ĐẶT VẤN ĐỀ

Hiện nay rất nhiều bài toán nhận dạng sử dụng học sâu để giải quyết do học sâu có thể giải quyết các bài toán với số lượng lớn, kích thước đầu vào lớn với hiệu năng cũng như độ chính xác vượt trội so với các phương pháp phân lớp truyền thống.

Mạng nơ-ron tích chập (Convolutional Neural Networks - CNN) là một trong những mô hình học sâu 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 tốt nghiệp này, chúng em đi vào nghiên cứu về mạng neural cũng như các mạng Tích chập (Convolution) và mạng Hồi quy để áp dụng trong việc nhận dạng hệ thống và nhận dạng giọng nói.

Nội dung đồ án bao gồm 3 chương chính:

- Chương 1: Cơ sở lý thuyết về mạng nơ-ron và mạng nơ-ron tích chập
- Chương 2: Ứng dụng mạng CNN và Hồi quy vào nhận dạng hệ thống
- Chương 3: Ứng dụng mạng CNN vào nhận dạng giọng nói

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

CNN: Mạng nơ-ron tích chập (Convolutional Neural Network)

TDL: Khối trễ

ARMA: Autoegressive Moving Average Model

$u(t)$: Tín hiệu vào

$y(t)$: Tín hiệu ra

$G(z)$: Hàm truyền dạng không liên tục

$G(s)$: Hàm truyền dạng liên tục

E: Hàm entropy

$IW^{1,1}$: Ma trận trọng số đầu vào thứ nhất của lớp neural thứ nhất

$LW^{1,1}$: Ma trận trọng số phản hồi từ đầu ra về lớp neural thứ nhất

DFT: Biến đổi Fourier rời rạc (Discrete Fourier Transform)

FFT: Biến đổi Fourier nhanh (Fast Fourier Transform)

$\omega(n)$: Hàm cửa sổ

$H_m(k)$: Bộ lọc thứ $m-1$ trong ngân hàng bộ lọc

DANH MỤC BẢNG, BIỂU

DANH MỤC HÌNH VẼ, ĐỒ THỊ

LỜI MỞ ĐẦU

Trong 5 năm học tập tại trường *Đại học Bách Khoa Hà Nội*, nhà trường và thầy cô không chỉ truyền đạt cho chúng em những kiến thức chuyên môn về ngành mà còn giáo dục cho chúng em về lý tưởng đạo đức trong cuộc sống. Đây là những hành trang không thể thiếu cho cuộc sống và sự nghiệp. Chúng em xin bày tỏ lòng biết ơn sâu sắc đến quý thầy cô trong *Khoa Điện* đặc biệt là các thầy cô trong bộ môn *Điều khiển tự động* đã tận tình chỉ bảo, dẫn dắt chúng em đến ngày hôm nay.

Riêng đối với đồ án này, chúng em xin bày tỏ lòng biết ơn sâu sắc đến Thầy *TS. Nguyễn Hoài Nam* là giáo viên hướng dẫn đã tận tình chỉ bảo và hướng dẫn, cũng như tạo mọi điều kiện hỗ trợ trong quá trình thực hiện. Do thời gian làm đồ án hạn chế nên không tránh khỏi những thiếu sót chúng em kính mong quý thầy cô tận tình chỉ dẫn thêm.

Chúng em xin chân thành cảm ơn!

TP.Hà Nội, ngày 01 tháng 06 năm 2019

Nhóm sinh viên thực hiện

Đỗ Sơn Lâm

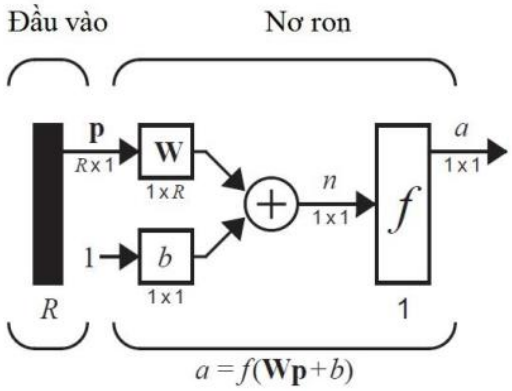
Dương Bá Hải Đăng

CHƯƠNG 1. CƠ SỞ 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



Hình 1.1. Mạng neural 1 lớp [7]

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.

Bảng 1.1. Hàm truyền [1]

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:

$$a = f(\mathrm{W}p + b) \tag{1.1}$$

$$\text{Trong đó } p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix}; \mathrm{W} = \begin{bmatrix} \mathrm{w}_{1,1} \\ \mathrm{w}_{1,2} \\ \vdots \\ \mathrm{w}_{1,R} \end{bmatrix} \tag{1.2}$$

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 được 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

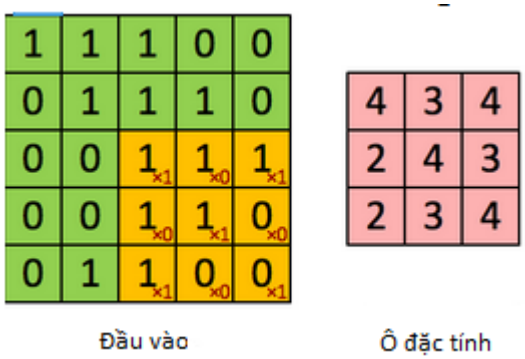
thông minh như nhận diện khuôn mặt người dùng, phát triển xe hơi tự lái hay drone giao hàng tự động.

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 bài báo cáo này, 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) [5]

1.2.2. Tích chập (Convolution)

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.



Hình 1.2. Hình minh họa tích chập

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.

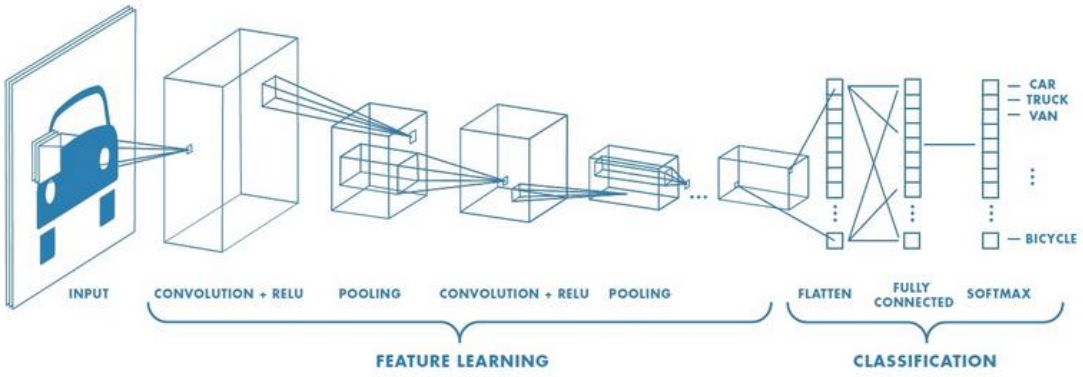
Sliding Window (hay còn gọi là kernel, filter hoặc feature detect) là một ma trận có kích thước nhỏ như trong ví dụ trên là 3x3. Convolution hay tích chập là nhân từng phần tử bên trong ma trận 3x3 với ma trận bên trái. Kết quả được một ma trận gọi là Convoled feature được sinh ra từ việc nhận ma trận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



Hình 1.3. Các layer mạng CNN [11]

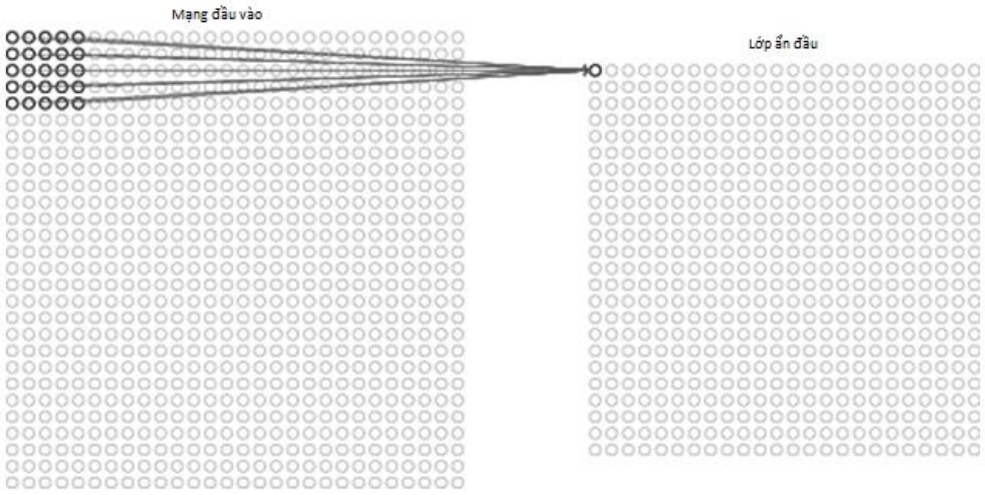
a) Lớp ảnh đầu vào

Lớp ảnh đầu vào (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) Lớp tích chập

Một lớp tích chập (Convolutional layer) 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.



Hình 1.4. Bộ lọc 5x5 đi qua lớp ảnh đầu vào để ra lớp mới

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:

$$OutputSize = \frac{InputSize - FilterSize + 2 * Padding}{Stride} + 1 \qquad (1.3)$$

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:

$$\sum neurons = MapSize * NumberOfFilters \tag{1.4}$$

Trong đó $MapSize = Width * Height$ (của $OutputSize$) (1.5)

c) Lớp chuẩn hóa

Lớp chuẩn hóa (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) Lớp kích hoạt

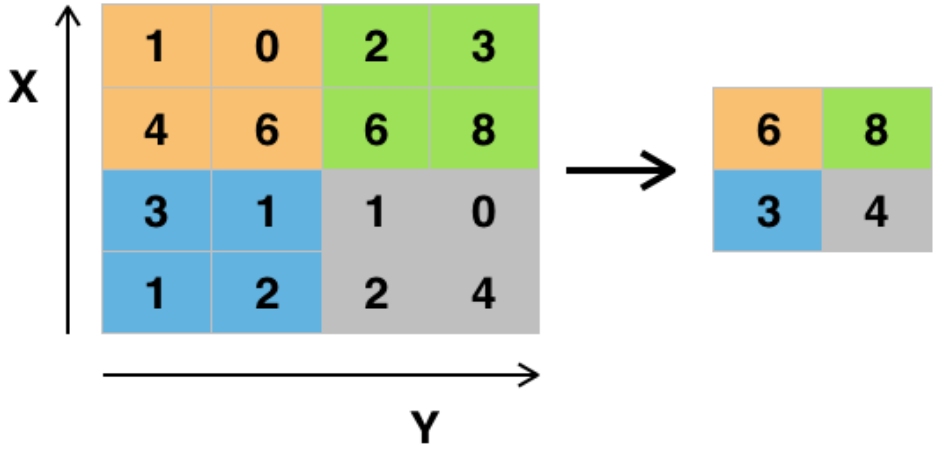
Sau lớp Batch Normalization, ta sử dụng hàm kích hoạt ReLU (Rectified Linear Units). Lớp ReLU làm cho với mỗi thành phần đầu vào mà giá trị nhỏ hơn 0 thì được đặt bằng 0.

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{1.6}$$

Trước đây, các nhóm nghiên cứu khác thường sử dụng hàm kích hoạt là hàm Tanh hoặc hàm Sigmoid để huấn luyện mô hình neural network. 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) Lớp gộp



Hình 1.5. Hoạt động của lớp gộp lấy giá trị lớn nhất

Mục đích của gộp rất đơn giản, nó làm giảm số thông số 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).

Overlapping Max-pooling layer là một loại Max-pooling layer, nhưng một window của bước này sẽ có một phần chồng lên window của bước tiếp theo. Trong trường hợp cụ thể chúng ta đang xét tới ở bài toán này, ta sử dụng pooling với kích thước là 3x3 và bước nhảy là 2 giữa các pooling. Nghĩa là giữa pooling này và pooling khác sẽ trùng nhau 1 pixel. Các thí nghiệm thực tế đã chứng minh rằng việc sử dụng overlapping giữa các pooling giúp giảm độ lỗi top-1 error 0.4% và top-5 error là 0.3% khi so với việc sử dụng pooling có kích thước 2x2 và bước nhảy 2. Công thức tính kích thước đầu ra của Pooling layer tương tự như với Convolutional layer:

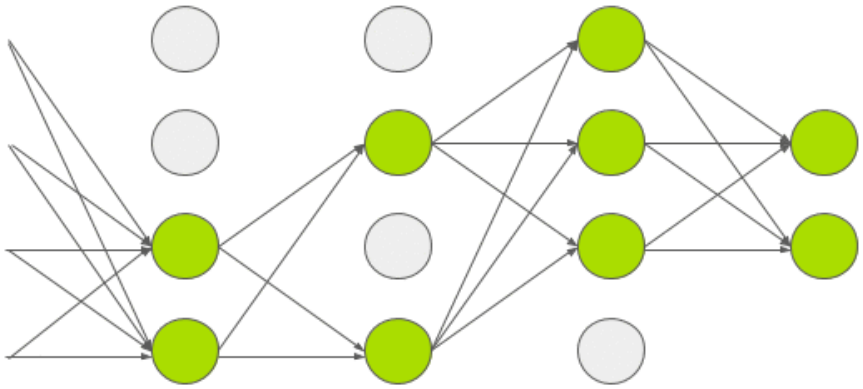
$$OutputSize = \frac{InputSize - Poolingwindow + 2 * Padding}{Stride} + 1 \qquad (1.7)$$

Sử dụng hàm ***maxPooling2dLayer***

f) Kĩ thuật loại bỏ (Dropout)

Neural network chính bản thân nó có khả năng học được những gì được dạy, tuy nghiên, nếu quá trình huấn luyện không tốt, mô hình có khả năng sẽ chỉ nhận dạng được tốt những gì chúng đã được học, có khi chính xác đến 100%, nhưng với những thứ ta cho ngoài phần đã dạy thì độ chính xác bị giảm mạnh. Và kết quả Neural network sẽ hoạt động tốt trên tập huấn luyện, nhưng chúng không rút ra được bản chất chính của vấn đề, và kết quả trên tập kiểm tra (test) sẽ rất kém. Người ta gọi trường hợp này là *overfitting*.

