COLOMBIA POTENCIA DE LA VIDA

TIC

Actividad 3

Redes Neuronal Convolucional aplicadas en MNIST reconocimiento de dígitos manuscritos

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

UTP Universidad Tecnológica de Pereira

Redes Neuronal Convolucional aplicadas en MNIST reconocimiento de dígitos manuscritos

Este código implementa una Red Neuronal Convolucional (CNN) para el reconocimiento de dígitos manuscritos utilizando el conjunto de datos MNIST.

Importa las librerías a utilizar:

```
import keras
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Carga del Conjunto de Datos:
Utiliza Keras para cargar el conjunto de datos MNIST, que consiste en imágenes de dígitos escritos a mano junto con sus etiquetas correspondientes.

```
# Cargar el conjunto de datos MNIST
mnist = keras.datasets.mnist

# Cargue la división de entrenamiento y prueba del conjunto de datos MNIST
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
```

Redes Neuronal Convolucional aplicadas en MNIST reconocimiento de dígitos manuscritos

Normalización de los Datos:
Normaliza los valores de píxeles de las imágenes dividiéndolos por 255 para escalarlos al rango [0, 1].

```
# Normalizar los valores de píxeles del tren y probar las imágenes.
training_images  = training_images / 255.0
test_images = test_images / 255.0
```

Construcción del Modelo:
Define el modelo de clasificación utilizando Sequential, que es un modelo lineal de capas apiladas.
Las capas principales son:
Conv2D: Capa de convolución con 32 filtros de tamaño (3,3) y función de activación ReLU.
MaxPooling2D: Capa de agrupación máxima con un tamaño de ventana de (2,2).
Flatten: Capa para aplanar la salida de la capa convolucional.

Redes Neuronal Convolucional aplicadas en MNIST reconocimiento de dígitos manuscritos

Dos capas Dense completamente conectadas con activación ReLU y softmax, respectivamente.

```
# Construir el modelo de clasificación.
model = keras.models.Sequential([
    # Agregar convoluciones y max pooling
    keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),
    keras.layers.MaxPooling2D(2, 2),
    # Agrega las mismas capas que antes.
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')])

model.summary()
```

Compilación del Modelo:
Compila el modelo utilizando el optimizador Adam y la pérdida de entropía cruzada categórica dispersa.

```
# Compilar el modelo
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Redes Neuronal Convolucional aplicadas en MNIST reconocimiento de dígitos manuscritos

Entrenamiento del Modelo:

Entrena el modelo con los datos de entrenamiento durante 10 épocas.

```
# Entrenar el modelo
history = model.fit(training_images, training_labels, epochs=10)

# Graficar el historial de entrenamiento:
pd.DataFrame(history.history).plot(grid=True)
```

Redes Neuronal Convolucional aplicadas en MNIST reconocimiento de dígitos manuscritos

Evaluación del Modelo:

Evalúa el modelo en el conjunto de entrenamiento y en el conjunto de prueba para calcular la pérdida y la precisión.

```
# Evaluar el modelo en el conjunto de entrenamiento
loss, accuracy = model.evaluate(training_images, training_labels)
print("Pérdida en el conjunto de entrenamiento:", loss)
print("Precisión en el conjunto de entrenamiento:", accuracy)

# Evaluar el modelo con datos no vistos
loss, accuracy = model.evaluate(test_images, test_labels)
print("Pérdida en el conjunto de prueba:", loss)
print("Precisión en el conjunto de prueba:", accuracy)
```

Redes Neuronal Convolucional aplicadas en MNIST reconocimiento de dígitos manuscritos

Predicción:

Realiza una predicción en una imagen de prueba específica e imprime la etiqueta real y la clasificación prevista.

```
#predict
index = 1
print(f'Label: {test_labels[index]}')
classification = model.predict(test_images[index:index+1])
print(f'Classification:\n {classification.reshape(-1,1)}')
```

Al final, el código también grafica el historial de entrenamiento, mostrando cómo la pérdida y la precisión del modelo cambiaron durante el entrenamiento. Esto proporciona una visión general del rendimiento del modelo a lo largo del tiempo.

