CNN

Tinoco Sergio.
Instituto Politécnico Nacional, Escuela Superior de Cómputo.
Redes neuronales y aprendizaje profundo

25 de mayo de 2023

En esta práctica se va a implementar la red neuronal convolucional LeNet-5 cuya arquitectura está compuesta por 3 capas convolucionales y 2 capas totalmente conectadas como se muestra en la figura 1.

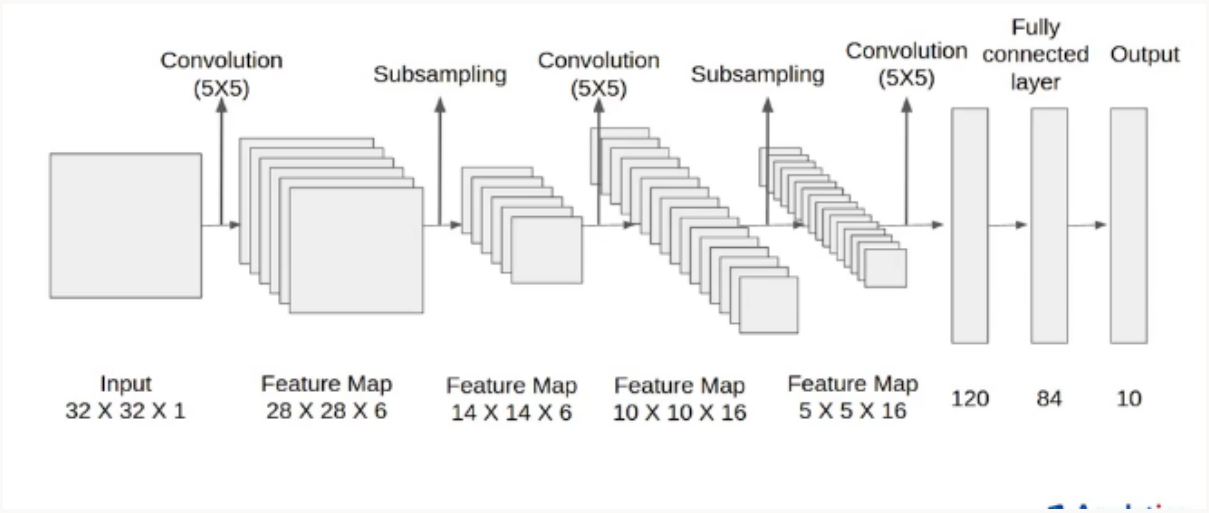

Figura 1. Arquitectura de LeNet-5.

Para ello se va a utilizar el dataset “MNIST” para el entrenamiento.

```
# Download and load the training data
trainset = datasets.MNIST('MNIST_data/', download=True, train=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
```

Se definen las capas convolucionales del modelo.

En la primera capa se tiene un kernel de 5x5 que produce un mapa de características de profundidad de 6.

Posteriormente se aplica el max pooling de la primera capa convolucional utilizando un kernel de 2x2 y un stride de 2.

En la segunda capa se tiene un kernel de 5x5 que produce un mapa de características de profundidad de 16.

Posteriormente se aplica el max pooling de la segunda capa convolucional utilizando un kernel de 2x2 y un stride de 2.

Finalmente una última capa convolucional de 5x5x120.

```
# De acuerdo al artículo de LeCun La primera capa está compuesta por 6 kernels de 5x5
self.conv1 = nn.Conv2d(1, 6, 5) # 1 canal de entrada 6 feature maps de salida, kernels de 5x5

# Después tenemos una capa maxpooling
# kernel_size=2, stride=2
self.pool1 = nn.MaxPool2d(2,2)

# Agregamos otra capa convolucional con 16 kernels de 5 x 5
self.conv2 = nn.Conv2d(6, 16, 5)

# Maxpooling
self.pool2 = nn.MaxPool2d(2,2)

# Agregamos una ultima capa convolucional con 120 kernels de 5 x 5
self.conv3 = nn.Conv2d(16, 120, 5)
```

Se definen las capas totalmente conectadas de la red neuronal.

La capa de entrada recibe un vector columna de 120 elementos que corresponde con la salida de la tercera capa convolucional, cuya salida es un vector de 120 elementos. La segunda capa totalmente conectada va a recibir dicho vector de 120 elementos y su salida va a contener 84 neuronas de acuerdo a la arquitectura de LeNet. Finalmente la capa de salida va a recibir dicho vector de 84 elementos y su salida corresponde con un vector de 10 elementos que corresponde con el número de clases definidas en el dataset utilizado.

```
# Capas totalmente conectadas
self.fc2 = nn.Linear(120,84)
self.relu2=nn.ReLU()
self.fc3 = nn.Linear(84,len(classes) )
```

Finalmente el modelo de la red quedaría de la siguiente forma:

```
LeNet5(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(16, 120, kernel_size=(5, 5), stride=(1, 1))
  (pool3): MaxPool2d(kernel_size=1, stride=1, padding=0, dilation=1, ceil_mode=False)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (relu2): ReLU()
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

Se calcula la exactitud del modelo antes del entrenamiento:

```
Porcentaje de exactitud antes de entrenar:  9.73
```

Como se puede observar el modelo tuvo una exactitud del 10%.

Se definen los hiper parámetros de la red.