Để tránh xảy ra overfitting, có một kĩ thuật ta có thể sử dụng được gọi là *dropout* (loại bỏ). Kĩ thuật này khá đơn giản, sẽ có một xác suất α mà một neural sẽ bị loại bỏ khỏi mô hình. Nếu một neural bị loại khỏi mô hình, nó sẽ không được tham gia vào quá trình lan truyền tiến hoặc lan truyền ngược. Cho nên mỗi giá trị input sẽ đi qua một kiến trúc mạng khác nhau, kết quả là giá trị của tham số trọng số sẽ tốt hơn và khó bị overfitting hơn. Trong quá trình test, toàn bộ network được sử dụng, tuy nhiên giá trị output sẽ được nhân với tham số α tương ứng với những neural đã bị loại bỏ trong quá trình training.



Hình 1.6. Dropout

g) Lớp tổng hợp

Thường thì sau các lớp Conv+Pooling thì sẽ là 2 lớp tổng hợp (Fully connected layer), 1 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 dùng hàm ***fullyConnectedLayer***

h) Lớp đầu ra

Softmax and Classification Layers:

- Softmax function :

$$a_r = P\left(c_r \middle| x,\theta\right) = \frac{P\left(x,\theta \middle| c_r\right)P\left(c_r\right)}{\sum_{j=1}^{k} P\left(x,\theta \middle| c_j\right)P\left(c_j\right)} = \frac{\exp\left(z_r\left(x,\theta\right)\right)}{\sum_{j=1}^{k}\exp\left(z_j\left(x,\theta\right)\right)} \tag{1.8}$$

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.

- 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:

$$E\left(0\right) = -\sum_{i=1}^{n}\sum_{j=1}^{k} t_{\mathrm{ij}} \ln y_j\left(x_i,\theta\right) \tag{1.9}$$

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

CHƯƠNG 2. ỨNG DỤNG MẠNG CNN VÀ HỒI QUY VÀO NHẬN DẠNG HỆ THỐNG

Trong rất nhiều trường hợp, sự hiểu biết về những quy luật giao tiếp bên trong hệ thống và bên ngoài đối với hệ thống chưa được rõ ràng. Để xây dựng được mô hình toán học của hệ thống một cách hoàn chỉnh theo ta sẽ sử dụng phường pháp thực nghiệm. Đây là phương pháp xác định mô hình toán học trên cơ sở quan sát tín hiệu vào *u(t)* và ra *y(t)* của hệ thống. Trong đó tín hiệu vào *u(t)* được chủ động chọn trước và chỉ cần quan sát tín hiệu ra *y(t)* gọi là phương pháp thực nghiệm chủ động. [5]

Ở đây ta sẽ xét bài toán thực nghiệm chủ động xác định hàm truyền đạt G(s) được chia thành 2 bước chính sau:

- *Phần 1*: Nhận dạng bậc của mô hình sử dụng mạng Convolutional Neural Network thông qua bài toán phân loại ảnh của đầu ra *y(t)* đã được chuẩn hóa với đầu vào là hàm dạng xung đơn vị:

$$u_T(t) = \begin{cases} 1 & 0 \le t < T_m \\ 0 & t \ge T_m \end{cases} \tag{2.1}$$

với T_m đủ lớn để *y* có thể đạt giá trị xác lập.

- *Phần 2*: Xác định tham số hàm truyền dạng không liên tục G(z) sử dụng mạng nơ-ron hồi quy với tín hiệu ra $y(t) = h(t)$ đã quan sát được từ đầu ra của hệ thống khi tín hiệu vào của nó là hàm $u(t) = 1(t)$ hoặc dạng khác.

Ở chương này ta sẽ nhận dạng 10 đối tượng có hàm truyền đạt dạng như sau:

Khâu quán tính bậc nhất: $G(s) = \dfrac{k}{1+Ts}$ (2.2)

Khâu quán tính bậc hai: $G(s) = \dfrac{k}{(1+T_1 s)(1+\mathrm{T}_2 s)} \qquad T_1 \neq T_2$ (2.3)

Khâu dao động bậc hai: $G(s) = \dfrac{k}{1 + 2DTs + \mathrm{T}^2 s^2} \qquad 0 < D < 1$ (2.4)

Khâu quán tính bậc ba:

$$G(s) = \frac{k}{(1 + T_1 s)(1 + \mathrm{T}_2 s)(1 + \mathrm{T}_3 s)} \quad T_1 \neq T_2 \neq T_3 \tag{2.5}$$

Khâu dao động bậc ba:

$$G(s) = \frac{k}{(1 + 2DTs + \mathrm{T}^2 s^2)(T_3 s + 1)} \qquad 0 < D < 1 \tag{2.6}$$

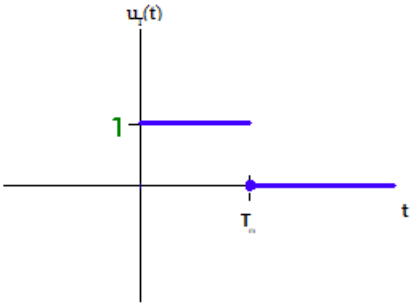
Và các khâu trên có thêm thành phần trễ: $e^{-s\tau}$

2.1. Xác định bậc hệ thống bằng mạng CNN

2.1.1. Dữ liệu huấn luyện

Dữ liệu huấn luyện là tập ảnh các đáp ứng đầu ra của đối tượng khi có đầu vào là hàm xung đơn vị:

$$u_T(t) = \begin{cases} 1 & 0 \le t < T_m \\ 0 & t \ge T_m \end{cases} \tag{2.7}$$



Hình 2.1. Tín hiệu xung đơn vị

Ở đây ta chọn T đủ lớn để tín hiệu ra đạt giá trị xác lập.

Các bộ mẫu bao gồm:

Khâu quán tính bậc nhất: $G(s) = \frac{k}{1+Ts}$ (2.8)

Khâu quán tính bậc hai: $G(s) = \frac{k}{(1+T_1 s)(1+\mathrm{T}_2 s)}$ $T_1 \neq T_2$ (2.9)

Khâu dao động bậc hai: $G(s) = \frac{k}{1+2DTs+\mathrm{T}^2 s^2}$ $0 < D < 1$ (2.10)

Khâu quán tính bậc ba:

$$G(s) = \frac{k}{(1+T_1 s)(1+\mathrm{T}_2 s)(1+\mathrm{T}_3 s)} \qquad T_1 \neq T_2 \neq T_3 \qquad (2.11)$$

Khâu dao động bậc ba:

$$G(s) = \frac{k}{(1+2DTs+\mathrm{T}^2 s^2)(T_3 s+1)} \qquad 0 < D < 1 \qquad (2.12)$$

Và các khâu trên có thêm thành phần trễ

Ảnh tín hiệu ra thu được khi tín hiệu vào của hệ thống là hàm xung chữ nhật đơn vị 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]

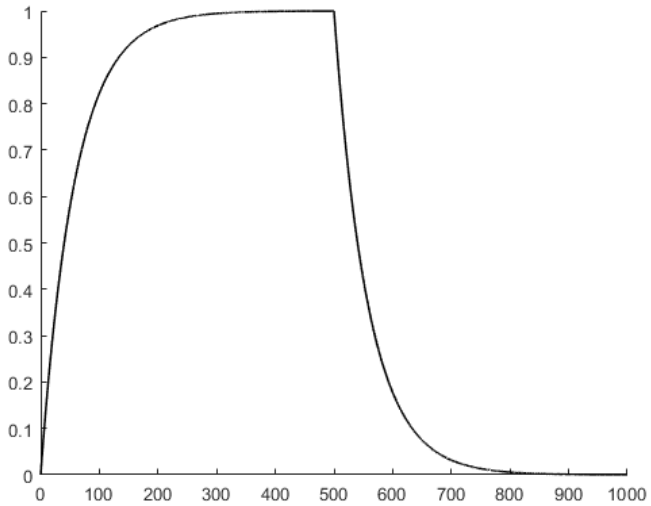
+ Đồ thị sau đó được lưu dưới dạng file có đuôi là ‘*.png’ với kích thước 96x96x3

+ Dữ liệu mẫu được tạo cho mỗi loại đối tượng là 2500 ảnh mẫu, với 80% dùng làm tập dữ liệu huấn luyện, 20% dùng làm tập xác nhận mạng trong khi huấn luyện.

+ Dải giá trị các thông số chạy mô phỏng:

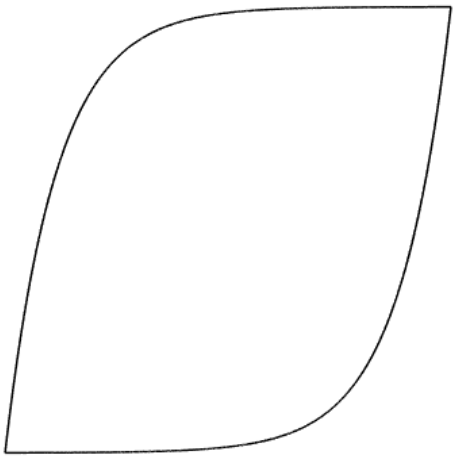
$$T_{\max} = 500; \tau \in [0;70]; k \in [0;99]; T_1, T_2, T_3 \in [1;40]; D \in (0;1)$$

Ví dụ tín hiệu ra:



Hình 2.2. Tín hiệu ra hàm bậc 1

Ảnh sau khi được xử lý có dạng như sau:



Hình 2.3. Ảnh đầu vào mạng CNN sau khi được xử lý

2.1.2. Thiết kế và huấn luyện mạng CNN

Chi tiết các lớp mạng nơ-ron được ghi ở phụ lục [1]

Dữ liệu đầu vào là ảnh 96 pixels x 96 pixels x 3 (ảnh màu). Dữ liệu đầu vào mỗi loại 2500 ảnh được chia 80% cho việc huấn luyện và 20% cho việc xác nhận mạng trong khi huấn luyện với số chu kì tối đa là 30, tốc độ học là 3e-5, sử dụng phương pháp tối ưu Adam.

- Lớp tích chập thứ nhất:

- 8 filter với kích thước 3x3, Stride = 1, Padding được tự chọn sao cho $OutputSize = InputSize$ (với trường hợp này thì Padding = 1).

- Như vậy số trọng số của 1 filter là $3\times3\times3=27$

=> Tổng số trọng số của 1 lớp convolutional là $(3\times3\times3+1)\times8=224$

$$OutputSize=\frac{(InputSize-\ filtersize+2\times Padding)}{Stride}+1=\frac{(96-3+2\times1)}{1}+1=96$$

- Số neurons của lớp convolutional 1 là $96\times96\times8=73728$ neurons

- Tiếp theo là 2 lớp batchNormalizationLayer và reluLayer để tăng tốc độ train và loại bỏ các trọng số âm.

- Lớp gộp đầu tiên:

- Ta sử dụng pooling window với kích thước là 3x3, Stride = 2, Padding được chọn sao cho $OutputSize = InputSize$ (ở đây thì Padding = 1).

- Khi đó thì $OutputSize=96$.

- Lớp tích chập thứ 2:

- Đầu vào có kích thước 96x96

- Số filters là 16, filter size 3x3, Stride = 1, Padding = 1.

- Số trọng số của 1 filter: $3\times3\times3=27$

- Tổng số trọng số của 1 lớp convolutional: $(3\times3\times3+1)\times16=448$

- Output size: $\frac{96-3+2\times1}{1}+1=96$

- Tổng số neurals: $96\times96\times16=147456$

- Lớp gộp thứ 2:

- Tương tự như lớp đầu tiên, ta sử dụng pooling window với kích thước là 3x3, Stride = 2, Padding được chọn sao cho $OutputSize = InputSize$ (ở đây thì Padding = 1).

- Khi đó thì $OutputSize = 96$.

- Lớp tích chập cuối cùng:

- Đầu vào có kích thước 96×96

- Số filters là 32, filter size 3x3, Stride = 1, Padding = 1.

- Số trọng số của 1 filter: $3\times3\times3=27$

- Tổng số trọng số của 1 lớp convolutional: $(3\times3\times3+1)\times32=896$

- Output size: $\frac{96-3+2\times1}{1}+1=96$

- Tổng số neurals: $96\times96\times32=294912$

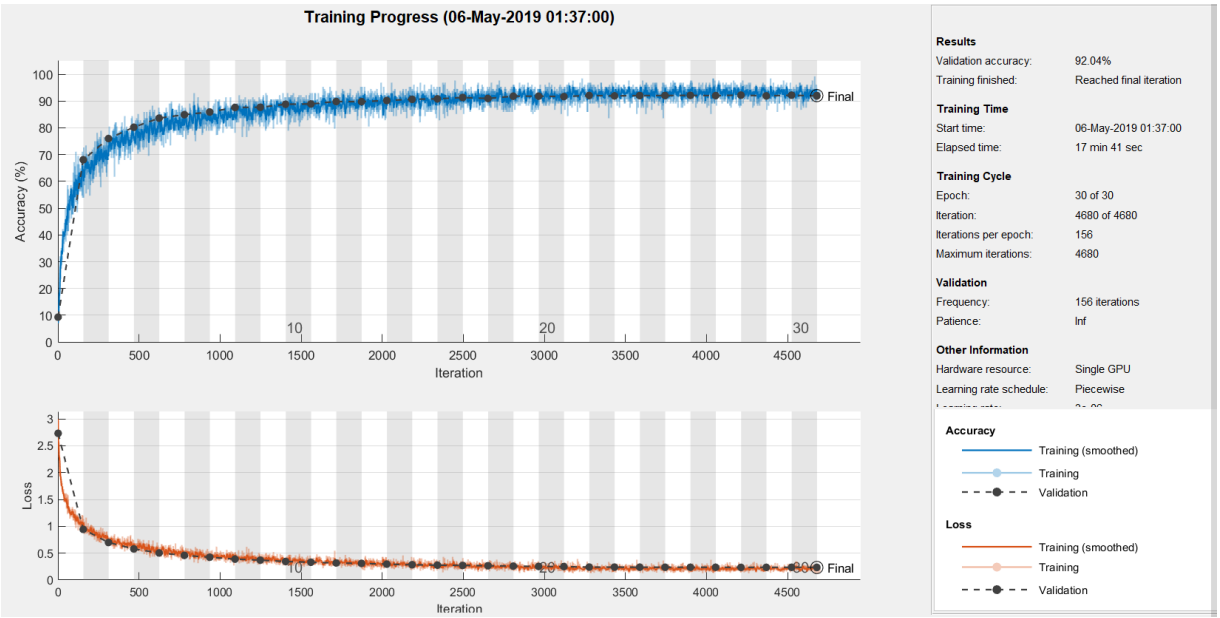
- Lớp loại bỏ:

- Ta tạo một lớp dropout với xác suất một neural bị loại bỏ trong khi training là 20% để giảm khả năng mạng của ta bị overfitting.