Al ejecutar el código obtendrá

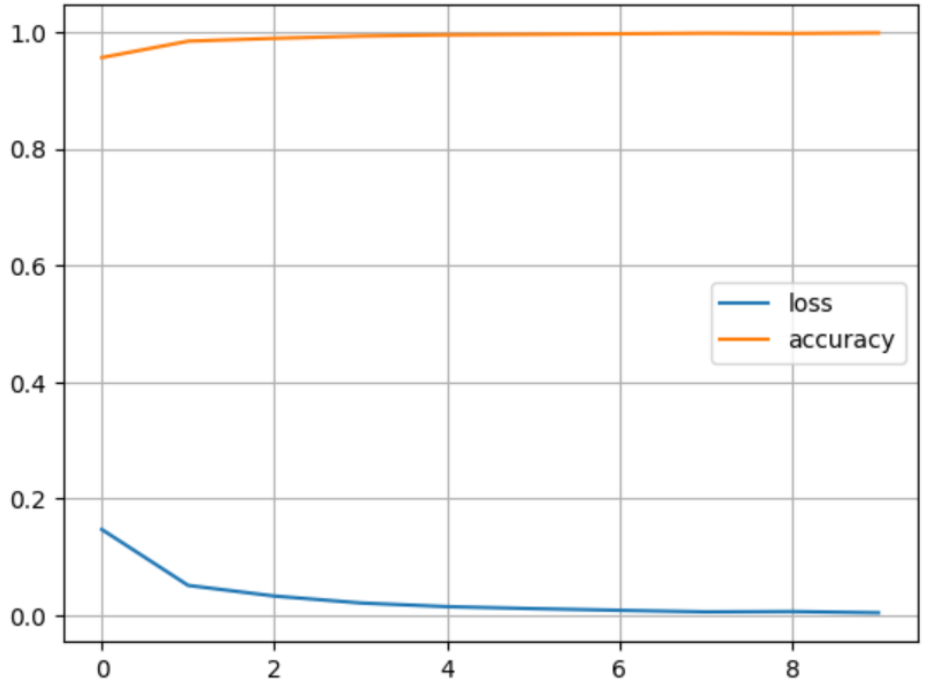
La arquitectura de la red

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 flatten_1 (Flatten)         (None, 5408)              0

 dense_2 (Dense)             (None, 128)               692352

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 693962 (2.65 MB)
Trainable params: 693962 (2.65 MB)
Non-trainable params: 0 (0.00 Byte)
```

Al ejecutar el código obtendrá

visualización del proceso de entrenamiento

```
Epoch 1/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.1466 - accuracy: 0.9566
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0506 - accuracy: 0.9847
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0324 - accuracy: 0.9896
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0205 - accuracy: 0.9934
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0142 - accuracy: 0.9955
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0108 - accuracy: 0.9964
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0079 - accuracy: 0.9973
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0053 - accuracy: 0.9984
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0058 - accuracy: 0.9979
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0037 - accuracy: 0.9990
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0024 - accuracy: 0.9993
```



Al ejecutar el código obtendrá

Evaluación del modelo en el conjunto de entrenamiento y prueba

```
Pérdida en el conjunto de entrenamiento: 0.002443862846121192
Precisión en el conjunto de entrenamiento: 0.9992833137512207
313/313 [==============================] - 1s 2ms/step - loss: 0.0609 - accuracy: 0.9865
Pérdida en el conjunto de prueba: 0.06089621037244797
Precisión en el conjunto de prueba: 0.9865000247955322
```

Predicción de un ejemplo

```
Label: 2
1/1 [==============================] - 0s 125ms/step
Classification:
 [[2.9093849e-11]
 [4.2438487e-08]
 [9.9999988e-01]
 [6.1310955e-15]
 [1.0083485e-19]
 [1.6886853e-19]
 [8.0925041e-08]
 [1.3714481e-17]
 [2.3612695e-15]
 [5.5636638e-19]]
```

Al ejecutar el código obtendrá

Después de realizar el entrenamiento vamos a ejecutar un código que permite Visualizar las convoluciones y Pooling
Este código realiza la visualización de las convoluciones y el agrupamiento (pooling) en una red neuronal convolucional (CNN)

Preparación de los Datos:
Imprime las etiquetas de las primeras 30 imágenes del conjunto de prueba para entender qué dígitos representan.

```
print(test_labels[:30])
```

```
[7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1]
```

Al ejecutar el código obtendrá

Selección de Imágenes de Interés:
Selecciona tres imágenes específicas del conjunto de prueba para la visualización. Estas imágenes se identifican por sus índices: FIRST_IMAGE, SECOND_IMAGE y THIRD_IMAGE.

