TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN ĐIỆN

BỘ MÔN ĐIỀU KHIỂN TỰ ĐỘNG

--------oOo--------

ĐỒ ÁN THIẾT KẾ
HỆ THỐNG ĐIỀU KHIỂN TỰ ĐỘNG

ỨNG DỤNG MẠNG CONVOLUTIONAL NEURAL NETWORK TRONG NHẬN DẠNG HỆ THỐNG ĐIỀU KHIỂN

Giáo viên hướng dẫn : TS. Nguyễn Hoài Nam

Sinh viên thực hiện : Dương Bá Hải Đăng – 20141016

Đỗ Sơn Lâm – 20142474

Đinh Trung Kiên – 20142385

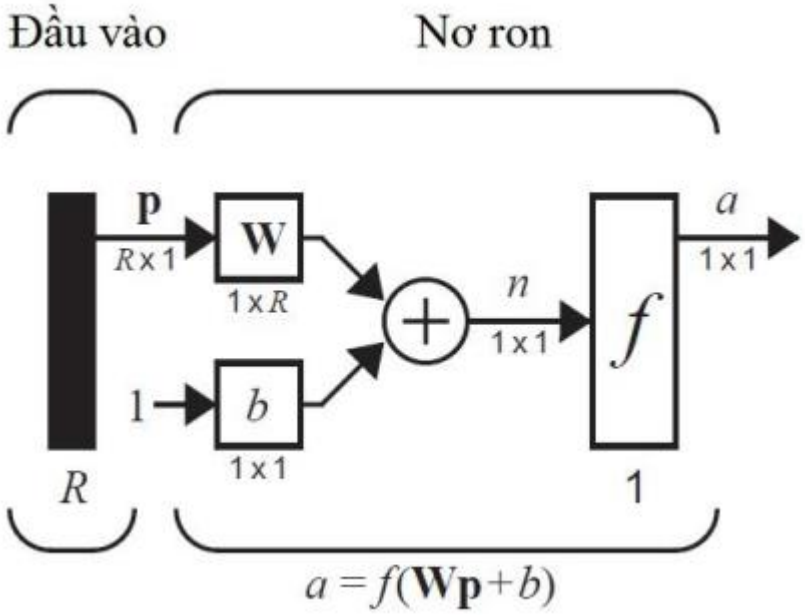
Mục lục

I. LÝ THUYẾT

1.1 Giới thiệu về mạng neural

Định nghĩa: Mạng nơron nhân tạo, Artificial Neural Network (ANN) là một mô hình xử lý thông tin phỏng theo cách thức xử lý thông tin của các hệ nơron sinh học. Nó được tạo nên từ một số lượng lớn các phần tử (nơron) kết nối với nhau thông qua các liên kết (trọng số liên kết) làm việc như một thể thống nhất để giải quyết một vấn đề cụ thể nào đó. Một mạng nơron nhân tạo được cấu hình cho một ứng dụng cụ thể (nhận dạng mẫu, phân loại dữ liệu) thông qua một quá trình học từ tập các mẫu huấn luyện. Về bản chất học chính là quá trình hiệu chỉnh trọng số liên kết giữa các nơron.

-Cấu trúc mạng Neural nhân tạo



Các thành phần cơ bản của 1 mạng Neural bao gồm:

+ Tập đầu vào: Là các tín hiệu vào (input signals) của nơron, các tín hiệu này thường được đưa vào dưới dạng một vector R chiều.
+ Tập các liên kết: Mỗi liên kết được thể hiện bởi một trọng số liên kết – Synaptic weight. Trọng số liên kết giữa tín hiệu vào thứ j với nơron k thường được kí hiệu là w_{kj} . Thông thường, các trọng số này được khởi tạo một cách

ngẫu nhiên ở thời điểm khởi tạo mạng và được cập nhật liên tục trong quá trình học mạng.

+ Bộ tổng (Summing function): Thường dùng để tính tổng của tích các đầu vào n với trọng số liên kết của nó.
+ Ngưỡng (còn gọi là một độ lệch - bias): Ngưỡng b này thường được đưa vào như một thành phần của hàm truyền.
+ Hàm truyền f (Transfer function): Hàm f này được dùng để giới hạn phạm vi đầu ra của mỗi nơron. Nó nhận đầu vào là kết quả của hàm tổng và ngưỡng.
+ Đầu ra a: Là tín hiệu đầu ra của một nơron, với mỗi nơron sẽ có tối đa là một đầu ra.