- Lớp tổng hợp:

- Lớp này sẽ lấy đầu vào từ lớp trước đó và nối lại thành 1 vector 10 chiều ứng với 10 mẫu cần nhận dạng.

2.1.3. Kết quả huấn luyện



Hình 2.4. Kết quả huấn luyện mạng CNN

Ta thấy độ chính xác của mạng neural đạt được là 92.04% sau 30 chu kì huấn luyện.

- Các đầu ra để phân loại bao gồm:

- Bậc 1

- Bậc 1 có trễ

- Bậc 2

- Bậc 2 có trễ

- Dao động bậc 2

- Dao động bậc 2 có trễ

- Bậc 3

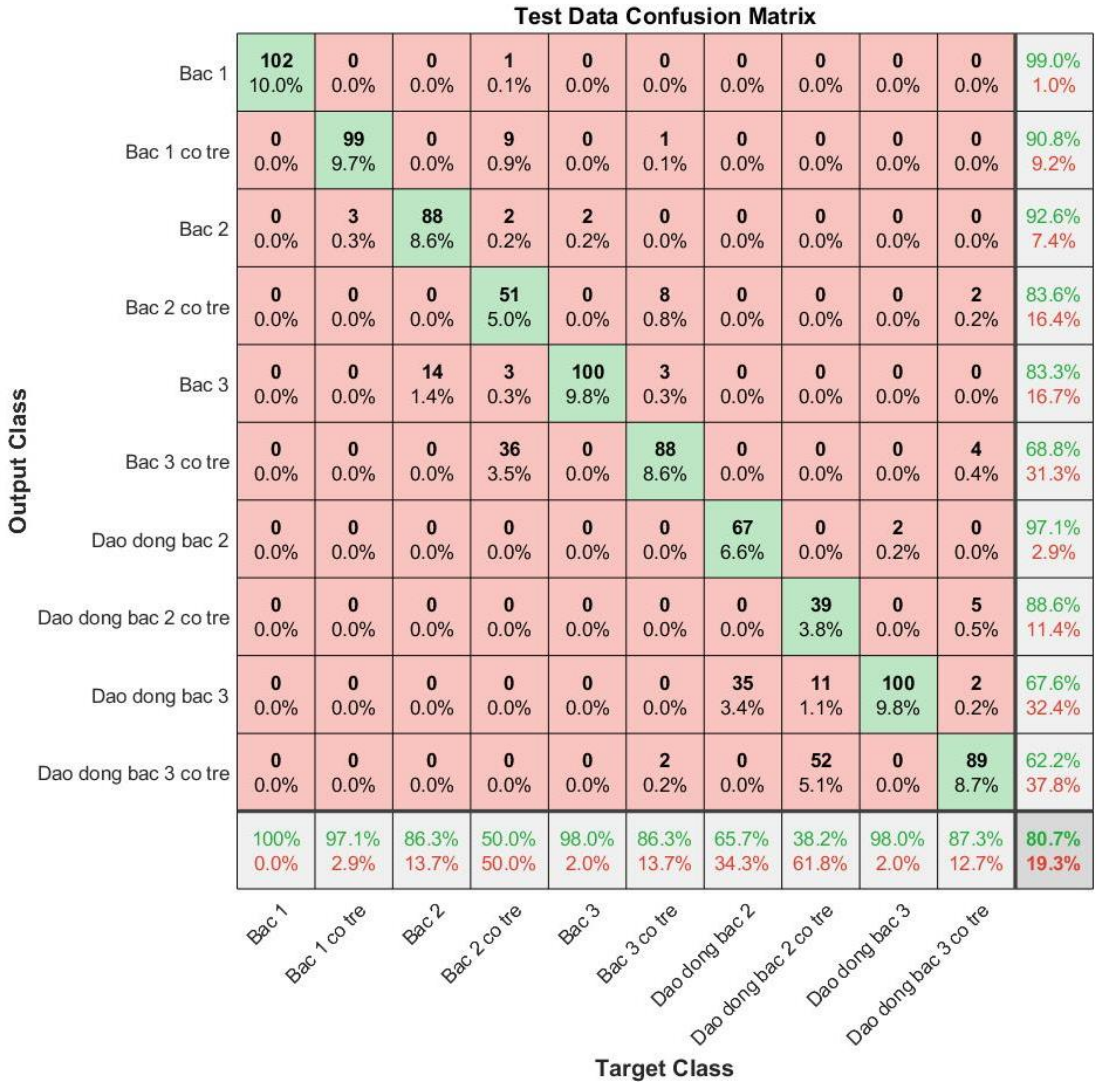
- Bậc 3 có trễ

- Dao động bậc 3

- Dao động bậc 3 có trễ

Ta tạo mẫu mới với thông số nằm trong dải mới:

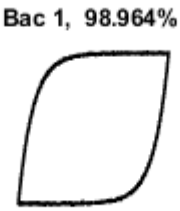
$$T_{\max} = 500; \tau \in [0;100]; k \in [0;150]; T_1, T_2, T_3 \in [1;50]; D \in (0;1)$$



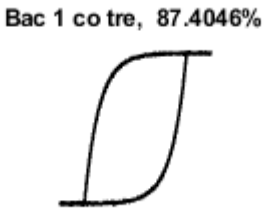
Hình 2.5. Bảng phân bố tỷ lệ 102 mẫu kiểm tra mới

Qua hình trên có thể nhận thấy rằng đối với nhiều mẫu, sự khác nhau là rất ít nên đã xảy ra nhiều trường hợp mạng nhận dạng nhầm giữa bậc đúng của mô hình với bậc của mô hình có dạng tương tự mẫu.

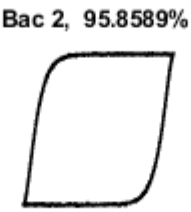
Một số mẫu mới được đưa vào mạng neural vừa huấn luyện để kiểm tra:



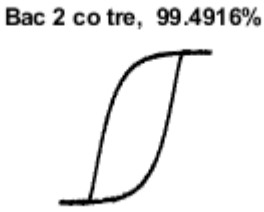
Hình 2.6. Bậc 1



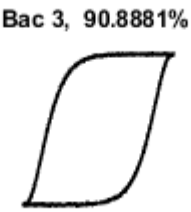
Hình 2.11. Bậc 1 có trễ



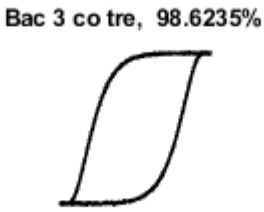
Hình 2.7. Quán tính bậc 2



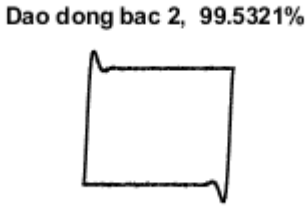
Hình 2.12. Quán tính bậc 2 có trễ



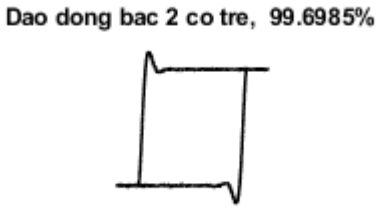
Hình 2.8. Bậc 3



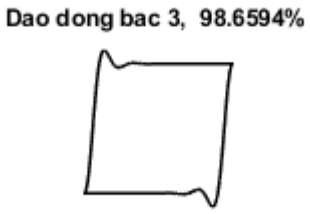
Hình 2.13. Bậc 3 có trễ



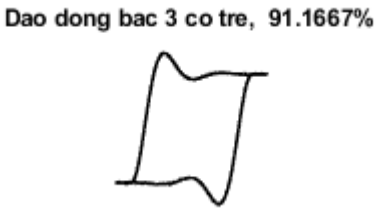
Hình 2.9. Dao động bậc 2



Hình 2.14. Dao động bậc 2 có trễ



Hình 2.10. Dao động bậc 3



Hình 2.15. Dao động bậc 3 có trễ

2.2. Xác định tham số mô hình ARMA bằng mạng hồi quy

2.2.1. Tạo mạng nơ-ron

Sau khi đã nhận dạng được bậc của mô hình toán học của đối tượng ta sẽ xác định tham số bằng phương pháp thực nghiệm chủ động với tín hiệu đầu vào u(t) đặt trước là hàm 1(t) và tín hiệu ra y(t) = h(t). Bộ thông số nhận được với 1 chu kỳ trích mẫu T_a.

Với thành phần delay ta sử dụng hàm delayest để xác định.

Với dạng $G(s)$ đã xác định được trước tiên ta giả sử giữa hàm truyền và đối tượng điều khiển không có sai lệch, đưa hàm truyền $G(s)$ về dạng mô hình hàm truyền không liên tục:

$$G(z)=\frac{Y(z)}{U(z)}=\frac{b_0+b_1z^{-1}+...+b_{n_b}z^{-n_b}}{1+a_1z^{-1}+a_2z^{-2}+...+a_{n_a}z^{-n_a}}z^{-n_k} \qquad (2.13)$$

Với $U(z),Y(z)$ là ảnh z của dãy giá trị các tín hiệu vào ra $\{u_k\},\{y_k\}$ của đối tượng với chu kỳ trích mẫu T_a; n_b,n_a lần lượt là bậc tử và mẫu của hàm truyền z; n_k xác định được nhờ hàm delayest của phần mềm Matlab.

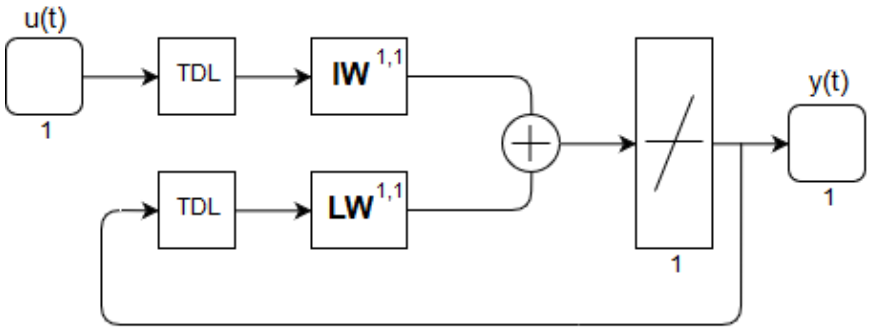
=> Sử dụng ý nghĩa $z^{-i}x_k=x_{k-1}$ của biến z, phương trình (2.13) được biểu diễn lại dưới dạng phương trình tương đương viết trực tiếp theo quan hệ vào ra của tín hiệu trong miền thời gian [3] :

$$y_k\left(1+a_1z^{-1}+a_2z^{-2}+...+a_{n_a}z^{-n_a}\right)=u_kz^{-n_k}\left(b_0+b_1z^{-1}+...+b_{n_b}z^{-n_b}\right)$$
$$y_k+a_1y_{k-1}+...+a_{n_a}y_{k-n_a}=b_0u_{k-n_k}+b_1u_{k-1-n_k}+...+b_{n_b}u_{k-n_b-n_k} \quad (2.14)$$

Bỏ qua thành phần nhiễu ta viết lại được như sau:

$$y_k=\sum_{i=0}^{n_b}b_iu_{k-i-n_k}-\sum_{i=1}^{n_a}a_iy_{k-i} \quad (2.15)$$

Để nhận dạng được các tham số của mô hình (2.13) ta sử dụng 1 mạng neural xấp xỉ mô hình đối tượng



Hình 2.16. Mạng neural mô tả đối tượng

Với đầu vào là các giá trị đo được $\{u_k\},\{y_k\}$ $k=0,...,N$ với chu kỳ trích mẫu T_a. Các khối TDL là khối delay tương ứng với các giá trị trọng số IW, LW

$$IW^{1,1}=\begin{bmatrix}b_0 & b_1... & b_{n_b}\end{bmatrix};\quad LW^{1,1}=\begin{bmatrix}-a_1 & -a_2... & -a_{n_a}\end{bmatrix} \quad (2.16)$$

Từ đó ta được mô hình mạng neural mô tả đối tượng là mạng neural 1 lớp, 1 đầu vào, hàm truyền là "purelin".

