

Trường Đại học Bách Khoa Hà Nội  
Viện Công nghệ Thông Tin và Truyền Thông



Đồ án Tốt nghiệp Đại học

# Áp dụng mô hình học End-to-End giả lập hành vi trong công nghệ xe tự lái

Nguyễn Trung Sơn

Hà Nội, 05/2019

Trường Đại học Bách Khoa Hà Nội  
Viện Công nghệ Thông Tin và Truyền Thông



Đồ án Tốt nghiệp Đại học

# Áp dụng mô hình học End-to-End giả lập hành vi trong công nghệ xe tự lái

Sinh viên thực hiện      Nguyễn Trung Sơn

Người hướng dẫn      PGS.TS. Nguyễn Khanh Văn

Hà Nội, 05/2019

# Lời cam kết

Họ và tên sinh viên: Nguyễn Trung Sơn. ....

Điện thoại liên lạc: 0345418307 ..... Email: trungson077@gmail.com

Lớp: CNTT 2-4. .... Hệ đào tạo: Chính Quy. ....

Tôi – *Nguyễn Trung Sơn* – cam kết Đồ án Tốt nghiệp (ĐATN) là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *PGS.TS. Nguyễn Khanh Văn*. Các kết quả nêu trong ĐATN là trung thực, là thành quả của riêng tôi, không sao chép theo bất kỳ công trình nào khác. Tất cả những tham khảo trong ĐATN – bao gồm hình ảnh, bảng biểu, số liệu, và các câu từ trích dẫn – đều được ghi rõ ràng và đầy đủ nguồn gốc trong danh mục tài liệu tham khảo. Tôi xin hoàn toàn chịu trách nhiệm với dù chỉ một sao chép vi phạm quy chế của nhà trường.

*Hà Nội, ngày 27 tháng 5 năm 2019*

Tác giả ĐATN

*Nguyễn Trung Sơn*

# Lời cảm ơn

Lời đầu tiên, em xin gửi lời cảm ơn chân thành và sâu sắc đến thầy PGS.TS Nguyễn Khanh Văn giảng viên của bộ môn Công Nghệ Phần Mềm, viện Công Nghệ Thông Tin và Truyền Thông trường Đại học Bách Khoa Hà Nội, người đã tận tình hướng dẫn, chỉ bảo em trong suốt quá trình làm đồ án tốt nghiệp.

Đồng thời, em cũng gửi lời cảm ơn đến các thầy cô giáo trường Đại học Bách Khoa Hà Nội nói chung và các thầy cô trong viện Công Nghệ Thông Tin và Truyền Thông nói riêng đã tận tình dạy bảo cho em trong suốt quá trình học tập tại trường.

Cuối cùng, em xin bày tỏ lòng biết ơn đến gia đình, người thân cũng như bạn bè đã giúp đỡ và động viên em trong suốt quá trình học tập cũng như làm đồ án này.

Do kiến thức bản thân còn hạn chế, nên đồ án còn có nhiều thiếu sót. Vì vậy, em kính mong nhận được sự chỉ bảo và đóng góp ý kiến của thầy cô cũng như các bạn.

Bách Khoa, tháng 5 năm 2019

Em xin chân thành cảm ơn !

# Tóm tắt

Với sự phát triển của khoa học hiện đại, những chiếc xe ô tô không người lái tưởng chừng chỉ có trong những bộ phim khoa học viễn tưởng, giờ đây hoàn toàn có thể trở thành hiện thực. Công nghệ này vẫn đang được thử nghiệm và hứa hẹn là một công cụ hữu ích giúp các lái xe có thể an toàn hơn. Trong cuộc sống thường ngày, những tai nạn giao thông có thể đến bất cứ lúc nào, chỉ một phút sơ sẩy không chú ý của người điều khiển.

Các công ty lớn như Tesla , FPT , Apple, Ford , Google... đang đổ xô đi tìm hướng và chạy theo công nghệ cho tương lai của xe tự lái. Trong vòng chưa đầy 60 năm, nhân loại đã đi từ chỗ không có máy bay đến việc phóng phi thuyền đầu tiên lên Mặt Trăng. Trong vòng 20 năm, con người cũng đã đi từ chỗ sáng tạo ra những website thô sơ đến việc sử dụng trí tuệ nhân tạo để đánh bại cao thủ cờ vây thế giới. Công nghệ phát triển nhanh đến chóng mặt, và một sự đột phá tương tự có thể sớm được nhìn thấy ở những chiếc ô tô tự lái.

Có rất nhiều yếu tố công nghệ góp mặt vào trong sự ra đời của Công nghệ Xe tự lái như: Công nghệ nhận diện làn đường, các vật thể như đèn giao thông, các phương tiện khác và người đi bộ được xây dựng trên nền tảng OpenCV, công nghệ tích hợp lái điện sử dụng các vi mạch như Arduino, Raspberry Pi, giúp điều khiển quá trình đánh lái của xe theo những yêu cầu của vấn đề trên, công nghệ sử dụng các cảm biến để tạo ra các trường hợp cho xe tự di chuyển. Trong đồ án này em đã tìm hiểu và phát triển đề tài “Áp dụng mô hình học End-to-End giả lập hành vi trong công nghệ xe tự lái”. Nó là một module khá quan trọng trong công nghệ này, để xe tự lái có thể ra quyết định rẽ các hướng khi đi trên đường.

# Mục lục

Lời cam kết .....	iii
Lời cảm ơn .....	iv
Tóm tắt.....	v
Mục lục.....	vi
Danh mục hình vẽ.....	ix
Danh mục công thức.....	xiii
Danh mục các từ viết tắt.....	xiv
Danh mục thuật ngữ.....	xv
Chương 1 Giới thiệu đề tài.....	1
1.1 Đặt vấn đề .....	1
1.2 Mục tiêu và phạm vi đề tài.....	1
1.3 Định hướng giải pháp.....	2
1.4 Bố cục đồ án.....	3
Chương 2 Khảo sát và tổng quan .....	4
2.1 Khảo sát hiện trạng .....	4
2.2 Tổng quan về xe tự lái .....	5
2.2.1 Xu hướng và sự phát triển.....	5
2.2.2 Cách thức hoạt động của xe tự lái .....	6

2.2.3 Ứng dụng tại Việt nam [2] .....	7
<b>2.3 Phương pháp giả lập hành vi trong xe tự lái .....</b>	<b>9</b>
2.3.1 Mô hình trực quan về giả lập hành vi .....	9
2.3.2 Hệ thống DAVE-2 giả lập.....	11
2.3.3 End-to-end trong kiến trúc Deep Neural Network .....	13
2.3.4 Mô hình học End-to-end training mạng CNN để giả lập hành vi .....	14
2.3.5 Ứng dụng của CNNs trong bài toán giả lập hành vi .....	14
<b>Chương 3 Công nghệ sử dụng.....</b>	<b>16</b>
<b>3.1 Mạng Neural.....</b>	<b>16</b>
3.1.1 Mạng neural sinh học .....	16
3.1.2 Mạng neural nhân tạo .....	17
3.1.3 Xây dựng mạng Neural.....	18
<b>3.2 Mạng Neural tích chập.....</b>	<b>19</b>
3.2.1 Giới thiệu về mạng Neural tích chập .....	19
3.2.2 Kiến thức về mạng Tích chập .....	20
3.2.3 Phép chập khối .....	27
3.2.4 Các kiến trúc của mạng Neural tích chập .....	30
3.2.5 Một số mô hình mạng Neural tích chập.....	32
<b>3.3 Một số kiến thức cơ sở.....</b>	<b>34</b>
3.3.1 Sai số toàn phương trung bình .....	34
3.3.2 Thuật toán tối ưu hóa Adam .....	35
3.3.3 Hàm kích hoạt ReLU và ELU .....	38
<b>Chương 4 Mô hình và các kỹ thuật trong việc xử lý bài toán giả lập hành vi ..</b>	<b>41</b>
<b>4.1 Xây dựng bài toán .....</b>	<b>41</b>

<b>4.2 Cân bằng dữ liệu.....</b>	<b>43</b>
<b>4.3 Các kỹ thuật xử lý ảnh tăng cường.....</b>	<b>45</b>
<b>4.4 Huấn luyện với CNN .....</b>	<b>50</b>
4.4.1 Thử nghiệm 1 .....	51
4.4.2 Thử nghiệm 2 .....	51
4.4.3 Thử nghiệm 3 .....	53
4.4.4 Mô hình cuối .....	53
<b>4.1 Xây dựng demo .....</b>	<b>58</b>
4.1.1 Thư viện và công cụ sử dụng .....	58
4.1.2 Kết quả đạt được .....	59
<b>4.2 Triển khai .....</b>	<b>60</b>
<b>Chương 5 Thực nghiệm và đánh giá kết quả .....</b>	<b>61</b>
<b>Chương 6 Kết luận và hướng phát triển .....</b>	<b>62</b>
<b>Tài liệu tham khảo.....</b>	<b>63</b>



# Danh mục hình vẽ

<b>Hình 1</b> Mô tả cách thức xe tự lái .....	4
<b>Hình 2</b> Dịch vụ cho thuê xe tự lái.....	6
<b>Hình 3</b> Xe tự lái Tesla model X.....	7
<b>Hình 4</b> Xe tự lái đầu tiên tại Việt Nam của công ty FPT .....	8
<b>Hình 5</b> Hệ thống thu thập dữ liệu DAVE-2 [3].....	9
<b>Hình 6</b> Mô hình dữ liệu huấn luyện DAVE-2 [3] .....	10
<b>Hình 7</b> Giao diện màn hình training .....	11
<b>Hình 8</b> Góc lái giữa do camera ghi lại .....	12
<b>Hình 9</b> Góc lái bên phải do camera ghi lại.....	12
<b>Hình 10</b> Góc lái bên trái do camera ghi lại .....	13
<b>Hình 11</b> Mô hình học End-to-end.....	13
<b>Hình 12</b> Mạng được đào tạo được sử dụng để tạo các lệnh lái từ một camera trung tâm phía trước.....	14
<b>Hình 13</b> Các đặc trưng của một con đường trong CNNs .....	14
<b>Hình 14</b> Đặc trưng của khu rừng qua CNNs .....	15
<b>Hình 15</b> Mạng Neural sinh học .....	16
<b>Hình 16</b> Mạng neural nhân tạo [4].....	17
<b>Hình 17</b> Kiến trúc một mạng Neural [4] .....	18
<b>Hình 18</b> Bộ lọc được sử dụng trong lớp tích chập đầu tiên là các ma trận kích thước <b>3x3</b> [5] .....	21

<b>Hình 19</b> Nhân chập bộ lọc F1 với ma trận ảnh đầu vào của số 7 [5] .....	22
<b>Hình 20</b> Nhân với filter F1 [5] .....	23
<b>Hình 21</b> Ví dụ về bộ lọc cạnh với đầu vào là ảnh số viết tay. [5] .....	23
<b>Hình 22</b> Ma trận đầu vào được bao quanh bởi đường viền phụ kích thước p (giá trị 0). [5] .....	24
<b>Hình 23</b> Nhân chập với bước sai (trượt) $s = 2$ .....	26
<b>Hình 24</b> Phép nhân chập khối - áp dụng cho ảnh màu RGB kích thước 6x6 .....	27
<b>Hình 25</b> Ba lớp của bộ lọc có thể được cấu hình khác nhau để phát hiện đặc trưng trên một, hai hoặc cả ba kênh màu của ảnh đầu vào. [5].....	28
<b>Hình 26</b> Hai bộ lọc kích thước 3x3x3 được sử dụng để phát hiện đồng thời cạnh đứng và ngang của ảnh đầu vào (hệ RGB). [4] .....	29
<b>Hình 27</b> Kiến trúc của một lớp [4] .....	30
<b>Hình 28</b> Các ký hiệu cơ bản của một CNN [4] .....	31
<b>Hình 29</b> Ví dụ một CNN cơ bản được sử dụng cho bài toán phân loại ảnh (“mèo” hay “không phải mèo”).....	32
<b>Hình 30</b> Densenet(2016) .....	32
<b>Hình 31</b> Các block của Densenet.....	33
<b>Hình 32</b> LeNet(1998).....	34
<b>Hình 33</b> sự khác như khi có và không có momentum .....	36
<b>Hình 34</b> Sự khác nhau giữa khi ta updata lên Nesterov Mommentum.....	36
<b>Hình 35</b> Đồ thị hàm ReLU .....	38
<b>Hình 36</b> Đồ thị hàm ELU .....	39
<b>Hình 37</b> Thông số của tập nhãn.....	42
<b>Hình 38</b> Sơ đồ dữ liệu trong file driving_log .....	43

<b>Hình 39</b> Dữ liệu sau khi được được lọc .....	44
<b>Hình 40</b> Xử lý khi tăng góc lái bên phải và giảm góc lái bên trái.....	44
<b>Hình 41</b> Biểu đồ dữ liệu sau khi được cân bằng và tách thành 2 tập train và valid .....	45
<b>Hình 42</b> kỹ thuật Zooming .....	45
<b>Hình 43</b> Kỹ thuật Panning.....	46
<b>Hình 44</b> Kỹ thuật Flipping .....	46
<b>Hình 45</b> Kỹ thuật điều chỉnh độ sáng .....	47
<b>Hình 46</b> Tổng hợp các kỹ thuật tăng cường ảnh .....	47
<b>Hình 47</b> Cấu trúc RGB sang YUV .....	48
<b>Hình 48</b> RGB sang YUV .....	48
<b>Hình 49</b> Hình ảnh của tập training đã được kết hợp.....	49
<b>Hình 50</b> Hình ảnh của tập validation sau khi kết hợp.....	49
<b>Hình 51</b> Sơ đồ các kỹ thuật xử lý ảnh tăng cường .....	49
<b>Hình 52</b> Kiến trúc CNN của NVIDIA paper [3] .....	50
<b>Hình 53</b> Kết quả mô hình thử nghiệm một .....	51
<b>Hình 54</b> Kết quả mô hình thử nghiệm hai.....	52
<b>Hình 55</b> Kết quả mô hình thử nghiệm 3 .....	53
<b>Hình 56</b> Kết quả mô hình cuối .....	54
<b>Hình 57</b> Kiến trúc mô hình cuối.....	55
<b>Hình 58</b> Đặc trưng khi mạng học tìm đường lớp Conv thứ nhất .....	56
<b>Hình 59</b> Đặc trưng khi mạng học tìm đường lớp Conv thứ 2 .....	57

# Danh mục bảng

<b>Bảng 1</b> Ba góc của camera .....	41
<b>Bảng 2</b> Thông số các số lần thử nghiệm .....	58
<b>Bảng 3</b> Các công cụ sử dụng .....	59
<b>Bảng 4</b> Tiêu chí thống kê .....	59

# Danh mục công thức

<b>Công thức 1</b> MSE.....	35
<b>Công thức 2</b> Adagrad .....	36
<b>Công thức 3</b> Tổng quán công thức Adadelta [7].....	37
<b>Công thức 4</b> RMSprop [7] .....	37
<b>Công thức 5</b> Công thức tối ưu hóa Adam [7] .....	38
<b>Công thức 6</b> ReLU và đạo hàm của ReLU .....	38
<b>Công thức 7</b> Hàm ELU [1] .....	40

# Danh mục các từ viết tắt

<b>CAN</b>	Controller Area Network Mạng điều khiển khu vực
<b>NN</b>	Neural Network Mạng lưới thần kinh
<b>lr</b>	Learning rate Tỷ lệ học
<b>CNN</b>	Convolutional neural network Mạng tích chập
<b>FNN</b>	Feed-forward neural network Mạng neural truyền thẳng
<b>RNN</b>	Recurrent neural network Mạng neural hồi quy
<b>Conv</b>	Convolution Tích chập
<b>FC</b>	Fully connected Lớp kết nối đầy đủ

# Danh mục thuật ngữ

<b>End-to-end</b>	Mô hình học không có sự can thiệp của con người của deep learning
<b>Thị giác máy tính</b>	Một lĩnh vực bao gồm các phương pháp thu nhận, xử lý ảnh kỹ thuật số, phân tích và nhận dạng các hình ảnh và, nói chung là dữ liệu đa chiều từ thế giới thực để cho ra các thông tin số hoặc biểu tượng, ví dụ trong các dạng quyết định.
<b>Deep learning</b>	Học sâu, một lĩnh vực nhỏ trong machine learning
<b>Giả lập hành vi</b>	Dựa vào những kinh nghiệm từ dữ liệu cho trước để tạo lại cũng như cải thiện lại hành vi của dữ liệu.
<b>Filter</b>	Mã trận để lọc cạnh của hình ảnh để ra một hình ảnh với những tính chất đặc trưng trong mạng CNNs
<b>DAVE-2 system</b>	Hệ thống thu thập và huấn luyện dữ liệu.

# Chương 1 Giới thiệu đề tài

## 1.1 Đặt vấn đề

Xe không người lái đang là một trong những xu thế phát triển chủ đạo của ngành công nghiệp xe hơi trong vài năm qua. Tuy nhiên, xuất phát điểm của loại phương tiện này thì đã có từ gần một thế kỷ trước, nghĩa là trước cả cuộc cách mạng công nghiệp lần thứ ba. Con người chúng ta đã đạt được những bước tiến rất dài, từ những công nghệ điều khiển từ xa bằng sóng vô tuyến ban đầu cho tới việc áp dụng các hệ thống lidar, radar hay cảm biến thông minh trên xe tự lái ở thời điểm hiện tại. Đích đến của công nghệ này là giảm thiểu ùn tắc giao thông và nâng cao độ an toàn cũng như sự tiện lợi cho người sử dụng.

Khi những chiếc xe tự lái không cần nhiều không gian để di chuyển và đỗ, đường phố sẽ có nhiều chỗ hơn cho người đi bộ và người đi xe đạp với các làn sang đường hoặc làn chuyên dụng. Khi mà tầm nhìn không còn là vấn đề, các cải tiến khác cho người đi bộ như tăng bóng râm hoặc lối đi có rào che có thể được bổ sung.

Chính vì thế đã có những kỹ thuật, những thuật toán luôn được cải thiện để cho xe tự lái càng ngày càng tốt hơn và hoàn thiện hơn, một trong những kỹ thuật đó là giúp cho xe có khả năng tự lái trên các khung đường mà không cần đến con người điều khiển. Nhưng có hàng ngàn giải thuật và mô hình được áp dụng, vậy giải thuật nào và mô hình nào là tốt nhất?

Ta sẽ sang phần 1.2 sau đây để tìm hiểu về điều này.

## 1.2 Mục tiêu và phạm vi đề tài

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



Phần lớn những chiếc xe tự lái hiện nay đều sử dụng hệ thống bao gồm hàng loạt camera và cảm biến, bên cạnh các công cụ chỉ đường và vô số chương trình máy tính. Tuy nhiên cách tiếp cận như vậy đã đạt tới giới hạn của nó. Trên thực tế, những chiếc xe tự lái do các công ty lớn như Google phát triển, đã đạt được một số thành tựu nhất định, song chưa đủ tốt để có thể trở nên thông dụng, và cũng chưa đủ thông minh để tự làm chủ trong vô số các trường hợp có thể xảy đến trong quá trình lưu thông. Vì vậy, điều cần thiết ở đây là một hệ thống thông minh hơn, nhưng không cần trang bị thêm các cảm biến hoặc nhiều chương trình.

Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến sẽ giúp cho chúng ta xây dựng được hệ thống thông minh với độ chính xác cao như vậy. CNNs giúp chúng ta trích chọn những đặc trưng của ảnh rồi từ đó phát hiện vật thể mới qua những đặc trưng đó dựa vào “kinh nghiệm” từ những đặc trưng cũ. Chúng ta sẽ tìm hiểu kỹ hơn về CNNs ở Chương 3 Phần 3.2

Mục tiêu cuối cùng của đồ án này là áp dụng các nguyên tắc học sâu từ CNNs kết hợp với các kỹ thuật xử lý hình ảnh để dạy một chiếc xe ô tô có khả năng tự lái trong một ứng dụng lái xe mô phỏng mà không cần bất cứ sự can thiệp nào từ con người ở bên ngoài, đây cũng chính là end-to-end learning: xu hướng mới của các mô hình machine learning hiện đại (2.3.2).

### **1.3 Định hướng giải pháp**

Từ những phân tích ở trên, trong đồ án này em đã tìm hiểu và áp dụng phương pháp End-to-end learning(2.3.4) sử dụng hệ thống DAVE-2 từ NVIDIA paper [1] và trình mô phỏng của UDACITY.

Giả lập hành vi là một phương pháp mà các kỹ năng nhận thức của con người có thể được nắm bắt và tái tạo trong một chương trình máy tính. Khi chủ thể con người thực hiện kỹ năng, hành động của người đó được ghi lại cùng với tình huống làm phát sinh hành động. Dữ liệu của những thông số này được sử dụng làm đầu vào cho một chương trình học tập. Phương pháp này có thể được sử dụng để xây dựng các hệ thống điều khiển tự động cho các nhiệm vụ phức tạp mà lý thuyết điều khiển cổ điển không phù hợp. Nó cũng có thể được sử dụng cho đào tạo.