Hàm truyền

Tên hàm	Mô tả toán học	Ứng dụng
a = hardlim(n)	$a = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}$	Mạng Perceptron
a = purelin(n)	$a = n$	Mạng Adaline
a = logsig(n)	$a = \frac{1}{1+e^{-n}}$	Mạng nhiều lớp, thuật toán lan truyền ngược
a = tansig(n)	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$	Mạng nhiều lớp, thuật toán lan truyền ngược
a = poslin(n)	$a = \begin{cases} 0 & n < 0 \\ n & n \geq 0 \end{cases}$	Mạng Hamming
a = satlins(n)	$a = \begin{cases} -1 & n < -1 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$	Mạng Hopfield

Xét về mặt toán học , cấu trúc của một mạng neuron được biểu diễn bằng biểu thức sau:

$$a = f(\mathrm{W}p + b)$$

Trong đó $p = \begin{bmatrix} p_1 \\ p_2 \\ : \\ p_R \end{bmatrix}; \mathrm{W} = \begin{bmatrix} \mathrm{w}_{1,1} \\ \mathrm{w}_{1,2} \\ : \\ \mathrm{w}_{1,R} \end{bmatrix}$

Như vậy nơron nhân tạo nhận các tín hiệu đầu vào, xử lý (nhân các tín hiệu này với trọng số liên kết, tính tổng các tích thu đoợc rồi gửi kết quả tới hàm truyền), và cho một tín hiệu đầu ra (là kết quả của hàm truyền).

-Một số loại mạng neuron:

+ Mạng perceptron: Có thể giải quyết bài toán phân loại với đường biên tuyến tính
+ Mạng nhiều lớp (mở rộng của mạng perceptron): Có thể giải quyết bài toán phân loại bất kỳ, xấp xỉ một hàm phi tuyến bất kỳ.
+ Mạng neuron động: Nhận dạng và điều khiển các hệ thống động học và phi tuyến
+ Mạng nhớ: ứng dụng để phân loại mẫu(chữ viết,ảnh)
+ Mạng Adaline: ứng dụng như bộ lọc thích nghi
+ Mạng RBF: ứng dụng xấp xỉ hàm, phân loại mẫu

1.2. Giới thiệu mạng Convolutional Neural Network

1.2.1. Định nghĩa

Những năm gần đây, ta đã chứng kiến được nhiều thành tựu vượt bậc trong ngành Thị giác máy tính (Computer Vision). Các hệ thống xử lý ảnh lớn như Facebook, Google hay Amazon đã đưa vào sản phẩm của mình những chức năng thông minh như nhận diện khuôn mặt người dùng, phát triển xe hơi tự lái hay drone giao hàng tự động.

Với sự phát triển phần cứng mạnh mẽ cho phép tính toán song song hàng tỉ phép tính, tạo tiền đề cho Mạng nơ-ron tích chập trở nên phổ biến và đóng vai trò quan trọng trong sự phát triển của trí tuệ nhân tạo nói chung và xử lý ảnh nói riêng. Một trong các ứng dụng quan trọng của mạng nơ-ron tích chập đó là cho phép các máy tính có khả năng "nhìn" và "phân tích", nói 1 cách dễ hiểu, Convnets được sử dụng để nhận dạng hình ảnh bằng cách đưa nó qua nhiều layer với một bộ lọc tích

chập để sau cùng có được một điểm số nhận dạng đối tượng. CNN được lấy cảm hứng từ vỏ não thị giác.

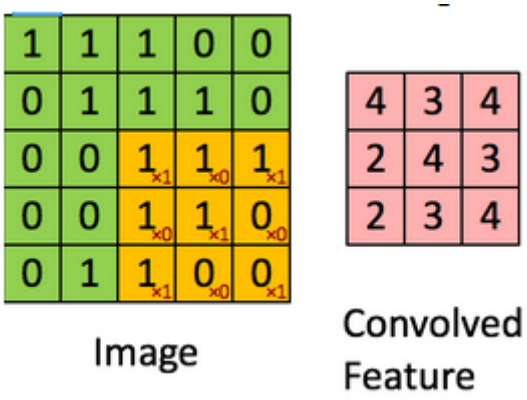
Mỗi khi chúng ta nhìn thấy một cái gì đó, một loạt các lớp tế bào thần kinh được kích hoạt, và mỗi lớp thần kinh sẽ phát hiện một tập hợp các đặc trưng như đường thẳng, cạnh, màu sắc,v.v.v của đối tượng. lớp thần kinh càng cao sẽ phát hiện các đặc trưng phức tạp hơn để nhận ra những gì chúng ta đã thấy.

Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay. Trong đồ án này, chúng ta sẽ trình bày về Convolution (tích chập) cũng như ý tưởng của mô hình CNNs trong phân lớp ảnh áp dụng trong bài toán nhận dạng hệ thống (Image Classification)

1.2.2. Convolution (tích chập)

Tích chập được sử dụng đầu tiên trong xử lý tín hiệu số (Signal processing). Nhờ vào nguyên lý biến đổi thông tin, các nhà khoa học đã áp dụng kĩ thuật này vào xử lý ảnh và video số.

Để dễ hình dung, ta có thể xem tích chập như một cửa sổ trượt (sliding window) áp đặt lên một ma trận. Ta có thể theo dõi cơ chế của tích chập qua hình minh họa bên dưới.



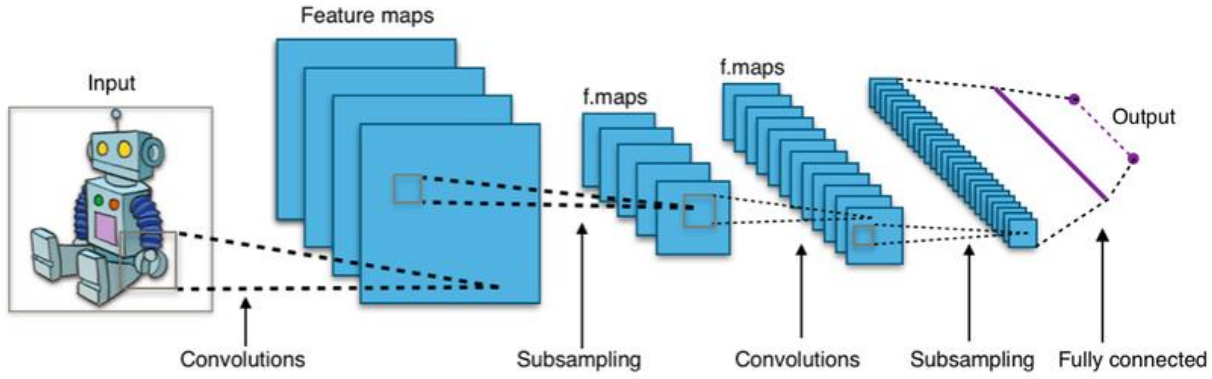
Các convolutional layer có các parameter(kernel) đã được học để tự điều chỉnh lấy ra những thông tin chính xác nhất mà không cần chọn các feature. Trong hình ảnh ví dụ trên, ma trận bên trái là một hình ảnh trắng đen được số hóa. Ma trận có kích thước 5x5 và mỗi điểm ảnh có giá trị 1 hoặc 0 là giao điểm của dòng và cột. Convolution hay tích chập là nhân từng phần tử trong ma trận 3.

Sliding Window hay còn gọi là kernel, filter hoặc feature detect là một ma trận có kích thước nhỏ như trong ví dụ trên là 3x3. Convolution hay tích chập là nhân từng phần tử bên trong ma trận 3x3 với ma trận bên trái. Kết quả được một ma trận gọi là Convoled feature được sinh ra từ việc nhận ma trận Filter với ma trận ảnh 5x5 bên trái.

1.2.3. Cấu trúc mạng CNN

Mạng CNN là một tập hợp các lớp Convolution chồng lên nhau và sử dụng các hàm nonlinear activation như ReLU và pooling (Subsampling layer) để kích hoạt các trọng số trong các node. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo.

Trong mô hình mạng truyền thẳng (feedforward neural network) thì mỗi neural đầu vào (input node) cho mỗi neural đầu ra trong các lớp tiếp theo. Mô hình này gọi là mạng kết nối đầy đủ (fully connected layer) hay mạng toàn vẹn (affine layer). Còn trong mô hình CNNs thì ngược lại. Các layer liên kết được với nhau thông qua cơ chế convolution. Layer tiếp theo là kết quả convolution từ layer trước đó, nhờ vậy mà ta có được các kết nối cục bộ. Như vậy mỗi neuron ở lớp kế tiếp sinh ra từ kết quả của filter áp đặt lên một vùng ảnh cục bộ của neuron trước đó.



a) Image input layer

Lớp Image input layer định dạng kích thước của các ảnh đầu vào của 1 mạng CNN. Sử dụng hàm imageInputLayer function. Kích thước của ảnh tương ứng với chiều cao, chiều rộng , và kênh màu của ảnh đó.

