Convolutional neural network (Mạng nơ-ron tích chập)

Đặt vấn đề:

- Chúng ta phân biệt các vật thể khác nhau như thế nào?

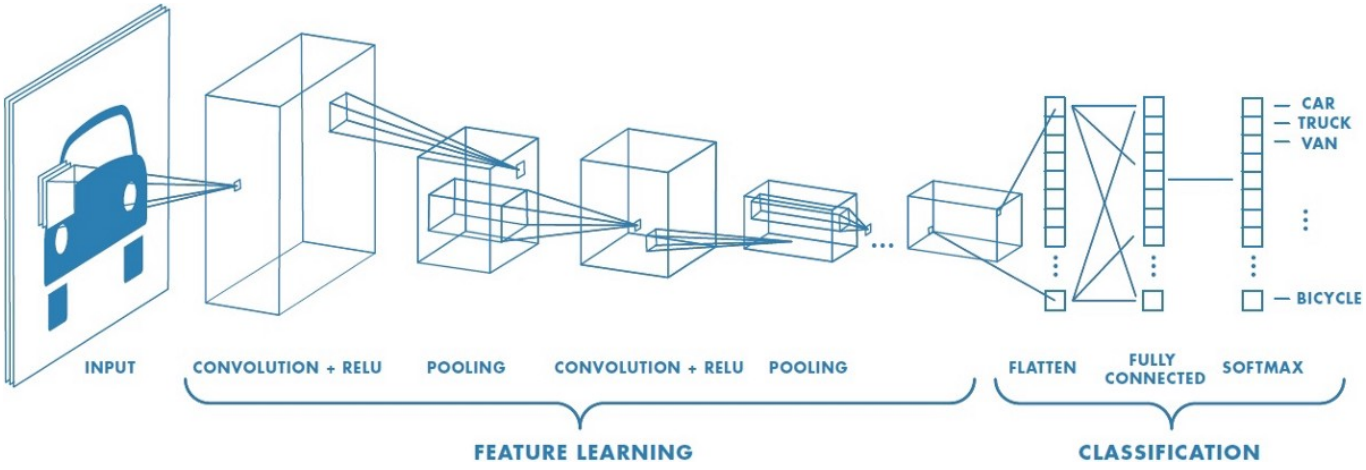
$\Rightarrow$ Các đặc trưng đường nét, cạnh, màu sắc, kích thước...

Làm thế nào để máy có thể hiểu 1 tấm ảnh?

1.Máy tính làm gì với CNN?

CNN cho phép các máy tính có khả năng “nhìn” và “phân tích”, nói 1 cách dễ hiểu, CNN được sử dụng để nhận dạng hình ảnh bằng cách đưa hình ảnh đó qua nhiều layer với một bộ lọc tích chập để sau cùng có được một điểm số nhận dạng đối tượng.

CNN có 02 phần chính: Lớp trích xuất đặc trưng của ảnh (Conv, Relu và Pool) và Lớp phân loại (FC và softmax).

