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

Deep Learning Report

Marlon David Bustos Paucar[1]

[1]Facultad de Ciencias, Universidad Nacional de Ingenieria
[1]marlon.bustos.p@uni.pe

February 28, 2022

Resumen

Actualmente, las redes neuronales convolucionales son usadas en la mayoría de trabajos que requieran la clasificacione de imágenes. En este protyecto, se utilizó redes neuronales convolucionales profundas (CNN) para aprender características, entrenar y realizar una clasificación de las imágenes de perros y gatos. El Dataset de este proyecto fue extraído de Kaggle. Este conjunto consta de 10000 imágenes de perros y gatos en formato RGB. Después de entrenar el modelo y ponerlo a prueba, el modelo es capaz de clasificar las imágenes con una precisión del 83.9 %.

Palabras Clave
Convolutional Neural Network, Deep Learning, Image Classification

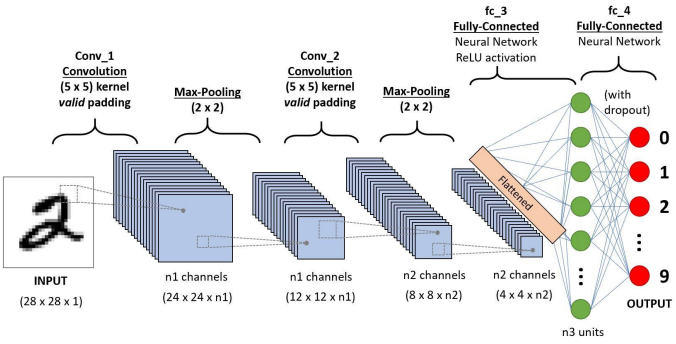

Figura 1: Convolutional Neural Network

1 Metodología

1.1 Redes Neuronales Convolucionales

El nombre "red neuronal convolucional" indica que la red emplea una operación matemática (convolución). La convolución es un tipo especializado de operación lineal. Las redes convolucionales son simplemente redes neuronales que utilizan la convolución en lugar de la multiplicación general de matrices en al menos una de sus capas. Una red neuronal convolucional consiste en una entrada y una capa de salida, así como múltiples capas ocultas. Las capas ocultas de una CNN típicamente consisten en una serie de capas convolucionales que conviven con una multiplicación u otro producto escalar. La función de activación suele ser una capa RELU y, posteriormente, le siguen convoluciones adicionales, como pooling layers, fully connected layers y normalization layers, denominadas capas ocultas porque sus entradas y salidas están enmascaradas por la función de activación y la convolución final. La convolución final, a su vez, a menudo implica backpropagation para ponderar con mayor precisión el producto final.

1.2 Dataset y Data Augmentation

El Dataset se divide en carpetas para cada clase y a la vez se divide en un conjunto para el entrnamiento y otro para realizar las pruebas (test). El conjunto de entrenamiento y el conjunto de prueba se componen de 2 carpetas, una para imágenes de gatos y otra para imágenes de perros. Hay 4000 imágenes de cada gato y perro para entrenamiento y 1000 imágenes de cada gato y perro para prueba. Las imágenes tienen diferentes formas y tamaños, pero para entrenar una CNN, las imágenes deben ser del mismo tamaño.

El redimensionamiento de las imagenes se lleva a cabo utilizando el módulo torchvision.transforms de Pytorch. Usándolo, las imágenes se redimensionan a 128 x 128. Además, para el entrenamiento de una red neuronal convolucional se necesita un gran conjunto de imágenes. Por lo tanto, el redimensionamiento se aplica a todo el dataset de imágenes.

También su usaron otras transformaciones co-

mo CenterCrop() que recorta la imagen dada en el centro, RandomHorizontalFlip() que voltea horizontalmente la imagen dada al azar con una probabilidad dada y RandomRotation() que rota la imagen dado un ángulo.

