes se obtiene 80 % del Lab. El 20 % restante es obtenido con un extra mile (e.g insights, comentarios, fun facts etc.) sobre los ejercicios.

Convolutional Neural Networks

En este primer ejercicio vamos a estar entrenando redes convolucionales para utilizarlas como clasificadores de imagenes. Para este primer ejercicio estaremos MNIST y CIFAR-10/100, ambos datasets son considerados "toy-problemz nos permitira iterar los entrenamientos rapidamente. En la vida real, los datasets (generalmente) utilizan imagenes de mayor resolución por lo que el entrenamiento tiende a ser mas lentos que en este ejercicio.

Data Exploration

- Descargar el dataset seleccionado de este link. El formato del dataset es imagenes en carpetas train y test, labels en dataframes *training_ labels.csv* y *test_ labels.csv*
- Familiarizarse con la data: contar cantidad de imágenes por clase, visualizar un poco las imágenes para corroborar congruencia con las labels. Realizar análisis extras sobre la data.
- Dividir training dataset en los subconjuntos training y validación
- Crear objetos datasets y verificar que que este retornando las imágenes y labels de forma esperada.

```
class CustomDataset(torch.data.Dataset):
    def __init__(self, folder_path, labels_path, labels_to_index, transforms=None):
        self.folder_path = folder_path
        self.labels_df = pd.read_csv(labels_path)
        unique_labels = pd.unique(self.labels['labels'])
        self.labels_to_index = labels_to_index
        self.transforms = transforms #Augmentation and ToTensor

    def __getitem__(self, index):
        '''Read index image + label and return it'''
        input_data = self.labels_df.loc[index, :]
        image = cv2.imread(os.path.join(self.folder_path, input_data['image_name']),
                           cv2.IMREAD_UNCHANGED) #Read image
        label = self.labels_to_index[input_data['label']] #Change label to number
        if self.transforms:
            image = self.transforms(image) #Run given transformations / augmentations
        return image, label

    def __len__(self):
        return len(self.labels) #Size of dataset
```

Model definition

- Crear modelo SimpleCNN el cual tiene tres capas convolucionales de 64 canales cada una y con kernels (3, 3, 5). Agregar padding para preservar el tamaño de la imagen a través del modelo. Max Pooling (kernel=2, stride=2) luego de la primera convolución y un par de capas lineales (densely connected) con 128 neuronas para clasificar las 10 clases. Todas las convoluciones llevan asociada la función de activación ReLu. Importar el modelo (e.g en un notebook), parsearle una imagen en blanco y corroborar que la salida es la esperada. Plotear como va cambiando la dimensionalidad de la imagen de entrada en cada uno de los puntos del modelo.

```python
from torch import nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self, nb_classes, img_size=(255, 255)):
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64,
                               kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=64,
                               kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=64,
                               kernel_size=5, padding=2)
        self.max_pool = nn.MaxPool2d((2, 2), stride=2)

        vector_size = img_size // 2 * img_size // 2 * 64
        self.fc = nn.Linear(vector_size, nb_classes)
        self.nb_classes = nb_classes

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.max_pool(x)
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))

        x = x.view(len(x), -1)
        return self.fc(x)
```

Training definition

Nota: Recordar guardar el modelo de mejor performance en validación durante el entrenamiento usando *torch.save* para posteriormente poder evaluar su performance en test

- Entrenar la CNN con el optimizador Stochastic Gradient Descent con momentum de 0.9 y learning rate de 0.01. A su vez, entrenar la misma red utilizando el optimizador *Adam*.
- Escribir script (guía en el training script) de entrenamiento. Agregar tqdm para visualizar progreso del entrenamiento. **OJO:** setear la random seed de torch y la de numpy para asegurar que múltiples iteraciones tienen el mismo modelo de initialización.
- Entrenar una red de capas MultiLayerPerceptrons (Linear) que tomen como input los flattened pixels de las imágenes (raw vector representation [normalizados]). El modelo debe incluir dropout (entre la capa linear y la activación) y debe utilizar ReLu activation.
- Entrenar dos modelos tradicionales (e.g SVM, Random Forest, XGBoost, KNN) que sirvan como baseline para clasificar los raw vectors, calcular métricas de validación y tunear hiperparámetros mas importantes.

model/metric	training accuracy	validation accuracy	test accuracy
SimpleCNN			
ResNet-50			
SimpleMLP			
SVM			
KNN			

Preguntas y comentarios a considerar:

- Plotear curvas de training loss y accuracy, validation loss y accuracy. Que se puede observar en las curvas? Generalizó el modelo? Overfiteo? Underfiteo? Si el modelo underfiteo incrementar el numero de epochs. Si overfiteo o estabilizo muy rápido, dividir el learning rate por 10 en la epoch donde estabiliza (utilizar algún tipo de learning rate scheduler). Comparar las distintas iteraciones de entrenamiento
- Que conclusiones se puede sacar sobre el entrenamiento? Como se compara el modelo que tiene learning rate scheduler con el modelo usando learning rate constante (en test set)?
- Escojer un modelo del Model Zoo de PyTorch. Entrenar modelo usando el mismo pipeline de entrenamiento que se ha estado trabajando. Nota: Es probable que se tenga que resizear la imagen dependniendo del modelo seleccionado
- Realizar tabla de comparación entre modelos baselines, CNN simple, CNN proveniente del model zoo y el MLP.
- Para el modelo de mejor performance en test, realizar un análisis de la distribución de los errores y determinar que clase es de mayor probabilidad de fallo.
- Para cada una de las clases, realizar curvas de precision y recall (PR curve) para todos los modelos y para todas las clases. Realizar un plot por clase.

Apéndice

training script

```
from torch import nn
from torch.data import DataLoader
import torch.optim as optim
from model import SimpleCNN
from dataset import CustomDataset, labels_to_index

training_dataset = CustomDataset(train_images_path, train_df_path,
                                 labels_to_index, transforms=transforms.ToTensor())
validation_dataset = CustomDataset(val_images_path, val_df_path,
                                   labels_to_index, transforms=transforms.ToTensor())

train_loader = DataLoader(training_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(validation_dataset, batch_size=1, shuffle=False)
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = SimpleCNN(len(labels_to_index), (image_height, image_width))
model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9)
metrics = {'train_loss' : [], 'train_acc' : [], 'val_loss' : [], 'val_acc' : []}
for epoch_index in range(nb_epochs):
    running_loss, running_corrects = 0.0, 0
    model.train()
    for batch_x, batch_y in train_loader:
        batch_x, batch_y = batch_x.to(device), batch_y.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        predictions = torch.max(outputs, 1)[1]
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * batch_x.size(0)
        running_corrects += torch.sum(predictions == batch_y.data)
    metrics['train_loss] += [running_loss / len(training_dataset)]
    metrics['train_acc'] += [running_corrects.double() / len(training_dataset)]

    running_loss, running_corrects = 0.0, 0
    model.eval()
    for batch_x, batch_y in val_loader:
        batch_x, batch_y = batch_x.to(device), batch_y.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        loss = criterion(outputs, labels)

        running_loss += loss.item() * batch_x.size(0)
        running_corrects += torch.sum(preds == batch_y.data)
    metrics['val_loss] += [running_loss / len(validation_dataset)]
    metrics['val_acc'] += [running_corrects.double() / len(training_dataset)]
```

Referencias

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[5] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.