Ở bài này kích thước của ảnh đầu vào là [96x96x3] tương ứng với kích thước ảnh là 96 pixels x 96 pixel với kênh màu là 3 tức là ảnh màu.

b) Convolutional Layer

+ Một lớp Convolutional bao gồm các neurons kết nối các feature map của ảnh đầu vào hoặc đầu ra của lớp trước nó. Lớp này tìm hiểu các feature trong 1 mảng nhỏ (filter a pixels x a pixels) khi được quét trên ảnh đầu vào. Kích thước của filter này có thể được chỉnh sửa để phù hợp với kích thước của ảnh đầu vào sử dụng hàm filterSize .
+ Trong mỗi vùng, mỗi một kết nối sẽ học một trọng số và mỗi neuron ẩn sẽ học một bias sử dụng hàm trainNetwork. Mỗi một vùng đấy gọi là một trường tiếp nhận cục bộ. Filter di chuyển dọc từ trái sang phải hoặc từ trên xuống dưới dọc theo ảnh đầu vào. Bước di chuyển được gọi là Stride. Vì vậy số lượng vùng quét có thể áp đặt được thông qua filterSize và Stride.
+ Số trọng số được sử dụng cho 1 filter là h*w*c lần lượt là chiều dài, rộng, số kênh màu của filter.Số filter sẽ quết định số kênh đầu ra của 1 lớp convolutional.
+ Khi 1 filter đi dọc theo ảnh nó sử dụng trọng số và bias cố định tạo ra 1 feature map. Vì vậy số feature maps (M) của 1 lớp convolutional sẽ bằng với số filters. Mỗi feature map có trọng số và bias riêng. Vậy tổng số thông số trong 1 lớp convolutional là ((h*w*c+1)*Number of filters) trong đó bias=1.
+ Kích thước đầu ra(chiều dài và rộng) của 1 lớp convolutional được tính theo công thức:

  Output size= (input size-filter size+2*Padding)/Stride +1 .

  Trong đó input size, filter size là kích thước của đầu vào và filter, Stride là bước nhảy của filter, Padding là số hang hoặc cột thêm vào rìa của ảnh đầu vào để điều chỉnh kích thước ảnh đầu ra. Output size phải là số nguyên dương để đảm bảo tất cả các phần của ảnh đều được xét đến trong lớp convolutional.
+ Số lượng neurons trong 1 lớp convolutional:

  Tổng số neurons bằng (Map size* Number of Filters)

  Trong đó Map size là chiều dài*rộng của output size.

c) Batch Normalization Layer

Lớp Batch Normalization được sử dụng giữa lớp convolutional và các hàm nonlinear activation như ReLU để tang tốc độ huấn luyện và giảm độ nhạy cảm. Để tạo 1 lớp Batch Normalization sử dụng hàm **batchNormalizationLayer**

d) ReLU Layer

Lớp ReLU làm cho với mỗi thành phần đầu vào mà giá trị nhỏ hơn 0 thì được đặt bằng 0.

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

ReLU được chứng minh giúp cho việc training các Deep Networks nhanh hơn rất nhiều. Sự tăng tốc này được cho là vì ReLU được tính toán gần như tức thời và gradient của nó cũng được tính cực nhanh với gradient bằng 1 nếu đầu vào lớn hơn 0, bằng 0 nếu đầu vào nhỏ hơn 0. Loại bỏ các giá trị âm của đầu vào và set về 0.

Tạo ra 1 lớp ReLU sử dụng hàm **reluLayer**

e) Max-Pooling Layer

Mục đích của pooling rất đơn giản, nó làm giảm số hyperparameter mà ta cần phải tính toán, từ đó giảm thời gian tính toán, tránh overfitting. Loại pooling ta thường gặp nhất là max pooling, lấy giá trị lớn nhất trong một pooling window. Pooling hoạt động gần giống với convolution, nó cũng có 1 cửa sổ trượt gọi là pooling window, cửa sổ này trượt qua từng giá trị của ma trận dữ liệu đầu vào (thường là các feature map trong convolutional layer), chọn ra một giá trị từ các gía trị nằm trong cửa sổ trượt (với max pooling ta sẽ lấy giá trị lớn nhất).Thường lớp này ta sẽ chọn pooling window trượt dọc theo đầu vào mà không bị trùng. Có nghĩa là bước nhảy sẽ bằng kích thước của window.Nếu đầu vào của lớp Pooling là h x h và pooling window là t x t thì đầu ra của lớp pooling của 1 kênh convolutional là h/x x h/x .Ta có thể sử dụng pooling window trượt tương tự như filter của lớp convolutional và output size cũng sẽ giống với công thức:

$$\frac{(inputsize - poolingwindow + 2 \times Padding)}{Stride}$$

Sử dụng hàm **maxPooling2dLayer**

f) Fully connected Layer

Thường thì sau các lớp Conv+Pooling thì sẽ là 2 lớp Fully connected, một layer để tập hợp các tất cả các feature layer mà đã được học ở các lớp trước, chuyển đổi dữ liệu từ 3-D, hoặc 2-D thành 1-D, tức chỉ còn là 1 vector. Lớp cuối cùng này tổng hợp tất cả các feature để nhận dạng các loại ảnh. Số neuron của layer này phụ thuộc vào số output mà ta muốn tìm ra. Giả sử với tập dữ liêu ta đang xét chẳng hạn, ta có tập 10 đối tượng. Vậy output sẽ có số neurons là 10. Để tạo 1 lớp fully Connected dung hàm **fullyConnectedLayer**

g) Output Layer

Softmax and Classification Layers:

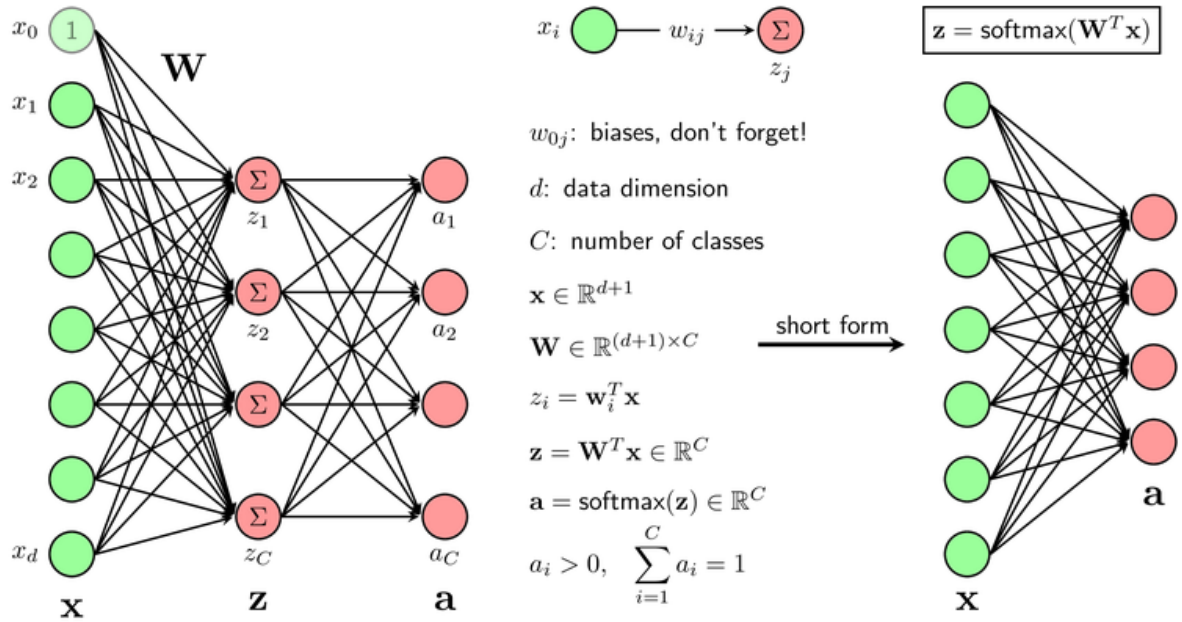
- Softmax function:

$$a_r = P\left(c_r \middle| x,\theta\right) = \frac{P\left(x,\theta \middle| c_r\right)P\left(c_r\right)}{\sum_{j=1}^{k} P\left(x,\theta \middle| c_j\right)P\left(c_j\right)} = \frac{\exp\left(z_r\left(x,\theta\right)\right)}{\sum_{j=1}^{k}\exp\left(z_j\left(x,\theta\right)\right)}$$

Với $0 \le P\left(c_r \middle| x,\theta\right) \le 1, \sum_{j=1}^{k} P\left(c_j \middle| x,\theta\right) = 1$

Với mỗi input x, $P\left(c_r \middle| x,\theta\right)$ thể hiện xác xuất để input đó rơi vào class r nếu biết tham số của mô hình.