```
epochs = 2
steps = 0
running_loss = 0
print_every = 40
```

Se realiza el entrenamiento:

```
Epoch: 1/2..  Training Loss: 1.665..  Test Loss: 0.969..  Test Accuracy: 0.647
Epoch: 1/2..  Training Loss: 0.849..  Test Loss: 0.777..  Test Accuracy: 0.702
Epoch: 1/2..  Training Loss: 0.717..  Test Loss: 0.739..  Test Accuracy: 0.714
Epoch: 1/2..  Training Loss: 0.697..  Test Loss: 0.671..  Test Accuracy: 0.731
Epoch: 1/2..  Training Loss: 0.634..  Test Loss: 0.645..  Test Accuracy: 0.746
Epoch: 1/2..  Training Loss: 0.603..  Test Loss: 0.615..  Test Accuracy: 0.759
Epoch: 1/2..  Training Loss: 0.601..  Test Loss: 0.607..  Test Accuracy: 0.769
Epoch: 1/2..  Training Loss: 0.583..  Test Loss: 0.575..  Test Accuracy: 0.780
```

Resultados de la clasificación de imágenes.

```
Epoch: 2/2..  Training Loss: 0.372..  Test Loss: 0.404..  Test Accuracy: 0.851
Epoch: 2/2..  Training Loss: 0.347..  Test Loss: 0.396..  Test Accuracy: 0.857
Epoch: 2/2..  Training Loss: 0.381..  Test Loss: 0.369..  Test Accuracy: 0.863
Epoch: 2/2..  Training Loss: 0.329..  Test Loss: 0.377..  Test Accuracy: 0.864
Epoch: 2/2..  Training Loss: 0.361..  Test Loss: 0.427..  Test Accuracy: 0.841
```

Figura 2. Resultados de la exactitud del modelo.

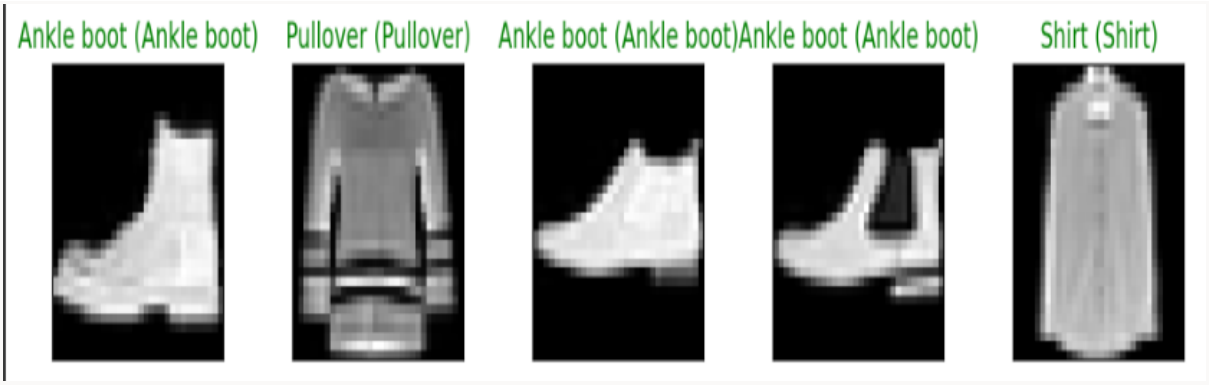

Figura 3. Resultados de las primeras 5 imágenes de prueba.

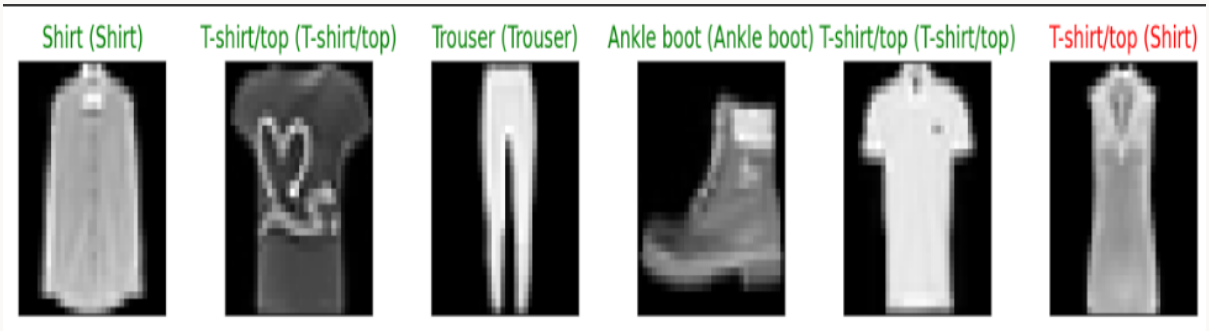

Figura 4. Resultados de las últimas 5 imágenes de prueba.

Conclusiones

Como se pudo observar en los resultados, la pérdida fue disminuyendo con cada época, alcanzando un valor menor a 0.4 lo cual no está mal para un entrenamiento con 2 épocas, si bien este resultado podría mejorar ajustando más algunos hiperparametros como el tamaño del batch, la función de pérdida, el optimizador, la tasa de aprendizaje etc.

LeNet es una arquitectura de red neuronal relativamente simple ya que no cuenta con una gran cantidad de capas o neuronas comparado con otros modelos más complejos y aún así su efectividad resulta ser adecuada para muchos problemas de clasificación de imágenes, lo que la hace un buen modelo de clasificación.