Biểu thức toán học diễn tả mạng:

$$y_k=purelin\left(IW^{1,1}\underline{u}_{TDL}+LW^{1,1}\underline{y}_{TDL}\right)$$
$$=\begin{bmatrix}b_0 & ... & b_{n_b}\end{bmatrix}\begin{bmatrix}u_{k-n_k}\\ .\\ .\\ u_{k-n_b-n_k}\end{bmatrix}+\begin{bmatrix}-a_1 & ... & -a_{n_a}\end{bmatrix}\begin{bmatrix}y_{k-1}\\ .\\ .\\ y_{k-n_a}\end{bmatrix}$$

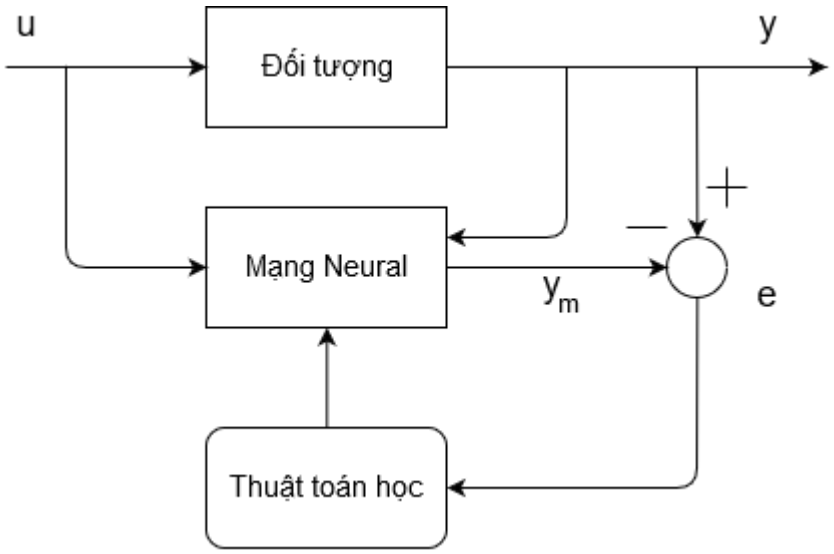
$$\Leftrightarrow y_k = \sum_{i=0}^{n_b} b_i u_{k-i-n_k} - \sum_{i=1}^{n_a} a_i y_{k-i} \quad (2.17)$$

2.2.2. Huấn luyện

Trên thực tế giữa 2 vế của đẳng thức (2.15) tồn tại một sai lệch và đó chính là sai lệch mô hình với đối tượng [3] :

$$e_k = y_k - \left(\sum_{i=0}^{n_b} b_i u_{k-i-n_k} - \sum_{i=1}^{n_a} a_i y_{k-i} \right) \quad (2.18)$$

Bài toán nhận dạng dựa trên cơ sở cực tiểu hóa hàm mô tả sai lệch mô hình:



Hình 2.17. Sơ đồ quá trình huấn luyện mạng

Hình trên biểu diễn sơ đồ huấn luyện tham số cho mạng neural dựa trên sai số e giữa đầu ra của mạng và đầu ra $\{y_k\}$.

Ở đây luật học em chọn là trainlm (Levenberg-Marquardt) [8] với hàm mục tiêu mô tả tổng bình phương sai lệch giữa mô hình hàm truyền và đối tượng.

Các dạng hàm truyền rời rạc của các mẫu trong bài báo cáo:

Khâu tích phân bậc nhất: $G(z) = \dfrac{b_1 z^{-1}}{1 + a_1 z^{-1}}$ (2.19)

Khâu quán tính bậc hai: $G(z) = \dfrac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$ (2.20)

Khâu dao động bậc hai: $G(z) = \dfrac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$ (2.21)

Khâu quán tính bậc ba: $G(z) = \dfrac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}$ (2.22)

Khâu dao động bậc ba: $G(z) = \dfrac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}$ (2.23)

Các khâu trên với thành phần trễ z^{-n_k}

2.3. Một số kết quả nhận dạng bậc và tham số

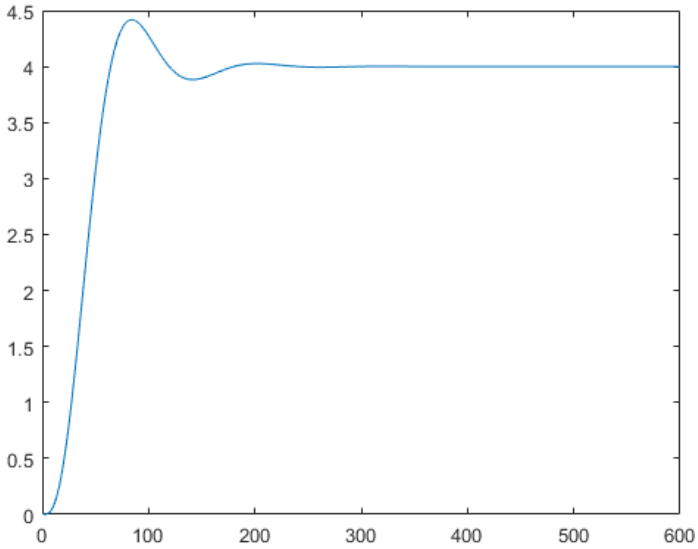
2.3.1. Bộ tham số lý tưởng

$$G(s) = \frac{4}{6s^3 + 5.9s^2 + 3.45s + 1} \quad (2.24)$$

Chuyển về dạng rời rạc với chu kỳ trích mẫu 0.1s là:

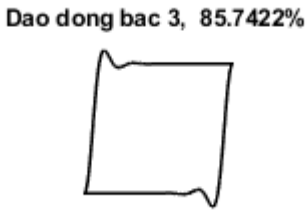
$$G(z) = \frac{1.084e-4z^{-1} + 4.23e-4z^{-2} + 1.032e-4z^{-3}}{1 - 2.901z^{-1} + 2.807z^{-2} - 0.9063z^{-3}} \quad (2.25)$$

Với bộ dữ liệu u(t), y(t) là lý tưởng với chu kỳ trích mẫu 0.1s.



Hình 2.18. Đáp ứng y(t)

Sau khi đưa vào mạng CNN đã huấn luyện ở mục 0 ta được kết quả:



Hình 2.19. Kết quả nhận dạng bậc bộ tham số lý tưởng

Nhận dạng hệ thống là mô hình dao động bậc 3 với độ chính xác 85.7422%

Bảng phân phối tỷ lệ % của các đầu ra như sau:

Bảng 2.1. Bảng phân phối tỷ lệ nhận dạng bậc mô hình

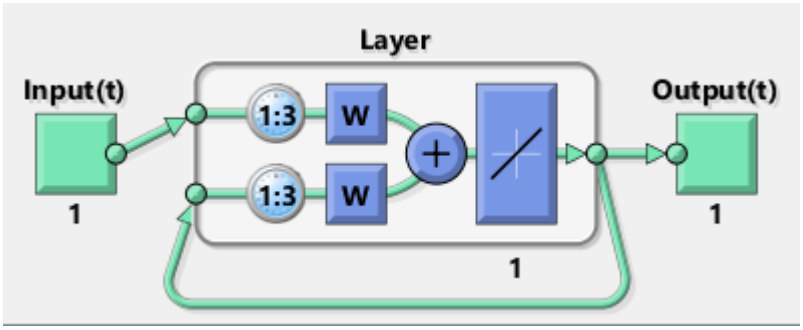
Bậc 1	Bậc 1 trễ	Dao động bậc 3	Dao động bậc 3 trễ	Quán tính bậc 2	Quán tính bậc 2 trễ	Quán tính bậc 3	Quán tính bậc 3 trễ	Dao động bậc 2	Dạo động bậc 2 có trễ
0.0000	0.0000	0.8574	0.1415	0.0000	0.0001	0.0003	0.0006	0.0000	0.0001

Từ đó ra xác định được mô hình hàm truyền rời rạc:

$$G(z) = \frac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}} \qquad (2.26)$$

Với $n_a = n_b = 3; n_k = 0$.

Mạng Neural mô phỏng hàm truyền của đối tượng



Hình 2.20. Mạng mô phỏng đối tượng dao động bậc 3

Sau khi huấn luyện sau 30 chu kì và so sánh với Toolbox System Identification của Matlab ta nhận được kết quả:

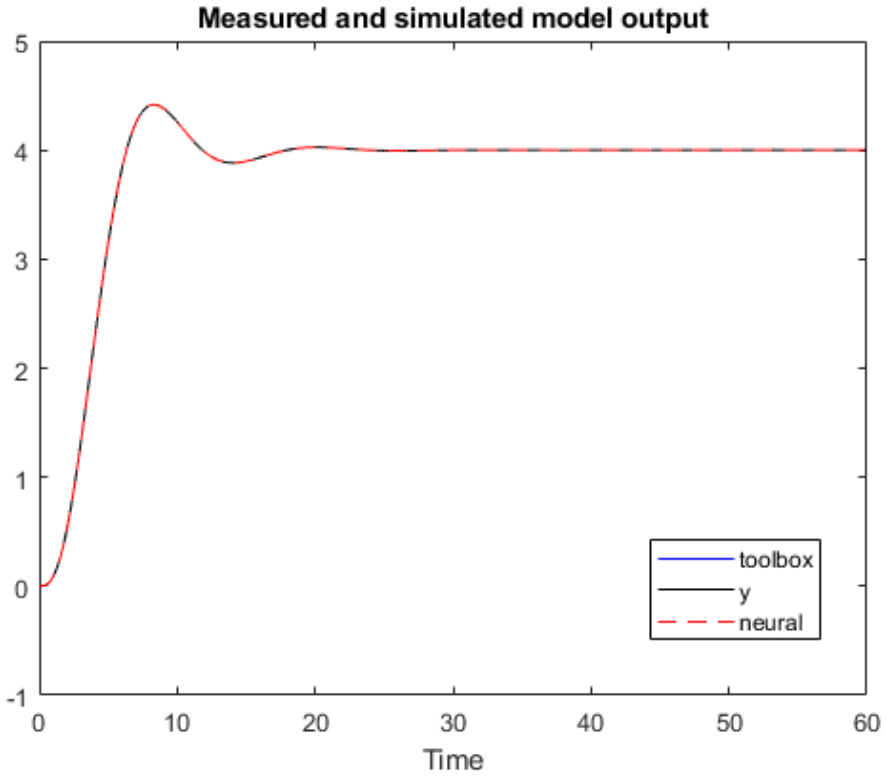
$$\text{Neural: } G(z) = \frac{2.116e-4 z^{-1} + 2.115e-4 z^{-2} + 2.115e-4 z^{-3}}{1 - 2.9008 z^{-1} + 2.8073 z^{-2} - 0.9063 z^{-3}} \qquad (2.27)$$

$$\text{Toolbox: } G(z) = \frac{2.1155e-4 z^{-1} + 2.1154e-4 z^{-2} + 2.1154e-4 z^{-3}}{1 - 2.9008 z^{-1} + 2.8073 z^{-2} - 0.9063 z^{-3}} \qquad (2.28)$$

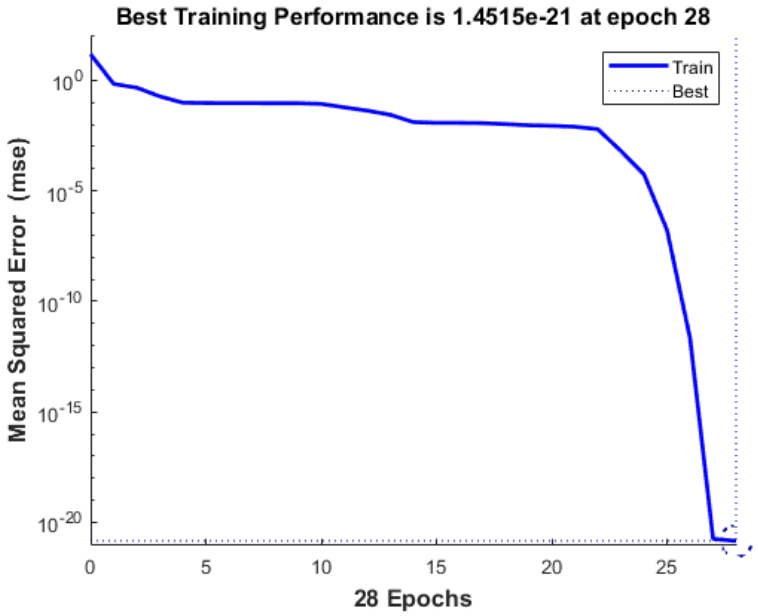
- Hàm G(s):

$$\text{Neural: } G(s) = \frac{4}{6s^3 + 5.899s^2 + 3.45s + 1} \tag{2.29}$$

$$\text{Toolbox: } G(s) = \frac{4}{6s^3 + 5.899s^2 + 3.45s + 1} \tag{2.30}$$



Hình 2.21. Bảng so sánh đầu ra của các phương pháp

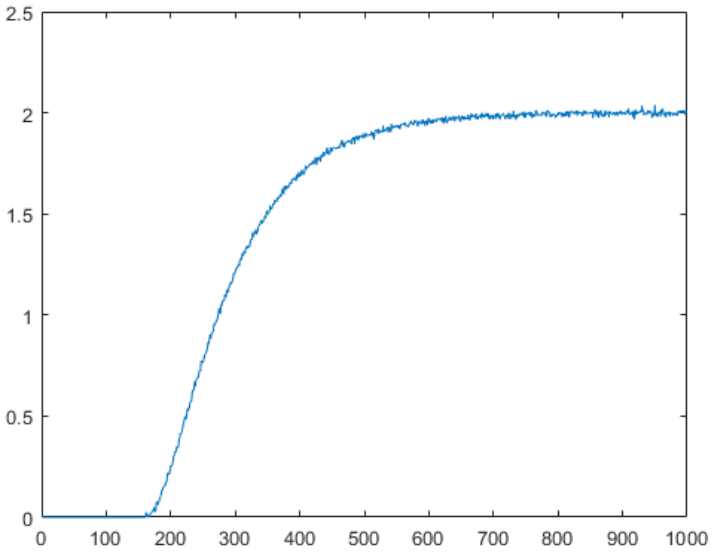


Hình 2.22. Hiệu suất sử dụng phương pháp bình phương tối thiểu

Hiệu suất đạt mức 1.45e-21 sau 28 chu kì.