Phương pháp giả lập khá đơn giản, ta sẽ thu thập lại dữ liệu bằng cách lái xe trong chế độ training trong trình mô phỏng mà UDACITY cung cấp(2.3.2), trình mô phỏng có rất nhiều

điểm tương đồng giống như lái xe trong thế giới thật, hình ảnh được chụp trong môi trường mô phỏng (đường, điểm, phong cảnh) là một hình ảnh gần đúng có thể chụp được trong thế giới thực. Tiếp đó ta sẽ biểu diễn những dữ liệu thu thập được trên các biểu đồ, học tăng cường, sử dụng các kỹ thuật xử lý ảnh để tạo ra các điều kiện thời tiết, môi trường giả ...

Sau khi đã thu thập dữ liệu và xử lý xong đó ta sẽ dùng mạng CNNs để huấn luyện các dữ liệu này với nhau. Hệ thống tự động học các biểu diễn bên trong của các bước xử lý cần thiết như phát hiện các tính năng đường hữu ích chỉ với góc lái của con người làm tín hiệu huấn luyện. Kết quả cuối cùng của việc huấn luyện này là để dạy cho xe trong trình mô phỏng có “kinh nghiệm” tự lái dựa trên dữ liệu thu thập được của ta khi lái xe ở trên.

## **1.4 Bố cục đề án**

Phần còn lại của báo cáo đề án tốt nghiệp này được tổ chức như sau.

Chương 2, em sẽ nêu về các khảo sát và tổng quan hóa về từ lịch sử , ứng dụng , cách thức hoạt động của xe tự lái, phần sau nữa là phương pháp giả lập hành vi cho xe tự lái và tại sao nó lại quan trọng đến như vậy.

Chương 3, em sẽ nói về những kiến thức về mạng neural, mạng neural tích chập, các hàm kích hoạt ... em đã sử dụng trong phạm vi đề án của mình, tại sao lại phải dùng mô hình end-to-end mà ko dùng mô hình khác.

Sau khi đã tìm hiểu được các kiến thức để áp dụng thì Trong Chương 4, em sẽ nêu ra những kỹ thuật em đã sử dụng và tối ưu hóa dữ liệu, phân chia dữ liệu, dùng python Socketio để liên kết, cách sử dụng trình mô phỏng để tạo dữ liệu ....

Chương 5, em sẽ nói về những lần thực nghiệm và kết quả của mình khi làm đề án.

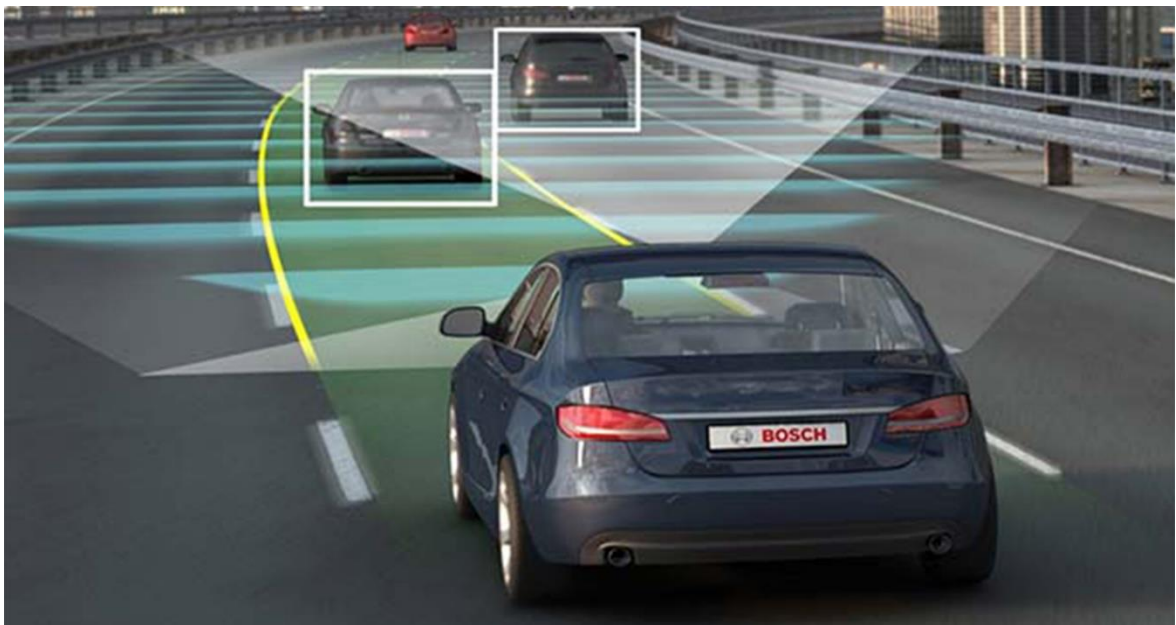
Chương 6, là chương cuối nêu ra những kết luận và hướng phát triển của em cho đề án của mình sau này.

## Chương 2 Khảo sát và tổng quan

Chương 2 sẽ trình bày tổng quan về công nghệ sử dụng trong xe tự lái, lịch sử, ứng dụng... Tại sao xe tự lái lại phát triển mạnh như vậy. Cùng với đó trong Chương 2 sẽ trình bày tóm tắt về mô hình trực quan của End-to-End và cách áp dụng nó vào trong bài toán giả lập hành vi quan trọng như thế nào.

### 2.1 Khảo sát hiện trạng

Hai thập kỉ về trước, xe không người lái (driverless car) là thứ người ta chỉ có thể thấy trên những bộ phim khoa học viễn tưởng. Nhưng với sự tiến bộ của công nghệ, giờ đây tính năng này đã trở nên khá phổ biến trên những mẫu xe hiện đại. [1]



**Hình 1** Mô tả cách thức xe tự lái

Tesla giới thiệu Autopilot lần đầu tiên trên mẫu sedan Model S vào năm 2015, đem đến cho các phương tiện khả năng tự vận hành trong một số hoàn cảnh nhất định. Năm ngoái, nhà sản xuất nâng cấp phần cứng trên tất cả các dòng xe nhằm hỗ trợ hệ thống tự lái mới nhất có tên gọi Enhanced Autopilot. Theo đó, những xe Tesla sản xuất từ tháng 10/2016 trở đi đều được trang bị 8 camera, đảm bảo phạm vi quan sát 360 độ xung quanh chiếc xe. Mỗi camera có tầm nhìn xa tới 250 mét, 12 cảm biến siêu âm giúp phương tiện phát hiện và tránh những chướng ngại vật cứng cũng như mềm. Với sự hỗ trợ từ hệ thống radar, xe Tesla có thể nhìn xuyên qua sương mù, mưa lớn, bụi bẩn, thậm chí một chiếc xe phía trước. Tất nhiên thứ hỗ trợ cho xe hệ thống xe tesla chính là dựa trên mô hình mạng End-to-End để giả lập hành vi sẽ trình bày tiếp ở phần sau của báo cáo này.

Hiện nay không có quá nhiều mạng hay mô hình áp dụng được cho xe tự lái vì công nghệ này mới nổi lên trong vài năm gần đây, ta có thể kể tên vài mạng và mô hình nổi bật như Recurrent neural network, các biến thể của mạng CNN do người dùng và người học tự phát triển trên quá trình học và nghiên cứu, end-to-end và các biến thể của nó ...

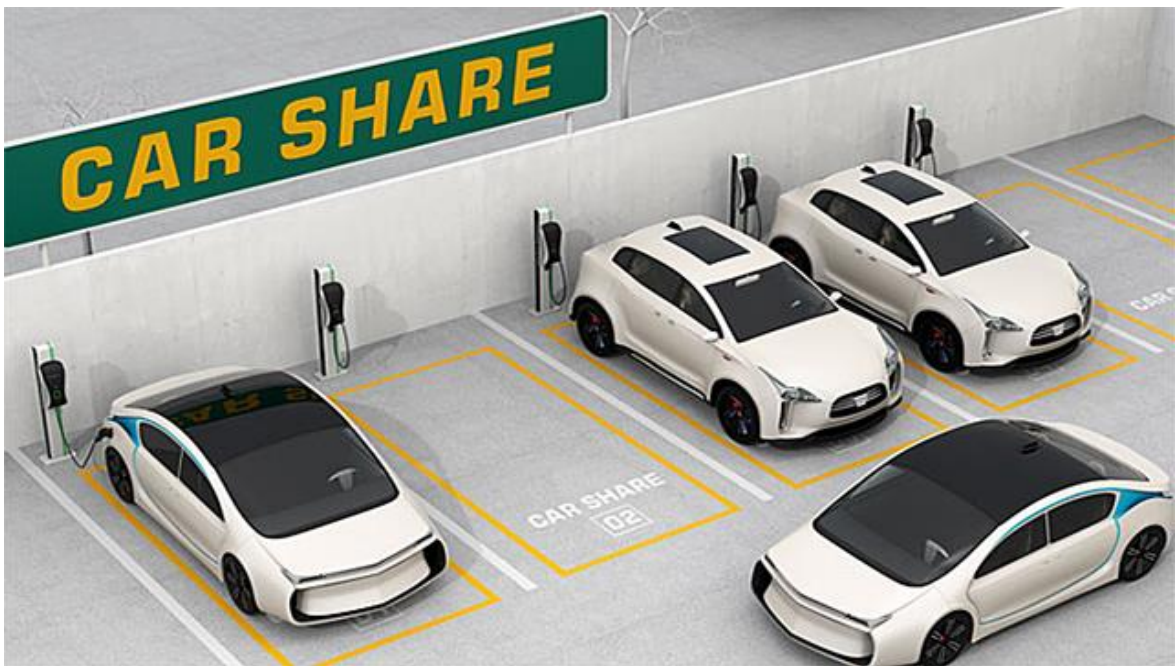
Từ những thực tế nêu trên em đã tìm hiểu và áp dụng mô hình mạng End-to-end của NVIDIA trong đồ án này để tạo ra một model tốt từ những dữ liệu, kinh nghiệm lái xe của con người để xe tự lái có thể học và mô phỏng lại theo cách tốt nhất có thể và hy vọng trong tương lai sẽ còn tiếp tục phát triển nó lên nữa .

## **2.2 Tổng quan về xe tự lái**

### **2.2.1 Xu hướng và sự phát triển**

Với sự phát triển xu hướng công nghệ 4.0, các tiện ích của việc đặt dịch vụ vận chuyển qua kênh trực tuyến, trong đó có công nghệ chia sẻ xe ngày càng phổ biến. Dịch vụ này đã xuất hiện ở nhiều quốc gia trên thế giới từ nhiều năm trước và được xem là một dịch vụ giao thông tiện ích cho các đô thị hiện đại, tiết kiệm chi phí và thân thiện với môi trường. Riêng tại thị trường Mỹ, các ứng dụng như Turo, Zipcar... rất phổ biến. Trong đó, Zipcar thành lập từ năm 2000, hiện đã có mặt tại 170 thành phố và 44 tiểu bang và hơn 300 trường đại học, cao đẳng... Với Turo, người dùng có thể dễ dàng thuê ô tô và di chuyển tại hơn 4.500 thành

phổ và hơn 300 sân bay tại Mỹ. Tại Nga, dịch vụ car sharing có xu hướng phát triển mạnh mẽ. Theo Bloomberg, cuối năm 2018 có hơn 16.500 ô tô cho thuê tại thủ đô Moscow, đưa thành phố này trở thành khu vực có lượng xe cho thuê lớn nhất châu Âu và lớn thứ nhì thế giới. Sở Giao thông Vận tải Moscow ước tính, con số này sẽ tăng thêm 5.000 chiếc mỗi năm. Số xe cho thuê bùng nổ cũng góp phần thúc đẩy lượt sử dụng dịch vụ. Con số đã tăng hơn 4 lần, chạm mức 23 triệu lượt. Chủ các hãng cho thuê xe tự lái cho rằng, dù mô hình này tại Nga xuất phát chậm hơn, nhưng nhờ đó họ có thể tận dụng những công nghệ mới nhất để phát triển nhanh chóng.



**Hình 2** Dịch vụ cho thuê xe tự lái

### 2.2.2 Cách thức hoạt động của xe tự lái

Các thiết bị cần thiết đối với xe tự lái gồm có camera, radar, đèn laser, định vị GPS. Theo đó, một chiếc ô tô tự lái cần một hệ thống có khả năng kết hợp GPS và thông tin từ cảm biến thành những hành động thực tế như lái xe, tăng tốc, bóp phanh.

Diễn hình như hai mẫu xe Model S và Model X của hãng Tesla đều được cập nhật hệ thống Autopilot mới, cho phép xe tự vận hành trên nhiều kiểu đường khác nhau. Với việc sử dụng hệ thống mới, xe Tesla đã thông minh hơn và nhận biết được nhiều từ môi trường xung quanh hơn. Hệ thống Autopilot bản cũ cho phép xe Tesla hoạt động tốt trong điều kiện giao

thông phải dừng và khởi động liên tục trên đường thẳng, còn bắt đầu kém dần ở đường cong hay đường đèo khi mà xe thường mất tín hiệu từ đường. Bản cập nhật đã khắc phục được điều đó. Mặc dù đã hoàn thiện hơn trước nhưng hệ thống Autopilot vẫn cần phải lập trình để thông minh hơn. Ví dụ như trong video dưới đây, chiếc xe có lúc dừng lại khi phát hiện người đi lại ở ven đường, có lúc tỏ ra bối rối trước phương tiện xung quanh và cũng phải phanh lại.



**Hình 3** Xe tự lái Tesla model X

### **2.2.3 Ứng dụng tại Việt nam [2]**

Hiện giao thông ở những thành phố lớn, đông đúc của Việt Nam như Hà Nội, thành phố Hồ Chí Minh vẫn còn là một thách thức để ứng dụng xe tự lái. Dự kiến, mất khoảng 10 năm nữa để cập nhật, phát triển công nghệ, xe tự lái mới có thể chạy trên đường phố ở Việt Nam. Tuy nhiên, trước mắt, xe tự lái được nghiên cứu để ứng dụng trong sân golf, trong kho hàng, di chuyển giữa các toà nhà trong khu công nghệ cao Hoà Lạc, hay trong sân bay bằng xe tự lái là điều hoàn toàn khả thi ở Việt Nam. Với sự bùng nổ của trí tuệ nhân tạo, nhiều hãng ô tô hiện đang mạnh tay đầu tư cho xe không người lái. Tuy nhiên, sân chơi này giờ không còn là của các hãng ô tô truyền thống mà đã dịch chuyển sang các công ty phần mềm khi 90%



sáng tạo của ô tô nằm ở đây. Và các nhà khoa học của Việt Nam đang bắt kịp rất nhanh với xu thế này.

Ở Việt Nam thì xe tự lái được bắt đầu xây dựng là từ năm ngoái tức năm 2018 do FPT đầu quân và được thử nghiệm tại TP. Hồ Chí Minh . **Hình 4** bên dưới chiếc ô tô tự lái do nhóm sinh viên một trường đại học nghiên cứu, thử nghiệm có khả năng di tự di chuyển với tốc độ 25 km/h, tự phát hiện và tránh các vật thể. Về phần mềm, chiếc xe được tích hợp và ứng dụng những công nghệ tiên tiến như path planning (thuật toán tìm đường, lên kế hoạch di chuyển),behavioral cloning (mô phỏng hành động của con người), thiết bị lidar, deep learning (học sâu)... Phần cứng chủ yếu mua từ các thương hiệu lớn như NVIDIA, camera của ZED hay lidar của Velodyne... tương tự phần cứng sử dụng trên các hệ thống của Tesla hay Google, Mercedes. Phần cứng mà FPT tự phát triển là hệ thống điều khiển vô-lãng. Ban đầu, nhóm nghiên cứu đã tác động vào mô tơ của hệ thống trợ lực lái điện nhưng bất thành, gây cháy mô tơ và ECU của xe. Một kỹ sư cho biết, hệ thống điều khiển vô-lãng phức tạp như hiện tại chỉ là tạm thời, sẽ được tối giản trong tương lai.

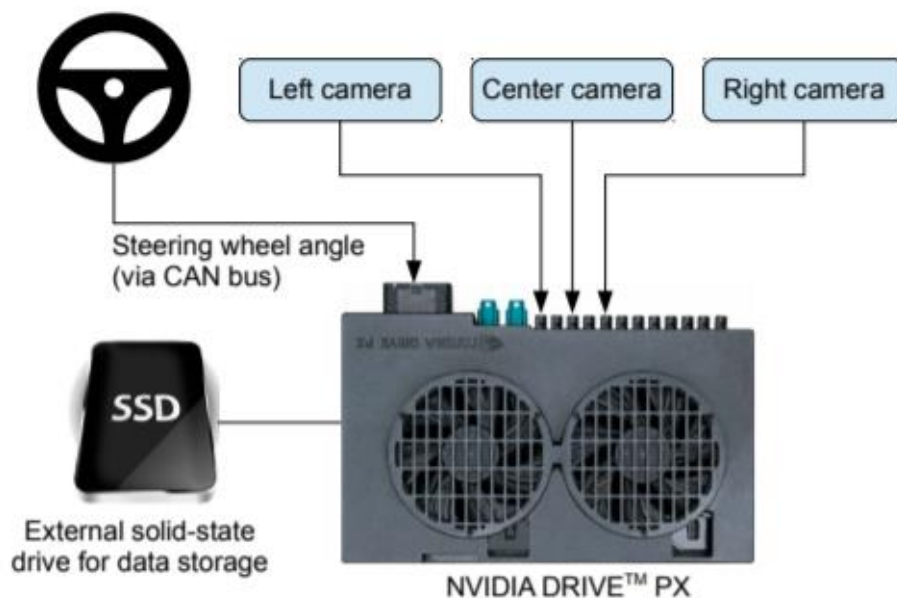


**Hình 4** Xe tự lái đầu tiên tại Việt Nam của công ty FPT

## 2.3 Phương pháp giả lập hành vi trong xe tự lái

Tổng quan trong phần này sẽ em nói về mô hình Dave-2 của NVIDIA phát triển cho xe tự lái, cách giả thích và mô hình của hệ thống Dave-2 quan trọng như thế nào trong giả lập hành vi trên kinh nghiệm của người lái.

### 2.3.1 Mô hình trực quan về giả lập hành vi



**Hình 5** Hệ thống thu thập dữ liệu DAVE-2 [3]

CNN ra đời đã tạo ra một cuộc cách mạng về nhận dạng các khuôn mẫu , Trước khi áp dụng rộng rãi các CNN, hầu hết các tác vụ nhận dạng mẫu được thực hiện bằng cách sử dụng giai đoạn ban đầu của tính năng trích xuất thủ công, sau đó là trình phân loại. Điểm đột phá của CNN là các tính năng được học tự động, từ các ví dụ đào tạo. Cách tiếp cận CNN đặc biệt mạnh mẽ trong các tác vụ nhận dạng hình ảnh vì hoạt động tích chập nắm bắt được bản chất 2D của hình ảnh. Ngoài ra, bằng cách sử dụng tích chập hạt nhân để quét toàn bộ hình ảnh, tương đối ít thông số cần phải học so với tổng số lượng phép tính. các

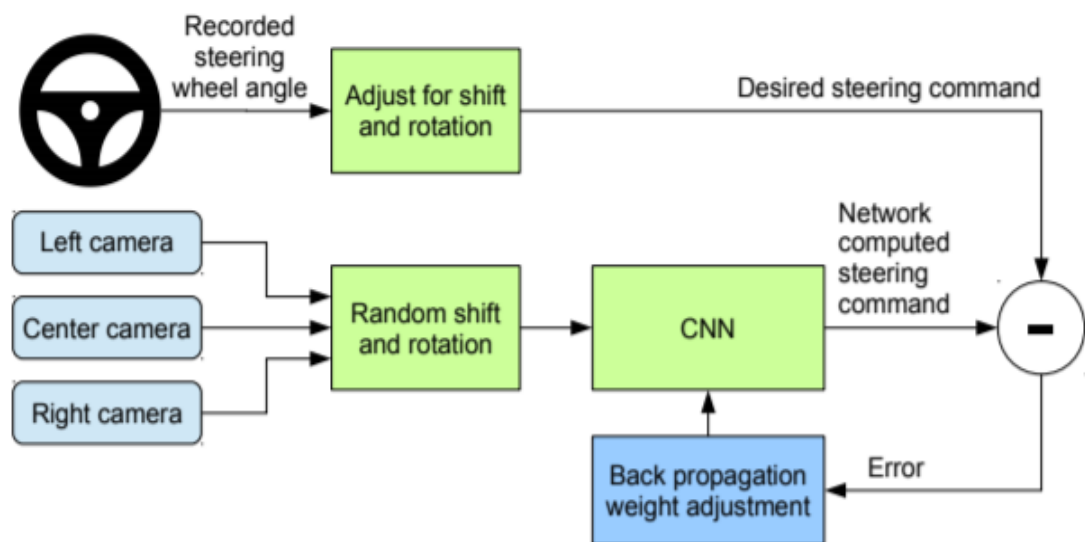


thuật toán học tập CNN đã được triển khai trên các đơn vị xử lý đồ họa song song (GPU) giúp tăng tốc đáng kể việc học và suy luận.

DAVE được đào tạo về giờ lái xe của con người trong những môi trường tương tự, nhưng không giống nhau. Dữ liệu huấn luyện bao gồm video từ hai camera kết hợp với tay lái trái và phải.

