1st Summer School Physical Engineering UNI 2022-Lima,Perú

Reporte Deep Learning

Mirian Andrea Geronimo Aparicio[1]

[1]Facultad de Ciencias, Universidad Nacional de Ingeniería
[1]mgeronimoa@uni.pe

March 3, 2022

Resumen

En este trabajo mostramos un algoritmo de una red neuronal para la Clasificación de imágenes con un Dataset del juego Piedra-Papel-Tijera y así aprender sobre los fundamentos básicos de Redes Neuronales. Hacemos uso del framework Pytorch por su facilidad de usar respecto a otros frameworks. Analizamos las diferencias con algunos cambios como Data Augmentation, Shuffle, Learning Rate con el objetivo de ver cuál nos proporciona una mejor precisión.

Palabras Clave
Pytorch, Red Neuronal, Train Accuracy.

1 Metodología

El Deep Learning, es una rama del Machine Learning que introduce datos de entrada a través de una arquitectura de red neuronal. Esta red neuronal se puede entender como una función que tiene parámetros que son entrenados a través de un número de capas ocultas que procesan los datos, permitiendo así a la máquina "profundizar" en su aprendizaje haciendo conexiones y ponderando los inputs para obtener cada vez mejores resultados.

Pytorch es uno de los frameworks más utilizados en las redes neuronales. Tiene como objetivo la implementación y entrenamiento de modelos de Deep Learning de una manera sencilla y eficiente por su enfoque a la programación de tensores, diferenciación automática y su capacidad para ejecutarse en GPU (lo que acelera el entrenamiento de los modelos).

Veamos el algoritmo. Importamos las librerías necesarias para la clasificación de imágenes utilizando un dataset del juego Piedra Papel o Tijera.

```
import torch
import torch.nn as nn
import torchvision
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np
import pandas as pd
from collections import defaultdict
import os
import PIL
from scipy.io import wavfile
import torchaudio
from IPython.display import Audio, display
```

Definimos nuestro device:

```
dev = torch.device("cuda:0" if
torch.cuda.is_available() else "cpu")
```

Cargamos la carpeta .json luego de descargarlo desde nuestra cuenta en Kaggle y lo movemos a otro directorio.

```
!mkdir .kaggle
!mv kaggle.json .kaggle/
!mv .kaggle ~/
```

Descargamos y descomprimimos el dataset que contiene diferentes imágenes del juego Piedra Papel o tijera[1]:

```
!kaggle datasets download drgfreeman/rockpaperscissors
!unzip rockpaperscissors.zip
```

Definimos las funciones para entrenar el modelo.

```
  def evaluate(model, loader, crit):
  model.eval()
  total = 0
  corrects = 0
  avg_loss = 0
  for x, y in loader:
    x = x.to(dev)
    y = y.to(dev)
    o = model(x)
    loss = crit(o,y)
    avg_loss += loss.item()
    corrects += torch.sum(torch.argmax(o,axis=1) == y).item()
    total += len(y)
  acc = 100* corrects / total
  avg_loss /= len(loader)
  return avg_loss, acc

  def train_one_epoch(model, train_loader, crit, optim):
  model.train()
  total = 0
  corrects = 0
  avg_loss = 0
  for x, y in train_loader:
    optim.zero_grad()
    x = x.to(dev)
    y = y.to(dev)
    o = model(x)
    loss = crit(o,y)
    avg_loss += loss.item()
    loss.backward()
    optim.step()
    corrects += torch.sum(torch.argmax(o,axis=1) == y).item()
    total += len(y)
  acc = 100 * corrects / total
  avg_loss /= len(train_loader)
  return avg_loss, acc

def train(model, train_loader, test_loader, crit, optim, epochs = 20):
for epoch in range(epochs):
    train_loss, train_acc = train_one_epoch(model, train_loader,crit, optim)
    test_loss, test_acc = evaluate(model, test_loader, crit)
    print(f"epoch: {epoch}, train loss: {train_loss}, train acc: {train_acc}%,


    test loss: {test_loss}, test acc: {test_acc}%")
```

Aquí usamos dos tipos adicionales de data augmentation, el primero torchvision.transforms.RandomRotation(15) y luego torchvision.transforms.RandomHorizontalFlip() .

```
  from torchvision.transforms import transforms
  img_transform = torchvision.transforms.Compose([
  torchvision.transforms.RandomRotation(15),
  torchvision.transforms.RandomHorizontalFlip(),
  torchvision.transforms.Resize((224,224)),
  torchvision.transforms.ToTensor()
])
```

Creamos la carpeta data para mover ahí las carpetas paper, rock y scissors:

```
!mkdir data
!mv paper data
!mv rock data
!mv scissors data
```

Separamos el dataset para entrenar y para el test.

```
dataset = torchvision.datasets.ImageFolder("data",transform=img_transform)
N = len(dataset)
train_size = int(N * 0.8)
test_size = N - train_size
train_ds, test_ds = torch.utils.data.random_split(dataset, [train_size,test_size])
```