Hình vẽ dưới đây thể hiện hàm softmax dưới dạng neural network



- Classification layer thường đi sau softmax layer. Ở lớp này giá trị từ hàm softmax được lấy ra và chỉ định cho mỗi input vào 1 trong k classes bằng việc sử dụng hàm entropy:

$$E(0) = -\sum_{i=1}^{n}\sum_{j=1}^{k} t_{ij} \ln y_j(x_i, \theta)$$

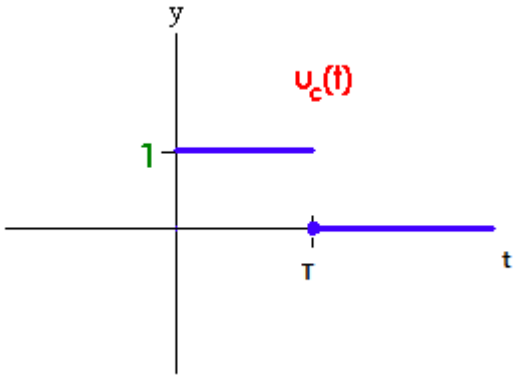
Trong đó t_{ij} là chỉ số chỉ ra rằng mẫu i thuộc về class j, $y_j(x_i, \theta)$ là đầu ra của mẫu i được lấy từ hàm softmax.

II. ỨNG DỤNG MẠNG CNN VÀO NHẬN DẠNG HỆ THỐNG TRONG ĐIỀU KHIỂN

2.1. Dữ liệu huấn luyện

- Dữ liệu training là ảnh các đáp ứng đầu ra của đối tượng khi có đầu vào là hàm Step

Function : $u_T(t) = \begin{cases} 1 & 0 \le t < T \\ 0 & t \ge T \end{cases}$



Ở đây ta chọn T=500 đủ lớn để tín hiệu ra đạt gần giá trị t vô cùng

- Các bộ mẫu bao gồm:

Khâu tích phân bậc nhất: $G(s) = \dfrac{k}{1+Ts}$

Khâu quán tính bậc hai: $G(s) = \dfrac{k}{(1+T_1 s)(1+T_2 s)}$ $\quad T_1 \neq T_2$

Khâu dao động bậc hai: $G(s) = \dfrac{k}{1+2DTs+T^2 s^2}$ $\quad 0 < D < 1$

Khâu quán tính bậc ba: $G(s) = \dfrac{k}{(1+T_1 s)(1+T_2 s)(1+T_3 s)}$ $\quad T_1 \neq T_2 \neq T_3$

Khâu dao động bậc ba: $G(s) = \dfrac{k}{(1+2DTs+T^2 s^2)(T_3 s+1)}$ $\quad 0 < D < 1$

Và các khâu trên có thêm thành phần trễ

- Ảnh tín hiệu ra thu được khi tín hiệu vào của hệ thống là hàm step function sẽ được xử lý lại:

+ Thời gian mô phỏng là 2T với đồ thị nửa thời gian T cuối được vẽ ngược lại với nửa thời gian T đầu.

+ Sau đó đồ thị được thu gọn theo 2 trục từ dải ban đầu về dải [0,1]

+ Ảnh sau khi export sẽ ở dưới dạng đuôi ‘png’ với kích thước 96x96x3

- Dữ liệu mẫu được tạo cho mỗi thư mục là 2500 ảnh mẫu.

2.2. Huấn luyện mạng CNN:

+ Ta sử dụng mạng neural bao gồm 15 lớp neurons :

```
- layers = [
- imageInputLayer([96 96 3])
- convolution2dLayer(6,32,'Padding',0,'Stride',2)
- batchNormalizationLayer
- reluLayer
- maxPooling2dLayer(2,'Stride',2)
-   convolution2dLayer(4,16,'Padding',0,'Stride',2)
-   batchNormalizationLayer
-   reluLayer
-   maxPooling2dLayer(2,'Stride',2)
-   convolution2dLayer(2,8,'Padding',1)
-   batchNormalizationLayer
-   reluLayer
-   fullyConnectedLayer(10)
-   softmaxLayer
-   classificationLayer];
```

+ Dữ liệu đầu vào là ảnh 96 pixels x 96 pixels x 3(ảnh màu)

+ <u>Lớp convolutional thứ nhất</u>

convolution2dLayer(6,32,'Padding',0,'Stride',2)

với 32 filters kích thước 6x6 , bước nhảy quét Stride=2 và không thêm Padding

-Như vậy số trọng số của 1 filter là 6x6x3=108. Số feature map = 32.