Theo nhiều cách, DAVE-2 được lấy cảm hứng từ công trình tiên phong của Pomerleau, người năm 1989 đã chế tạo ra xe tự lái trên mặt đất trong hệ thống Mạng thần kinh (ALVINN). Nó đã chứng minh rằng một mạng lưới thần kinh được đào tạo cuối cùng thực sự có thể điều khiển một chiếc xe hơi trên đường công cộng.

**Hình 6** cho thấy một sơ đồ khối đơn giản hóa của hệ thống thu thập dữ liệu để đào tạo dữ liệu cho DAVE-2. Ba camera được gắn phía sau kính chắn gió của chiếc xe thu thập dữ liệu. Video đóng dấu thời gian từ các camera được chụp đồng thời với góc lái được áp dụng bởi người lái. Lệnh lái này có được bằng cách chạm vào Controller Area Network (CAN).



**Hình 6** Mô hình dữ liệu huấn luyện DAVE-2 [3]

Đào tạo với dữ liệu từ chỉ người lái xe là không đủ. Mạng phải học cách phục hồi từ những sai lầm từ người lái, nếu không xe sẽ tự trượt ra khỏi đường, do đó ta phải đào tạo dữ liệu tăng cường

Hệ thống lấy dữ liệu CAN từ người lái và học thêm với những hình ảnh chụp được từ 3 camera gắn ở xe đã tạo nên một hệ thống hoàn chỉnh .

Ở **Hình 6** Hình ảnh được đưa vào CNN sau đó tính toán một lệnh chỉ đạo được đề xuất. Lệnh được đề xuất được so sánh với mong muốn lệnh cho hình ảnh đó và các trọng số của CNN được điều chỉnh để đưa đầu ra CNN đến gần hơn với đầu ra mong muốn. Việc điều chỉnh trọng lượng được thực hiện bằng cách sử dụng Lan truyền ngược. Sau khi được đào tạo, mạng có thể tạo ra chỉ đạo từ hình ảnh video của một camera trung tâm.

### 2.3.2 Hệ thống DAVE-2 giả lập

Như đã nói ở phần 2.3.1 em sẽ xây dựng mô hình học End-to-end dựa trên ứng dụng mô phỏng ô tô tự lái là phần mềm mã nguồn mở được phát triển bởi Udacity, được viết bằng Unity ( công cụ dùng để phát triển game). Vì mô hình này dựa trên cách hoạt động của DAVE-2



**Hình 7** Giao diện màn hình training

Ứng dụng có 2 chế độ: training mode (sẽ cho dữ liệu về ô tô tự lái để train mô hình) và autonomous mode (sau khi học được mô hình rồi thì đây là phần ô tô tự lái). Ở ô tô có gắn

3 cameras (trái, giữa, phải) Ô tô có thể di chuyển sang bên trái ( $\leftarrow$ ), sang bên phải ( $\rightarrow$ ), tăng tốc ( $\uparrow$ ), giảm tốc ( $\downarrow$ ) Trong phần training mode, ở mỗi vị trí ô tô di chuyển thì sẽ cho ta các dữ liệu: ảnh ở 3 camera, góc lái của vô lăng, tốc độ xe, độ giảm tốc (throttle) và phanh (brake).

Chế độ “Anonymous mode” trong trình mô phỏng, trong đó chiếc xe tương tác với một server để trao đổi từ xa hướng dẫn chiếc xe. Trình mô phỏng gửi hình ảnh camera của server và server dự kiến sẽ trả lời với các góc lái và hiển thị kết quả ra màn hình. Vì vậy, nó không chỉ là trình giả lập lái xe để lấy dữ liệu đào tạo, nó hoàn toàn cần thiết để đánh giá giải pháp cho mô hình này xem có hiệu quả hay không, ta sẽ nói rõ điều này hơn ở Chương 4.

Dữ liệu training set sẽ được lấy từ training mode của phần mềm mô phỏng.



**Hình 8** Góc lái giữa do camera ghi lại



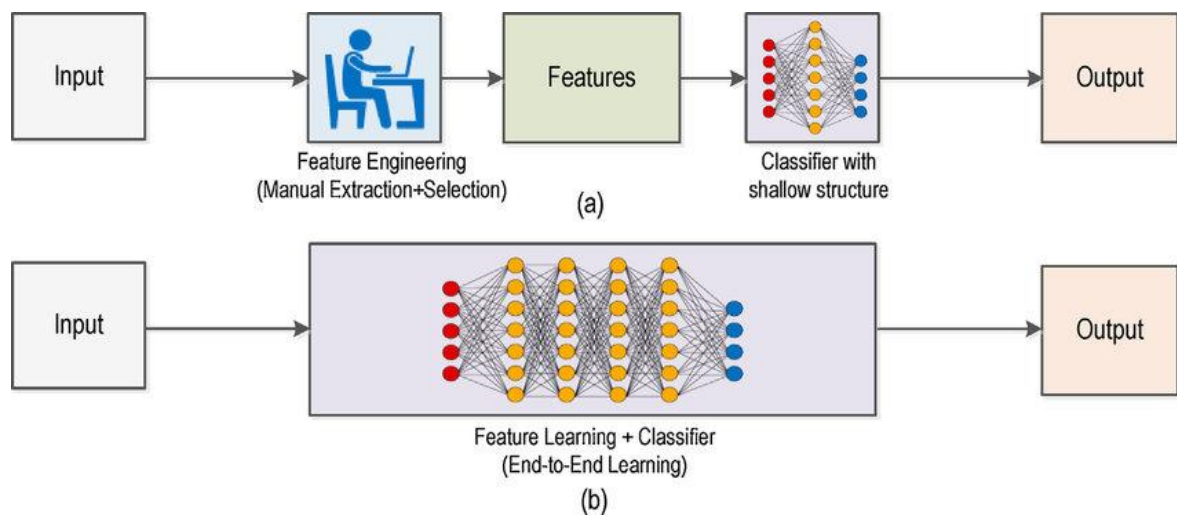
**Hình 9** Góc lái bên phải do camera ghi lại



**Hình 10** Góc lái bên trái do camera ghi lại

### 2.3.3 End-to-end trong kiến trúc Deep Neural Network

Ưu điểm vượt trội của các kiến trúc Deep Neural Network là khả năng End-to-end Learning, tức là khả năng học được các quy luật gán nhãn chuỗi từ tập dữ liệu gán nhãn trước mà không cần có bất cứ sự can thiệp của con người. Điều này đã loại bỏ đi nhược điểm của nhóm phương pháp phải dựa trên kiến thức ngôn ngữ để lựa chọn các đặc trưng. Tuy nhiên cũng tồn tại nhược điểm liên quan đến kích cỡ tập dữ liệu huấn luyện. Thông thường, các kiến trúc Deep Neural Network yêu cầu tập dữ liệu khá lớn để có thể đạt được độ chính xác cao.



**Hình 11** Mô hình học End-to-end



### 2.3.4 Mô hình học End-to-end training mạng CNN để giả lập hành vi

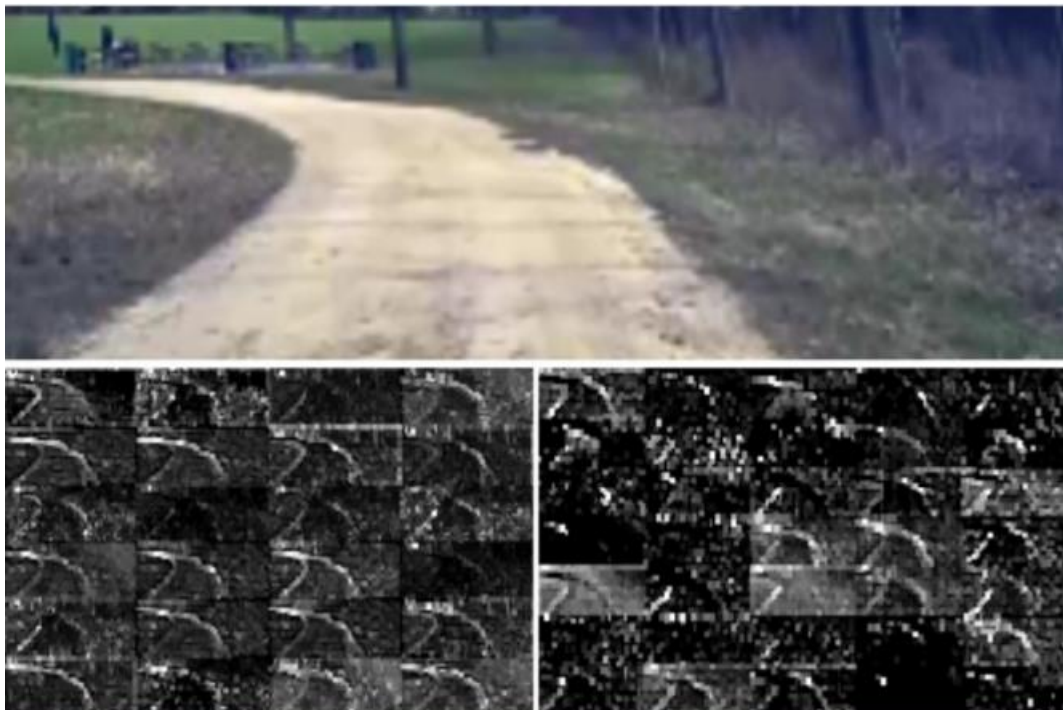


**Hình 12** Mạng được đào tạo được sử dụng để tạo các lệnh lái từ một camera trung tâm phía trước.

Huấn luyện dữ liệu được thu thập từ simulator từ phần 2.3.2 đã nói, mô hình sẽ tự học từ trên chính kinh nghiệm của người lái xe trên simulator. Nó sẽ tự học trên dữ liệu của người lái và kết hợp với các góc lái với nhau để cho ra một mô hình tốt nhất cho xe có thể tự ra quyết định khi cần rẽ .

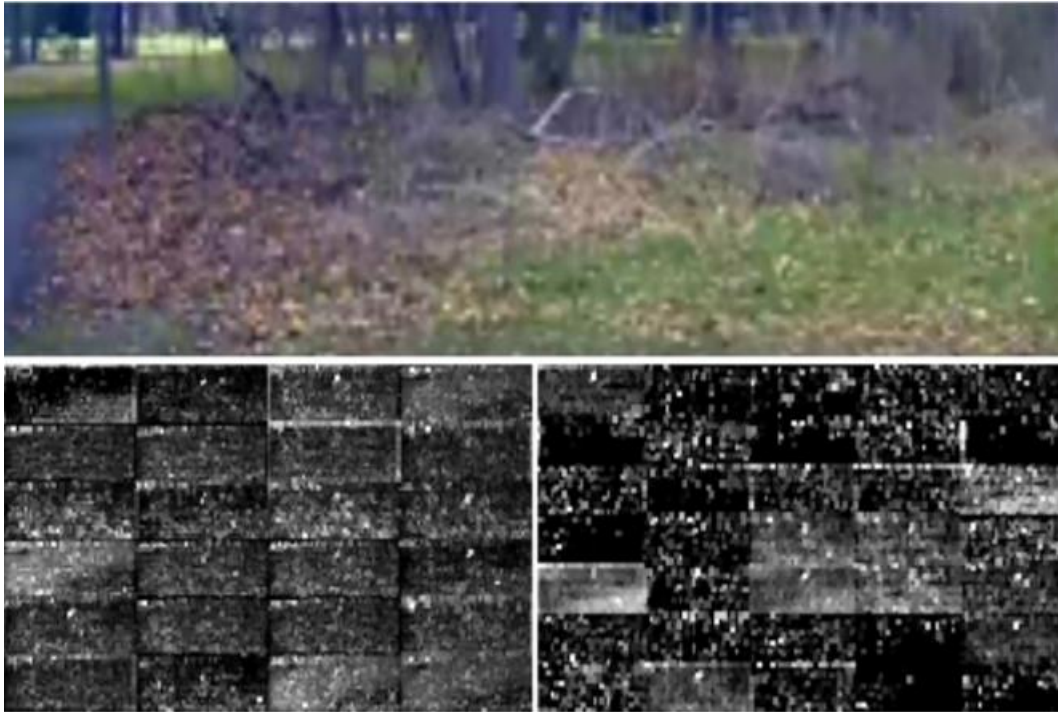
### 2.3.5 Ứng dụng của CNNs trong bài toán giả lập hành vi

**Hình 13** và **Hình 14** dưới hai lớp hình ảnh đặc trưng cho hai đầu vào khác nhau là con đường và khu rừng. Trong trường hợp con đường thì các đặc trưng hiển thị rõ đường viền và đường cong còn trong khu rừng thì bao gồm phần lớn các đặc trưng bị nhiễu.



**Hình 13** Các đặc trưng của một con đường trong CNNs

Các hình ảnh đặc trưng bên dưới của một con đường thể hiện rất rõ qua những đường viền trắng và cong. Từ những đặc trưng như vậy mạng CNN sẽ tự mình phát hiện thế nào là một con đường dựa trên những đặc điểm đó và huấn luyện sao cho hiệu quả nhất (Hình ảnh con đường này là một hình ảnh mới chưa được qua đào tạo để phát hiện các đường viền).



**Hình 14** Đặc trưng của khu rừng qua CNNs

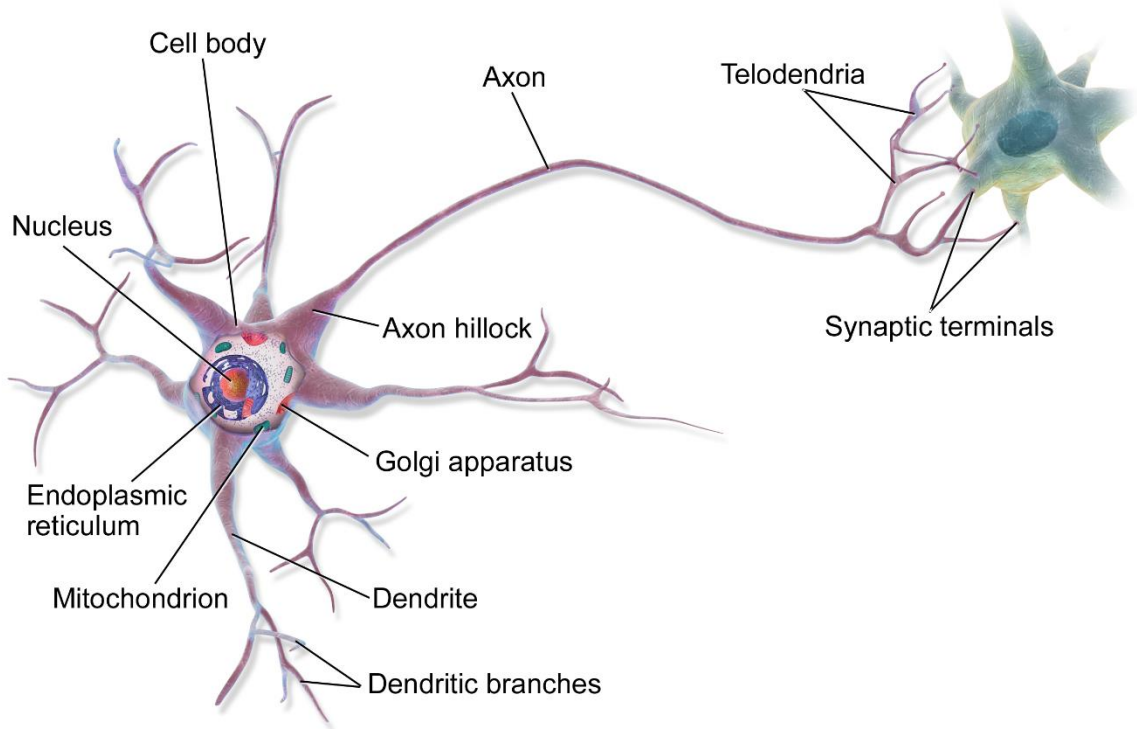
Đây là một ví dụ điển hình về việc phát hiện đây không phải là một con đường. Các đặc trưng chủ yếu bên dưới là các vùng nhiễu không nhìn rõ, và mạng CNN đã không nhận được bất kỳ một đặc trưng hữu ích nào để phát hiện đây là một con đường.

Nhờ hai hình ảnh trên ta đã chứng minh bằng thực nghiệm rằng CNN có thể nhận ra một bức ảnh dựa vào các đặc trưng của bức ảnh đó, hệ thống mạng CNN học nhận dạng viền của một con đường mà không cần hướng dẫn hay trích xuất thủ công từ con người.

## Chương 3 Công nghệ sử dụng

### 3.1 Mạng Neural

#### 3.1.1 Mạng neural sinh học



**Hình 15** Mạng Neural sinh học

Tri thức của loài người cho đến nay là rất phong phú, sâu rộng và đa dạng từ thế giới vi mô như nguyên tử, điện tử, hạt nhân, các hạt cơ bản, ... đến những hiểu biết vĩ mô về trái đất, về hệ mặt trời, hệ thiên hà, ... . hiểu biết về thế giới tự nhiên và xã hội, về các ngành khoa học, kỹ thuật khác nhau như: toán, lý, hóa, công nghệ thông tin và cả những hiểu biết về bản thân con người. Tuy vậy sự hiểu biết của chúng ta về chính bộ não của chúng ta lại rất ít.

Công trình nghiên cứu về não đầu tiên thuộc về Ramond- Cajál (1911), ông cho rằng hệ thần kinh cấu tạo từ các noron. Con người có khoảng 1011 noron, 1015 khớp kết nối. Các khớp khi mới sinh ít kết nối với nhau, chúng được kết nối nhờ quá trình học.

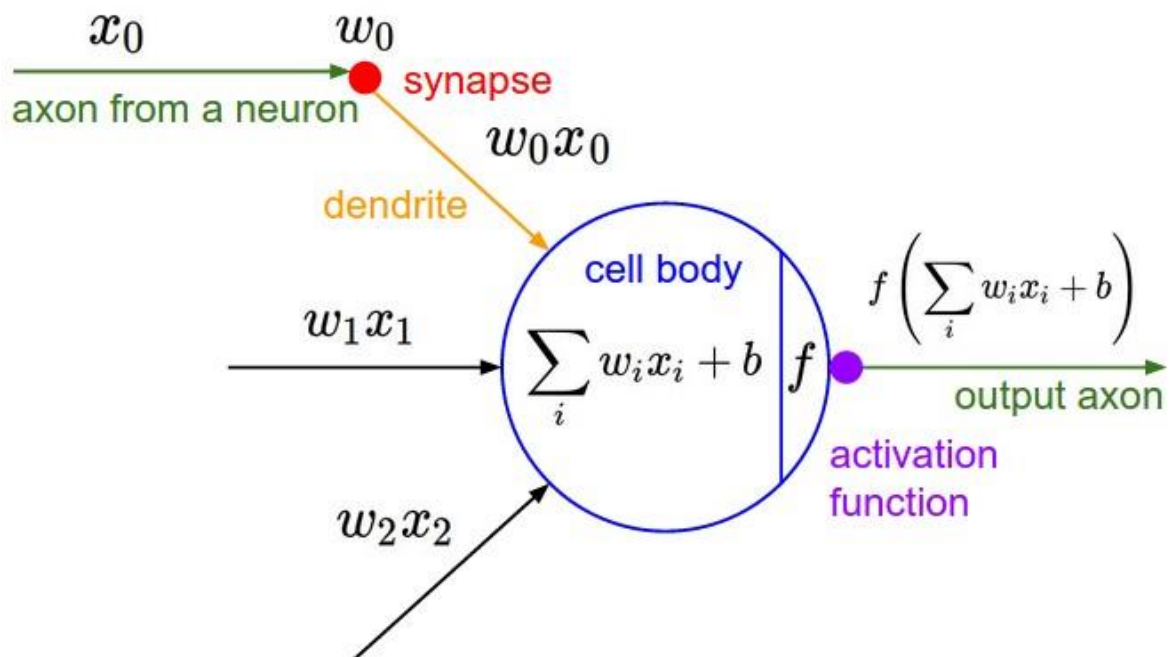
Đến nay con người chúng ta đã hiểu biết rằng các quá trình tính toán trong bộ não sinh học khác với các máy tính số. Bộ não xử lý chậm hơn ( $10^{-3}$  s) so với các thiết bị vật lý ( $10^{-9}$  s) tuy nhiên tốn ít năng lượng hơn và thực hiện được nhiều chức năng hơn.

#### Các thành phần chính của neuron:

- **Khớp kết nối (synapse):** Kết nối các neuron nhờ các tính chất hóa lý
- **Sợi nhánh (dendrites):** Thu nhận thông tin về nhân neuron qua khớp.
- **Thân tế bào:** Tổng hợp tín hiệu.
- **Trục cảm ứng (axon):** đưa tín hiệu ra và truyền tới các neuron khác qua các khớp kết nối.