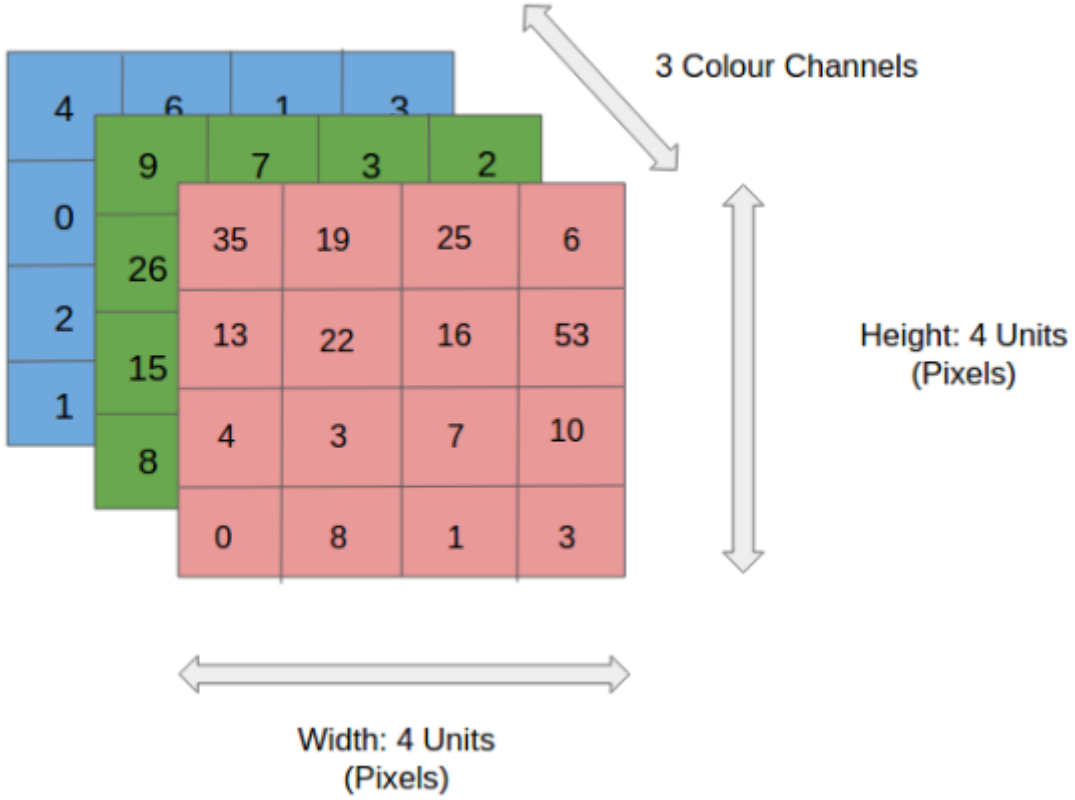


Đầu vào (dữ liệu training):

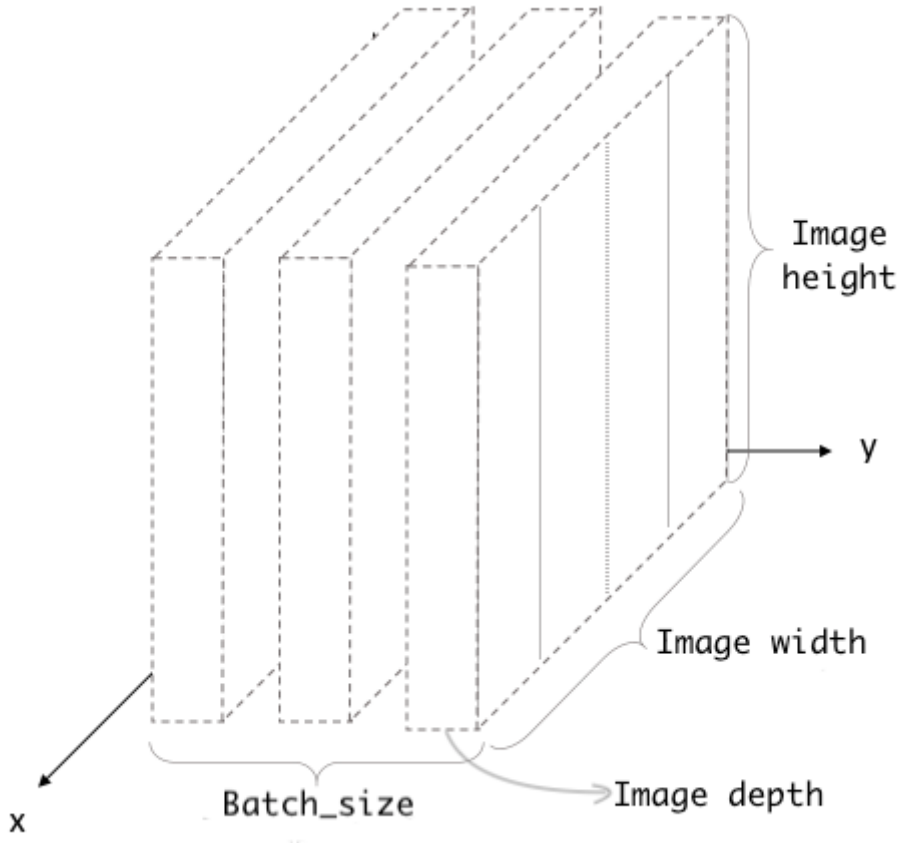
Input đầu vào là một bức ảnh được biểu diễn bởi ma trận pixel với kích thước: [W x H x D]

