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

Reporte Deep Learning

Michael Jordy Rojas Ruiz[1]

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

25 de febrero de 2022

Resumen

Este proyecto está enfocado en el uso de redes neuronales, mediante el framework PyTorch, para la clasificación de imágenes. Se utilizan dos optimizadores diferentes (Stochastic Gradient Descent y Adam) con la finalidad de comparar el porcentaje de precisión y la rapidez entre ambos. Además, se analizan los resultados obtenidos por la manera en la que se ingresan las imágenes al modelo para ser entrenadas y testeadas, donde se corrobora que es más beneficioso entrenar el modelo haciéndolas ingresar de manera aleatoria.

Palabras Clave
Visión computacional, redes neuronales, clasificación de imágenes.

1 Metodología

En el desarrollo de este proyecto se desarrolló como tema introductorio el uso de las librerías de PyTorch con la finalidad de aprender los fundamentos básicos de las Redes Neuronales, las que serán optimizadas con *Stochastic Gradient Descent* (SGD) y posteriormente con *Adam*.

Al modelo creado se le añade la función de activación ReLU, así como también un filtro de convolución con *padding* y *stride* por defecto, es decir padding = 0 y stride = 1. Así como también se le agrega el *Max Pooling layer* y el *BatchNorm layer*.

Para el entrenamiento del modelo se utiliza un *dataset*, obtenido de la página web de kaggle, llamado dogs and cats dataset. Dentro de esta carpeta se encuentran dos carpetas, una será utilizada para el entrenamiento del modelo y la otra para ponerla a prueba.

2 Resultados

Como se mencionó anteriormente se utilizan las librerías de PyTorch en Google Colab:

```
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
```

Se selecciona el *device* a utilizar:

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

Finalmente para obtener el *dataset* de la página de kaggle, se tiene que descargar el archivo kaggle.json y subirlo al notebook:

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

Se escoge el *dataset*, en este caso, la carpeta dogs and cats dataset:

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

Las funciones utilizadas para entrenar el modelo son las mismas que se enseñaron durante las clases (ver el Apéndice A). Ahora, antes de ingresar las imágenes para el entrenamiento y ponerlo a prueba se hacen unas transformaciones a las imágenes:

```
img_transform = torchvision.transforms.Compose([
  torchvision.transforms.RandomRotation(20),
  torchvision.transforms.RandomHorizontalFlip(),
  torchvision.transforms.Resize((224,224)),
  torchvision.transforms.ToTensor()
])
```

Aquí se hacen algunos cambios, con el comando *torchvision.transforms.RandomRotation(20)* se hace girar la imagen en un ángulo de 20 grados con cierta probabilidad, luego con *torchvision.transforms.RandomHorizontalFlip()* se gira

horizontalmente la imagen dada con cierta probabilidad (p=0.5). Se hace un cambio en el tamaño de las imágenes a 224×224 con el comando *torchvision.transforms.Resize((224,224))* y por último se convierte a tensor la imagen mediante *torchvision.transforms.ToTensor()*.

Se coloca el *path* de las carpetas que se utilizarán para el entrenamiento y para la prueba:

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

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

Para el entrenamiento del modelo se modifica el lote o paquete *(batch size)* a 200 imágenes con un *shuffle = True*, posteriormente se hará que *shuffle = False*.

```
train_dl = torch.utils.data.DataLoader(train_ds,
batch_size=200,shuffle=True)
test_dl = torch.utils.data.DataLoader(test_ds,
batch_size=200,shuffle=True)
```

Ahora es cuando se define el modelo:

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

  nn.Conv2d(64,128,3,bias=False),
  nn.BatchNorm2d(128),
  nn.ReLU(inplace=True),
  nn.MaxPool2d(2),