#### 3.1.2 Mạng neural nhân tạo

Mạng nơ-ron nhân tạo là sự mô phỏng toán học của mạng nơ-ron sinh học. Một mạng neuron nhân tạo được xây dựng từ những thành phần cơ sở là những nơ-ron nhân tạo gồm nhiều đầu vào và một đầu ra. Các đầu vào tiếp nhận kích thích từ đầu ra của những nơ-ron khác hoặc từ môi trường. Mỗi nơ-ron vào có một bộ trọng số nhằm khuếch đại tín hiệu kích thích sau đó tất cả sẽ được cộng lại. Tín hiệu sau đó sẽ được tiếp tục biến đổi nhờ một hàm phi tuyến, thường gọi là hàm kích hoạt. Và cuối cùng tín hiệu sẽ được đưa đến đầu ra của neuron để lại trở thành đầu vào của các nơ-ron khác hoặc trở thành tín hiệu ra của toàn bộ mạng.

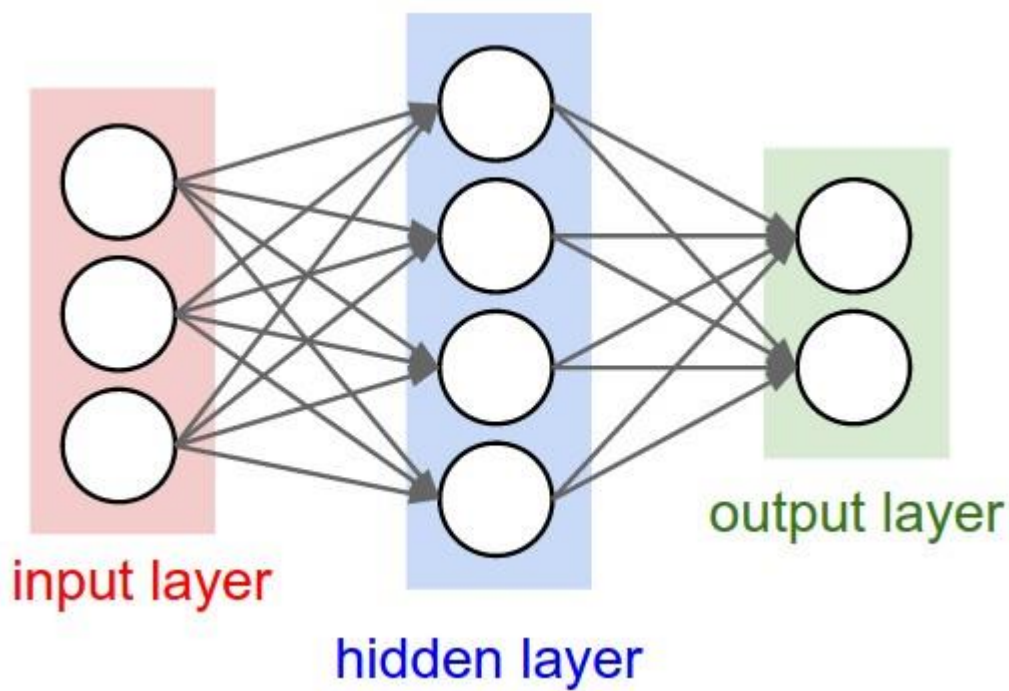


**Hình 16** Mạng neural nhân tạo [4]



- $w$  là trọng số liên kết, thể hiện mức độ quan trọng, độ mạnh của dữ liệu đầu vào đối với quá trình xử lý thông tin. Quá trình học của một mạng neuron thực chất là quá trình điều chỉnh các trọng số liên kết của các dữ liệu đầu vào để được kết quả đầu ra như mong muốn.
- $f$  là hàm kích hoạt, có nhiệm vụ chuẩn hóa output. Nếu không có  $f$  thì output sẽ nằm trong khoảng  $(-\infty; +\infty)$  thì ta sẽ không thể biết được khi nào output fire hoặc không. Chú ý hàm kích hoạt buộc phải là hàm phi tuyến vì nếu nó là hàm tuyến tính thì khi kết hợp với phép tuyến tính  $w x$  thì kết quả thu được sẽ là một thao tác tuyến tính, dẫn tới việc sử dụng hàm  $f$  là vô nghĩa.
- $\theta$  là ngưỡng (threshold) nếu đặt  $-\theta = b$  thì ta gọi  $b$  là bias. Coi bias như một input bổ sung, thay vì có  $n$  input thì sẽ có input thứ  $n+1$  với giá trị bằng  $b$  và trọng số  $w = 1$ . Điều này giúp cho việc so sánh giá trị input sẽ thuận lợi hơn thay vì mỗi bài toán ta tự định nghĩa một số threshold khác nhau thì bây giờ ta quy về so sánh với 0.

### 3.1.3 Xây dựng mạng Neural



**Hình 17** Kiến trúc một mạng Neural [4]

Một mạng NN sẽ có 3 kiểu tầng:

- **Tầng vào (input layer):** Là tầng bên trái cùng của mạng thể hiện cho các đầu vào của mạng.

- **Tầng ra (output layer):** Là tầng bên phải cùng của mạng thể hiện cho các đầu ra của mạng.
- **Tầng ẩn (hidden layer):** Là tầng nằm giữa tầng vào và tầng ra thể hiện cho việc suy luận logic của mạng.

Deep Learning Neural Network là khái niệm chỉ mạng nơron có hai tầng ẩn trở lên. Mạng nơron có hai loại:

- Mạng nơron truyền thẳng (feed-forward neural network – FNN): Mỗi nơron nhận input từ các nơron ở lớp trước nó và gửi tín hiệu này đến các nơron ở các lớp kế tiếp
- Mạng nơron hồi quy (recurrent neural network – RNN): là một mạng mà nơron chứa vòng lặp bên trong chính nó. Nói cách khác mỗi nơron trong mạng RNN nó khả năng lưu lại thông tin mà nó vừa xử lý trước đó. Ví dụ chuỗi các đầu vào  $x_0...x_n$  là những sự kiện xảy ra theo thứ tự thời gian. Những sự kiện này đều có mối liên hệ về thông tin với nhau và thông tin của chúng sẽ được giữ lại để xử lý sự kiện tiếp theo trong mạng nơron hồi quy. Vì tính chất này, mạng nơron hồi quy phù hợp cho những bài toán với dữ liệu đầu vào dưới dạng chuỗi và các sự kiện trong chuỗi có mối liên hệ với nhau trong những bài toán về xử lý ngôn ngữ tự nhiên như phân loại ngữ nghĩa (semantic classification), nhận dạng giọng nói (speech recognition),...

## 3.2 Mạng Neural tích chập

Mạng nơron tích chập (CNN hay ConvNet) là mạng nơron (Wikipedia, bài báo, video) phổ biến nhất được dùng cho dữ liệu ảnh. Bên cạnh các lớp liên kết đầy đủ (FC layers), CNN còn đi cùng với các lớp ẩn đặc biệt giúp phát hiện và trích xuất những đặc trưng - chi tiết (patterns) xuất hiện trong ảnh gọi là Lớp Tích chập (Convolutional Layers). Chính những lớp tích chập này làm CNN trở nên khác biệt so với mạng nơron truyền thống và hoạt động cực kỳ hiệu quả trong bài toán phân tích ảnh.

### 3.2.1 Giới thiệu về mạng Neural tích chập

Nhìn vào một bức ảnh, một người với thị giác bình thường có thể dễ dàng mô tả nội dung, nhận biết và phát hiện các đối tượng được thể hiện trong bức ảnh cũng như vị trí chính xác của chúng. Tuy nhiên, việc này (đọc và hiểu một bức ảnh) khó khăn hơn nhiều đối với máy

tính khi “nó” “nhìn” mỗi bức ảnh chỉ đơn thuần là một ma trận số (tập hợp các điểm ảnh - pixel biểu diễn dưới dạng số theo một hệ cụ thể thường là RGB (Red - Green - Blue)). Mục tiêu chính của Thị giác máy tính (Computer Vision) - một nhánh của trí tuệ nhân tạo (Artificial Intelligence) là tìm ra câu nối giữa ma trận số này và thông tin ngữ nghĩa ẩn chứa trong ảnh. Thị giác máy tính tập trung giải quyết những bài toán như:

- **Phân loại ảnh, miêu tả ảnh**
- **Phát hiện vật thể trong ảnh:** Xe, con người, đèn giao thông, etc.
- **Tạo ảnh với những phong cách khác nhau:** Hiện thị nội dung ngữ nghĩa của ảnh gốc theo những phong cách khác nhau.

Mạng Nơ-ron truyền thống (Neural Network) hoạt động không thực sự hiệu quả với dữ liệu đầu vào là hình ảnh. Nếu coi mỗi điểm ảnh là một thuộc tính (feature), một ảnh RGB kích thước  $(64 \times 64 \times 64)$  có  $12288 \times 12288 (=64 \times 64 \times 3 = 64 \times 64 \times 3)$  thuộc tính. Nếu kích thước ảnh tăng lên  $1000 \times 1000 \times 1000 \times 1000$ , chúng ta có 33 triệu (3M3M) thuộc tính cho mỗi ảnh đầu vào. Nếu sử dụng mạng liên kết đầy đủ (*fully connected NN*) và giả sử lớp thứ 2 có  $1000 \times 1000$  thành phần (units/ neurons), ma trận trọng số sẽ có kích thước  $1000 \times 3M \times 1000 \times 3M$  tương đương với 3B3B trọng số cần huấn luyện (learning). Điều này yêu cầu khối lượng tính toán cực lớn (expensive computational cost) và thường dẫn đến overfitting do không đủ dữ liệu huấn luyện.

### 3.2.2 Kiến thức về mạng Tích chập

**Lớp tích chập được dùng để phát hiện và trích xuất đặc trưng - chi tiết của ảnh.**

Giống như các lớp ẩn khác, lớp tích chập lấy dữ liệu đầu vào, thực hiện các phép chuyển đổi để tạo ra dữ liệu đầu vào cho lớp kế tiếp (đầu ra của lớp này là đầu vào của lớp sau). Phép biến đổi được sử dụng là phép tính tích chập. Mỗi lớp tích chập chứa một hoặc nhiều bộ lọc - bộ phát hiện đặc trưng (filter - feature detector) cho phép phát hiện và trích xuất những đặc trưng khác nhau của ảnh.

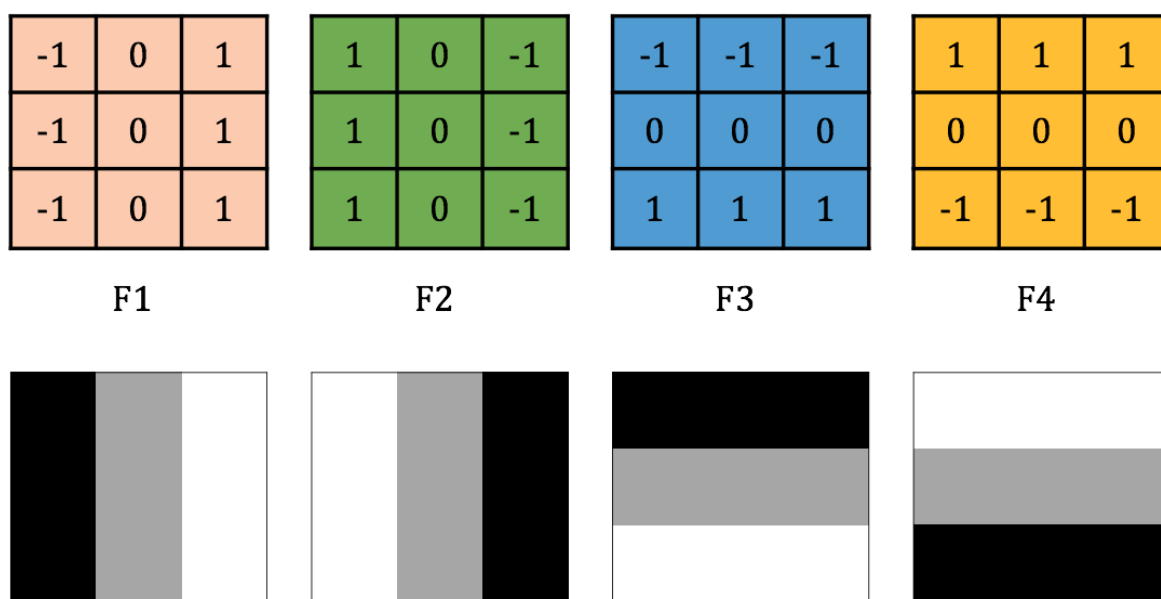
***Đặc trưng** của ảnh là gì? Đặc trưng ảnh là những chi tiết xuất hiện trong ảnh, từ đơn giản như cạnh, hình khối, chữ viết tới phức tạp như mắt, mặt, chó, mèo, bàn, ghế, xe, đèn giao thông, v.v.. Bộ lọc phát hiện đặc trưng là bộ lọc giúp phát hiện và trích xuất các đặc trưng của ảnh, có thể là bộ lọc góc, cạnh, đường chéo, hình tròn, hình vuông, v.v. [5]*

**Bộ lọc ở lớp tích chập càng sâu thì phát hiện các đặc trưng càng phức tạp.**

Độ phức tạp của đặc trưng được phát hiện bởi bộ lọc tỉ lệ thuận với độ sâu của lớp tích chập mà nó thuộc về. Trong mạng CNN, những lớp tích chập đầu tiên sử dụng bộ lọc hình học (geometric filters) để phát hiện những đặc trưng đơn giản như cạnh ngang, dọc, chéo của bức ảnh. Những lớp tích chập sau đó được dùng để phát hiện đối tượng nhỏ, bán hoàn chỉnh như mắt, mũi, tóc, v.v. Những lớp tích chập sâu nhất dùng để phát hiện đối tượng hoàn chỉnh như: chó, mèo, chim, ô tô, đèn giao thông, v.v. Để hiểu cách thức hoạt động của lớp tích chập cũng như phép tính tích chập, hãy cùng xem ví dụ về bộ lọc phát hiện cạnh (edge filters/detectors) dưới đây.

### Ví dụ về bộ lọc cạnh

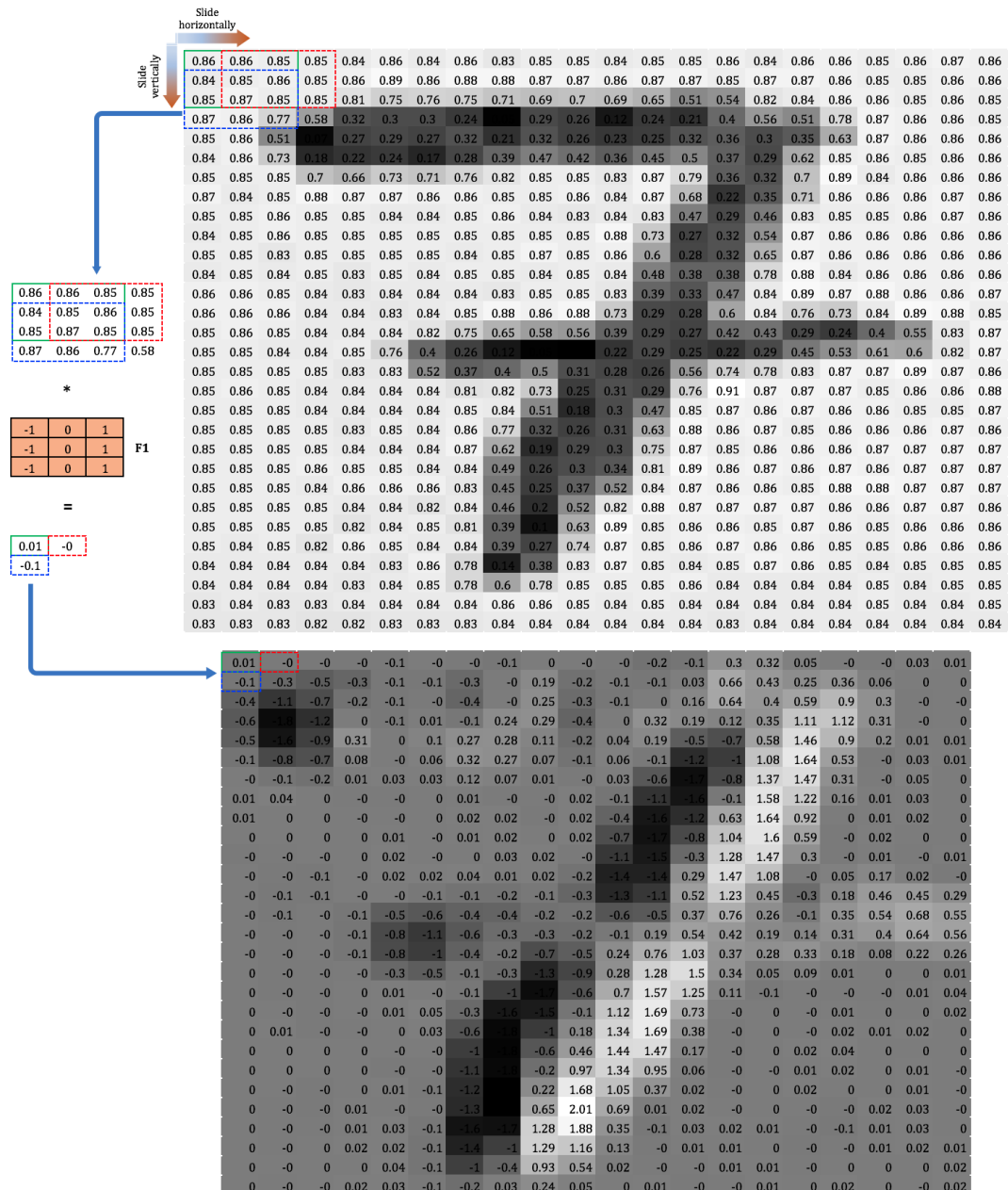
Trong ví dụ này, CNN được sử dụng để phân loại tập các ảnh viết tay của các số từ 00 tới 99. Đầu vào là những bức ảnh trắng đen (Gray Scale) và được biểu diễn bởi một ma trận các điểm ảnh với kích thước cố định  $h \times w$ . Lớp tích chập đầu tiên của CNN sử dụng 4 bộ lọc kích thước  $3 \times 3$ :  $F_1, F_2, F_3, F_4$  với giá trị tương ứng như trong **Hình 18**. Các giá trị tại mỗi ô của các bộ lọc có thể được biểu diễn bởi màu sắc tương ứng với Đen (-1), Xám (0), Trắng (1) như trong hình dưới đây.



**Hình 18** Bộ lọc được sử dụng trong lớp tích chập đầu tiên là các ma trận kích thước **3x3** [5]

Để minh họa cho phép nhân chập, chúng ta sử dụng đầu vào là một bức ảnh viết tay của số 7, biểu diễn dưới dạng ma trận  $30 \times 22$  và áp dụng riêng biệt từng bộ lọc ở trên. Phép nhân tích

chập được thực hiện bằng cách trượt ma trận lọc  $3 \times 3$  trên ma trận ảnh đầu vào  $32 \times 22$  (bộ lọc dịch sang phải/ xuống dưới 1 cột/ hàng mỗi một lần trượt) cho đến khi nó đi qua hết tất cả các vùng kích thước  $3 \times 3$ . Việc trượt ma trận lọc trên ma trận đầu vào được gọi là “chập” (convolving). Như minh họa trong **Hình 19**, ma trận F1 được chập với từng vùng (block - region) điểm ảnh kích thước  $3 \times 3$  của ảnh đầu vào. Tại mỗi vị trí di chuyển của ma trận F1, giá trị đầu ra được tính bằng tích chập (dot-product) của ma trận F1 với vùng bao phủ tương ứng.



**Hình 19** Nhân chập bộ lọc F1 với ma trận ảnh đầu vào của số 7 [5]

Ô đầu tiên ( 0 , 0 ) của ma trận đầu ra (giá trị 0.01) ra là kết quả của phép nhân chập giữa ma trận  $F_1$  với góc trái trên cùng của ma trận đầu vào và được tính như sau:

$$\begin{array}{|c|c|c|} \hline 0.86 & 0.86 & 0.85 \\ \hline 0.84 & 0.85 & 0.86 \\ \hline 0.85 & 0.87 & 0.85 \\ \hline \end{array} \quad \begin{array}{c} * \\ * \\ * \end{array} \quad \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} = \begin{array}{l} 0.86*(-1) + 0.86*0 + 0.85*1 \\ + 0.84*(-1) + 0.85*0 + 0.86*1 \\ + 0.85*(-1) + 0.87*0 + 0.85*1 \end{array} = \boxed{0.01}$$

3x3 block      Dot product (Element-wise)       $F_1$

**Hình 20** Nhân với filter  $F_1$  [5]

Từ các ma trận đầu ra kích thước  $28 \times 20$  , chúng ta thấy được cả bốn bộ lọc  $F_1, F_2, F_3$  và  $F_4$  đều được sử dụng để phát hiện cạnh trong bức ảnh (thể hiện bởi những điểm ảnh sáng hơn) (**Hình 21**):

- $F_1$ : Phát hiện cạnh đứng phải.
- $F_2$ : Phát hiện cạnh đứng trái.
- $F_3$ : Phát hiện cạnh ngang dưới.
- $F_4$ : Phát hiện cạnh ngang trên.