- W: chiều rộng
- H: chiều cao
- D: Là độ sâu, hay dễ hiểu là số lớp màu của ảnh.

> Ví dụ ảnh RGB sẽ là 3 lớp ảnh Đỏ, Xanh Dương, Xanh.



Kích thước input và output của môn hình CNN:



Batch_size là số tấm ảnh chúng ta đưa vào để train. Image_depth là số ma trận biểu thị cường độ sáng các màu. Ví dụ:

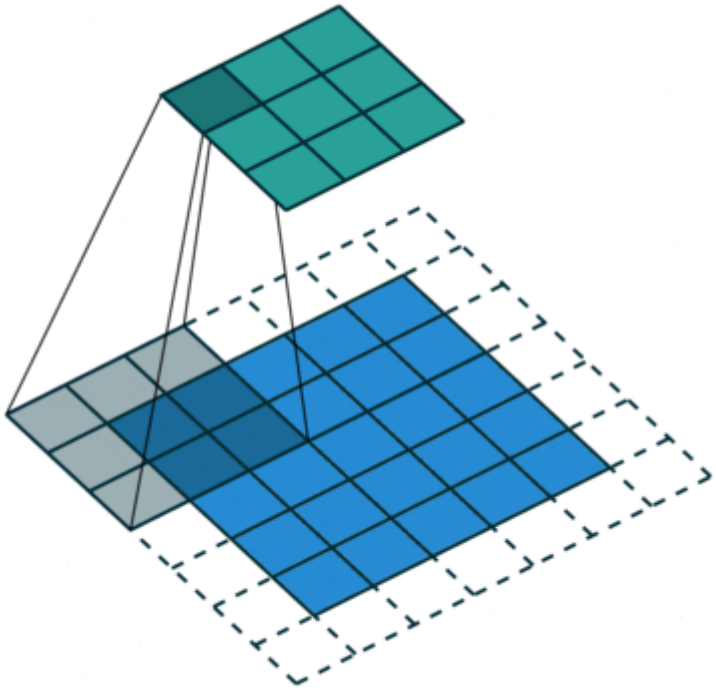
- Đối với n tấm ảnh RBG thì batch_size là n, Image_depth là 3
- Đối với n tấm ảnh Gray thì Batch_size tương tự,nhưng Image_depth là 1.

2. Conv Layer:

Mục tiêu của các lớp tích chập là trích xuất các đặc trưng của ảnh đầu vào.

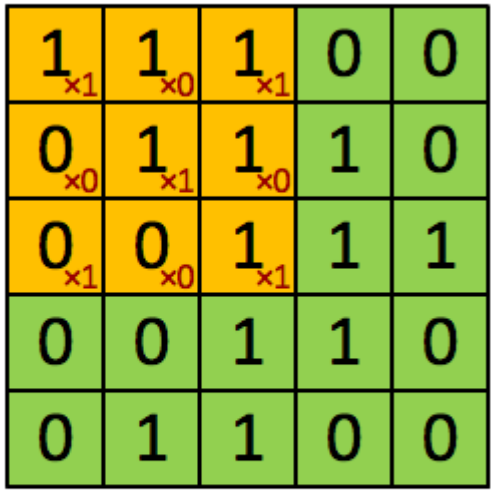
Ví dụ:

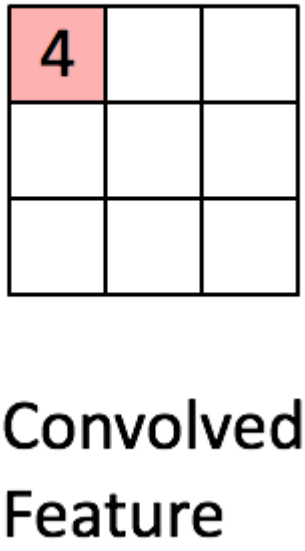
- Lớp Conv so khớp bộ lọc (filter) với từng phần của tấm ảnh:



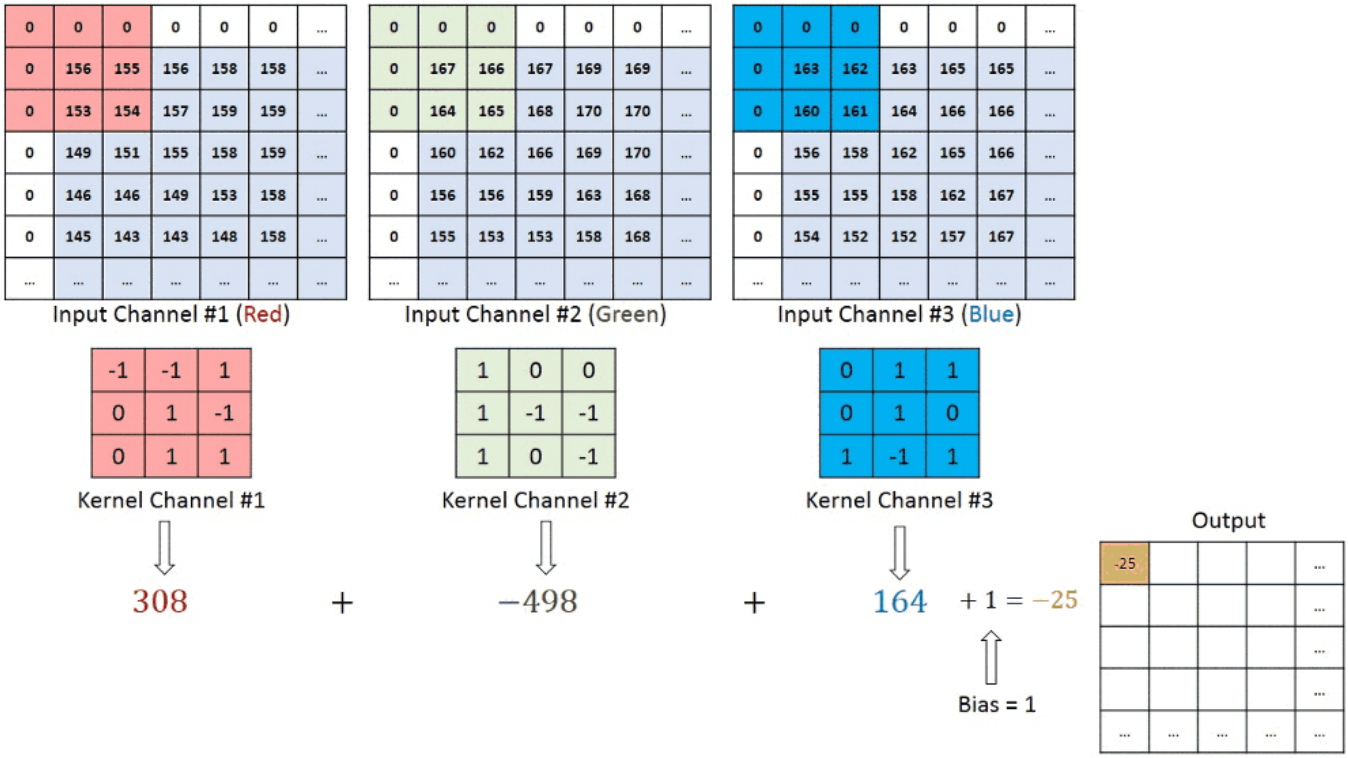
`

- Đối với 1 kênh màu:





- Đối với nhiều kênh màu:

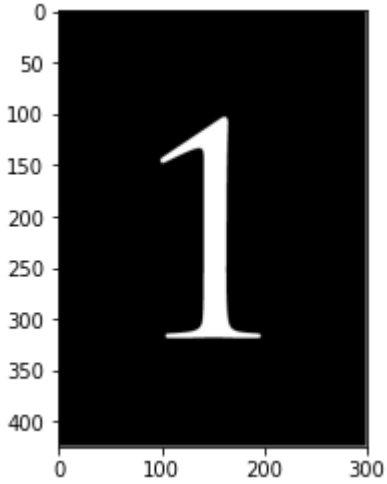


- Ví dụ 1 tấm ảnh thực tế thì qua lớp conv layer trông như thế nào?

```
 1
 2
 3  def conv2d_11(image, fil, stride = 1, pad = 0):
 4      n_H_old, n_W_old = image.shape
 5      f, f = fil.shape
 6      image_pad = np.pad(image, pad_width=pad, mode = 'constant', const
 7      n_H_new = int((n_H_old - f + 2*pad)/stride) + 1
 8      n_W_new = int((n_W_old - f + 2*pad)/stride) + 1
 9      image_res = np.zeros((n_H_new, n_W_new))
10      for h in range(n_H_new):
11          for v in range(n_W_new):
12              h_start = h*stride
13              h_end = h_start + f
14              v_start = v*stride
15              v_end = v_start + f
16              image_res[h, v] = np.sum(image_pad[h_start:h_end, v_start
17      return image_res
18  # add noise
19  import numpy as np
20  import matplotlib.pyplot as plt
21  import skimage
22  import skimage.io
23  fn = '/so1.jpg'
24  img_rgb = skimage.io.imread(fn )
25  img_gray = (skimage.color.rgb2gray(img_rgb)*255).astype(np.int)
26
27
28
```

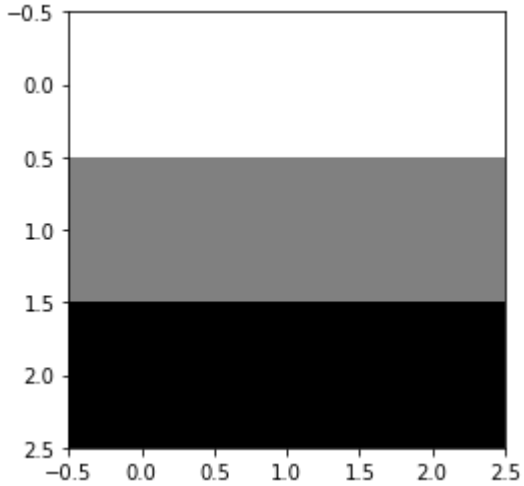
```
plt.imshow(img_gray, cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7f3137bd6d68>
```



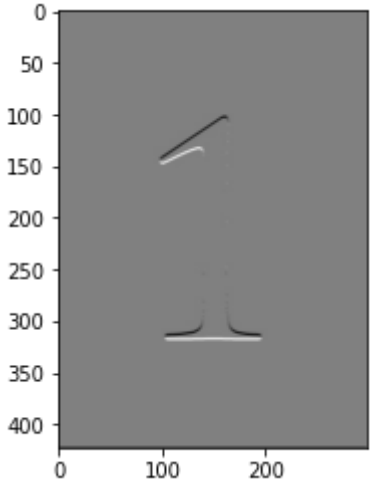

```
fil = np.array([[1, 1, 1],[0, 0, 0],[-1, -1, -1]])
img_ver_edge = conv2d_11(img_gray, fil)
plt.imshow(fil, cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7f31379bc208>
```