  nn.Flatten(),
  nn.Linear(128,2)
).to(dev)
```

Se hace uso de los siguientes layers:

- Conv2d, la cual se encarga de detectar figuras o patrones dentro de la imagen.
- BatchNorm2d, su función es evitar que los números "exploten" al multiplicar o sumar los números.
- ReLU, es una función de activación la cual transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran. Tiene buen comportamiento con las imágenes y buen desempeño en redes convolucionales.
- MaxPool2d, escoge el máximo valor de cada submatriz.
- Flatten, hace un reshape del penúltimo tensor obtenido.
- Linear, finalmente se hace una transformación lineal indicando los valores entrantes y salientes. Como en nuestro caso tenemos solo dos clases (perros y gatos), se coloca el número 2.

2.1 Optimizer Stochastic Gradient Descent

Se utiliza el optimizador SGD para el entrenamiento del modelo con un *learning rate* igual 0,1 para 10 épocas:

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

Luego, se pone a prueba el modelo. Se ha hecho una pequeña modificación; dado que la variable "pred" tiene como valores 0 o 1, donde el valor 0 indica que la figura es un gato y el valor 1 indica que la figura es un perro, entonces se hace que imprima el nombre de la clase de animal a la que pertenece la imagen:

```
model.eval()
idx = 1
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()
if pred==1:
  print('Es un perro')
if pred==0:
  print('Es un gato')
plt.imshow(x_numpy)
```

2.1.1 Shuffle = True

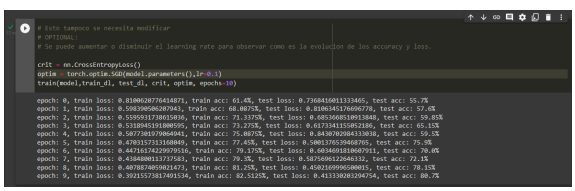

Figura 1: Resultados por épocas en el entrenamiento del modelo. Con un *test accuracy* de 80,7 % y un *test loss* de 0,4133.

```
model.eval()
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()
if pred==1:
  print('Es un perro')
if pred==0:
  print('Es un gato')
plt.imshow(x_numpy)
```

```
Es un gato
<matplotlib.image.AxesImage at 0x7f8963b0bd50>
```

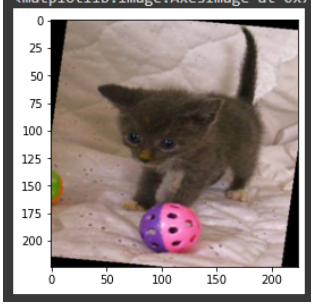

Figura 2: Prueba del modelo con la imagen de un gato.

```
model.eval()
idx = 54
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()
if pred==1:
  print('Es un perro')
if pred==0:
  print('Es un gato')
plt.imshow(x_numpy)
```

```
Es un perro
<matplotlib.image.AxesImage at 0x7f896320ba10>
```

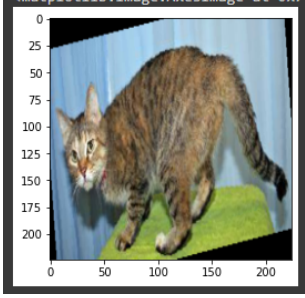

Figura 4: Prueba del modelo con la imagen de un gato.

2.1.2 Shuffle = False

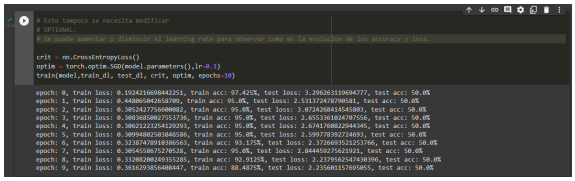

Figura 5: Resultados por épocas en el entrenamiento del modelo. Con un *test accuracy* de 50,0 % y un *test loss* de 2,2356.

```
model.eval()
idx = 1000
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()
if pred==1:
  print('Es un perro')
if pred==0:
  print('Es un gato')
plt.imshow(x_numpy)
```

```
Es un perro
<matplotlib.image.AxesImage at 0x7f89638c2910>
```

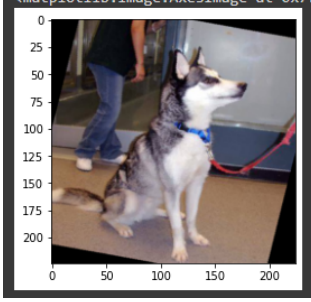

Figura 3: Prueba del modelo con la imagen de un perro.