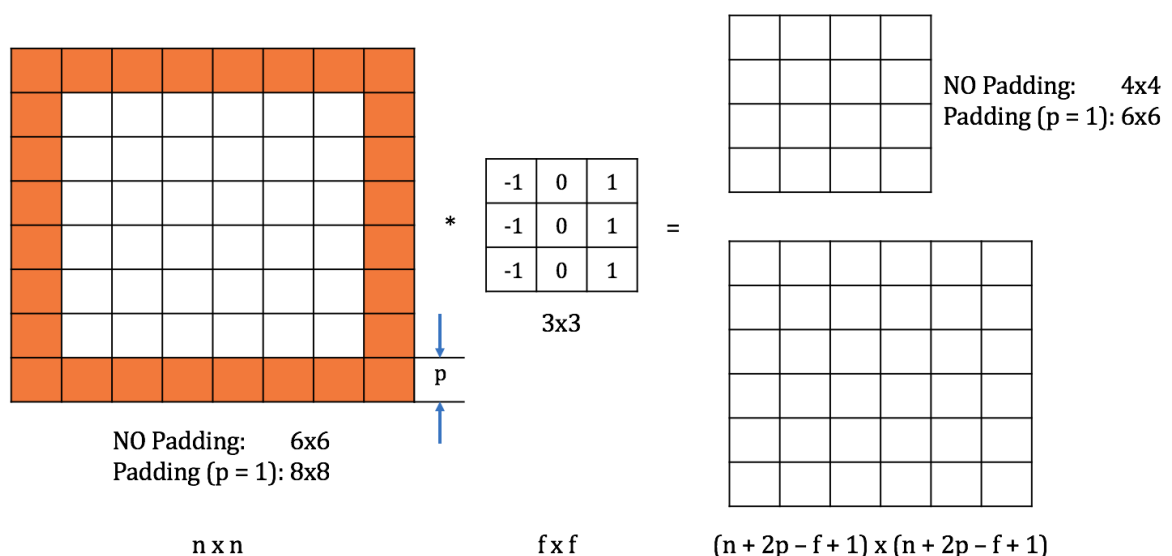
**Hình 21** Ví dụ về bộ lọc cạnh với đầu vào là ảnh số viết tay. [5]

Các bộ lọc cạnh: Rất nhiều bộ lọc cạnh đã được đề xuất với sự khác biệt nhỏ về kích thước và giá trị. Các ma trận lọc này thường có kích thước  $3 \times 3$  với các giá trị nhỏ trong khoảng  $-5$  tới  $5$  và đối xứng. Trong hình 4, bộ lọc Sobel được sử dụng để phát hiện các cạnh của giỏ hoa quả (ảnh màu) và cho kết quả khá tốt.

Nếu ta có tập dữ liệu huấn luyện lớn và hiệu năng tính toán cao, với những tập ảnh kích thước lớn và nhiều chi tiết phức tạp, các bộ lọc cạnh có thể được huấn luyện tự động từ tập dữ liệu. Nghĩa là các giá trị của ma trận lọc được coi như tham số của một mạng nơ-ron và huấn luyện (sử dụng back-propagation chẳng hạn) để có một tập giá trị tối ưu. Với cách tiếp cận này, các bộ lọc tạo ra có thể phát hiện không chỉ cạnh đứng hay ngang mà còn có thể những cạnh nghiêng một góc lẻ như  $40^\circ, 45^\circ, 70^\circ$ .

## Padding

- Lấy ví dụ với ma trận đầu vào kích thước  $6 \times 6$ . Nếu ta nhân chập với bộ lọc kích thước  $3 \times 3$ , kết quả thu được là một ma trận đầu ra kích thước  $4 \times 4$  vì chỉ có  $4 \times 4$  vị trí trên ma trận đầu vào để đặt ma trận lọc. Tổng quát hoá, nếu ta nhân chập ma trận đầu vào kích thước  $n \times n$  với bộ lọc kích thước  $f \times f$ , ta thu được kết quả là một ma trận kích thước  $(n - f + 1) \times (n - f + 1)$ . Mỗi một lần áp dụng phép nhân chập, kích thước của ảnh bị giảm xuống, và vì thế chúng ta chỉ có thể thực hiện nó một vài lần trước khi ảnh trở nên quá nhỏ.
- Điểm ảnh ở khoảng trung tâm của ma trận đầu vào được bao phủ bởi rất nhiều vùng  $3 \times 3$  nghĩa là được sử dụng để tính nhiều giá trị đầu ra, trong khi những điểm ảnh ở góc hoặc cạnh chỉ được sử dụng 1 hoặc 2 lần vì chỉ bị bao phủ bởi 1 hoặc 2 vùng  $3 \times 3$ . Vì thế chúng ta đánh mất rất nhiều thông tin (có thể quan trọng) tại các vùng gần cạnh của ảnh.



**Hình 22** Ma trận đầu vào được bao quanh bởi đường viền phụ kích thước  $p$  (giá trị 0). [5]

Để khắc phục hai nhược điểm trên, một đường viền phụ (padding) được thêm vào xung quanh ma trận đầu vào. Việc thêm đường viền phụ làm tăng kích thước của ma trận đầu vào, dẫn tới tăng kích thước ma trận đầu ra. Từ đó độ chênh lệch giữa ma trận đầu ra với ma trận đầu vào gốc giảm. Những ô nằm trên cạnh/ góc của ma trận đầu vào gốc cũng lùi sâu vào bên trong hơn, dẫn tới được sử dụng nhiều hơn trong việc tính toán ma trận đầu ra, tránh được việc mất mát thông tin.

Trong **Hình 22**, ma trận đầu vào kích thước  $6 \times 6$  được thêm vào đường viền phụ kích thước 1 ( $p = 1$ ), trở thành ma trận  $8 \times 8$ . Khi nhân chập ma trận này với bộ lọc  $3 \times 3$ , chúng ta thu được ma trận đầu ra  $6 \times 6$ . Kích thước của ma trận đầu vào (gốc) được duy trì. Những điểm ảnh nằm ở cạnh của ma trận đầu vào gốc được sử dụng nhiều lần hơn (4 lần với những điểm ảnh ở góc).

Theo quy ước, các ô trên đường viền phụ có giá trị bằng không,  $p$  là kích thước của đường viền phụ. Trong hầu hết các trường hợp, đường viền phụ đối xứng trái-phải, trên-dưới so với ma trận gốc, vì thế kích thước của ma trận đầu vào được tăng lên  $2p$  mỗi chiều. Ma trận đầu ra do đó có kích thước  $(n + 2p - f + 1) \times (n + 2p - f + 1)$ .

Tuỳ theo giá trị của  $p$ , chúng ta có hai trường hợp chính:

- Nhân chập không dùng đường viền phụ (**valid convolution**) - NO padding:

$$(n \times n) * (f \times f) \Rightarrow (n - f + 1) \times (n - f + 1)$$

- Nhân chập không làm thay đổi kích thước đầu vào (same convolution): Kích thước đường viền phụ được tính theo công thức:

$$n + 2p - f + 1 = n \Rightarrow p = \frac{f - 1}{2}$$

Theo quy ước, kích thước bộ lọc  $f$  là số lẻ vì hai lý do chính sau:



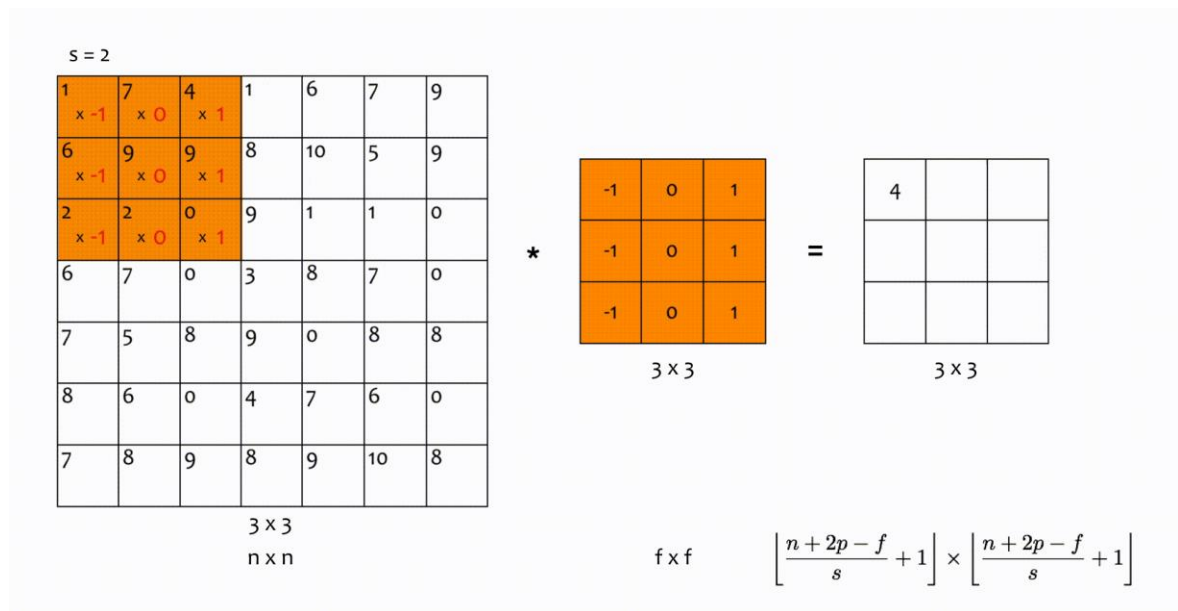
- Nếu  $f$  là số chẵn, chúng ta phải thêm vào bên trái của ma trận đầu vào nhiều hơn bên phải (hoặc ngược lại), việc này dẫn tới hệ đầu vào không đối xứng (**asymetric**).
- Nếu  $f$  là số lẻ, ma trận đầu vào có một điểm ảnh ở trung tâm. Trong lĩnh vực thị giác máy tính, việc có một nhân tố khác biệt (**distinguisher**) - một điểm đại diện cho vị trí của bộ lọc thường mang lại hiệu năng cao cho bài toán.

### Nhân chập sai (strided convolutions)

Trong phép nhân chập ở trên, bộ lọc trượt trên ma trận đầu vào 1 hàng/ cột trong mỗi bước di chuyển. Tuy nhiên, giá trị này có thể bằng 2, 3 hoặc lớn hơn. Số hàng/ cột mà bộ lọc trượt qua trong một bước di chuyển ký hiệu là  $s$ . Kích thước ma trận đầu ra lúc này được tính bởi:

$$\left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-s}{s}+1\right)$$

Nếu  $n+2p-f$  không chia hết cho  $s$ , chúng ta lấy chặn dưới ( $\lfloor \rfloor$ ) như trong hình minh hoạ dưới đây.



**Hình 23** Nhân chập với bước sai (trượt)  $s = 2$

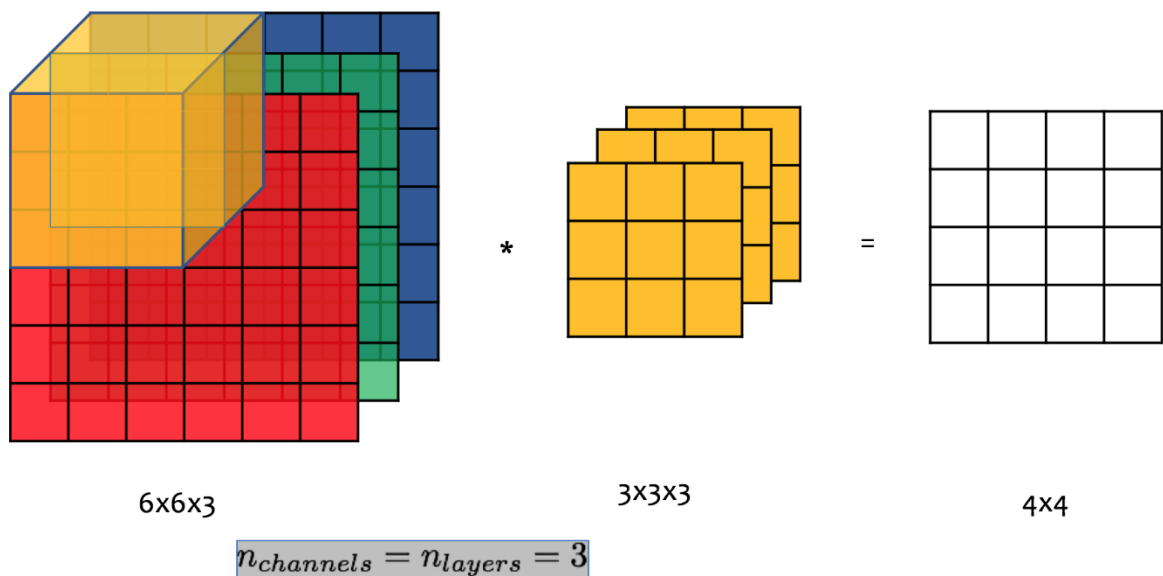
Trong lĩnh vực toán học thuần túy, phép toán nhân chập được định nghĩa hơi khác so với phía trên. Trước khi thực hiện nhân chập (element-wise/ dot-product) và lấy tổng của các kết quả thu được, bộ lọc (filter) được lật lần lượt theo trục ngang và trục dọc (flipped filter). Ma trận đầu ra được tính dựa trên ma trận đầu vào và bộ lọc đã được lật này. Phép toán

“nhân chập” được trình bày ở trên (thực hiện trực tiếp trên ma trận đầu vào và bộ lọc gốc) được gọi là tương quan chéo (cross-correlation). Tuy nhiên, theo quy ước trong ML và DL, phép tương quan chéo (cross-correlation) được gọi là phép nhân chập (convolution).

### 3.2.3 Phép chập khối

#### Phép chập khối với một bộ lọc

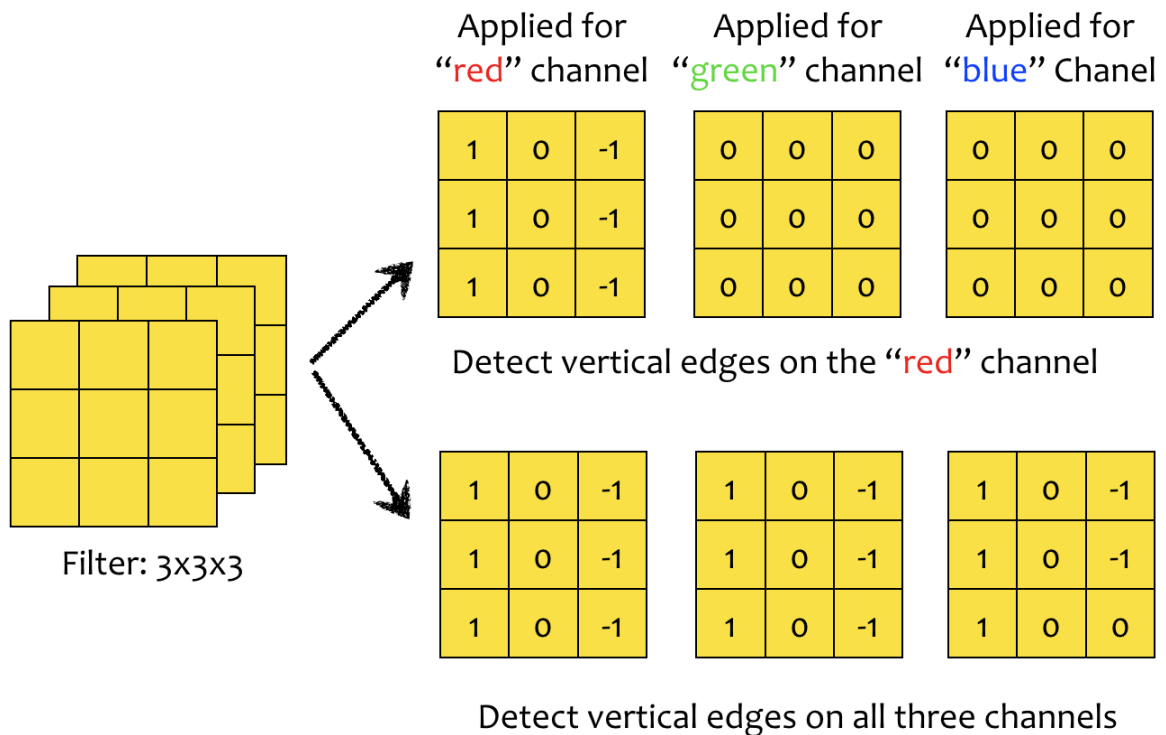
Các ví dụ ở trên sử dụng ảnh trong hệ gray và được biểu diễn dưới dạng ma trận 2 chiều (2D). Phép nhân chập cũng có thể dùng cho ảnh màu (3D images). Giả sử chúng ta có ảnh đầu vào kích thước  $6 \times 6$  được biểu diễn trong hệ RGB. Ma trận đầu vào do đó có kích thước  $6 \times 6 \times 3$  ( 3 kênh màu). Bộ lọc được sử dụng do đó cũng phải có 3 lớp tương ứng với 3 kênh màu đỏ, xanh lục và xanh lam.



**Hình 24** Phép nhân chập khối - áp dụng cho ảnh màu RGB kích thước  $6 \times 6$

*Khối lọc (filter cube) được dịch chuyển trên khối ma trận đầu vào. Mỗi lớp của bộ lọc được nhân chập với phần diện tích bị bao phủ bởi nó trên kênh tương ứng của ma trận đầu vào. Tại một vị trí cụ thể của khối lọc, giá trị tại ô tương ứng của trận đầu ra (ma trận 2 chiều) là tổng của ba tích thu được.*

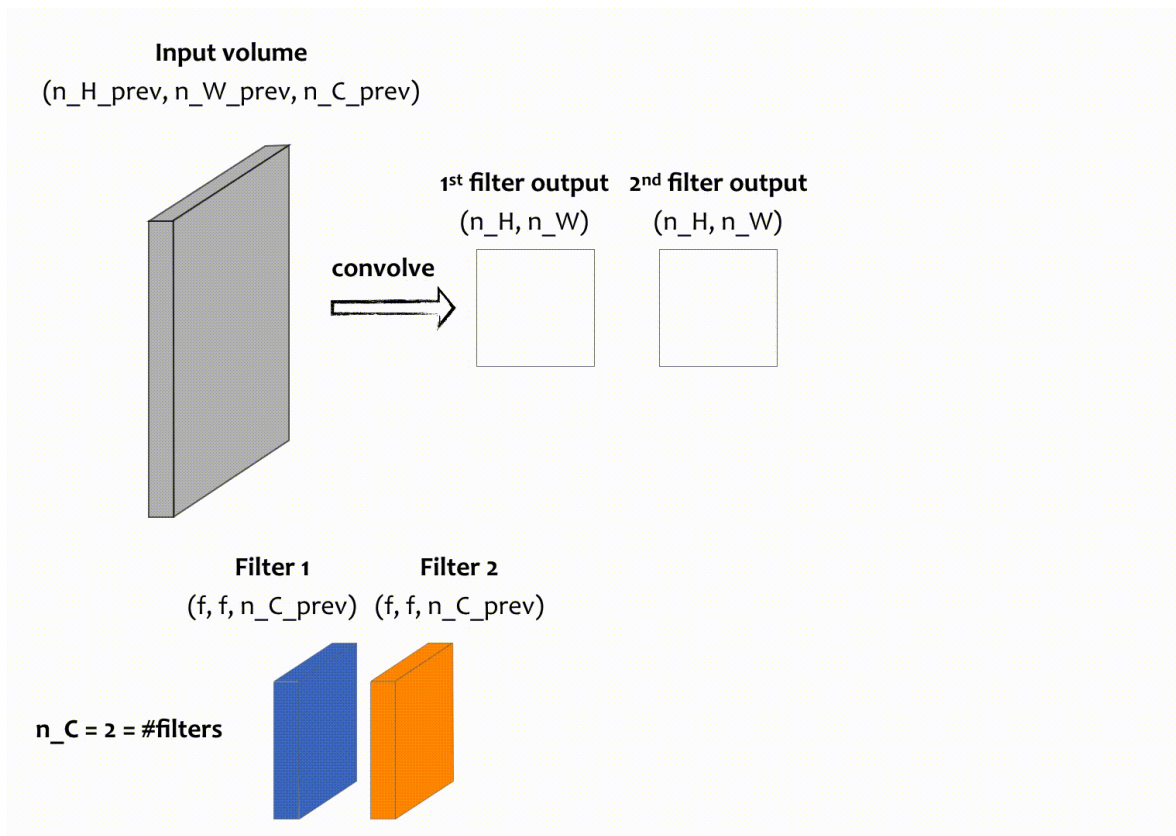
Phép chập khối có thể được sử dụng để phát hiện và trích xuất đặc trưng (chi tiết) ảnh trên từ kênh màu. Ví dụ, để phát hiện cạnh trên kênh màu đỏ ( red ), chúng ta đặt bộ phát hiện cạnh ở lớp đầu tiên của bộ lọc và thiết lập hai lớp kế tiếp bằng 0 (không thực hiện gì cả). Tương tự, để phát hiện cạnh trên cả ba kênh màu, bộ phát hiện cạnh được đặt ở cả ba lớp của bộ lọc khối (**Hình 25**).