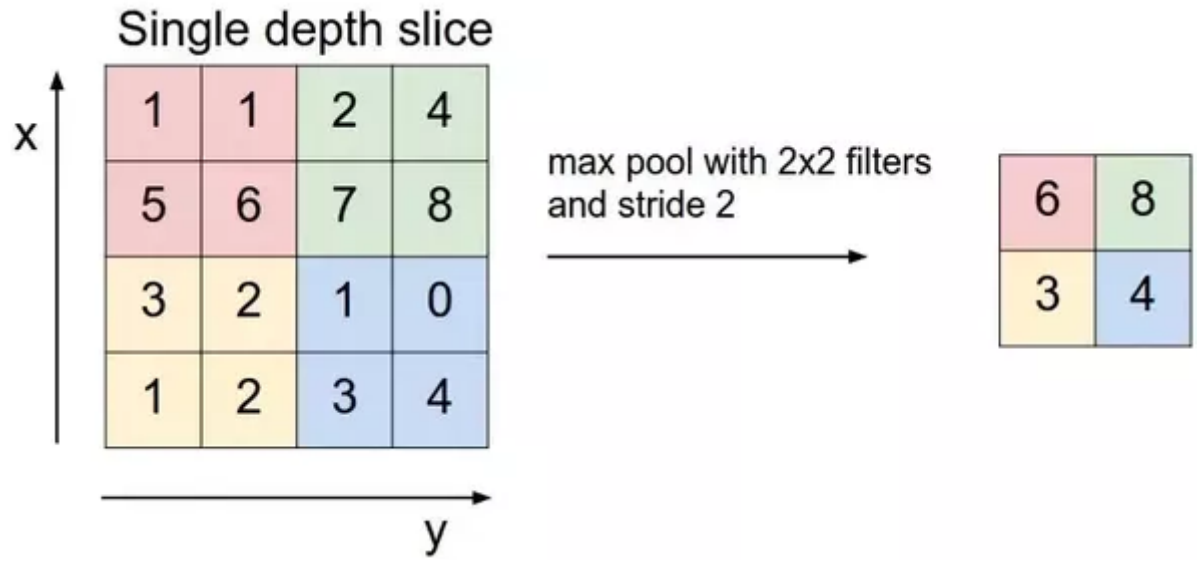

```
plt.imshow(img_ver_edge, cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7f3137992c50>
```