2.3.2. Bộ tham số có thêm nhiễu

$$\text{Hàm } G(s) = e^{-16s}\frac{2}{40s^2+14s+1} \tag{2.31}$$



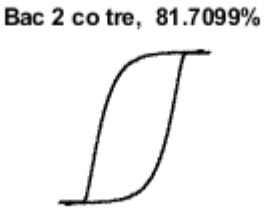
Hình 2.23. Đầu ra y(t)

Chuyển về dạng rời rạc với chu kỳ trích mẫu 0.1s là:

$$G(z) = z^{-160} \times \frac{2.471e-4z^{-1} + 2.442e-4z^{-2}}{1-1.965z^{-1} + 0.9656z^{-2}} \quad (2.32)$$

Với bộ dữ liệu u(t), y(t) là lý tưởng với chu kỳ trích mẫu 0.1s.

Sau khi đưa vào mạng CNN đã huấn luyện ở mục 0 ta được kết quả:



Hình 2.24. Kết quả nhận dạng bậc mô hình

Nhận dạng hệ thống là mô hình quán tính bậc 2 có trễ với độ chính xác 81.7099%

Bảng phân phối tỷ lệ % của các đầu ra như sau:

Bảng 2.2. Bảng phân phối tỷ lệ nhận dạng bậc mô hình

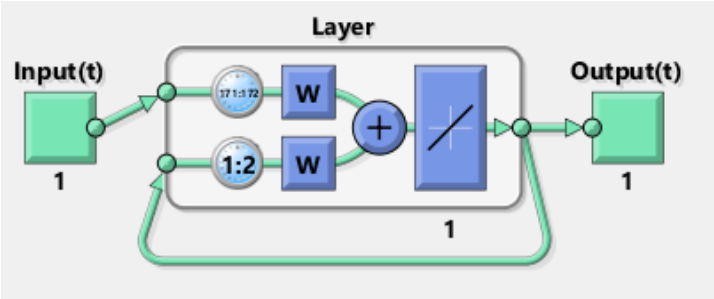
Bậc 1	Bậc 1 trễ	Dao động bậc 3	Dao động bậc 3 trễ	Quán tính bậc 2	Quán tính bậc 2 trễ	Quán tính bậc 3	Quán tính bậc 3 trễ	Dao động bậc 2	Dạo động bậc 2 có trễ
0.0002	0.0006	0.0000	0.0014	0.0000	0.8171	0.0001	0.1805	0.0000	0.0000

Từ đó ra xác định được mô hình hàm truyền rời rạc:

$$G(z) = z^{-n_k} \times \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad (2.33)$$

Sử dụng hàm delayest ta xác định được $n_a = n_b = 2; n_k \approx 170$

Mạng nơ-ron mô phỏng hàm truyền của đối tượng



Hình 2.25. Mạng mô phỏng đối tượng

Sau khi huấn luyện sau 21 chu kì và so sánh với Toolbox System Identification của Matlab ta nhận được kết quả:

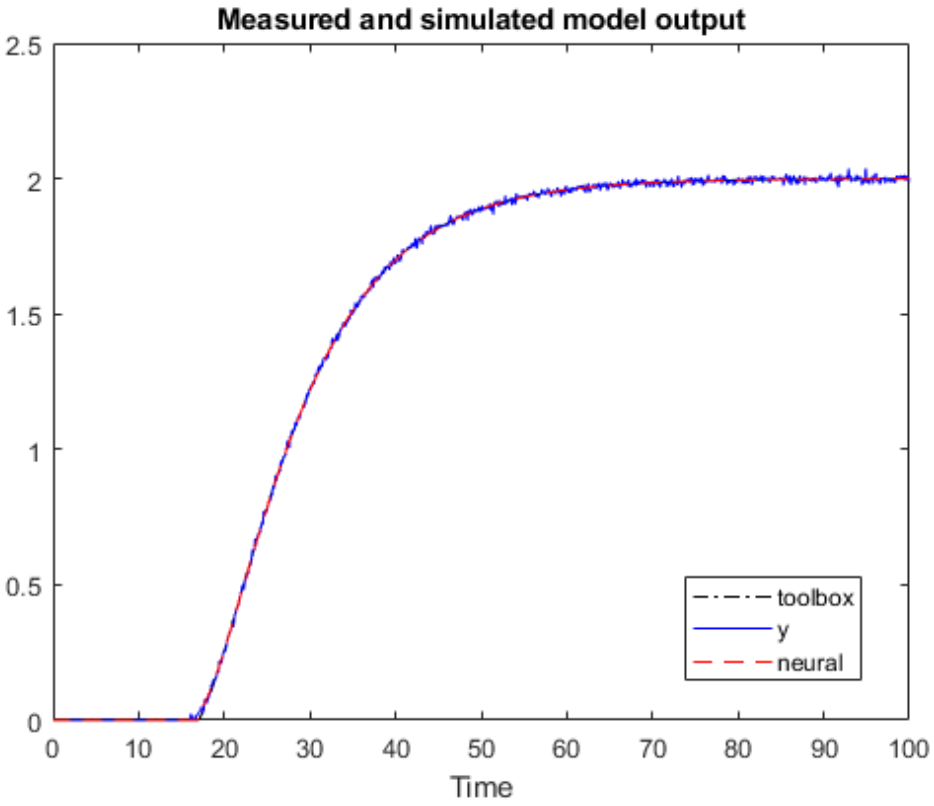
$$\text{Neural: } G(z) = z^{-170} \times \frac{2.466e-4z^{-1} + 2.465e-4z^{-2}}{1 - 1.9652z^{-1} + 0.9654z^{-2}} \tag{2.34}$$

$$\text{Toolbox: } G(z) = z^{-170} \times \frac{5.859e-3z^{-1} + 5.401e-3z^{-2}}{1 - 1.967z^{-1} + 0.9673z^{-2}} \tag{2.35}$$

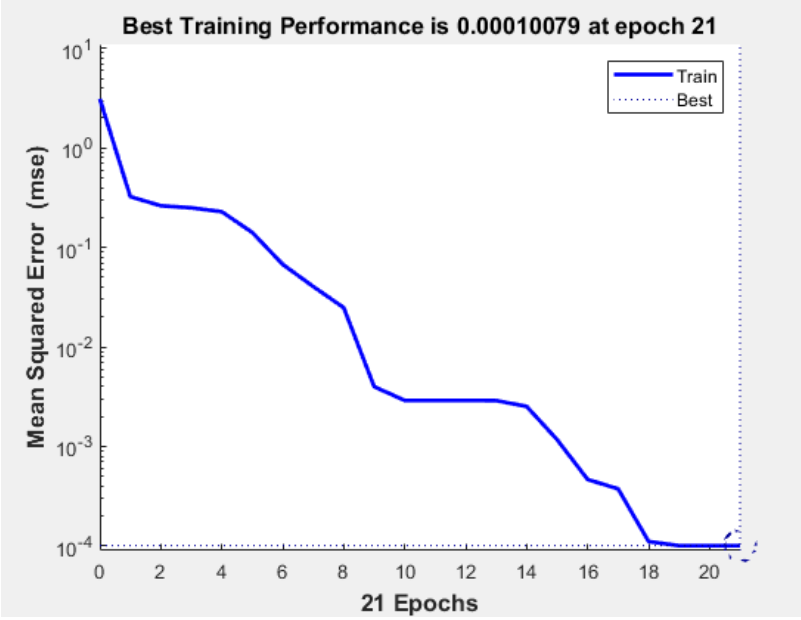
- Hàm G(s):

$$\text{Neural: } G(s) = e^{-17s} \frac{2.0076}{40s^2 + 14.064s + 1.0036} \tag{2.36}$$

$$\text{Toolbox: } G(s) = e^{-17s} \frac{2.0336}{40s^2 + 13.3s + 0.8132} \tag{2.37}$$



Hình 2.26. Sơ đồ so sánh đầu ra giữa các phương pháp

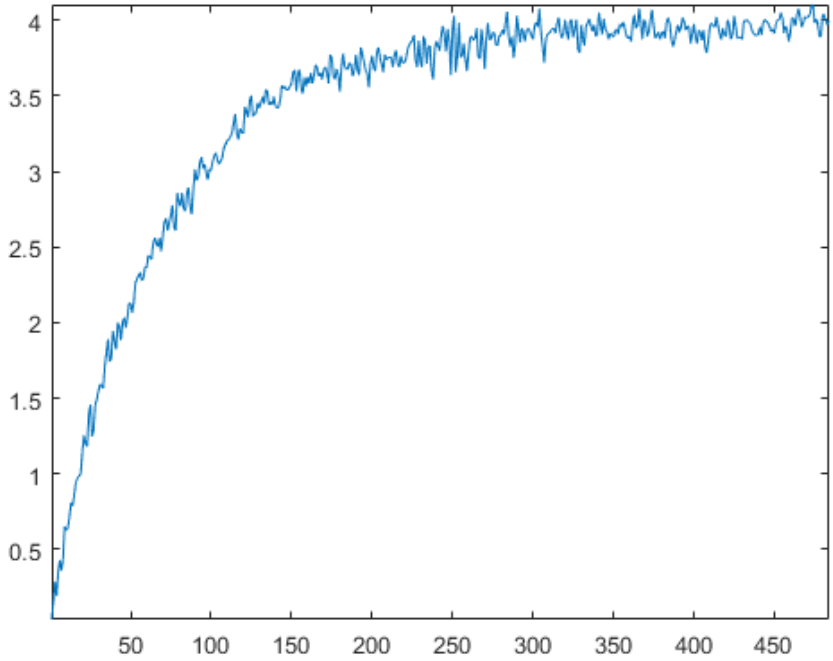


Hình 2.27. Hiệu suất

Hiệu suất đạt 1.0079e-4 sau 21 chu kì.

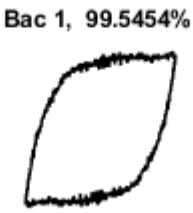
2.3.3. Bộ tham số thực

Đối tượng nhận dạng là động cơ một chiều với bộ tham số $\{u_k\},\{y_k\}$ là quan hệ điện áp - dòng điện. Đầu vào là điện áp 18V, chu kỳ lấy mẫy $T_a = 5e-6s$.



Hình 2.28. Sơ đồ bộ tham số thực

Sau khi đưa vào mạng CNN đã huấn luyện ở mục 0 ta được kết quả:



Hình 2.29. Kết quả nhận dạng đối tượng động cơ một chiều

Nhận dạng hệ thống là mô hình quán tính bậc nhất với độ chính xác 99.5454%

Bảng phân phối tỷ lệ % của các classes như sau:

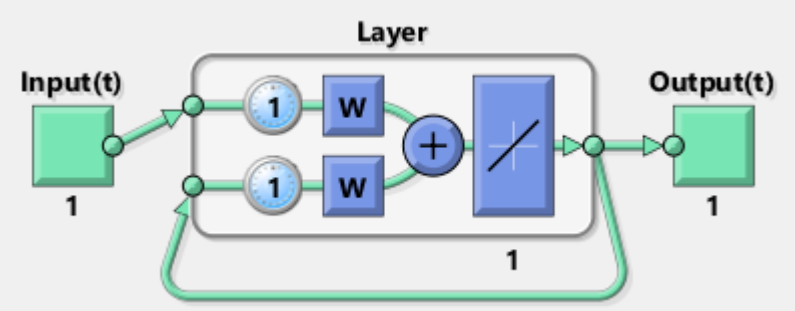
Bảng 2.3. Bảng phân phối tỷ lệ nhận dạng bậc mô hình

Bậc 1	Bậc 1 trễ	Dao động bậc 3	Dao động bậc 3 trễ	Quán tính bậc 2	Quán tính bậc 2 trễ	Quán tính bậc 3	Quán tính bậc 3 trễ	Dao động bậc 2	Dạo động bậc 2 có trễ
0.9955	0.0002	0.0000	0.0000	0.0001	0.0000	0.0000	0.0000	0.0042	0.0000

Từ đó ra xác định được mô hình hàm truyền là dạng bậc nhất $G(s)=\frac{k}{1+Ts}$

Mô hình rời rạc $G(z)=\frac{b_1 z^{-1}}{1+a_1 z^{-1}}$ $n_a=n_b=2; n_k=0$

Mạng neural mô phỏng hàm truyền của đối tượng



Hình 2.30. Mạng mô phỏng đối tượng

Sau khi huấn luyện sau 12 chu kì và so sánh với Toolbox System Identification của Matlab ta nhận được kết quả:

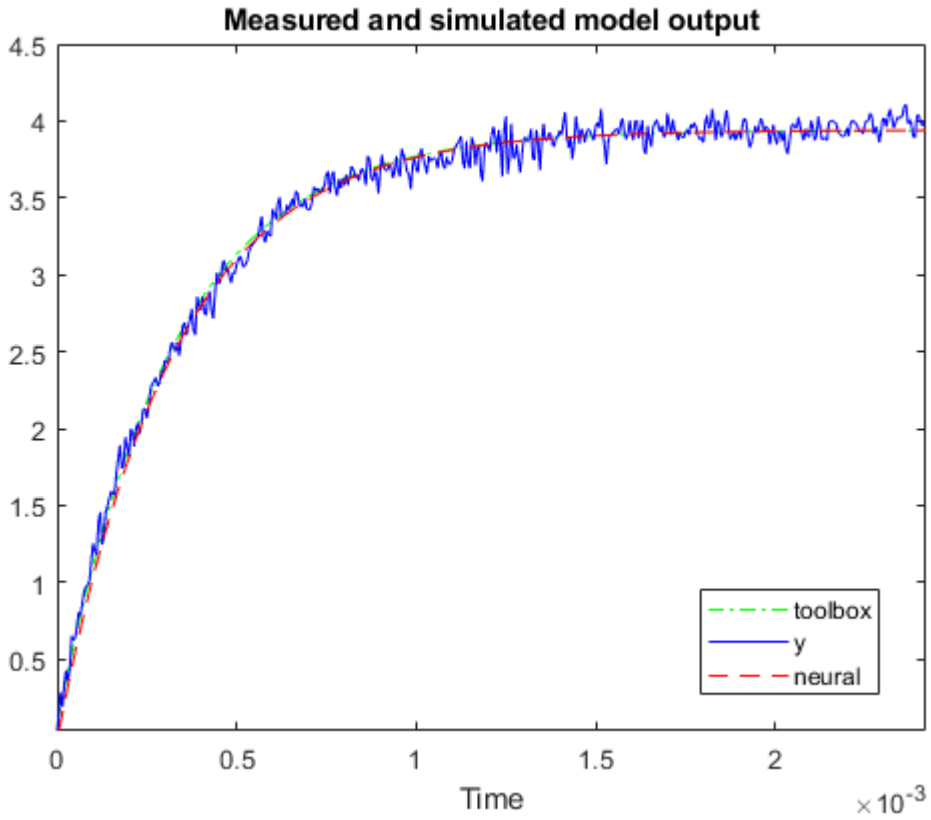
$$\text{Neural: } G(z)=\frac{0.0034z^{-1}}{1-0.9846z^{-1}} \tag{2.38}$$

$$\text{Toolbox: } G(z)=\frac{0.003406z^{-1}}{1-0.9844z^{-1}} \quad \text{với } T_a=5.10^{-6} \tag{2.39}$$

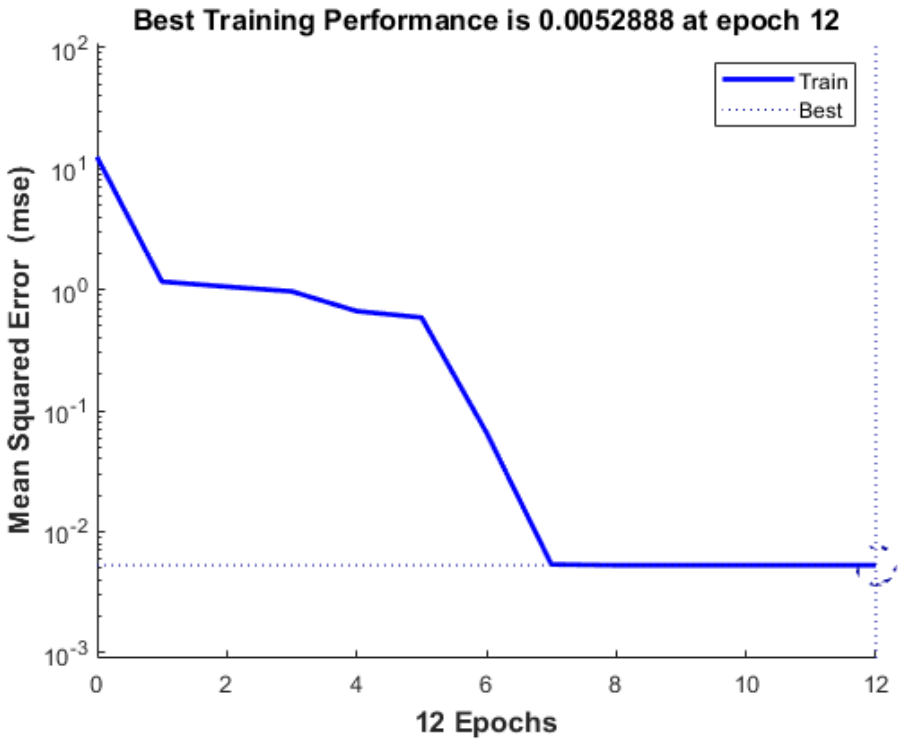
- Hàm G(s):

$$\text{Neural: } G(s) = \frac{0.2199}{3.224e-4s+1} \tag{2.40}$$

$$\text{Toolbox: } G(s) = \frac{0.2183}{3.1797e-4s+1} \tag{2.41}$$



Hình 2.31. Bảng so sánh đầu ra giữa các phương pháp



Hình 2.32. Hiệu suất

Hiệu suất đạt 5.2888e-3 sau 12 chu kì.

2.4. Kết luận chương

Bài toán nhận dạng được chia làm hai phần rõ ràng: Nhận dạng bậc và nhận dạng thông số. Đặc biệt phương pháp Nhận dạng bậc sử dụng mạng CNN là khá mới. Tuy nhiên, mạng đang được xây dựng vẫn chưa tối ưu, có thể được phát triển thêm bằng cách thay đổi cấu trúc mạng, tối ưu hóa đầu vào bằng các cách xử lý ảnh nhằm giảm số lượng thông số của mạng, tăng tốc độ huấn luyện.

CHƯƠNG 3. ỨNG DỤNG MẠNG CNN VÀO NHẬN DẠNG GIỌNG NÓI

Ở chương này, ta sẽ sử dụng mạng CNN để nhận dạng 5 lệnh giọng nói: "tiến", "lùi", "trái", "phải", "dừng" cho việc điều khiển xe hai bánh tự cân bằng.

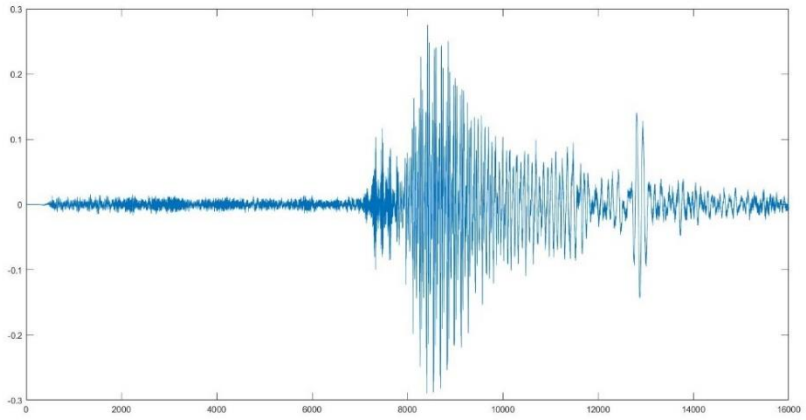
3.1. Dữ liệu huấn luyện

Mẫu dữ liệu huấn luyện là các đoạn ghi âm 1 giây, mỗi đoạn ghi âm chỉ chứa một từ cần nhận dạng. Mỗi từ cần nhận dạng sẽ bao gồm hơn 500 đoạn ghi âm của hơn 100 người khác nhau là sinh viên của trường Đại học Bách Khoa Hà Nội, được chia vào một thư mục riêng.

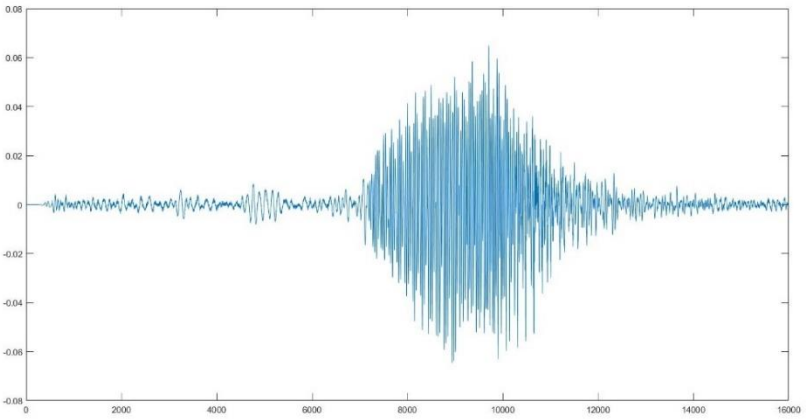
Ngoài các lệnh ta đã đặt ra để nhận dạng như "tiến", "lùi", "trái", "phải", "dừng", ta sẽ sử dụng thêm các mẫu huấn luyện là các từ khác, các mẫu này đã có sẵn trên mạng cung cấp bởi TensorFlow – Một thư viện phần mềm mã nguồn mở dành cho máy học trong nhiều loại hình tác vụ nhận thức và hiểu ngôn ngữ, một sản phẩm của Google. Các mẫu này giúp máy tính có thể phân biệt được đâu là lệnh cần nhận diện, đâu là một từ nào đó khác không nằm trong số lệnh cần nhận dạng nêu trên.

Thêm đó là các mẫu được đặt làm âm thanh nền, để phân biệt khi nào đang có giọng nói, khi nào là chỉ các âm thanh ngoài môi trường đang được thu lại.

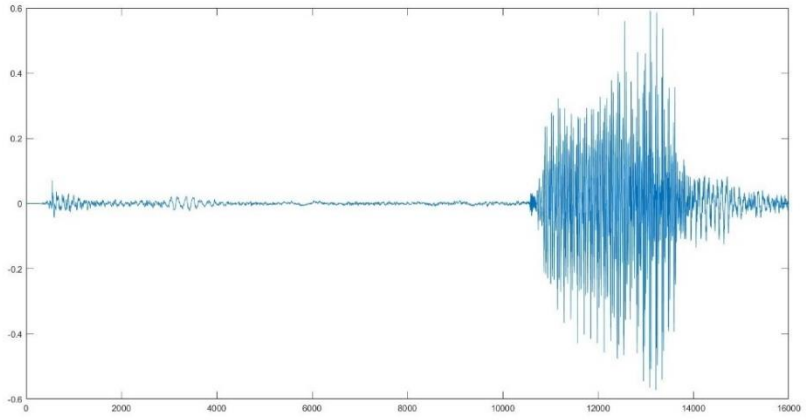
Mỗi mẫu huấn luyện sẽ được ghi âm và đưa về với tốc độ lấy mẫu là 16kHz (16,000 mẫu mỗi giây), và được lưu lại với định dạng .wav



Hình 3.1. Một mẫu huấn luyện lệnh "dừng"



Hình 3.2. Một mẫu huấn luyện lệnh "lùi"



Hình 3.3. Một mẫu huấn luyện lệnh "tiến"

3.2. Tiền xử lý dữ liệu mẫu âm thanh

Mục đích của bước này là đưa mẫu huấn huyện từ dạng sóng âm sang dạng quang phổ bằng biến đổi Fourier. Ta có thể truyền trực tiếp dạng sóng âm vào mạng nơ-ron nhưng cố gắng nhận diện cấu trúc âm thanh trực tiếp bằng những mẫu này rất khó. Thay vào đó, chúng ta giải quyết vấn đề dễ hơn với dạng quang phổ của âm thanh.

Để đưa về dạng quang phổ, ta sẽ thực hiện các bước sau.

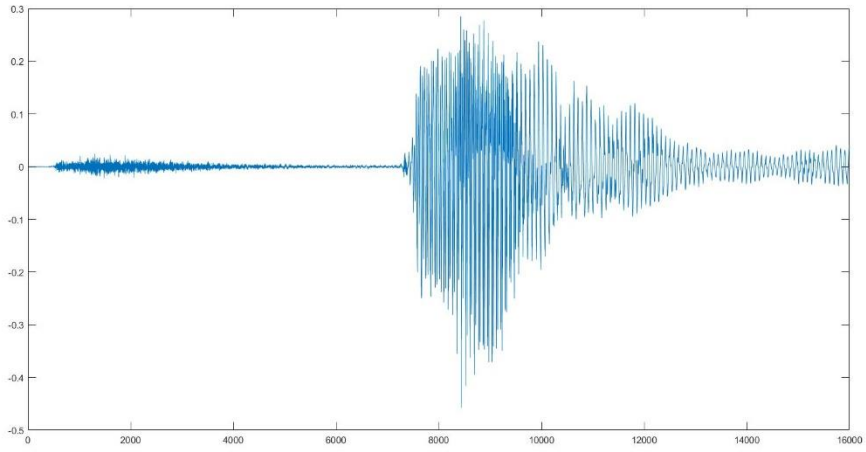
3.2.1. Bộ lọc hiệu chỉnh

Tín hiệu tiếng nói $s(n)$ được đưa qua bộ lọc số bậc thấp để phổ đồng đều hơn, giảm ảnh hưởng gây ra cho các xử lý tín hiệu sau này, ngoài ra cũng tránh được các vấn đề về tính toán trong khâu biến đổi Fourier sau này.

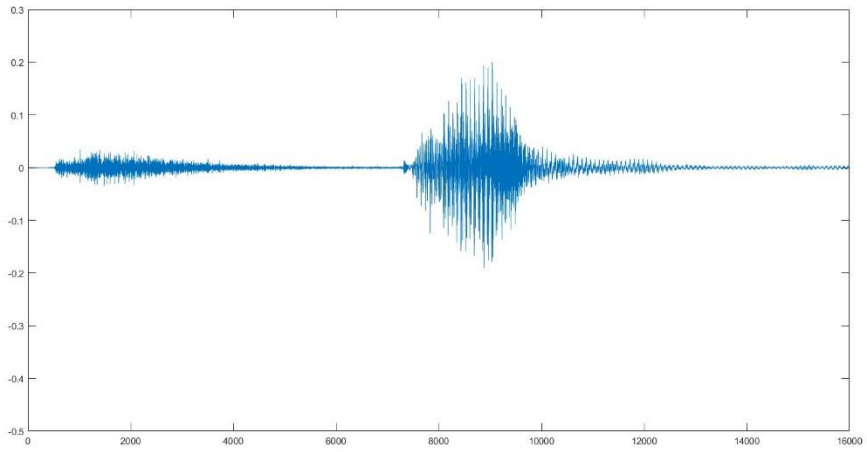
Bộ lọc hiệu chỉnh được áp dụng vào tín hiệu $s(n)$ thường cố định bậc một, và quan hệ giữa tín hiệu ra với tín hiệu vào tuân theo phương trình:

$$x(n) = s(n) - \alpha.s(n-1) \tag{3.1}$$

Hệ số α thường được chọn là 0.95 hoặc 0.97. Ở đây ta chọn $\alpha = 0.97$ vì nó loại bỏ các tín hiệu tần số thấp tốt hơn.