2 Resultados

- Importación de librerías de Python:

```
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
```

- Descarga del dataset de perros y gatos. (Se usó el dataset de perros y gatos)

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

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

!kaggle datasets download chetankv
/dogs-cats-images
!unzip dogs-cats-images.zip
```

- Entrenamiento de la red neuronal (no se modificó nada)

```
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}%")
```

- Data augmentation: Se usaron las transformaciones CenterCrop() que recorta la imagen dada en el centro, RandomHorizontalFlip() que voltea horizontalmente la imagen dada al azar con una probabilidad dada y RandomRotation() que rota la imagen dado un ángulo.

```
img_transform = torchvision.transforms.Compose([
  torchvision.transforms.Resize((128,128)),
  torchvision.transforms.CenterCrop(128),
  torchvision.transforms.RandomHorizontalFlip(p=0.5),
  torchvision.transforms.RandomRotation(degrees = 10),
  torchvision.transforms.ToTensor(),
])
```

- Creación de los datasets de entrenamiento y test.

```
train_ds = torchvision.datasets.ImageFolder("/content/dataset
/training_set",transform=img_transform)
test_ds = torchvision.datasets.ImageFolder("/content/dataset
/test_set",transform=img_transform)
```

- Creación de los dataloaders con un batch size = 128 y shuffle = True.

  Cuándo se usó "Shuffle = False"se obtuvieron malos resultados con respecto a la precisión con la que el modelo clasificaba a las imágenes. (test-accuracy = 62 %)

  El parámetro Shuffle es necesario para evitar la asignación no aleatoria al conjunto de entrenamiento y prueba. Con shuffle=True se divide los datos al azar. Por ejemplo, supongamos que tenemos datos de clasificación binaria y están ordenados por etiquetas. Si se dividen en proporciones 80:20 para entrenar y testear, los datos de prueba contendrán solo las etiquetas de una clase. Shuffle = True evita esto.

```
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)
```

- Creación del modelo: La primera capa del modelo clasificador secuencial es una capa Conv2d, la siguiente capa es una capa de batch normalization seguida de una capa de activación ReLU y una por último una capa de max pooling, se usó, de manera intercalada, este conjunto de capas por 4 veces cambiando los parámetros como era requerido.

```
model = nn.Sequential(

    nn.Conv2d(3,16,7,bias = False),
    nn.BatchNorm2d(16),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),
    nn.Conv2d(16,32,3,bias = False),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),
    nn.Conv2d(32,32,5,bias = False),
    nn.BatchNorm2d(32),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),
    nn.Conv2d(32,64,3,bias = False),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2),


    nn.Flatten(),
    nn.Linear(1600,512),
    nn.ReLU(inplace=True),
    torch.nn.Sigmoid(),
    nn.Linear(512,256),
    nn.ReLU(inplace=True),
    nn.Linear(256,128),
    nn.ReLU(inplace=True),
    nn.Linear(128,2),

).to(dev)
```

- Entrenamiento de la red neuronal. Cuando se disminuyó el learning rate el train-accuraccy y test-accuraccy disminuían ligeramente con respecto a los valores que tenían con el learning rate de 0.1 y cuando se aumento el valor del learning rate se obtuvieron resultados ligeramente mejores que cuando se usaba el learning rate preestablecido de 0.1

```
crit = nn.CrossEntropyLoss()
optim = torch.optim.SGD(model.parameters(),lr=0.1)
train(model,train_dl, test_dl, crit, optim,
epochs=20)
```

- Resultados:

```
epoch: 0, train loss: 0.332712839283640 4, train acc: 85.35%, test loss: 1.07756033167243, test acc: 59.65%
epoch: 1, train loss: 0.314913268363665, train acc: 86.9875%, test loss: 0.649353891611 0992, test acc: 72.65%
epoch: 2, train loss: 0.305397758881251 04, train acc: 86.725%, test loss: 0.866319727152586, test acc: 64.35%
epoch: 3, train loss: 0.303685277227371 43, train acc: 86.7125%, test loss: 0.495374269783496 86, test acc: 78.35%
epoch: 4, train loss: 0.303080716339943, train acc: 87.375%, test loss: 0.595583930611610 4, test acc: 70.95%
epoch: 5, train loss: 0.281367355159350 8, train acc: 88.0375%, test loss: 0.991702280938625 3, test acc: 63.7%
epoch: 6, train loss: 0.285409425459210 83, train acc: 87.8%, test loss: 0.448331732302904 13, test acc: 80.05%
epoch: 7, train loss: 0.283472764350119 1, train acc: 88.0625%, test loss: 0.572965782135725, test acc: 74.95%
epoch: 8, train loss: 0.261393912964397, train acc: 88.8625%, test loss: 0.353472290560803 14, test acc: 85.25%
epoch: 9, train loss: 0.260613546480992 63, train acc: 88.7125%, test loss: 0.528958051359177, test acc: 76.8%
epoch: 10, train loss: 0.255615391191982 07, train acc: 89.0375%, test loss: 0.390289412811398 5, test acc: 82.45%
epoch: 11, train loss: 0.259424171987034 05, train acc: 88.8625%, test loss: 0.411552306264638 9, test acc: 80.75%
epoch: 12, train loss: 0.257264561832897 25, train acc: 89.4375%, test loss: 0.349359342828393, test acc: 84.8%
epoch: 13, train loss: 0.256640945226926 63, train acc: 88.95%, test loss: 0.413823195731313, test acc: 84.8%
epoch: 14, train loss: 0.237991400890880 17, train acc: 90.2625%, test loss: 0.859931517392396 9, test acc: 64.65%
epoch: 15, train loss: 0.230560150411393 9, train acc: 90.325%, test loss: 0.533032495528459 5, test acc: 78.05%
epoch: 16, train loss: 0.235715878624764 7, train acc: 90.275%, test loss: 0.434814652428039 7, test acc: 81.9%
epoch: 17, train loss: 0.220586125221517 35, train acc: 91.125%, test loss: 0.495030181482434 3, test acc: 79.65%
epoch: 18, train loss: 0.220738167465011, train acc: 90.675%, test loss: 0.328202974051237 1, test acc: 86.4%
epoch: 19, train loss: 0.216767374012205 33, train acc: 90.7875%, test loss: 0.310817966237664 2, test acc: 86.15%
```

Figura 2: Tabla de resultados por épocas.

3 Discusión de resultados

- Cuando el learning rate es demasiado grande, La gradiente descendente puede aumentar en lugar de disminuir el training-loss. Cuando la tasa de aprendizaje es demasiado pequeña, el entrenamiento no solo es más lento, sino que puede atascarse permanentemente con un alto training-loss. En este proyecto se escogió un learning rate de 0.1.
- El parámetro Shuffle es necesario para evitar la asignación no aleatoria al conjunto deentrenamiento y prueba.
- Se obtuvo un test-accuraccy de 86.15 %
- La precisión del entrenamiento y test aumentan a medida que se aumenta el número de épocas.
- Se podrían obtener mejores resultados con un dataset más grande y más capas en el modelo.

Conclusiones

En este proyecto se trabajó con una red neuronal convolucional profunda para la clasificación de imágenes de perros y gatos. A pesar de usar solo un subconjunto de las imágenes, se obtuvo una precisión del 83.4 %

Agradecimientos

Mi más sincero agradecimiento al profesor Elvin Mark Muños Vega por brindarnos sus conocimientos y guiarnos en este proyecto. También agradezco al equipo de Aepif por contactar a los profesores, organizar y hacer posible esta escuela de verano que ha sido una experiencia muy valiosa en mi formación profesional.

Referencias

[1] PyTorch. (2019). Documentación de PyTorch. 27/02/2022, de PyTorch Sitio web: https://pytorch.org/docs/stable/index.html

[2] chetanimravan. (2018). Dogs Cats Images. 27/02/2022, de Kaggle Sitio web: https://www.kaggle.com/chetankv/dogs-cats-images