Cargamos los datos por lotes

```
train_dl = torch.utils.data.DataLoader(train_ds,


batch_size=128,shuffle=True)
test_dl = torch.utils.data.DataLoader(test_ds,


batch_size=128,shuffle=True)
```

Creamos nuestro modelo utilizando varias capas y lo enviamos al device:

```
    model = nn.Sequential(
    nn.Conv2d(3,32,kernel_size=3),
    nn.MaxPool2d(2),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),
    nn.Conv2d(32,128,kernel_size=3),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.5),
    nn.Flatten(),
    nn.Linear(86528,256),
    nn.ReLU(inplace=True),
    nn.Linear(256,10)
).to(dev)
```

Definimos el criterio y usamos el optimizador SGD, y entrenamos:

```
crit = nn.CrossEntropyLoss()
optim = torch.optim.SGD(model.parameters(),lr=0.1)
train(model,train_dl, test_dl, crit, optim, epochs=10)
```

Muestra el resultado de la imagen de índice idx=10:

```
model.eval()
# idx = 10
idx = 720
x, y = test_ds[idx]
x_numpy = x.numpy().transpose(1,2,0)
N, H, W = x.shape
x = x.reshape(1,N,H,W)
pred = torch.argmax(model(x.to(dev)).cpu()).item()
print(pred)
plt.imshow(x_numpy)
```

2 Resultados

```
# Esto es solo para poder ver un ejemplo de las imagenes que se estan usando
plt.imshow(train_ds[10][0].numpy().transpose(1,2,0))
```

```
<matplotlib.image.AxesImage at 0x7f5c112fd610>
```

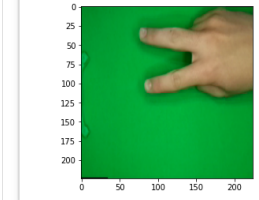

Figura 1: Un ejemplo de las imágenes que se están usando

```
epoch: 0, train loss: 17.359481147357396, train acc: 32.514285714285714%, test loss: 1.0966284275054932, test acc: 31.27853881278539%
epoch: 1, train loss: 1.1343910268374853, train acc: 38.0%, test loss: 1.0981332659721375, test acc: 35.616438356164384%
epoch: 2, train loss: 1.0988031455448695, train acc: 34.057142857142856%, test loss: 1.0978454649448395, test acc: 35.616438356164384%
epoch: 3, train loss: 1.0985160384859358, train acc: 33.94285714285714%, test loss: 1.0961987674236298, test acc: 34.018264840182646%
epoch: 4, train loss: 1.0906927670751299, train acc: 35.94285714285714%, test loss: 1.0740599632263184, test acc: 52.968036529680374%
epoch: 5, train loss: 1.736742981842586, train acc: 39.657142857142854%, test loss: 1.0974907875061035, test acc: 35.616438356164384%
epoch: 6, train loss: 1.0992103729929243, train acc: 33.94285714285714%, test loss: 1.0973010659217834, test acc: 35.616438356164384%
epoch: 7, train loss: 1.0990496107510157, train acc: 33.94285714285714%, test loss: 1.0976466536521912, test acc: 35.616438356164384%
epoch: 8, train loss: 1.0988129973411156, train acc: 33.94285714285714%, test loss: 1.097700685262268, test acc: 35.616438356164384%
epoch: 9, train loss: 1.0988528217588152, train acc: 33.94285714285714%, test loss: 1.0975313484668732, test acc: 35.616438356164384%
```

Figura 2: Resultado del entrenamiento con Shuffle=True y Learning Rate=0.1.

```
epoch: 0, train loss: 13.085404481206622, train acc: 33.771428571428574%, test loss: 1.0985330641269684, test acc: 36.30136986301371%
epoch: 1, train loss: 1.0977198311260767, train acc: 35.25714285714286%, test loss: 1.139359474182129, test acc: 34.018264840182646%
epoch: 2, train loss: 1.0979803119386946, train acc: 32.685714285714285%, test loss: 1.0677132606506348, test acc: 35.616438356164384%
epoch: 3, train loss: 1.1186736396380834, train acc: 34.4%, test loss: 1.0992609560489655, test acc: 32.876712328767134%
epoch: 4, train loss: 1.0971350244113378, train acc: 35.94285714285714%, test loss: 1.1009989380836487, test acc: 48.173515981735164%
epoch: 5, train loss: 1.093778329236167, train acc: 37.657142857142854%, test loss: 1.0973095297813416, test acc: 32.876712328767134%
epoch: 6, train loss: 1.1295624034745353, train acc: 35.02857142857143%, test loss: 1.098881483078003, test acc: 34.018264840182646%
epoch: 7, train loss: 1.0993557146617345, train acc: 32.971428571428574%, test loss: 1.0987411737442017, test acc: 34.018264840182646%
epoch: 8, train loss: 1.098678035395486, train acc: 33.42857142857143%, test loss: 1.0988595787371048, test acc: 34.018264840182646%
epoch: 9, train loss: 1.0986521925245012, train acc: 33.371428571428574%, test loss: 1.0988840162754059, test acc: 32.876712328767134%
```