**Hình 25** Ba lớp của bộ lọc có thể được cấu hình khác nhau để phát hiện đặc trưng trên một, hai hoặc cả ba kênh màu của ảnh đầu vào. [5]

### Phép chập khối với nhiều bộ lọc

Tại một lớp tích chập, nhiều bộ lọc có thể được sử dụng cùng lúc để phát hiện những đặc trưng khác của ảnh ví dụ như cạnh đứng, ngang hay nghiêng 45. Trong **Hình 24** hai bộ lọc kích thước  $3 \times 3 \times 3$  được sử dụng cùng lúc để đồng thời phát hiện cạnh đứng và ngang. Với mỗi bộ lọc, ta thu được ma trận có kích thước  $4 \times 4$  như đã trình bày ở trên. Hai ma trận này được nhập lại (stack together) tạo thành một ma trận đầu ra duy nhất kích thước  $4 \times 4 \times 2$  (**Hình 26**).



**Hình 26** Hai bộ lọc kích thước  $3 \times 3 \times 3$  được sử dụng để phát hiện đồng thời cạnh đứng và ngang của ảnh đầu vào (hệ RGB). [4]

## Lớp Pooling

Lớp Pooling được sử dụng trong CNN để giảm kích thước đầu vào, tăng tốc độ tính toán và hiệu năng trong việc phát hiện các đặc trưng. Có nhiều hướng Pooling được sử dụng, trong đó phổ biến nhất là pooling theo giá trị cực đại (max pooling) và pooling theo giá trị trung bình (average pooling).

Pooling theo giá trị cực đại (Max pooling):

*Nếu một đặc tính được phát hiện ở một vùng nào đó bị bao phủ bởi bộ lọc, giá trị cao nhất trong vùng sẽ được giữ lại tuy nhiên chưa ai giải thích được tại sao cách tiếp cận này lại hoạt động tốt trong thực nghiệm.*

Pooling theo giá trị trung bình (Average Pooling):

Thay vì lấy giá trị cực đại, pooling theo giá trị trung bình lấy trung bình của tất cả các giá trị trong vùng bị bao phủ bởi bộ lọc khi nó trượt trên ma trận đầu vào. Tuy nhiên pooling theo giá trị trung bình rất ít khi được sử dụng, hầu hết các CNNs hiện nay sử dụng pooling theo giá trị cực đại.

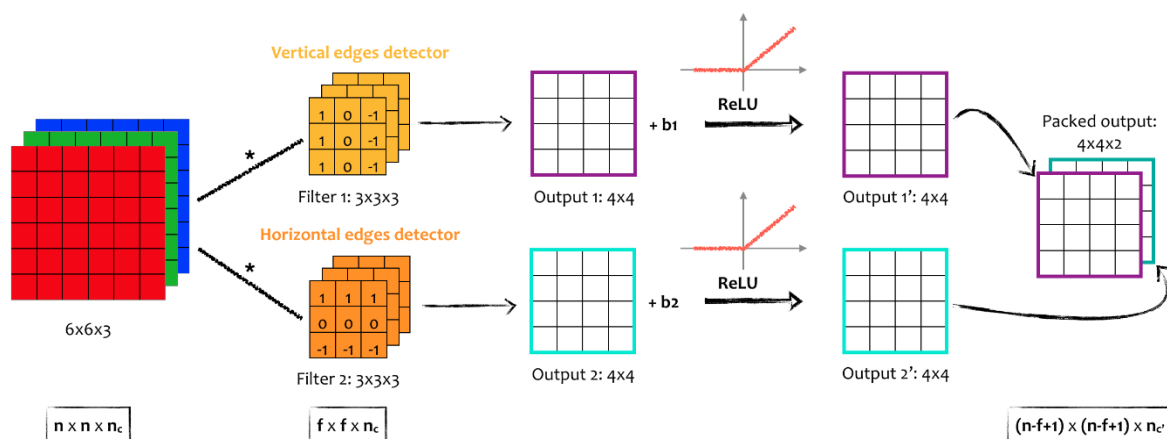
### 3.2.4 Các kiến trúc của mạng Neural tích chập

#### Mạng CNN một lớp

Phép chập với nhiều bộ lọc trình bày ở trên có thể chuyển thành CNN một lớp bằng cách cộng thêm vào mỗi ma trận ra  $4 \times 4$  một số thực  $b$  (bias) và đưa chúng qua một hàm kích hoạt không tuyến tính (non-linear activation function), ví dụ như ReLU. Kết hợp hai ma trận thu được, ta được khối ma trận ra kích thước  $4 \times 4 \times 2$ .

#### Tham số:

Có bao nhiêu tham số trong một lớp tích chập sử dụng 10 bộ lọc kích thước  $3 \times 3 \times 3$ ? Trả lời: Mỗi bộ lọc kích thước  $3 \times 3 \times 3$  có 27 tham số, cộng thêm bias  $b$ . Vì thế, có tất  $28 \times 10 = 280$  tham số cho 10 bộ lọc. Lưu ý rằng số lượng tham số là cố định và hoàn toàn không phụ thuộc vào kích thước của ảnh đầu vào ( $1000 \times 1000$  hay  $5000 \times 5000$ ). Đây là một tính chất quan trọng của lớp tích chập giúp CNN tránh được rủi ro overfitting do có quá nhiều tham số nếu sử dụng lớp liên kết đầy đủ (như đã trình bày ở dưới).



Hình 27 Kiến trúc của một lớp [4]

#### Ký hiệu

If layer  $l$  is a convolution layer, so:

- $f^{[l]}$  = filter size;
- $p^{[l]}$  = padding;
- $s^{[l]}$  = stride;
- $n_c^{[l]}$  = number of filters

Size

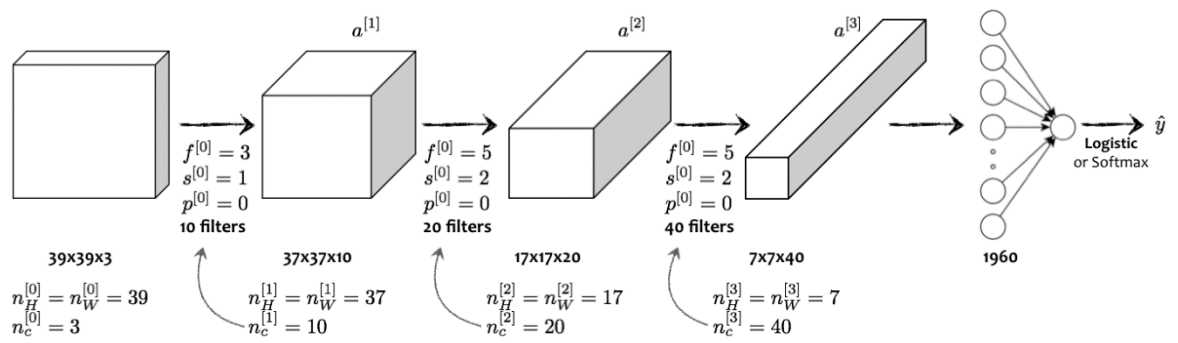
- Input:  $n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$
- Output:  $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
- Each filter:  $f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$  #filter's layers = #input channels
- Weights:  $f^{[l]} \times f^{[l]} \times n_C^{[l-1]} \times n_C^{[l]}$  ( $n_C^{[l]}$  is the number of filters in layer  $l$ )
- Activations:  $a^{[l]} \Rightarrow n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$ ;  $A^{[l]} \Rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$  ( $m$  is number of training examples)
- Bias:  $n_C^{[l]} - (1, 1, 1, n^{[l]})$

**Hình 28** Các ký hiệu cơ bản của một CNN [4]

## CNN đơn giản

Hãy lấy ví dụ về một CNN nhiều lớp (deep CNN) lấy đầu vào là ảnh X và xác định xem nó có phải là ảnh **mèo** hay không (bài toán phân loại ảnh - **image classification**). CNN được thiết lập như sau:

- **Ảnh đầu vào** có kích thước  $39 \times 39 \times 3$  (hệ RGB).
- Lớp tích chập **đầu tiên** sử dụng 10 bộ lọc,  $f^{[1]} = 5$ , bước trượt  $s^{[1]} = 1$ , không padding  $p^{[1]} = 0$
- Lớp tích chập **thứ hai** sử dụng 20 bộ lọc,  $f^{[2]} = 5$ , bước trượt  $s^{[2]} = 2$ , không padding  $p^{[2]} = 0$ .
- Lớp tích chập **cuối cùng** sử dụng 40 bộ lọc,  $f^{[3]} = 5$ , bước trượt  $s^{[3]} = 2$ , không padding  $p^{[3]} = 0$
- **Phẳng hoá** (flatten - unroll) ma trận khối thu được thành một vector cột chứa 1960 phần tử.
- Sử dụng lớp logistic regression để thu về kết quả: 0 (không phải mèo), 1 (mèo).



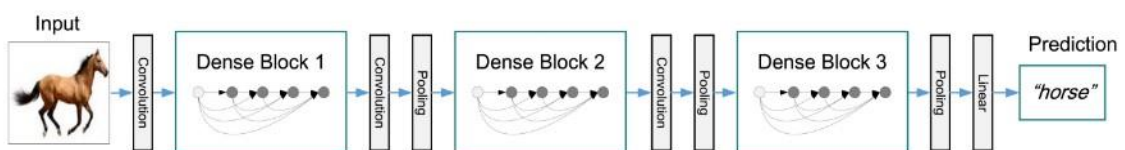
**Hình 29** Ví dụ một CNN cơ bản được sử dụng cho bài toán phân loại ảnh (“mèo” hay “không phải mèo”)

**Hình 29** minh họa một ví dụ cơ bản của CNN. Cấu trúc của các CNNs khá tương đồng nên việc lựa chọn các siêu thông số sử dụng trong các CNNs thường được chú trọng hơn: kích thước của bộ lọc, giá trị của bước trượt (  $s$  ), độ rộng đường viền phụ padding (  $p$  ), số lượng bộ lọc dùng trong một lớp chập (  $n_c$  ).

### 3.2.5 Một số mô hình mạng Neural tích chập

#### Densenet(2016)

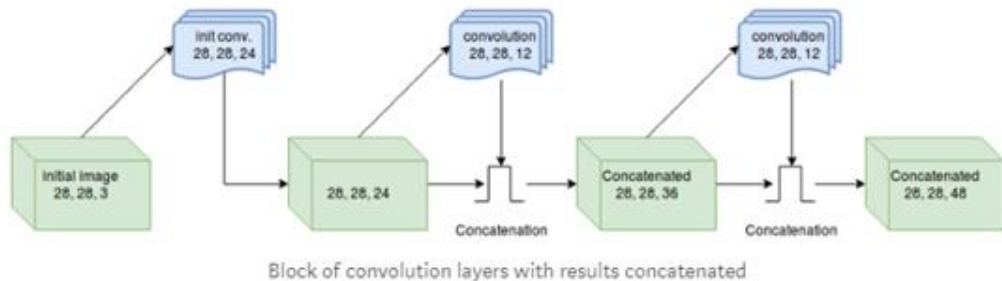
Densenet (Dense connected convolutional network) là một trong những network mới nhất cho visual object recognition. Nó cũng gần giống Resnet nhưng có một vài điểm khác biệt. Densenet có cấu trúc gồm các dense block và các transition layers. Được stack dense block-transition layers-dense block- transition layers như hình vẽ. Với CNN truyền thống nếu chúng ta có  $L$  layer thì sẽ có  $L$  connection, còn trong densenet sẽ có  $L(L+1)/2$  connection.



**Hình 30** Densenet(2016)

- Hãy tưởng tượng ban đầu ta có 1 image size (28,28,3). Đầu tiên ta khởi tạo feature layer bằng Conv tạo ra 1 layer size (28,28,24). Sau mỗi layer tiếp theo (Trong dense block ) nó sẽ tạo thêm  $K=12$  feature giữa nguyên width và height. Khi đó output tiếp theo sẽ là (28,28,24+12),(28,28,24+12+12). Ở mỗi dense block sẽ có normalization,

nonlinearity và dropout. Để giảm size và depth của feature thì transition layer được đặt giữa các dense block, nó gồm Conv kernel size =1, average pooling (2x2) với stride = 2 nó sẽ giảm output thành (14,14,48) [4]



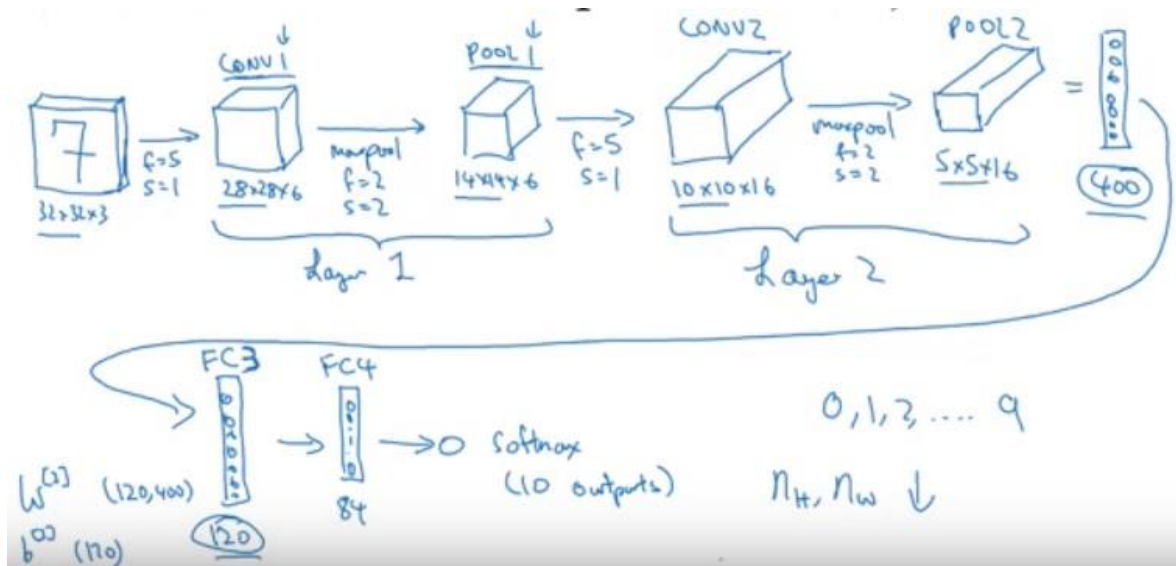
**Hình 31** Các block của Densenet

- Một số ưu điểm của Densenet:
  - Accuracy : Densenet training tham số ít hơn 1 nửa so với Resnet nhưng có same accuracy so trên ImageNet classification dataset.
  - Overfitting : DenseNet resistance overfitting rất hiệu quả.
  - Giảm được vanishing gradient.
  - Sử dụng lại feature hiệu quả hơn.

### LeNet(1998)

LeNet là một trong những mạng CNN lâu đời nổi tiếng nhất được Yann LeCun phát triển vào những năm 1998s. Cấu trúc của LeNet gồm 2 layer (Convolution + maxpooling) và 2 layer fully connected layer và output là softmax layer . Chúng ta cùng tìm hiểu chi tiết architect của LeNet đối với dữ liệu mnist (accuracy lên đến 99%). [4]





**Hình 32** LeNet(1998)

- Input shape 28x28x3
- Layer 1 :
  - Convolution layer 1 : Kernel 5x5x3 , stride = 1, no padding, number filter = 6 ,output = 28x28x6.
  - Maxpooling layer : pooling size 2x2, stride = 2, padding = "same", output = 14x14x6.
- Layer 2 :
  - Convolution layer 2 : kernel 5x5x6, stride = 1, no padding, number filter = 16, output = 10x10x16.
  - Maxpooling layer : pooling size = 2x2, stride = 2, padding = "same", output = 5x5x16.
  - Flatten output = 5x5x16 = 400
  - Fully connected 1 : output = 120
  - Fully connected 2 : output = 84
  - Softmax layer, output = 10 (10 digits). Nhược điểm của LeNet là mạng còn rất đơn giản và sử dụng sigmoid (or tanh) ở mỗi convolution layer mạng tính toán rất chậm.

### 3.3 Một số kiến thức cơ sở

#### 3.3.1 Sai số toàn phương trung bình

MSE đánh giá chất lượng của một ước lượng (ví dụ, một hàm toán học lập bản đồ mẫu dữ liệu của một tham số của dân số từ đó các dữ liệu được lấy mẫu) hoặc một yếu tố dự báo (ví

dự, một bản đồ chức năng có số liệu vào tùy ý để một mẫu của các giá trị của một số biến ngẫu nhiên). Định nghĩa của một MSE khác với những gì tương ứng cho dù là một trong những mô tả một ước lượng, hay một yếu tố dự báo.

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

### Công thức 1 MSE

Nếu  $\hat{Y}$  là một vector của  $n$  trị dự báo, và  $Y$  là vector các trị quan sát được, tương ứng với ngõ vào của hàm số phát ra dự báo, thì MSE của phép dự báo có thể ước lượng theo công thức trên.

### 3.3.2 Thuật toán tối ưu hóa Adam

Thuật toán Adam thể hiện sự áp đảo so với các thuật toán khác với 23% tỷ lệ xuất hiện trong các bài báo. Tỷ lệ sử dụng trong thực tế khá khó để ước lượng nhưng có thể còn cao hơn 23% vì nhiều bài báo không nhắc đến thuật toán tối ưu được sử dụng. [6]

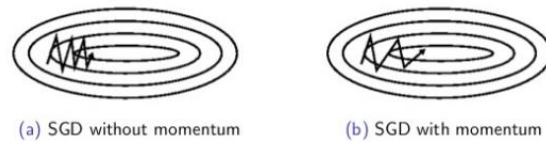
Lợi ích khi sử dụng Adam:

- Không khó khăn để implement
- Độ phức tạp hiệu quả
- Ít bộ nhớ yêu cầu.
- Thích hợp với các bài toán có độ biến thiên không ổn định và dữ liệu training phân mảnh.
- Các siêu tham số được biến thiên một cách hiệu quả và yêu cầu ít điều chỉnh

### SGD

Giả sử ta có vector tham số  $x$  và đạo hàm  $dx$ , thì form update cơ bản là:  $x += - \text{learning\_rate} * dx$

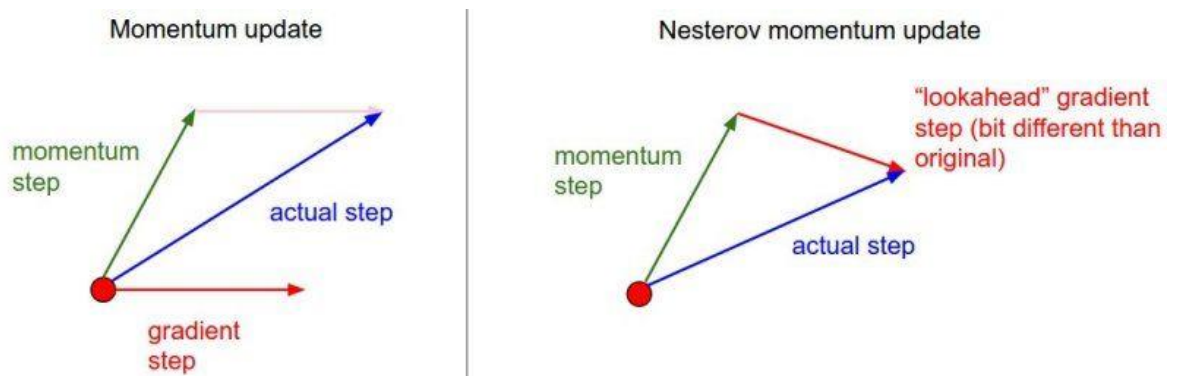
- SGD with Momentum (Momentum optimization):
  - Dưới góc nhìn vật lý, SGD với momentum giúp cho việc hội tụ có gia tốc, làm nhanh quá trình hội tụ trên các đường cong có độ dốc lớn, nhưng cũng đồng thời làm giảm sự dao động khi gần hội tụ.
  - Công thức tương tự trên nhưng chúng ta thêm tích vận tốc của lần trước vào và hệ số gamma thường bằng 0.9



**Hình 33** sự khác như khi có và không có momentum

### Nesterov accelerated gradient

- Cách thức này có 1 ít khác biệt so với momentum update, với momentum update ta tính toán đạo hàm tại vị trí hiện hành rồi sau đó làm 1 cú nhảy tới dựa trên vector momentum trước đó, với Nesterov momentum thay vì tính toán đạo hàm tại điểm hiện hành, chúng ta dựa vào vector momentum cũ để tính toán vị trí sắp tới, rồi sau đó mới dùng đạo hàm tại vị trí mới để correct lại:



**Hình 34** Sự khác nhau giữa khi ta updata lên Nesterov Mommentum

### Adagrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\epsilon I + \text{diag}(G_t)}} \cdot g_t, \quad (1)$$

#### Công thức 2 Adagrad

- Không giống như các cách thức trước, learning rate hầu như giống nhau cho quá trình learning, adagrad coi learning rate cũng là một tham số
- Nó update tạo các update lớn với các dữ liệu khác biệt nhiều và các update nhỏ cho các dữ liệu ít khác biệt
- Adagrad chia learning rate với tổng bình phương của lịch sử biến thiên (đạo hàm)
- Trong đó  $\epsilon$  là hệ số để tránh lỗi chia cho 0, default là  $1e-8$

- $G$  là một diagonal matrix nơi mà mỗi phần tử  $(i,i)$  là bình phương của đạo hàm vector tham số tại thời điểm  $t$

$$G_t = \begin{matrix} \mathbb{R}^{d \times d} \\ \left( \begin{array}{ccc} \square & \dots & \circ \\ \circ & \dots & \square \\ \circ & \dots & \circ \end{array} \right) \end{matrix}$$

- Một lợi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, thường sẽ để default là 0.01 và thuật toán sau đó sẽ tự động điều chỉnh.
- Một điểm yếu của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm learning rate cực kì nhỏ, làm việc training trở nên đóng băng.

