Redes Convolucionales

Las Redes neuronales convolucionales son un tipo de redes neuronales artificiales donde las «neuronas» corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico. Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones.

Cómo Funcionan

Las redes neuronales convolucionales consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general se añade una función para realizar un mapeo causal no-lineal.

Como cualquier red empleada para clasificación, al principio estas redes tienen una fase de extracción de características, compuesta de neuronas convolucionales , luego hay una reducción por muestreo y al final tendremos neuronas de perceptrón mas sencillas para realizar la clasificación final sobre las características extraídas.

La fase de extracción de características se asemeja al proceso estimulante en las células de la corteza visual. Esta fase se compone de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. Según progresan los datos a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo siendo estas activadas por características cada vez más complejas.

Como se logra que una red convolucional aprenda

Las Redes neuronales Convolucionales, CNN aprenden a reconocer una diversidad de objetos dentro de imágenes , pero para ello necesitan «entrenarse» de previo con una cantidad importante de «muestras» -lease más de 10.000, de ésta forma las neuronas de la red van a poder captar las características únicas -de cada objeto- y a su vez, poder generalizarlo – a esto es lo que se le conoce como el proceso de «aprendizaje de un algoritmo » . Nuestra red va a poder reconocer por ejemplo un cierto tipo de célula porque ya la ha «visto» anteriormente muchas veces, pero no solo buscará celulas semejantes sino que podrá inferir imagenes que no conozca pero que relaciona y en donde podrían existir similitudes , y esta es la parte inteligente del conocimiento

Ejemplo

Librerias

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras.datasets import mnist
import tensorflow as tf
```

Cargar Data

```
(train_samples, train_labels), (test_samples,test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

Procesamiento de los Datos (Estandarizacion)

```
train_samples = train_samples.reshape(train_samples.shape [0], 28, 28, 1)
test_samples = test_samples.reshape(test_samples.shape [0], 28, 28, 1)
train_samples = train_samples.astype(np.float32)
test_samples = test_samples.astype(np.float32)
train_samples = train_samples/255
test_samples = test_samples/255
```

```
c_train_labels = np_utils.to_categorical(train_labels, 10)
c_test_labels = np_utils.to_categorical(test_labels, 10)
```

Crear el Modelo

```
convnet = Sequential()
convnet.add(Convolution2D(32, 4, 4, activation=tf.keras.activations.relu, input_shape=(28,28,
convnet.add(MaxPooling2D(pool_size=(2,2)))
convnet.add(Convolution2D(32, 3, 3, activation=tf.keras.activations.relu))
convnet.add(MaxPooling2D((2,2), strides=(2,2), padding='same'))
convnet.add(Dropout(0.3))
```

```
convnet.add(Flatten())
convnet.add(Dense(10. activation=tf.keras.activations.softmax))
```

Compilar Modelo

```
convnet.compile(loss=tf.keras.losses.mean_squared_error, optimizer='sgd', metrics=['accuracy'
```

Entrenar Modelo

```
convnet.fit(train_samples, c_train_labels, batch_size=32, epochs=20, verbose=1)
```

```
Epoch 1/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0900 - accuracy: 0.13
Epoch 2/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0898 - accuracy: 0.17
Epoch 3/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0896 - accuracy: 0.20
Epoch 4/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0893 - accuracy: 0.23
Epoch 5/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0891 - accuracy: 0.25
Epoch 6/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0887 - accuracy: 0.27
Epoch 7/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0883 - accuracy: 0.28
Epoch 8/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0878 - accuracy: 0.29
Epoch 9/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0871 - accuracy: 0.29
Epoch 10/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0863 - accuracy: 0.29
Epoch 11/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0851 - accuracy: 0.29
Epoch 12/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0836 - accuracy: 0.29
Epoch 13/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0820 - accuracy: 0.30
Epoch 14/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0804 - accuracy: 0.32
Epoch 15/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0790 - accuracy: 0.33
Epoch 16/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0775 - accuracy: 0.35
Epoch 17/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0759 - accuracy: 0.38
Epoch 18/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0743 - accuracy: 0.39
Epoch 19/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0726 - accuracy: 0.42
Epoch 20/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0709 - accuracy: 0.44
<keras.callbacks.History at 0x7f9454364d90>
```

Probamos el Modelo

```
metrics = convnet.evaluate(test_samples, c_test_labels, verbose=1)
print()
print("%s: %.2f%%" % (convnet.metrics_names[1], metrics[1]*100))
predictions = convnet.predict(test_samples)
print(predictions.shape)
print(predictions)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0666 - accuracy: 0.548

accuracy: 54.82%
(10000, 10)
[[0.05059059 0.07069426 0.01916016 ... 0.25228304 0.09556083 0.22081254]
 [0.08034041 0.17502117 0.1984404  ... 0.02261334 0.04770268 0.02063787]
 [0.01580278 0.7415649  0.03589518 ... 0.03316012 0.03164867 0.03200641]
 ...
 [0.02507208 0.13871466 0.01486443 ... 0.23137373 0.11065524 0.22407784]
 [0.03921882 0.12511261 0.01552036 ... 0.21474238 0.09585022 0.21113762]
 [0.35220423 0.00869487 0.20146862 ... 0.01083724 0.04528471 0.01824366]]
```

✓ 1s completed at 10:59 AM