Convolutional Neural Network

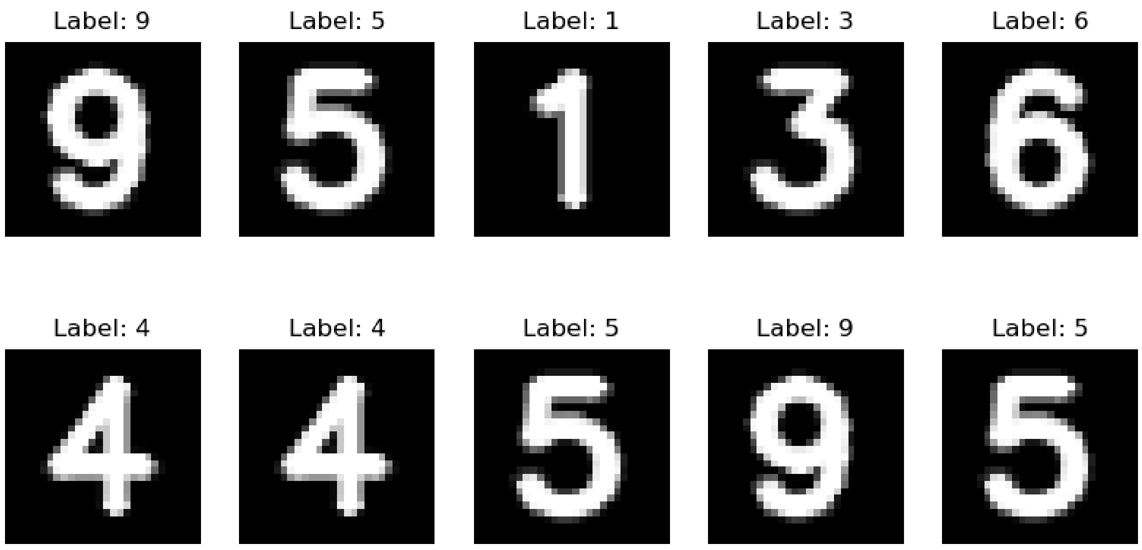
A convolutional neural network (CNN) is a type of artificial neural network that is particularly effective at analyzing visual imagery. It is designed to automatically and adaptively learn spatial hierarchies of features from input data. CNNs have become the go-to architecture for tasks such as image classification, object detection, facial recognition, and more.

The key components and operations within a CNN:

1. **Convolutional Layers:** are core building blocks of CNNs. Each layer consists of a set of learnable filters (AKA Kernels). These filters are small, spatially local, and applied across the entire input image to extract specific features such as edges, textures, or patterns. Convolution involves sliding the filter over the input image, performing element-wise multiplication with the local regions, and then summing up the results to produce a feature map.
2. **Activation Function:** After each convolution operation, a non-linear activation function (e.g., ReLU, sigmoid, tanh) is applied element-wise to introduce non-linearity into the network, enabling it to learn complex patterns and relationships in the data.
3. **Pooling Layers**: These layers are interspersed between convolutional layers to reduce the spatial dimensions (width and height) of the feature maps while preserving important information. Max pooling and average pooling are common pooling techniques used to downsample feature maps by taking the maximum or average value within each local region of the input.
4. **Fully Connected Layers:** Traditionally located towards the end of the network, these layers take the high-level features extracted by the convolutional layers and perform classification or regression tasks. Each neuron in a fully connected layer is connected to every neuron in the previous layer, allowing the network to learn complex relationships across the entire feature space.
5. **Softmax Layer:** typically used for classification tasks, converting the network's output into probability distributions over multiple classes.
6. **Loss Function:** Measures the difference between predicted and actual values during training.
7. **Optimization Algorithm:** Updates the network's parameters to minimize the loss function, commonly using stochastic gradient descent (SGD) or its variants.
8. **Backpropagation:** The process of computing gradients and updating parameters to improve the network's performance iteratively during training.

Dataset:

The dataset is created within the sample code below and we are also printing samples from the dataset, here is the screenshot of that sample:



Python Code:

```
import numpy as np
import cv2
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Function to generate a random image of a handwritten digit
def generate_digit_image(digit):
    image = np.zeros((28, 28), dtype=np.uint8)  # Create a blank 28x28
image
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(image, str(digit), (5, 22), font, 0.8, (255), 2,
cv2.LINE_AA)  # Draw the digit on the image
    return image

# Generate the dataset
num_samples = 10000
images = []
labels = []
```

```
for _ in range(num_samples):
    # Generate a random digit (0-9)
    digit = np.random.randint(0, 10)
    # Generate the corresponding image
    image = generate_digit_image(digit)
    # Append the image and label to the dataset
    images.append(image)
    labels.append(digit)

# Convert lists to NumPy arrays
images = np.array(images)
labels = np.array(labels)

# Visualize a few samples from the dataset
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(images[i], cmap='gray')
    plt.title('Label: {}'.format(labels[i]))
    plt.axis('off')
plt.show()

# Reshape images to fit the CNN input shape
images = images.reshape(-1, 28, 28, 1)

# Normalize pixel values to range [0, 1]
images = images.astype('float32') / 255.0

# One-hot encode labels
labels = to_categorical(labels)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels,
test_size=0.2, random_state=42)

# Build CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
```

```python
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=64,
validation_split=0.1)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

The code yielded:

```
  super().__init__(
Epoch 1/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 8s 39ms/step - accuracy: 0.7750 -
loss: 0.9102 - val_accuracy: 1.0000 - val_loss: 2.8289e-04
Epoch 2/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 4s 34ms/step - accuracy: 1.0000 -
loss: 2.2983e-04 - val_accuracy: 1.0000 - val_loss: 1.5034e-04
Epoch 3/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 4s 34ms/step - accuracy: 1.0000 -
loss: 1.2594e-04 - val_accuracy: 1.0000 - val_loss: 9.0492e-05
Epoch 4/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - accuracy: 1.0000 -
loss: 7.9572e-05 - val_accuracy: 1.0000 - val_loss: 6.0417e-05
Epoch 5/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 4s 32ms/step - accuracy: 1.0000 -
loss: 5.2256e-05 - val_accuracy: 1.0000 - val_loss: 4.3134e-05
Epoch 6/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 4s 33ms/step - accuracy: 1.0000 -
loss: 3.8565e-05 - val_accuracy: 1.0000 - val_loss: 3.2037e-05
Epoch 7/10
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 4s 32ms/step - accuracy: 1.0000 -
loss: 2.8835e-05 - val_accuracy: 1.0000 - val_loss: 2.4604e-05
Epoch 8/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 4s 32ms/step - accuracy: 1.0000 -
loss: 2.2270e-05 - val_accuracy: 1.0000 - val_loss: 1.9416e-05
Epoch 9/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 4s 32ms/step - accuracy: 1.0000 -
loss: 1.7662e-05 - val_accuracy: 1.0000 - val_loss: 1.5603e-05
Epoch 10/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 4s 32ms/step - accuracy: 1.0000 -
loss: 1.4357e-05 - val_accuracy: 1.0000 - val_loss: 1.2731e-05
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 -
loss: 1.1855e-05
Test accuracy: 1.0
```