```
FIRST_IMAGE=3
SECOND_IMAGE=10
THIRD_IMAGE=25

print(test_labels[FIRST_IMAGE])
print(test_labels[SECOND_IMAGE])
print(test_labels[THIRD_IMAGE])
```

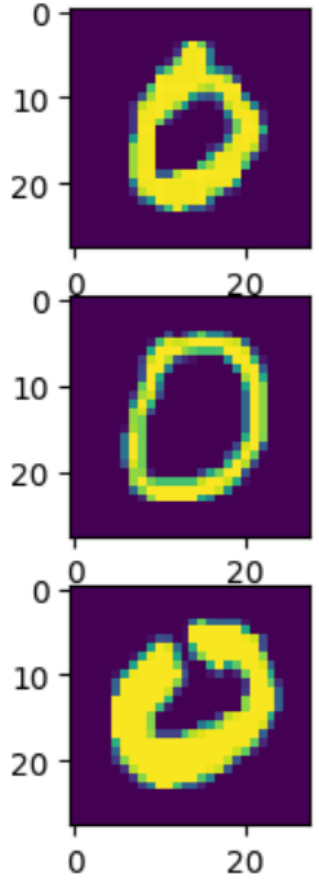
```
0
0
0
```

Al ejecutar el código obtendrá

Visualización de Imágenes:

Crea un arreglo de subtramas (3x1) y muestra las tres imágenes seleccionadas.

```
f, axarr = plt.subplots(3,1)
axarr[0].imshow(test_images[FIRST_IMAGE])
axarr[1].imshow(test_images[SECOND_IMAGE])
axarr[2].imshow(test_images[THIRD_IMAGE])
```



Al ejecutar el código obtendrá

Extracción de Activaciones:
Utiliza un modelo de activación para extraer las activaciones de la capa especificada (layer) para cada una de las imágenes seleccionadas. Se crea un modelo de activación que toma la entrada del modelo original y produce la salida de la capa de interés.

```
layer = 0
filter1 = 0
filter2 = 10

layer_outputs = [layer.output for layer in model.layers]
activation_model = keras.models.Model(inputs = model.input, outputs = layer_outputs)
```

Al ejecutar el código obtendrá

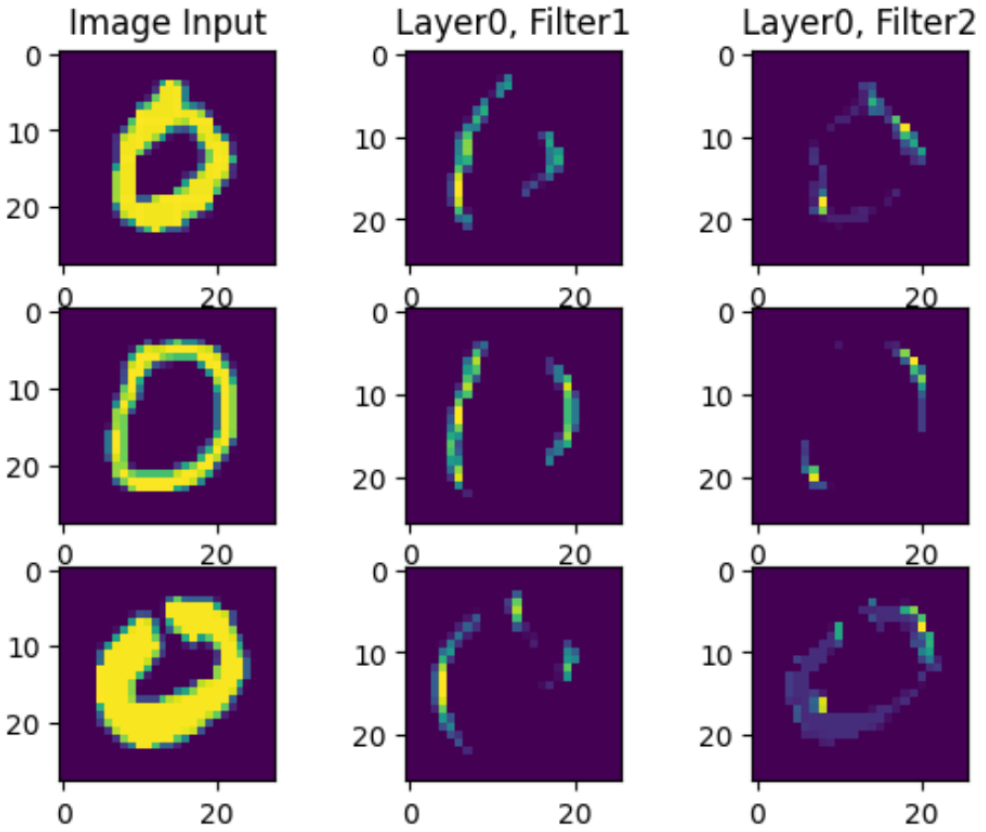
Visualización de Convoluciones:
Para cada imagen, muestra tres subtramas (3x3) que representan:

La imagen original.
La activación de los filtros especificados (filter1 y filter2) en la capa especificada (layer). Las activaciones se obtienen aplicando el modelo de activación a la imagen y seleccionando las salidas correspondientes a los filtros de interés.