### Adadelat

- Adadelat sinh ra để làm giảm nhược điểm của Adagrad (việc làm thay đổi learning rate theo tính đơn điệu giảm)
- Nó giới hạn sự tích lũy của độ biến thiên tới một giới hạn nhất định
- Để làm được điều trên nó đưa ra khái niệm, running average phụ thuộc vào trung bình trước và độ dốc hiện tại.

$$g_{t+1} = \gamma g_t + (1 - \gamma) \nabla \mathcal{L}(\theta)^2$$

$$x_{t+1} = \gamma x_t + (1 - \gamma) v_{t+1}^2$$

$$v_{t+1} = -\frac{\sqrt{x_t} + \epsilon \delta \mathcal{L}(\theta_t)}{\sqrt{g_{t+1}} + \epsilon}$$

$$\theta_{t+1} = \theta_t + v_{t+1}$$

**Công thức 3** Tổng quát công thức Adadelat [7]

### RMSprop

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta) \left( \frac{\partial C}{\partial w} \right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\partial C}{\partial w}$$

**Công thức 4** RMSprop [7]

- RMSprop sinh ra cũng để giải quyết vấn đề của Adagrad
- RMSprop giống với vector update đầu tiên của adadelat

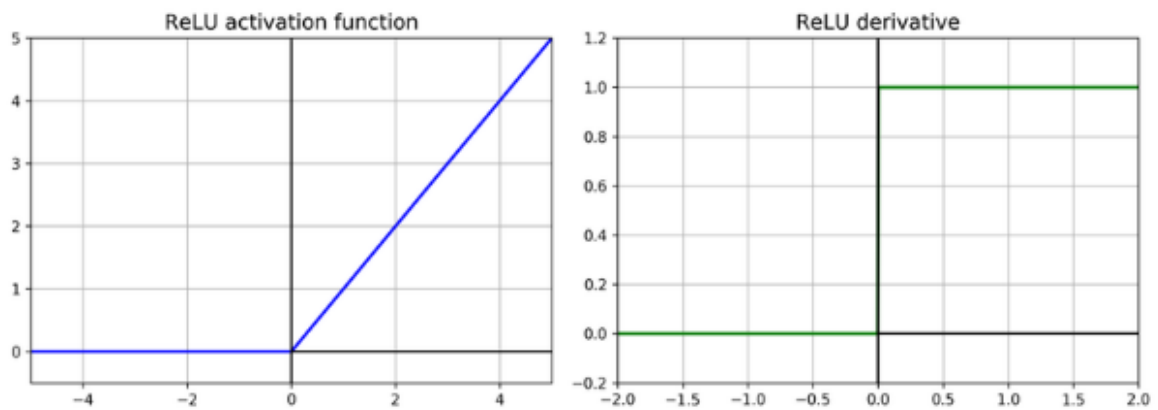
### Adam

- Giống với Adadelta và RMSprop, nó duy trì trung bình bình phương độ dốc (slope) quá khứ  $vt$  và cũng đồng thời duy trì trung bình độ dốc quá khứ  $mt$ , giống momentum.
- Trong khi momentum giống như một quả cầu lao xuống dốc, thì Adam lại giống như một quả cầu rất nặng và có ma sát (friction), nhờ vậy nó dễ dàng vượt qua local minimum và đạt tới điểm tối ưu nhất (flat minimum)
- Nó đạt được hiệu ứng Heavy Ball with Friction (HBF) nhờ vào hệ số  $(mt / \sqrt{vt})$

$$\begin{aligned}
\mathbf{g}_n &\leftarrow \nabla f(\boldsymbol{\theta}_{n-1}) \\
\mathbf{m}_n &\leftarrow (\beta_1 / (1 - \beta_1^n)) \mathbf{m}_{n-1} + ((1 - \beta_1) / (1 - \beta_1^n)) \mathbf{g}_n \\
\mathbf{v}_n &\leftarrow (\beta_2 / (1 - \beta_2^n)) \mathbf{v}_{n-1} + ((1 - \beta_2) / (1 - \beta_2^n)) \mathbf{g}_n \odot \mathbf{g}_n \\
\boldsymbol{\theta}_n &\leftarrow \boldsymbol{\theta}_{n-1} - a \mathbf{m}_n / (\sqrt{\mathbf{v}_n} + \epsilon),
\end{aligned}$$

**Công thức 5** Công thức tối ưu hóa Adam [7]

### 3.3.3 Hàm kích hoạt ReLU và ELU



**Hình 35** Đồ thị hàm ReLU

ReLU (Rectified Linear Unit) được sử dụng rộng rãi gần đây vì tính đơn giản của nó. Đồ thị của hàm ReLU được minh họa trên **Hình 35**(trái). Nó có công thức toán học:

$$RELU(z) = f(z) = \max(0, z)$$

$$f'(z) = \begin{cases} 1, & z > 0 \\ 0, & otherwise \end{cases}$$

**Công thức 6** ReLU và đạo hàm của ReLU

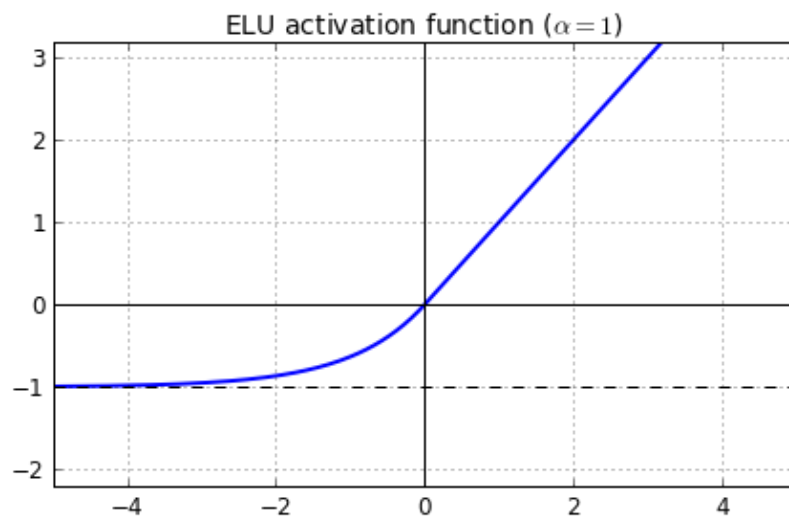
Ưu điểm chính của nó là:

- ReLU được chứng minh giúp cho việc training các Deep Networks nhanh hơn rất nhiều
- ReLU và các biến thể của nó thường được sử dụng thay cho Sigmoid function, do chúng hoạt động tốt hơn với các DNN. Cụ thể khi sử dụng ReLU activation function, chúng ta sẽ tránh được hiện tượng bão hòa đối với các giá trị dương và hàm này khá nhanh khi tính toán [8]

Nhược điểm chính:

- Tuy nhiên khi sử dụng ReLU chúng ta gặp phải một hiện tượng đó là Dying ReLUs, khi một số neuron sẽ chỉ cho ra giá trị là 0 trong quá trình training. Cụ thể khi weights của neuron được update và weighted sum của các inputs của neuron nhỏ hơn 0 dẫn đến output = 0 và gradient = 0. Nếu learning rate lớn, một lượng lớn neurons của network có thể sẽ die. Để giải quyết vấn đề này thì chúng ta sẽ dùng một số biến thể của ReLU.

## ELU



**Hình 36** Đồ thị hàm ELU

ELU là hàm cải tiến lên từ ReLU có công thức:

$$ELU(z) = \begin{cases} z, & z \geq 0 \\ \alpha [\exp(z) - 1], & \text{otherwise} \end{cases}$$

### Công thức 7 Hàm ELU [1]

ELU có performance khá tốt so với các biến thể trước đó của ReLU, với một số cải tiến như sau:

- Cho ra giá trị âm khi  $z < 0$ , giá trị trung bình của đầu ra sẽ gần với 0 -> tránh được hiện tượng vanishing gradients
- Gradient sẽ khác 0 khi  $z < 0$  -> tránh được hiện tượng dying ReLUs
- ELU là một hàm *khá* trơn -> việc sử dụng Gradient Descent sẽ hiệu quả hơn



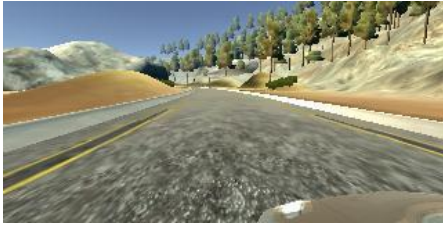
## Chương 4 Mô hình và các kỹ thuật trong việc xử lý bài toán giả lập hành vi

Ở Chương 2 và Chương 3 em đã khái quát những kiến thức cơ bản từ xe tự lái đến những kiến thức em sẽ sử dụng cho bài toán giả lập hành vi , và sang chương 4 em sẽ nói sâu hơn về cách thu thập dữ liệu , cân bằng dữ liệu , các kỹ năng huấn luyện mạng ....

### 4.1 Xây dựng bài toán

Bước đầu tiên của quá trình tự học là lựa chọn các Data Frame để sử dụng . Dữ liệu thu thập được sẽ được gán nhãn với các đường dẫn , phanh , các góc lái ...

Trong chế độ huấn luyện simulator sẽ ghi lại với tần số 10hz , ngoài ra tại một thời điểm nhất định nó sẽ ghi lại ba hình ảnh chụp từ máy ảnh trái , phải và giữa như **Bảng 1** dưới đây:

Trái	Giữa	Phải
		

**Bảng 1** Ba góc của camera

Sau khi chọn training mode thì sẽ phải làm quen với việc di chuyển của ô tô bằng các phím mũi tên. Đến khi lái mượt rồi thì chọn nút record. Chọn folder muốn lưu dữ liệu và chọn select.



Lái xe khoảng 10 phút sẽ ra khoảng 18000 ảnh (6000 ảnh từ mỗi camera). Ảnh màu từ camera ở ô tô có kích thước 320\*160

Sau khi ta lái xe sẽ thu thập được các dữ liệu hình ảnh và một tập **driving\_log** là nhãn của toàn bộ ảnh ta đã ghi lại ở trong simulator.

	A	B	C	D	E	F	G
1	C:\Users\A	C:\Users\A	C:\Users\A	0	0	0	0.649786
2	C:\Users\A	C:\Users\A	C:\Users\A	0	0	0	0.627942
3	C:\Users\A	C:\Users\A	C:\Users\A	0	0	0	0.62291
4	C:\Users\A	C:\Users\A	C:\Users\A	0	0	0	0.619162
5	C:\Users\A	C:\Users\A	C:\Users\A	0	0	0	0.615438
6	C:\Users\A	C:\Users\A	C:\Users\A	0	0	0	0.610506
7	C:\Users\A	C:\Users\A	C:\Users\A	0	0	0	0.606834
8	C:\Users\A	C:\Users\A	C:\Users\A	0	0	0	0.601971
9	C:\Users\A	C:\Users\A	C:\Users\A	0	0	0	0.59835
10	C:\Users\A	C:\Users\A	C:\Users\A	0	0.024006	0	0.620654
11	C:\Users\A	C:\Users\A	C:\Users\A	0	0.202773	0	0.707
12	C:\Users\A	C:\Users\A	C:\Users\A	0	0.382694	0	0.946799
13	C:\Users\A	C:\Users\A	C:\Users\A	-0.05	0.642727	0	1.434013
14	C:\Users\A	C:\Users\A	C:\Users\A	-0.25	0.863326	0	2.173052
15	C:\Users\A	C:\Users\A	C:\Users\A	-0.4	1	0	2.864847
16	C:\Users\A	C:\Users\A	C:\Users\A	-0.53543	1	0	3.791584
17	C:\Users\A	C:\Users\A	C:\Users\A	-0.43196	1	0	4.489107
18	C:\Users\A	C:\Users\A	C:\Users\A	-0.19096	1	0	5.422441
19	C:\Users\A	C:\Users\A	C:\Users\A	-0.00614	1	0	6.111838
20	C:\Users\A	C:\Users\A	C:\Users\A	0	1	0	7.026899
21	C:\Users\A	C:\Users\A	C:\Users\A	0	1	0	7.709986
22	C:\Users\A	C:\Users\A	C:\Users\A	0	1	0	8.614678
23	C:\Users\A	C:\Users\A	C:\Users\A	-0.05	1	0	9.288085
24	C:\Users\A	C:\Users\A	C:\Users\A	-0.25	1	0	10.16289
25	C:\Users\A	C:\Users\A	C:\Users\A	-0.41765	1	0	10.792

**Hình 37** Thông số của tập nhãn

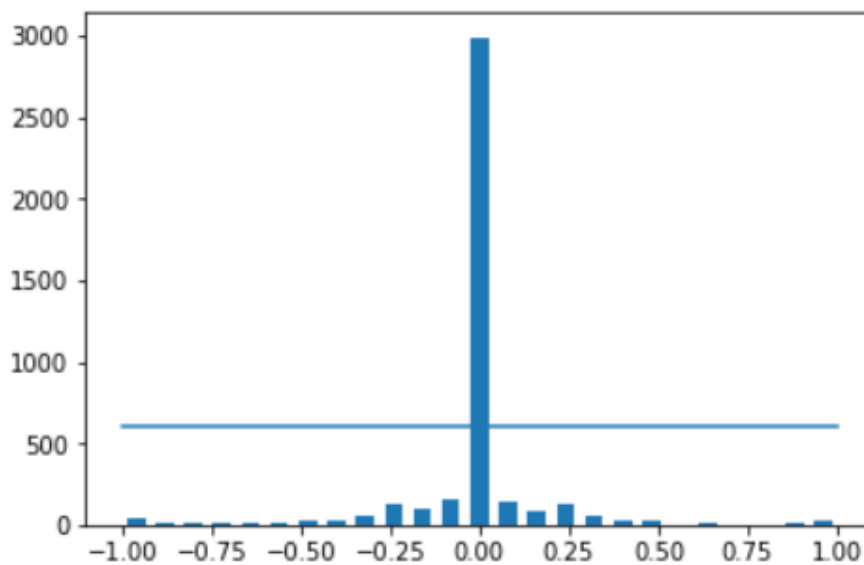
Như ta thấy trên **Hình 37** biểu diễn các thông số của nhãn dữ liệu như sau:

- Cột A biểu diễn đường dẫn ảnh camera trước của xe khi đi trên đường
- Cột B biểu diễn đường dẫn ảnh camera trái của xe khi đi trên đường
- Cột C biểu diễn đường dẫn ảnh camera phải của xe khi đi trên đường
- Cột D biểu diễn thông số góc lái có giá trị từ  $[-1,1]$
- Cột E biểu diễn thông số độ giảm tốc có giá trị từ  $[0,1]$
- Cột F biểu diễn thông số phanh có giá trị  $[0,1]$

- Cột G biểu diễn thông số tốc độ có giá trị [0,30]

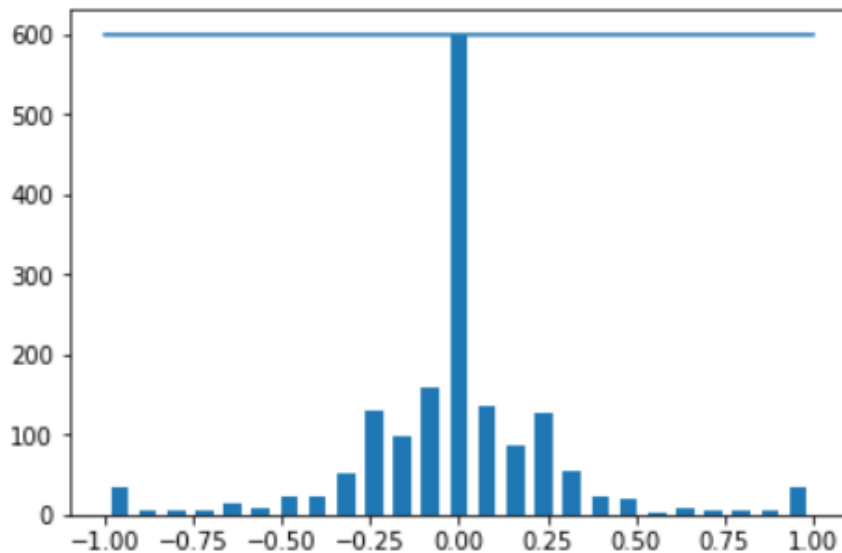
Dữ liệu thu thập được sử dụng trong đồ án là 12,159 ảnh và hơn 4053 khung dữ liệu từ tập nhãn. Trong khi training ở simulator có chưa rất nhiều khúc cua nông và đoạn đường thẳng. Do đó phần lớn các góc lái được ghi lại là 0. Do đó ta cần phải cân bằng lại dữ liệu và tiền xử lý hình ảnh để dữ liệu có thể trích chọn đặc trưng và có thể học tốt hơn, **phần 4.2** tiếp theo em sẽ nói sâu hơn về cách xử lý này.

## 4.2 Cân bằng dữ liệu



**Hình 38** Sơ đồ dữ liệu trong file driving\_log

Như trên biểu đồ **Hình 38** ta thấy số góc lái về hướng giữa tức  $steering\_angle = 0$  chiếm rất nhiều trong sơ đồ, vì thế nếu ta training với bộ dữ liệu gốc thì xe sẽ có xu hướng quyết định chạy theo đường thẳng thay vì rẽ nếu có khúc cua vì thế ta cần phải lược bỏ các phần từ 0 đến một giá trị nhất định nào đó để bộ dữ liệu có thể ra quyết định chính xác nhất khi training theo mạng CNN, ở đây ta sẽ loại bỏ tất cả các dữ liệu  $steering\_angle = 0$  lớn hơn 600.



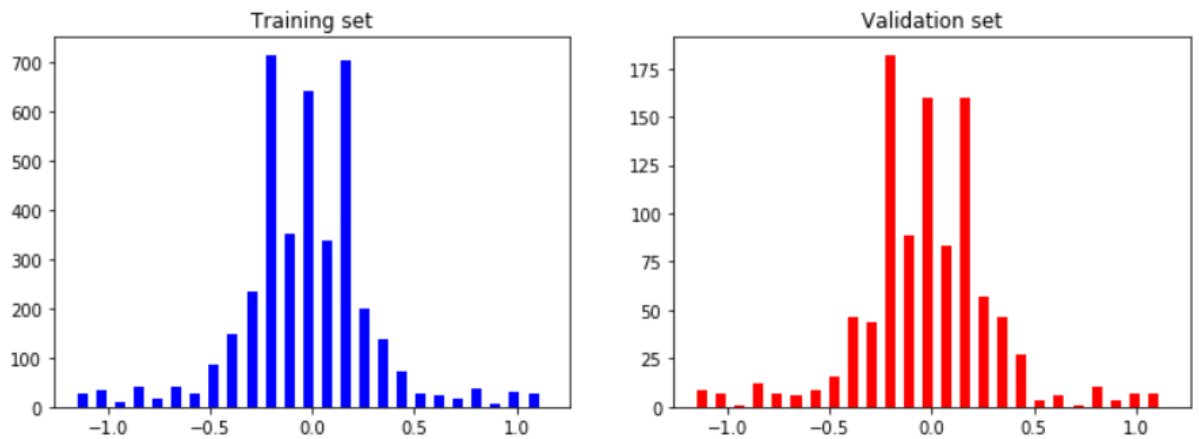
**Hình 39** Dữ liệu sau khi được lọc

Sau khi có dữ liệu được lọc thì nhìn vào biểu đồ **Hình 39** ta thấy dữ liệu của camera giữa ( $steering\_angle = 0$ ) và camera trái ( $steering\_angle < 0$ ) dường như nhiều hơn bên phải ( $steering\_angle > 0$ ), ta cần phải làm cho cân bằng dữ liệu của hai bên trái và phải bằng cách thêm 0.15 vào góc lái bên phải và -0.15 vào góc lái bên trái. Ý tưởng chính là camera bên phải phải di chuyển sang phải để vào giữa và camera trái di chuyển sang phải.

```
print(data.iloc[1])
def load_img_steering(datadir, df):
    image_path = []
    steering = []
    for i in range(len(data)):
        indexed_data = data.iloc[i]
        center, left, right = indexed_data[0], indexed_data[1], indexed_data[2]
        image_path.append(os.path.join(datadir, center.strip()))
        steering.append(float(indexed_data[3]))
        # left image append
        image_path.append(os.path.join(datadir, left.strip()))
        steering.append(float(indexed_data[3])+0.2)
        # right image append
        image_path.append(os.path.join(datadir, right.strip()))
        steering.append(float(indexed_data[3])-0.2)
    image_paths = np.asarray(image_path)
    steerings = np.asarray(steering)
    return image_paths, steerings
image_paths, steerings = load_img_steering(datadir + '/IMG', data)
print(image_paths)
```

**Hình 40** Xử lý khi tăng góc lái bên phải và giảm góc lái bên trái

Ngoài ra ta sẽ lấy ngẫu nhiên 20% dữ liệu training từ bộ dữ liệu ảnh gốc là 3991 ảnh và 20% tập valid từ tập training là 998 ảnh và sau đây là kết quả sau khi đã cân bằng dữ liệu và tách thành hai tập train và valid, còn tập test thì ta sẽ dùng simulator để thử nghiệm, ta có biểu đồ sau:

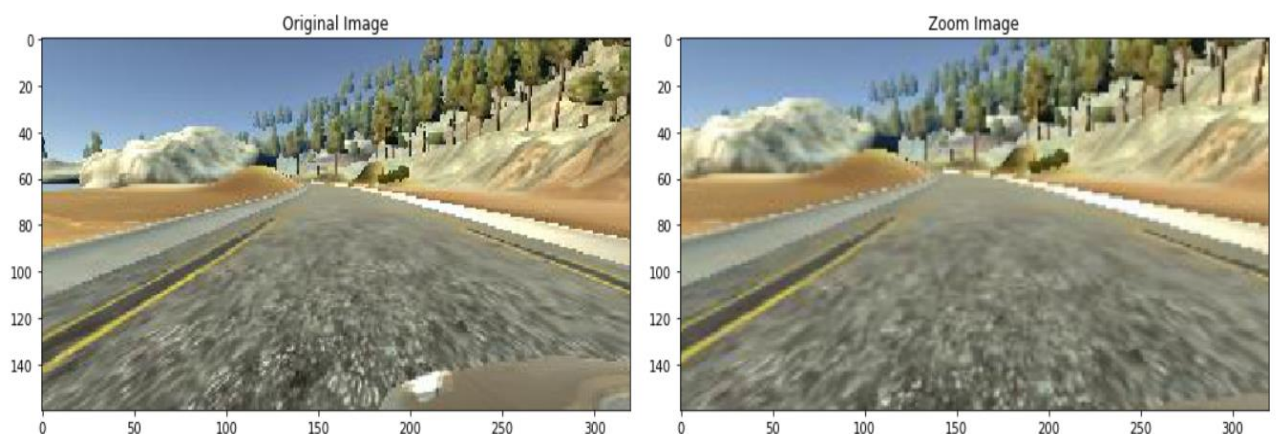


**Hình 41** Biểu đồ dữ liệu sau khi được cân bằng và tách thành 2 tập train và valid

### 4.3 Các kỹ thuật xử lý ảnh tăng cường

Ảnh gốc từ các camera của bộ dataset vẫn chưa thể đủ để huấn luyện mạng vì nó vẫn là một bộ dữ liệu khá đồng nhất và không có nhiều điều kiện trong đó , vì vậy ta cần các phương pháp xử lý ảnh tăng cường để tạo thêm các điều kiện thời tiết , vùng sáng tối , lật ảnh ... Để làm phong phú hơn cho tập dữ liệu.