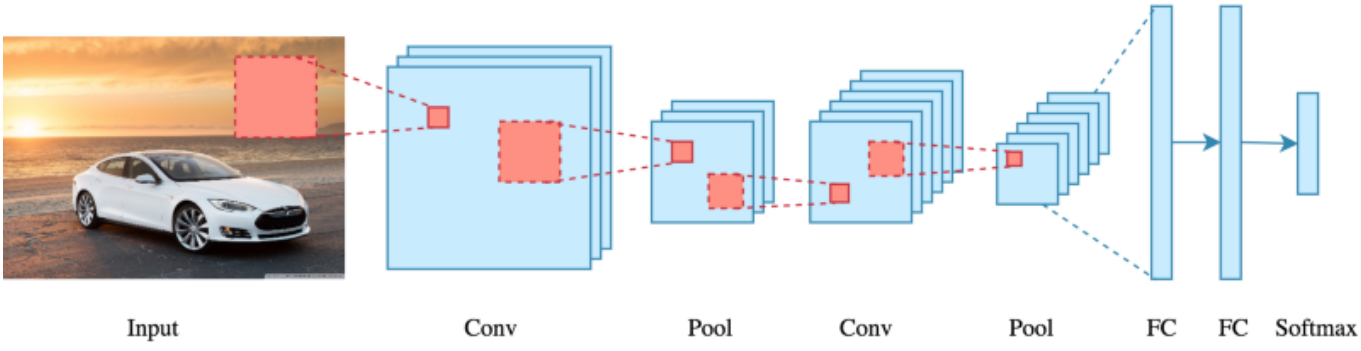


3. Pooling Layer:

Pool Layer thực hiện chức năng làm giảm chiều không gian của đầu vào và giảm độ phức tạp tính toán của model ngoài ra Pool Layer còn giúp kiểm soát hiện tượng overffiting. Số lượng weights càng nhiều thì độ phức tạp của model càng lớn và dễ dẫn đến overfit, hoặc quá ít thì không đủ mạnh với những dữ liệu phức tạp. Vì vậy ta cần cân nhắc kỹ việc lựa chọn model sao cho phù hợp với dữ liệu huấn luyện để cho ra một model có độ chính xác cao. Ví dụ: Max pooling với bộ lọc 2x2 và stride = 2. Bộ lọc sẽ chạy dọc ảnh. Và với mỗi vùng ảnh được chọn, sẽ chọn ra 1 giá trị lớn nhất và giữ lại.



4. Fully_Connected Layer (FC):



- Sau khi kết thúc phần trích xuất đặc trưng. Ta được 1 vector đặc trưng của ảnh. Thực chất khi kết thúc phần trích xuất, ta có thêm 1 lớp flatten, trải các ma trận thu được thành 1 vector đặc trưng. Vector này là input cho lớp fully connected layer.

Nhận diện chữ số viết tay

Data:

- Gồm 10000 tấm tải từ google, 1500 tấm tự tạo gồm 10 classes.
- Tập dữ liệu train: 9200 tấm
- Tập dữ liệu test: 2300 tấm

Model 1

Các bước tiền xử lí:

- Chuẩn hóa (normalize) giá trị các pixels về đoạn 0, 1.
- Resize các tấm ảnh về cùng kích thước 64×64

Mạng CNN:

- 2 convolution layer:
  - Lớp conv1: kích thước kernel là 3×3, gồm 30 filters, hàm kích hoạt là hàm relu.
  - Lớp conv2: kích thước kernel là 5×5, gồm 20 filters, hàm kích hoạt là hàm relu
- Dùng maxpooling layer2D kích thước 2×2 sau mỗi lớp conv.
- Sau đó là 1 lớp flatten
- Sử dụng 3 mạng Dense và 2 lớp Dropout tỉ lệ 0.25 để chống overfit.

Batch size = 32

Epochs = 10

Kết quả model

Accuracy Train: 0.9228

Accuracy Test: 0.9227

Nhận xét: model đồ có độ chính xác cao, không bị overfiting nhưng vì bộ data không được đa dạng và ảnh đầu vào là ảnh màu nên đối với các tấm ảnh có màu sắc nhiều thì model dự đoán không tốt

Model 2:

Các bước tiền xử lí:

- Chuyển ảnh màu về ảnh xám
- Chuẩn hóa (normalize) giá trị các pixels về đoạn 0, 1.
- Resize các tấm ảnh về cùng kích thước 32×32

Mạng CNN:

- 2 convolution layer:
  - Lớp conv1: kích thước kernel là 3×3, gồm 30 filters, hàm kích hoạt là hàm relu.
  - Lớp conv2: kích thước kernel là 5×5, gồm 20 filters, hàm kích hoạt là hàm relu

- Dùng maxpooling layer2D kích thước 2×2 sau mỗi lớp conv.
- Sau đó là 1 lớp flatten
- Sử dụng 3 mạng Dense và 2 lớp Dropout tỉ lệ 0.25 để chống overfit.

Batch size = 32

Epochs = 10

Kết quả model:

Accuracy Train: 0.9021

Accuracy Test: 0.8923

Model 3:

- Các bước tiền xử lí:
  - Chuyển về ảnh xám.
  - Chuẩn hóa (normalize) giá trị các pixels về đoạn 0, 1.
  - Resize các tấm ảnh về cùng kích thước 64×64

Mạng CNN:

- 5 lớp convolution layers.
- 4 dropout layers.
- 2 maxpooling layers.
- 1 lớp Flatten
- 1 lớp Dense layers.

Batch size = 32

Epochs = 10

- Accuracy Train: 0.9228
- Accuracy Test: 0.9227
- Nhận xét: model đồ có độ chính xác cao, không bị overfiting nhưng những ký tự khá giống nhau như số 1 và số 7 có thể bị nhầm lẫn,...

1