Tổng số trọng số của 1 lớp convolutional là $(6\times6\times3+1)\times32=3456$

- Output size

$$\frac{(inputsize-filtersize+2\times Padding)}{Stride}+1=\frac{(96-6+2\times0)}{2}+1=46$$

ảnh đầu ra là 46x46x3

- Số neurons của lớp convolutional 1 là $46\times46\times32=67712$ neurons
- Tiếp theo là 2 lớp batchNormalizationLayer và reluLayer để tăng tốc độ train và loại bỏ các trọng số âm.
- Sử dụng hàm maxPooling2dLayer(2,'Stride',2) lớp này sẽ quét ảnh với filter kích thước 2x2 và bước nhảy là 2 để tránh bị trùng. Sau đó trong các trọng số ở trong vùng bị quét sẽ chỉ lấy trọng số lớn nhất. Đầu ra của lớp này của 1 kênh lớp convolutional sẽ là 23x23.

+ Tương tự với lớp convolutional thứ 2 ta có:

convolution2dLayer(4,16,'Padding',0,'Stride',1)

- Đầu vào có kích thước 23x23
- Số filters là 16, filter size 4x4, bước nhảy 1 và không thêm Padding.
- Số trọng số của 1 filter = 4x4x3=48, Tổng số trọng số = (4x4x3+1)x16=784
- Output size= $(23-4+2\times0)+1=20$
- Số neurons = 20x20x16=6400
- Đầu ra sau lớp maxpooling là 10x10

+ Lớp convolutional cuối cùng:

convolution2dLayer(2,8,'Padding',0, 'Stride',1)

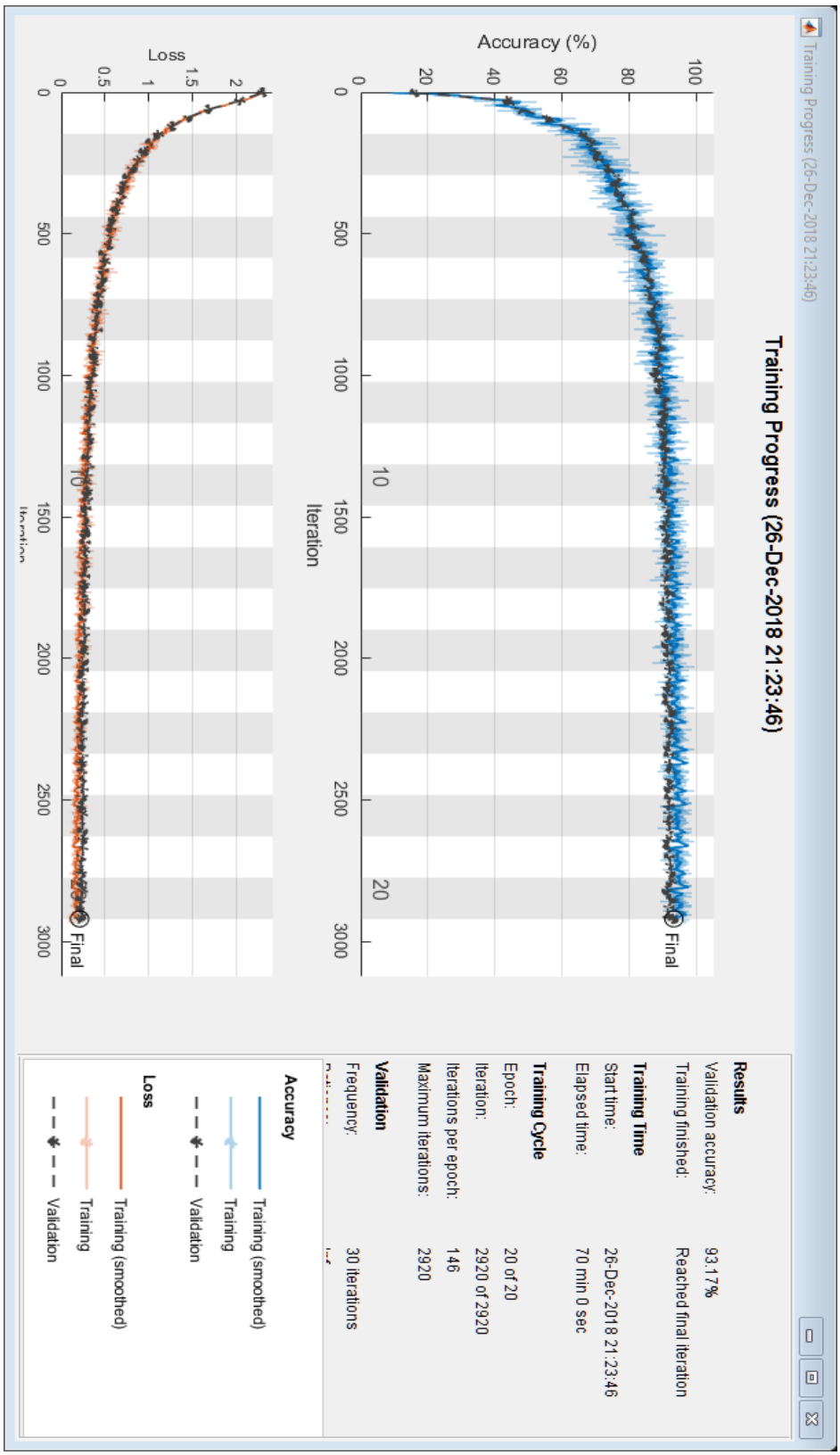
- Đầu vào có kích thước 10x10
- Số filters là 8, filter size 2x2, bước nhảy 1 và không thêm Padding.
- Số trọng số của 1 filter = 2x2x3=12, Tổng số trọng số =(12+1)x8=104
- Output size = 9
- Số neurons = 9x9x8=648
- Lớp fullyConnectedLayer(10) bao gồm 10 neurons bằng với số loại mẫu cần phân loại.