Hình 3.4. Lệnh "tiến" trước khi qua bộ lọc hiệu chỉnh



Hình 3.5. Lệnh "tiến" sau khi qua bộ lọc hiệu chỉnh

3.2.2. Phân khung

Sau khi lọc hiệu chỉnh, ta cần chia nhỏ tín hiệu thành từng khung với thời gian nhỏ. Lý do thực hiện bước này là tần số trong tín hiệu thường thay đổi theo thời gian, do đó trong hầu hết các trường hợp, ta không nên thực hiện biến đổi Fourier trên toàn bộ tín hiệu, bởi vì khi đó ta sẽ làm mất các đường viền tần số của tín hiệu theo thời gian. Để tránh trường hợp này, ta có thể giả thiết rằng các tần số của tín hiệu không thay đổi trong một khoảng thời gian ngắn. Do đó, bằng cách biến đổi Fourier cho từng đoạn thời gian nhỏ này sẽ cho ra được một xấp xỉ tốt của miền tần số của tín hiệu bằng cách ghép các khung nhỏ liền kề. Ngoài ra, việc chia nhỏ này sẽ giúp cho việc biến đổi Fourier được thực hiện nhanh hơn.

Với bài toán đặt ra trong đồ án này, ta chọn thời gian cho mỗi khung là 30ms, với bước nhảy để bắt đầu khung tiếp theo là 10ms (tức là 20ms cuối của khung trước sẽ trùng với 20ms đầu của khung sau).

Khi đó, từ tín hiệu ban đầu $x(n)_{N_0}$ với số giá trị là $N_0 = 16000$, ta chia thành các khung $x_i(n)_{N_1}$ với số giá trị mới là $N_1 = 0.03*16000 = 480$, tổng số khung là 98, hay $0 \le i \le 97$.

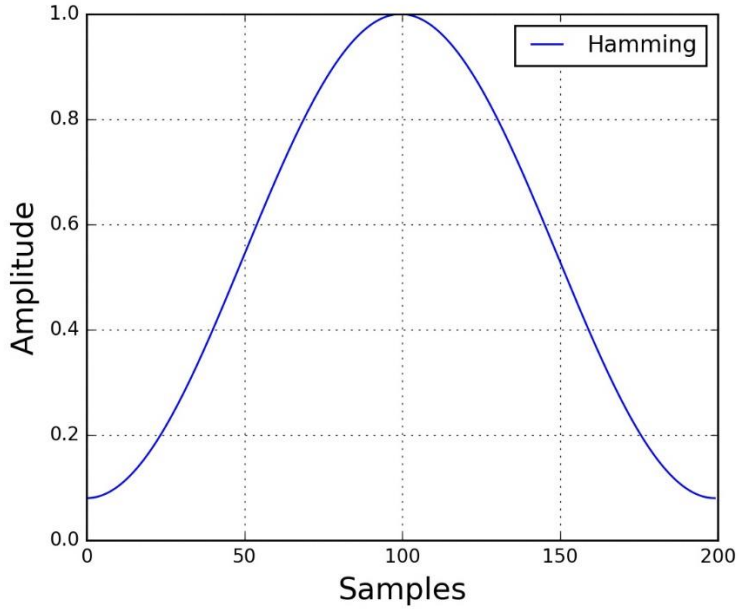
3.2.3. Áp dụng hàm cửa sổ

Sau khi chia tín hiệu ra thành các khung nhỏ, ta sẽ áp dụng hàm cửa sổ Hamming vào từng khung. Hàm này có công thức:

$$\omega(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N_1}\right) \tag{3.2}$$

Với $0 \le n \le N_1 - 1$, N_1 là độ dài mỗi khung nhỏ.

Biểu diễn hàm cửa sổ trên qua đồ thị ta được:



Hình 3.6. Hàm cửa sổ Hamming

Khi đó, từ một tín hiệu đã được chia nhỏ ban đầu $x_i(n)_{N_1}$, ta được tín hiệu mới là:

$$x_i^1(n)_{N_1} = x_i(n)_{N_1} \cdot \omega(n) \tag{3.3}$$

Dạng hàm cửa sổ ảnh hưởng rất lớn đến độ chính xác của việc xấp xỉ hàm tần số. Việc chọn hàm cửa sổ nào để xấp xỉ hóa hàm tần số, không chỉ phụ thuộc vào độ chính xác yêu cầu, mà còn phụ thuộc vào hạn chế về thời gian tính toán, vì hàm cửa sổ cho độ chính xác càng cao thì việc tính toán sẽ càng phức tạp, và thời gian tính sẽ

càng lớn. Do đó việc chọn hàm cửa sổ phụ thuộc vào sự tối ưu giữa độ chính xác yêu cầu và thời gian tính. Với bài toán này, sau khi thử với các hàm cửa sổ khác như cửa sổ chữ nhật, cửa sổ tam giác và cửa sổ Hanning, ta quyết định chọn cửa sổ Hamming vì cho ra độ chính xác của mạng CNN là cao nhất.

3.2.4. Biến đổi Fourier

Sau các bước chuẩn bị bên trên, bây giờ ta có thể thực hiện phép biến đổi Fourier. Ở bài toán này ta sử dụng biến đổi Fourier rời rạc DFT (*Discrete Fourier Transform)* cho dãy có độ dài hữu hạn:

$$X_i(k)_N = \sum_{n=0}^{N-1} x_i^1(n)_{N_1} e^{-jk\omega_1 n} \tag{3.4}$$

Với $\omega_1 = \frac{2\pi}{N}$. Ta chọn $N = 512$ để cho $N > N_1$ và khi đó ta đặt $x_i^1(n)_{N_1} = 0$ với $N_1 < n \le N-1$.

Tính trực tiếp DFT khi độ dài N lớn mất nhiều thời gian, ngay cả khi dùng máy tính. Để rút ngắn thời gian tính DFT, người ta xây dựng các thuật toán tính DFT nhanh, chúng được viết tắt theo tiếng Anh là FFT (*Fast Fourier Transforn).* Có rất nhiều thuật toán FFT khác nhau, nhưng với bài toán đặt ra ở đây, ta sử dụng hàm fft của Matlab, được tạo nên dựa trên một thư viện tên FFTW. [10]

3.2.5. Ngân hàng bộ lọc theo thang đo mel (Mel-scaled filter banks)

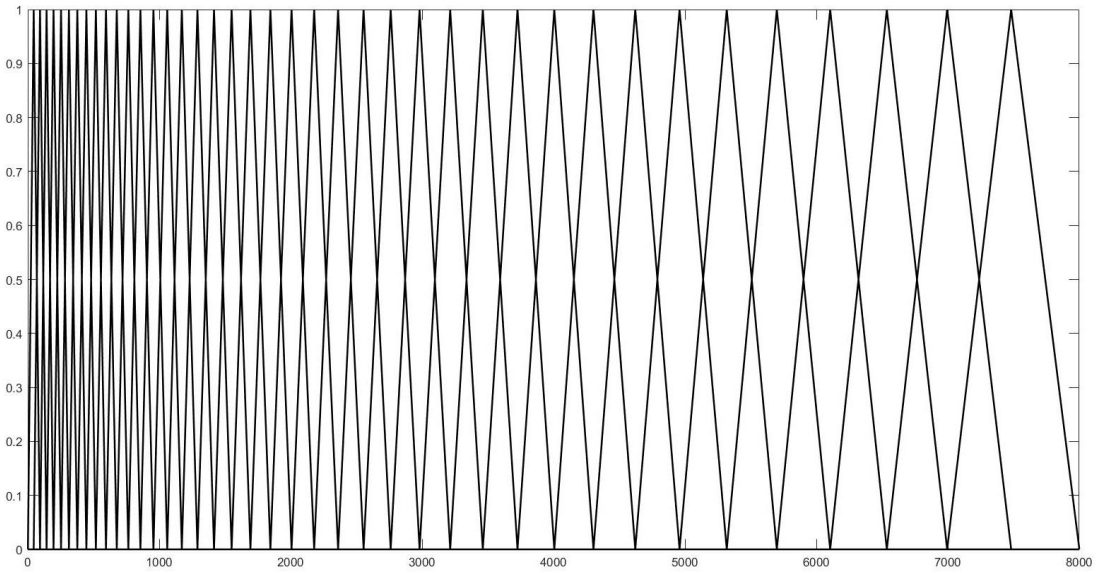
Phương pháp phân tích ngôn ngữ theo thang đo mel được sử dụng rộng rãi bởi tính hiệu quả của nó. Phương pháp này được xây dựng dựa trên sự cảm nhận của tai người đối với các dải tần số khác nhau. Với các tần số thấp (dưới 1000 Hz), độ cảm nhận của tai người là tuyến tính. Đối với các tần số cao, độ biến thiên tuân theo hàm logarit. Các băng lọc tuyến tính ở tần số thấp và biến thiên theo hàm logarit ở tần số cao được sử dụng để trích chọn các đặc trưng âm học quan trọng của tiếng nói.

Chúng ta có thể chuyển đổi giữa thang đo Hertz (f) và Mel (m) sử dụng các phương trình sau:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{3.5}$$

$$f = 700\left(10^{\frac{m}{2595}} - 1\right) \tag{3.6}$$

Một phương pháp để chuyển đổi sang thang mel là sử dụng băng lọc, trong đó mỗi bộ lọc có đáp ứng tần số dạng tam giác. Số băng lọc sử dụng thường là 40. Theo Nyquist, dải tần số được chọn thường sẽ bằng một nửa tần số lấy mẫu giọng nói, tức là từ 0 đến 8000 Hz. Ta có được biểu diễn của các bộ lọc như sau:



Hình 3.7. Biểu diễn 40 bộ lọc theo tần số Hz

Để có được các bộ lọc này, ta thực hiện theo các bước sau:

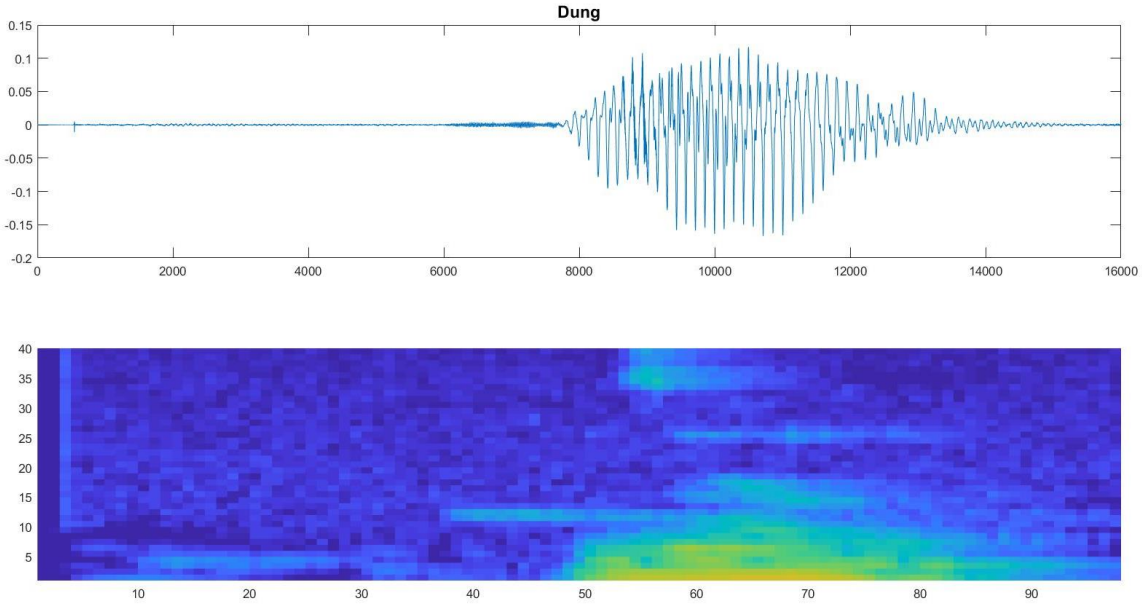
- Sử dụng phương trình (3.5) để chuyển đổi giá trị tần số lớn nhất và nhỏ nhất sang thang đo Mel. Ở trường hợp này, ta có 0 Hz là 0 Mels và 8000Hz là 2840 Mels.
- Do ta cần 40 bộ lọc đều nhau theo thang đo Mel nên ta cần tổng cộng là 42 điểm cách đều nhau từ 0 Mels đến 2840 Mels. Bằng tính toán ta được 42 điểm này như trong phụ lục [3].

- Sử dụng phương trình (3.6) để tính ngược lại thành 42 điểm theo thang đo Hertz, ta được 42 điểm như trong phụ lục [4].
- Bây giờ, để tạo các bộ lọc tam giác, ta sử dụng phương trình sau:

$$H_m(k)=\begin{cases} 0 & k<f(m-1) \\ \dfrac{k-f(m-1)}{f(m)-f(m-1)} & f(m-1)\le k<f(m) \\ 1 & k=f(m) \\ \dfrac{f(m+1)-k}{f(m+1)-f(m)} & f(m)<k\le f(m+1) \\ 0 & k>f(m+1) \end{cases} \tag{3.7}$$

Với $M=40$ là số bộ lọc cần tính toán, và $f(m)$ là $M+2$ tần số đã được chia đều ở bước trên.

Áp dụng ngân hàng bộ lọc này với tín hiệu cần xử lý, ta được quang phổ cuối cùng như sau:



Hình 3.8. Sóng âm và quang phổ của một mẫu ghi âm "dừng"

Quang phổ này chính là dữ liệu mà ta đưa trực tiếp vào mạng CNN để nhận dạng các lệnh.