```
f, axarr = plt.subplots(3,3)

axarr[0, 0].set_title("Image Input")
axarr[0,0].imshow(test_images[FIRST_IMAGE])
f1 = activation_model.predict(test_images[FIRST_IMAGE].reshape(1, 28, 28, 1))[layer]
axarr[0, 1].set_title("Layer"+ str(layer) +", Filter1")
axarr[0,1].imshow(f1[0, : , :, filter1])
axarr[0, 2].set_title("Layer"+ str(layer) +", Filter2")
axarr[0,2].imshow(f1[0, : , :, filter2])

axarr[1,0].imshow(test_images[SECOND_IMAGE])
f2 = activation_model.predict(test_images[SECOND_IMAGE].reshape(1, 28, 28, 1))[layer]
axarr[1,1].imshow(f2[0, : , :, filter1])
axarr[1,2].imshow(f2[0, : , :, filter2])

axarr[2,0].imshow(test_images[THIRD_IMAGE])
f3 = activation_model.predict(test_images[THIRD_IMAGE].reshape(1, 28, 28, 1))[layer]
axarr[2,1].imshow(f3[0, : , :, filter1])
axarr[2,2].imshow(f3[0, : , :, filter2])
```

Al ejecutar el código obtendrá

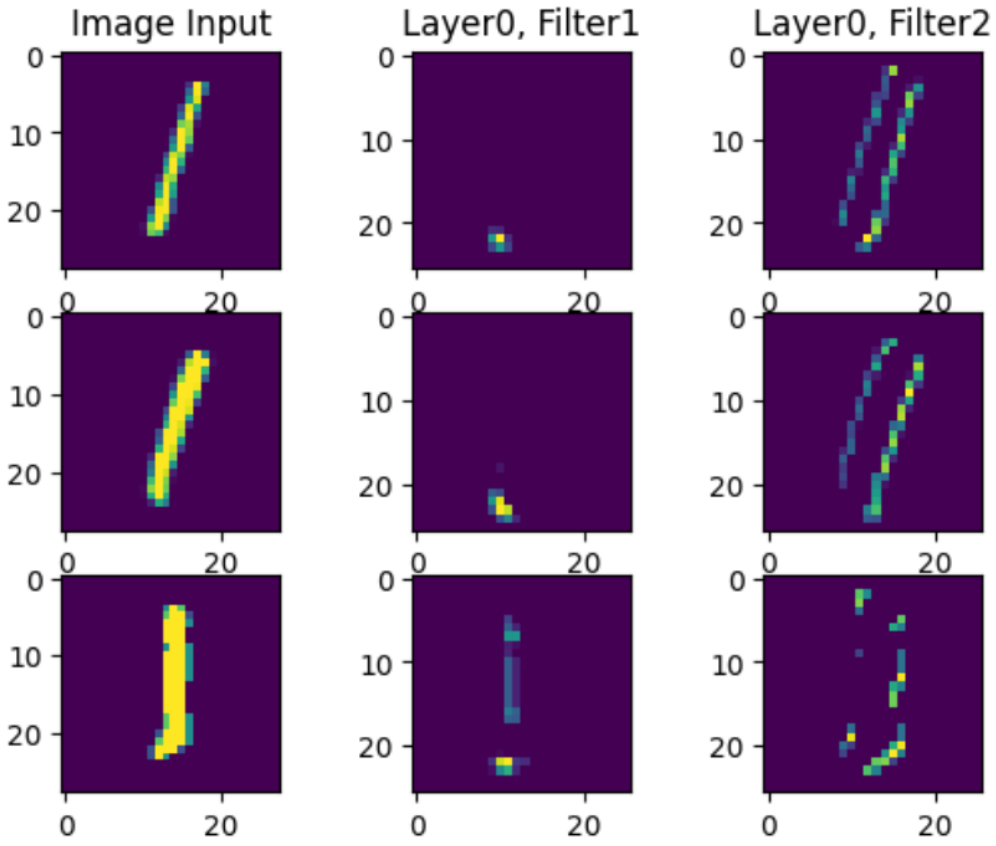
Este código proporciona una visualización detallada de las activaciones de los filtros en una capa específica de una CNN, lo que permite entender cómo la red neuronal procesa y extrae características de las imágenes de entrada. Esto es útil para comprender qué características son detectadas por la red y cómo influyen en la predicción final.
Al ejecutar el código se obtiene



Al ejecutar el código obtendrá

seleccionando elementos etiquetados como 7 tenemos

```
FIRST_IMAGE=2
SECOND_IMAGE=5
THIRD_IMAGE=14
```



COLOMBIA POTENCIA DE LA VIDA

TIC

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

UTP Universidad Tecnológica de Pereira