❖ Dữ liệu đầu vào mỗi loại 2500 ảnh được chia 75% cho việc training và 25% test với số epochs max là 20, tốc độ học là 10^-3

```
options = trainingOptions('sgdm', ...
 'MaxEpochs',60, ...
 'InitialLearnRate',1e-3, ...
 'ValidationData',imdsValidation, ...
 'ValidationFrequency',30, ...
```

```
    'Verbose',false, ...
    'Plots','training-progress');
```

2.3. Kết quả quá trình huấn luyện



Ta thấy độ chính xác của mạng neural đạt được là 93.17% sau 20 epochs huấn luyện.

Các classes để phân loại bao gồm:

-First-Order System

-First-Order System with delay

-Overdamped Second-Order System

-Overdamped Second-Order System with delay

-Underdamped Second-Order System

-Underdamped Second-Order System with delay

-Third-Order System

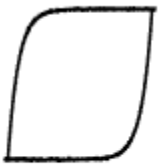
-Third-Order System with delay

-Fluctuated Third-Order System

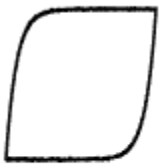
-Fluctuated Third-Order System with delay

Một số mẫu mới được đưa vào mạng neural vừa huấn luyện để kiểm tra:

First-Order System with delay



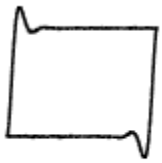
First-Order System



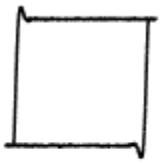
Overdamped Second-Order System

Overdamped Second-Order System with delay

Underdamped Second-Order System



Underdamped Second-Order System with delay



Fluctuated Third-Order System

Fluctuated Third-Order System with delay

Third-Order System

Third-Order System with delay

III. KẾT LUẬN

- Nhận xét: Mạng neural nhận dạng được tốt các hệ thống. Nhưng đối với đối tượng có thêm thành phần trễ nếu delay ít hoặc rất ít thì khó nhận dạng được.
- Hướng phát triển : Dựa vào mạng neural nhận được ta có thể xác định được dạng hàm truyền của đối tượng. Từ đó có thể xác định thông số của hàm truyền đối tượng một cách dễ dàng hơn.

Tài liệu tham khảo

1. Mark Hudson Beale, M. T. (n.d.). *Neural Network Toolbox™ User's Guide.*

T.Hagan, M. (n.d.). *Neural Network Design.*