```
model.eval()
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()
if pred==1:
  print('Es un perro')
if pred==0:
  print('Es un gato')
plt.imshow(x_numpy)
```

```
Es un perro
<matplotlib.image.AxesImage at 0x7f8962beaa10>
```

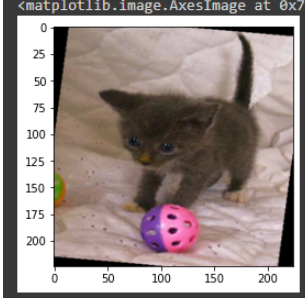

Figura 6: Prueba del modelo con la imagen de un gato.

Figura 7: Prueba del modelo con la imagen de un perro.

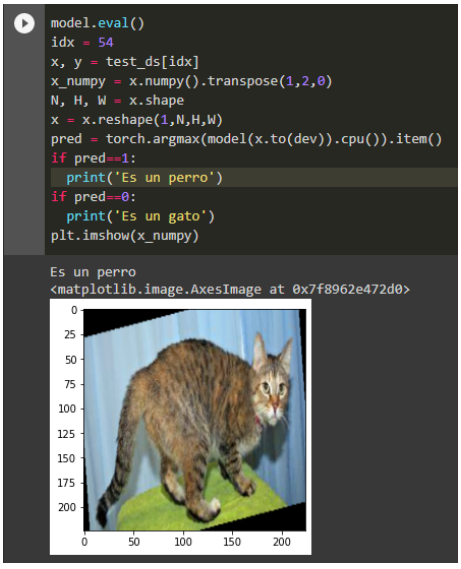

Figura 8: Prueba del modelo con la imagen de un gato.

2.2 Optimizer Adam

Ahora se utiliza el optimizador Adam para el entrenamiento del modelo con un *learning rate* igual 0,01 para 10 épocas:

```
crit = nn.CrossEntropyLoss()
optim = torch.optim.Adam(model.parameters(),lr=0.01)
train(model,train_dl, test_dl, crit, optim, epochs=10)
```

Luego, se pone a prueba el modelo. Se ha hecho una pequeña modificación; dado que la variable "pred" tiene como valores 0 o 1, donde el valor 0 indica que la figura es un gato y el valor 1 indica que la figura es un perro, entonces se hace que imprima el nombre de la clase de animal a la que pertenece la imagen:

```
model.eval()
idx = 1
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()
if pred==1:
  print('Es un perro')
if pred==0:
  print('Es un gato')
plt.imshow(x_numpy)
```

2.2.1 Shuffle = True

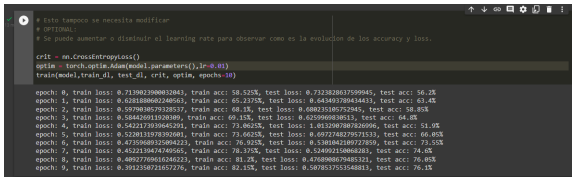

Figura 9: Resultados por épocas en el entrenamiento del modelo. Con un *test accuracy* de 76,1 % y un *test loss* de 0,5079.

Figura 10: Prueba del modelo con la imagen de un gato.

Figura 11: Prueba del modelo con la imagen de un perro.

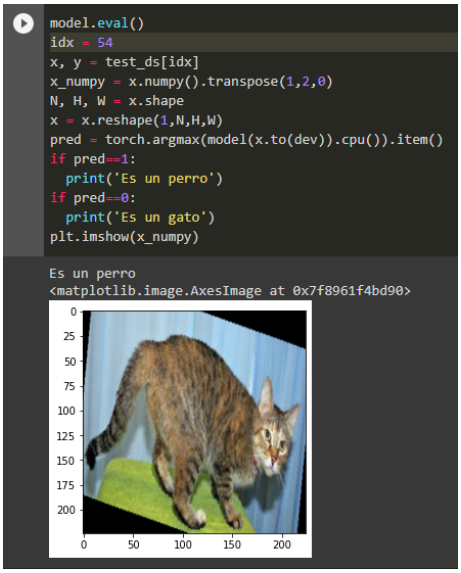

Figura 12: Prueba del modelo con la imagen de un gato.

2.2.2 Shuffle = False

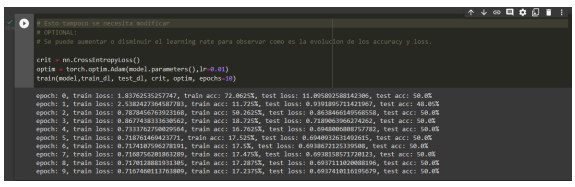

Figura 13: Resultados por épocas en el entrenamiento del modelo. Con un *test accuracy* de 50,0 % y un *test loss* de 0,6937.

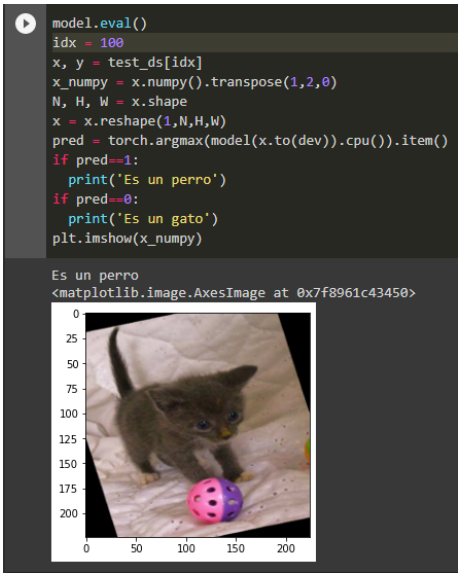

Figura 14: Prueba del modelo con la imagen de un gato.

Figura 15: Prueba del modelo con la imagen de un perro.

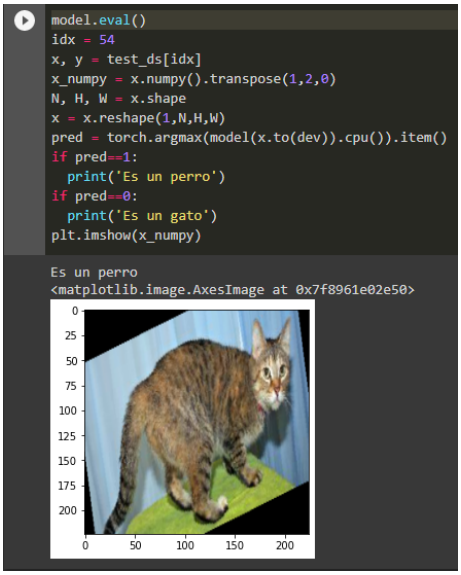

Figura 16: Prueba del modelo con la imagen de un gato.

3 Discusión de resultados

La primera gran diferencia de los resultados en el entrenamiento para cuando *suffle* = *True* y cuando *suffle* = *False* es el porcentaje alcanzado en el *test accuracy*; mientras que en el primer caso el porcentaje llega hasta un 81 % aproximadamente (optimizador SGD) o 76 % aproximadamente (optimizador Adam), en el segundo caso el porcentaje llega hasta 50 % (en ambos optimizadores). Esto posiblemente sea porque cuando se usa *suffle* = *False* las imágenes del lote que ingresa al modelo durante el entrenamiento y el testeo lo hacen en orden, es decir, ingresan primero las imágenes de la clase gato hasta culminar el proceso con las imágenes de la clase perro. Lo cual es contraproducente para este modelo, debido a que se entrena mejor para las imágenes de la clase perro, "olvidando" el entrenamiento que tuvo para las imágenes de la clase gato. Se puede corroborar observando las Figuras 6, 7, 6 (para el optimizador SGD) y las Figuras 14, 15, 16 (para el optimizador Adam).

En cuanto a rapidez, usando el optimizador Adam, el modelo termina ligeramente más rápido en comparación al optimizador SGD.

Conclusiones

Se ha podido ver que utilizar Google Colab es muy beneficioso para este tipo de aprendizaje, no es necesario tener una buena computadora, todo está en el notebook.

Además, se ha comprendido la utilidad de las redes neuronales en el campo de la visión computacional. Haciendo posible la obtención de resultados favorables al poner a prueba el modelo establecido, más aún cuando se ingresan de manera aleatoria las imágenes en el entrenamiento del modelo.

Agradecimientos

Muchas gracias al grupo de AEPIF por darse el tiempo de organizar este evento y brindarnos la oportunidad de conocer este curso tan interesante. A su vez un agradecimiento especial para el instructor Elvin Muñoz Vega por su esmero en la preparación de las clases y la paciencia para enseñarnos los temas desde lo más básico.

Referencias

[1] Diego Calvo. *Función de activación - Redes neuronales*. Link del documento. 2015.

[2] PyTorch. *TORCHVISION.TRANSFORMS*. Link del documento. 2017.

A Modelo de red neuronal utilizado

```
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}%")
```