Figura 3: Resultado del entrenamiento con Shuffle=False y Learning Rate=0.1.

```
epoch: 0, train loss: 1.2173091726643699, train acc: 40.4%, test loss: 1.0845973491668701, test acc: 33.333333333333336%
epoch: 1, train loss: 0.958542104278292, train acc: 54.628571428571426%, test loss: 1.0221275091171265, test acc: 51.36986301369863%
epoch: 2, train loss: 0.8856530572686877, train acc: 59.314285714285715%, test loss: 0.8461660593748093, test acc: 76.94063926940639%
epoch: 3, train loss: 0.6769964907850538, train acc: 73.48571428571428%, test loss: 0.5309355407953262, test acc: 86.07305936073061%
epoch: 4, train loss: 0.5979128841842923, train acc: 75.54285714285714%, test loss: 0.4839933291077614, test acc: 88.12785388127854%
epoch: 5, train loss: 0.5147144475153514, train acc: 79.77142857142857%, test loss: 0.5233568549156189, test acc: 75.34246575342466%
epoch: 6, train loss: 0.3731267622538975, train acc: 86.05714285714286%, test loss: 0.2919497117400169, test acc: 88.81278538812785%
epoch: 7, train loss: 0.3089507243462971, train acc: 88.97142857142858%, test loss: 0.22467298433184624, test acc: 93.37899543378995%
epoch: 8, train loss: 0.2647064306962916, train acc: 90.85714285714286%, test loss: 0.18471179902553558, test acc: 93.83561643835617%
epoch: 9, train loss: 0.24830225429364614, train acc: 91.37142857142857%, test loss: 0.5612506195902824, test acc: 73.97260273972603%
```

Figura 4: Resultado del entrenamiento con Shuffle=True y Learning Rate=0.01

```
model.eval()
# idx = 10
idx = 100
x, y = test_ds[idx]
x_numpy = x.numpy().transpose(1,2,0)
N, H, W = x.shape
x = x.reshape(1,N,H,W)
pred = torch.argmax(model(x.to(dev)).cpu()).item()
print(pred)
plt.imshow(x_numpy)
```

```
2
<matplotlib.image.AxesImage at 0x7fcc394d2ad0>
```

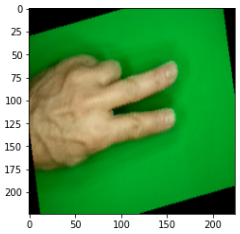

Figura 5: Resultado de predicción para la imagen idx=100 con Shuffle=True y Learning rate=0.01

3 Discusión de resultados

- Hicimos uso adicional de dos tipos de data Augmentation, Random Rotation y Random Horizontal, ambos permiten crear nuevas versiones de las imágenes originales del training set y así ayuda a nuestro modelo a aprender cómo se ve una imagen en general. Random Rotation realiza una rotación aleatoria de una imagen original en 15 grados(en nuestro modelo) con alguna probabilidad dada y Random Horizontal voltea horizontalmente una imagen original con una probabilidad dada.

- Cuando cambiamos el shuffle=True por shuffle=False se observa una ligera disminución en el accuracy, esto debido a que cuando elegimos shuffle=True los datos se reorganizan(mezclan) en cada época haciendo que los lotes en cada época no se parezcan produciendo una mejor precisión, de lo contrario (shuffle=False) puede que por ejemplo, cada clase vaya a un lote diferente, y en cada época, un el lote contenga la misma categoría lo que produciría una mala precisión.

- En nuestro entrenamiento usando el shuffle=True, obtenemos una mayor precisión(mejor accuracy) con el Learning Rate=0.01(Figura 4) que con el Learning Rate=0.1(figura 2), podría esperarse que suceda al revés pero también depende del Dataset que se utilice y la arquitectura del modelo. Así como también hay una mayor disminución del loss con rl=0.01.

Conclusiones

- Se obtiene una mejor precisión si escogemos shuffle=True.

- Los tipos de Data Augmentation ayuda a generalizar más fácilmente las imágenes u objetos y a que la red neuronal tenga un mejor desempeño con respecto a la perturbación indicada.

- Para nuestro Dataset y modelo, es más conveniente usar Learning Rate=0.01 que Learning rate=0.1.

Agradecimientos

Deseo expresar mis agradecimientos a AEPIF por organizar este 1st Summer School Physical Engineering 2022 y así contribuir al aprendizaje de muchos interesados en el campo. También agradecer al instructor Elvin Mark Muñoz Vega por su buena enseñanza y dedicación.

Referencias

[1] Rock-Paper-Scissors Images. Kaggle. https://www.kaggle.com/drgfreeman/rockpaperscissors

[2] Pytorch Documentation. Pytorch. https://pytorch.org/docs/stable/index.html