3.3. Thiết kế và huấn luyện mạng CNN

Đầu vào của mạng sẽ là các ảnh quang phổ đã được xử lý ở mục 3.2, với kích thước ảnh là [98 40].

Tổng số mẫu đưa vào mạng là hơn 8000 mẫu, trong đó có:

- 500 mẫu mỗi lệnh cần nhận dạng: “tiến”, “lùi”, “trái”, “phải”, “dừng”.
- 1500 mẫu âm thanh môi trường được tách ra từ các đoạn ghi âm nhiễu, môi trường thực tế. Các mẫu này sẽ giúp mạng nhận biết được khi nào đang có giọng nói và khi nào không.
- 4000 mẫu bao gồm các từ tiếng Anh được cung cấp bởi TensorFlow. Các mẫu này giúp mạng có thể phân biệt được 5 lệnh cần nhận dạng với các từ khác không phải lệnh.

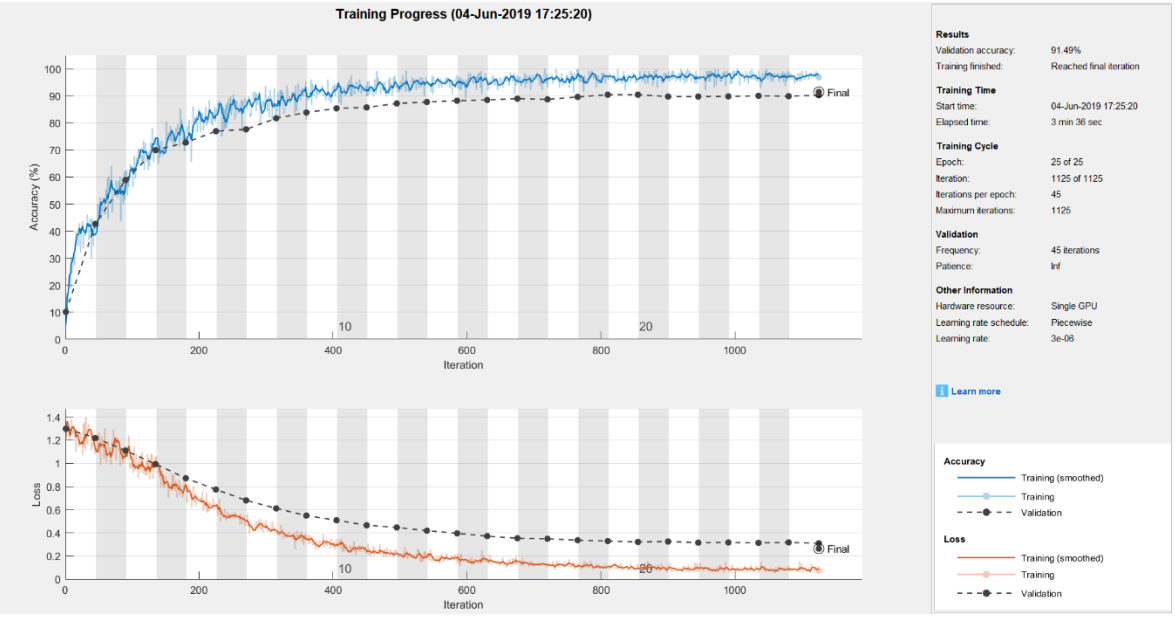
Chia 8000 mẫu huấn luyện thành hai tập:

- Tập huấn luyện: Với 80% số lượng mẫu huấn luyện trong mỗi thư mục, được chọn ngẫu nhiên.
- Tập xác nhận mạng trong khi huấn luyện: Với 20% số lượng mẫu huấn luyện trong mỗi thư mục, được chọn ngẫu nhiên.

Ta huấn luyện mạng với số chu kì tối đa là 25, tốc độ học là 0.00003, sử dụng phương pháp tối ưu Adam.

Chi tiết các lớp mạng được thể hiện trong phụ lục [2].

3.4. Kết quả huấn luyện



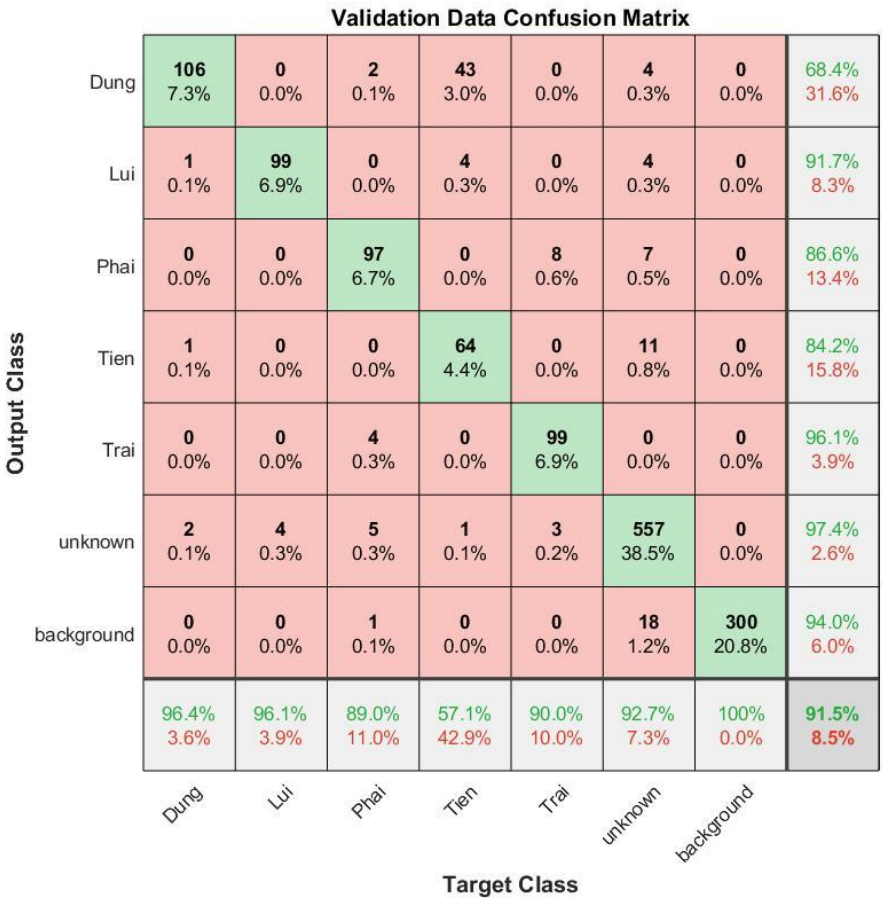
Hình 3.9. Kết quả huấn luyện mạng CNN

Ta thấy độ chính xác của mạng CNN là 91.49% sau 25 chu kì huấn luyện.

Mạng có 7 đầu ra bao gồm:

- Tiến
- Lùi
- Trái
- Phải
- Unknown (Những từ không nằm trong nhóm trên)
- Background (Âm thanh môi trường)

Kết quả của tập kiểm tra đối với mạng trong khi huấn luyện được biểu diễn chi tiết thông qua hình sau:



Hình 3.10. Bảng kết quả kiểm tra với tập xác nhận mạng

3.5. Kết luận chương

Trong chương vừa qua, chúng ta đã tạo ra được một mạng CNN nhận dạng được khá chính xác các lệnh đã đặt ra, đồng thời nhận dạng tốt những giọng mới chưa được đưa vào huấn luyện. Từ đó có thể áp dụng vào điều khiển xe hai bánh tự cân bằng. Tuy nhiên hiện tại chưa có mẫu giọng nữ nên mạng vẫn chưa nhận dạng được. Vì vậy để tăng độ chính xác, ta cần thu thập thêm nhiều mẫu từ nhiều người nhiều vùng miền và các giới tính khác nhau.

CHƯƠNG 4. KẾT LUẬN VÀ KIẾN NGHỊ

Trong đồ án tốt nghiệp này, cùng với sự hướng dẫn của *TS. Nguyễn Hoài Nam* chúng em đã hoàn thành được các nhiệm vụ của đồ án:

- Đưa ra một phương pháp nhận dạng hệ thống đã được đề xuất dựa trên cơ sở mạng CNN và mạng hồi quy. Phương pháp có thể nhận dạng nhiều loại đối tượng khác nhau từ bậc nhất đến bậc ba. Phương pháp đã được kiểm chứng thông qua mô phỏng và thực nghiệm.
- Mạng CNN nhận dạng được tốt bậc các đối tượng với tỉ lệ chính xác tương đối cao. Nhận dạng được tốt tham số với bộ tham số lý tưởng.
- Với nhận diện giọng nói, ta đã nhận dạng được khá chính xác các lệnh đã đặt ra với cả những giọng chưa được huấn luyện.

Những tồn tại và kiến nghị, đề xuất cải tiến đồ án:

- Phương pháp có thể được mở rộng cho các lớp đối tượng khác như bậc cao hơn, đối tượng có tích phân hoặc không ổn định. Độ chính xác có thể được cải thiện thêm nữa bằng cách thu thập thêm các mẫu nhận dạng.
- Nếu hằng số thời gian trễ quá nhỏ, hoặc với hằng số T lớn thì khó có thể nhận dạng được đúng bậc và loại mô hình. Để cải thiện, ta có thể sử dụng các ảnh mẫu với kích thước và độ phân giải lớn hơn, đưa ảnh về dạng binary giúp mạng xác định được bậc của mô hình tốt hơn.
- Ở bài toán nhận dạng tham số nếu có nhiễu tác động lớn làm cho kết quả đo $\{u_k\};\{y_k\}$ phản ánh không được sát thực tín hiệu hiện có của đối tượng. Điều này làm ảnh hưởng trực tiếp tới việc huấn luyện các trọng số IW và LW của mạng. Ta có thể phát triển xác định thành phần trễ n_k của đối tượng có thành phần trễ chính xác hơn so với hiện tại vẫn đang sử dụng hàm delayest của Matlab.
- Thu thập thêm dữ liệu lệnh để giúp tăng độ chính xác của mạng CNN nhận dạng giọng nói, đồng thời áp dụng mạng này vào việc điều khiển xe.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] **Nguyễn Hoài Nam**, "*Bài giảng điều khiển mờ và mạng nơron*".

[2] **Nguyễn Doãn Phước**, "*Lý thuyết điều khiển tuyến tính*", Nxb Khoa học và kỹ thuật.

[3] **Nguyễn Doãn Phước**, "*Tối ưu hóa trong điều khiển và điều khiển tối ưu*", Nhà xuất bản Bách Khoa Hà Nội, 2015.

[4] **Nguyễn Doãn Phước, Phan Xuân Minh**, "*Nhận dạng hệ thống điều khiển*", Nxb Khoa học và Kỹ thuật, 2001.

[5] **Vũ Hữu Tiệp**, "*Machine Learning cơ bản*", eBook, 2018.

[6] **Frigo, M., and S. G. Johnson**., "FFTW: An Adaptive Software Architecture for the FFT," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, 1998, pp. 1381 - 1384.

[7] **Mark Hudson Beale, Martin T.Hagan, Howard B. Demuth**, "*Neural Network Toolbox User's Guide*".

[8] **Martin T. Hagan, Howard B. Demuth, Mark Hudson Beale, Orlando De Jesús**, "*Neural Network Design 2nd Edition*".

[9] **W. Pete**, "*Speech Commands: A public dataset for single-word speech recognition*," 2017.

[10] "TensorFlow", Google, [Online]. Available: https://www.tensorflow.org/.

[11] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

PHỤ LỤC

[1] Ta sử dụng mạng neural bao gồm 16 lớp:

```
layers = [
    imageInputLayer(imgsize)

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    dropoutLayer(0.2)
    fullyConnectedLayer(numLabels)
    softmaxLayer
    classificationLayer];
```

[2] Ta sử dụng mạng với 23 lớp, các lớp được định nghĩa như trong mục 1.2.3:

```
layers = [
    imageInputLayer(imageSize)
```

```
convolution2dLayer(3,numF,'Padding','same')
batchNormalizationLayer
reluLayer

maxPooling2dLayer(3,'Stride',2,'Padding','same')

convolution2dLayer(3,2*numF,'Padding','same')
batchNormalizationLayer
reluLayer

maxPooling2dLayer(3,'Stride',2,'Padding','same')

convolution2dLayer(3,4*numF,'Padding','same')
batchNormalizationLayer
reluLayer

maxPooling2dLayer(3,'Stride',2,'Padding','same')

convolution2dLayer(3,4*numF,'Padding','same')
batchNormalizationLayer
reluLayer
convolution2dLayer(3,4*numF,'Padding','same')
batchNormalizationLayer
reluLayer

maxPooling2dLayer([1 timePoolSize])

dropoutLayer(dropoutProb)
fullyConnectedLayer(numClasses)
softmaxLayer];
```

[3]

- 42 điểm cách đều theo thang đo Mel đã tính toán là:

0.00	69.27	138.54	207.80	277.07	346.34
415.61	484.88	554.15	623.41	692.68	761.95
831.22	900.49	969.76	1039.02	1108.29	1177.56
1246.83	1316.10	1385.37	1454.63	1523.90	1593.17
1662.44	1731.71	1800.98	1870.24	1939.51	2008.78
2078.05	2147.32	2216.59	2285.85	2355.12	2424.39
2493.66	2562.93	2632.20	2701.46	2770.73	2840.00

[4]

- 42 điểm cách đều theo thang đo Herzt đã tính toán là:

0.00	44.37	91.56	141.74	195.10	251.84
312.18	376.34	444.57	517.12	594.28	676.32
763.57	856.35	955.01	1059.92	1171.48	1290.12
1416.27	1550.43	1693.08	1844.78	2006.10	2177.64
2360.06	2554.04	2760.31	2979.67	3212.93	3460.97
3724.74	4005.23	4303.50	4620.67	4957.95	5316.62
5698.02	6103.59	6534.88	6993.51	7481.21	8000.00