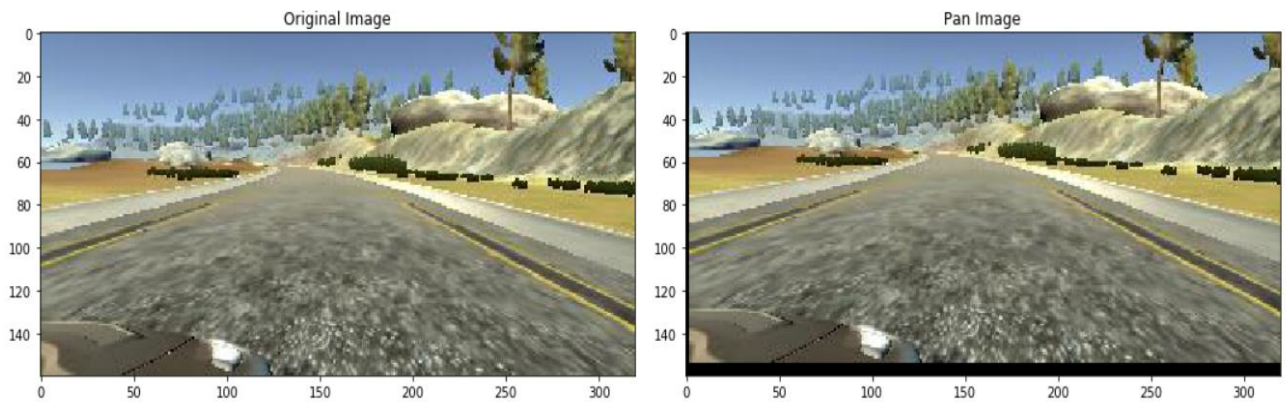
#### Zooming



**Hình 42** kỹ thuật Zooming

Ta thấy trên **Hình 42** phần mũi xe đã bị loại bỏ và ảnh đã được cắt và không bị mất đi kích thước ban đầu của ảnh là 320\*160

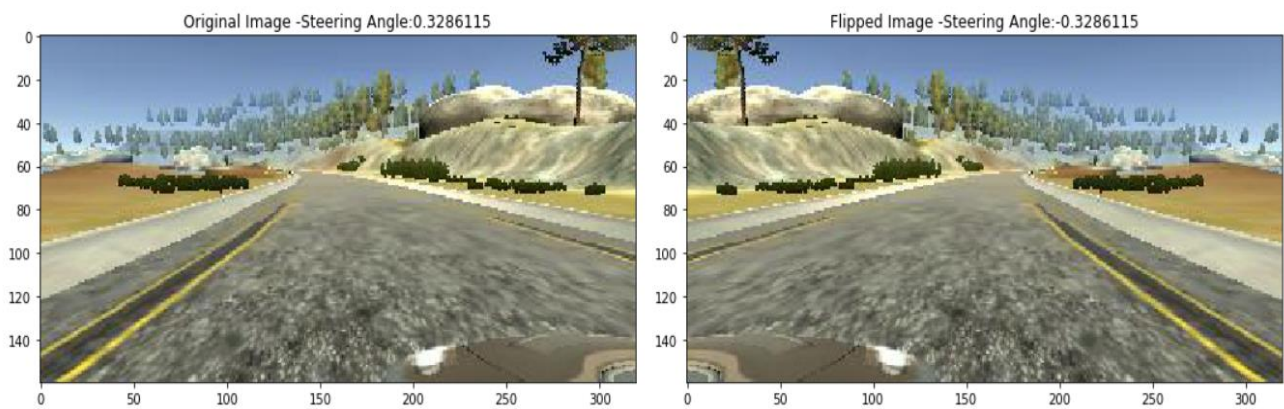
### Panning



**Hình 43** Kỹ thuật Panning

Ảnh trên **Hình 43** đã được thu lại và 2 trục đã được thu nhỏ lại và tạo thêm 2 viền đen, ta đã loại bỏ và tạo cho ảnh một trọng tâm vào đường mà không mất đi kích thước gốc

### Flipping

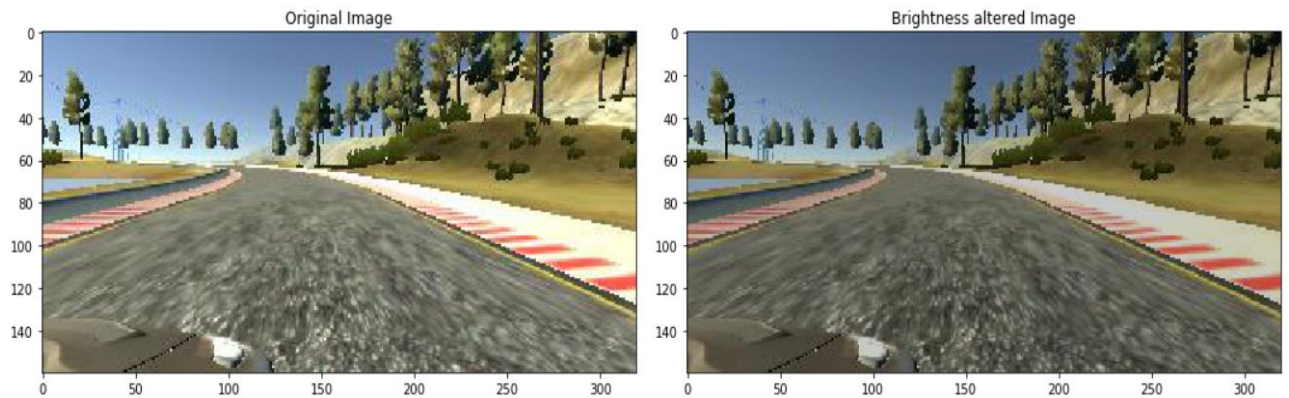


**Hình 44** Kỹ thuật Flipping

Vì như trên biểu đồ **Hình 41** ta thấy dữ liệu góc lái bên trái nhiều hơn góc lái bên phải nên ta sẽ đảo ngược góc theo chiều ngang hay bất kỳ góc lái tương ứng nào như trên **Hình 44**



## Điều chỉnh độ sáng

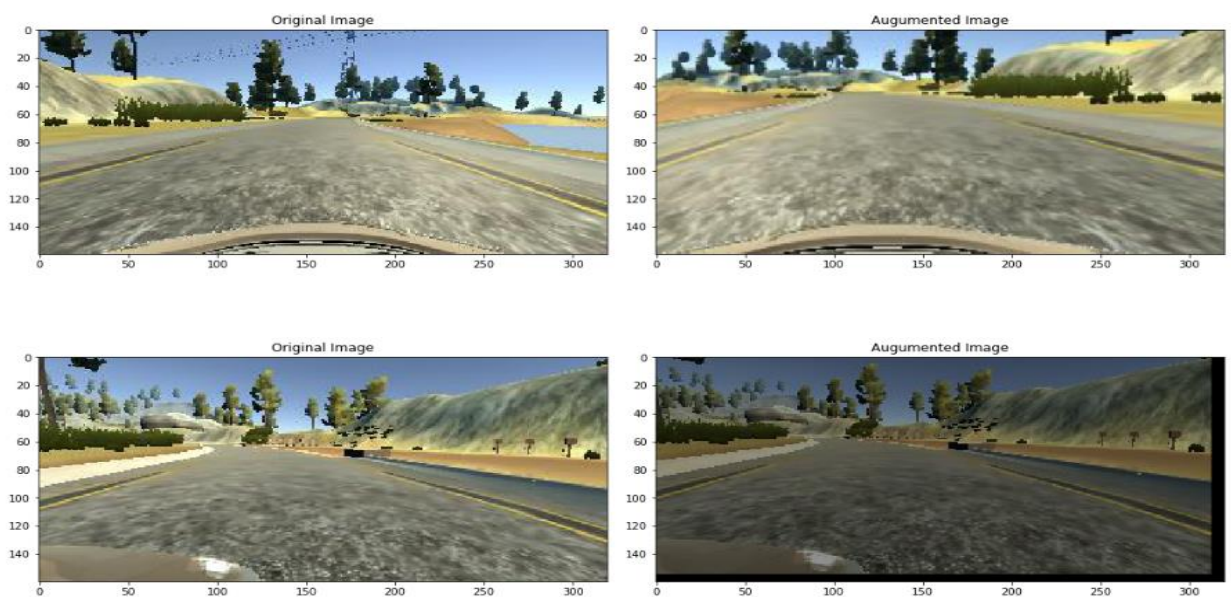


**Hình 45** Kỹ thuật điều chỉnh độ sáng

Trong kỹ thuật này ta sẽ điều chỉnh độ sáng tối của một bức ảnh gốc để mô phỏng lái xe trong các điều kiện khác nhau, ta có thể thấy hình ở trên có sự sáng tối rõ rệt.

## Kết hợp các kỹ thuật

Như đã trình bày ở trên, ta sử dụng 4 kỹ thuật tăng cường ảnh để làm cho bộ dữ liệu phong phú hơn ở bước này ta sẽ tạo ra một bộ lọc chỉ lấy 50% cho toàn bộ kỹ thuật ở trên vì nếu kết hợp 4 kỹ thuật này lại sẽ tạo ra một bộ dữ liệu khoảng  $12000 \times 4 = 48000$  cái ảnh, và điều này là không cần thiết, và vì thế ta sẽ lấy 50% của bộ dữ liệu đó và trộn lẫn với nhau và ta được một bộ ảnh kết hợp như **Hình 46**:



**Hình 46** Tổng hợp các kỹ thuật tăng cường ảnh

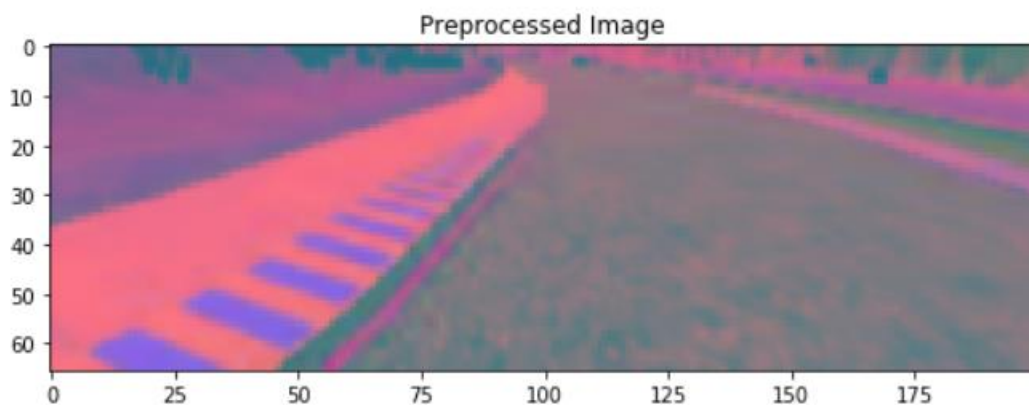
## Tiền xử lý hình ảnh

Trước khi cho vào training thì ta cần phải xử lý đưa ảnh từ kích thước 320\*160 về kích thước chuẩn của hình ảnh trong NVIDIA paper [3] là 200\*66 và chuyển các kênh màu RGB về kênh màu YUV và làm mờ ảnh để không bị nhiễu và cho kết quả tốt hơn khi training.

```
#Preprocessing Images
def img_preprocess(img):
    img = img[60:135, :, :]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img, (3,3), 0)
    img = cv2.resize(img, (200,66))
    img = img/255
    return img
```

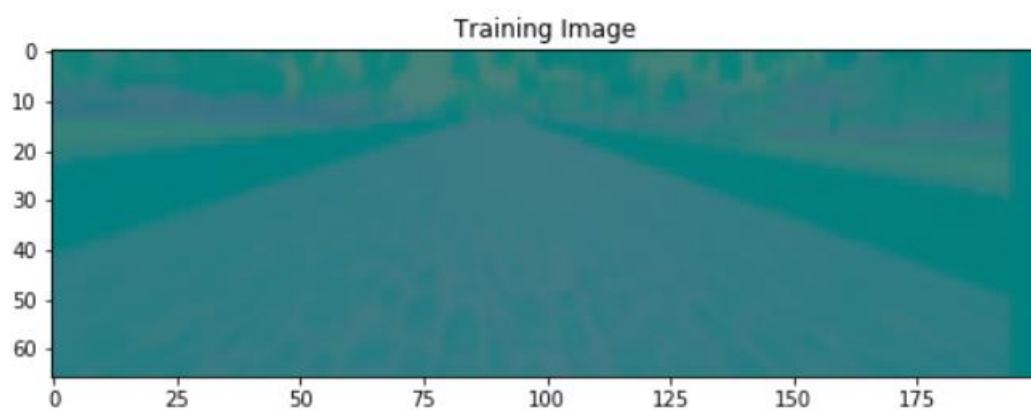
**Hình 47** Cấu trúc RGB sang YUV

Và ta thu được kết quả sau:

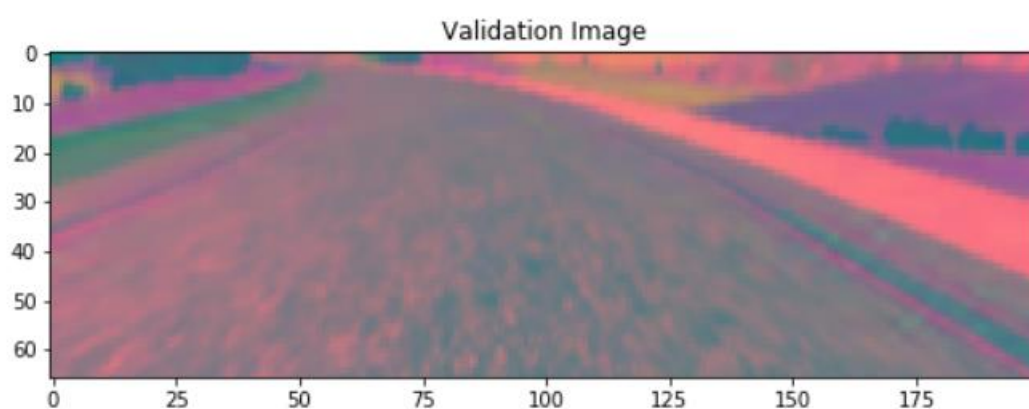


**Hình 48** RGB sang YUV

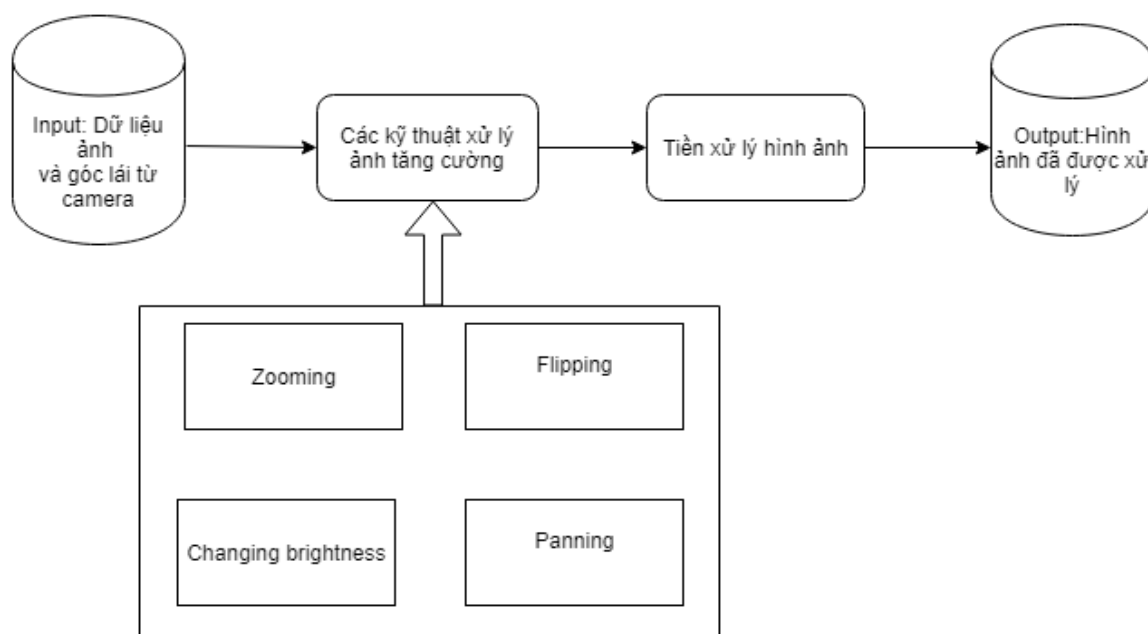
Cuối cùng ta sẽ kết hợp toàn bộ các kỹ thuật đã “Trộn lẫn” ở trên và hình ảnh đã được tiền xử lý thì ta sẽ được kết quả của hai tập training và valid như sau:



**Hình 49** Hình ảnh của tập training đã được kết hợp



**Hình 50** Hình ảnh của tập validation sau khi kết hợp



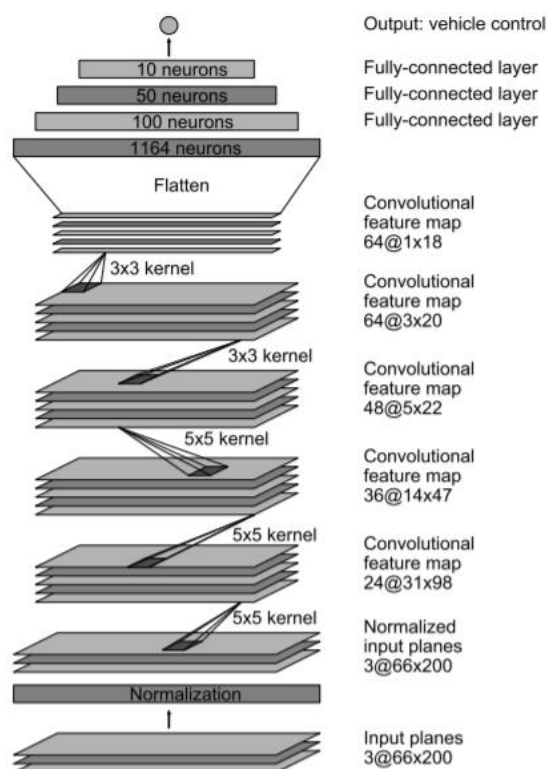
**Hình 51** Sơ đồ các kỹ thuật xử lý ảnh tăng cường



**Tại sao khi được xử lý thì sao hình ảnh tập validation không thay đổi?** Đơn giản bởi vì tập training set bao gồm dữ liệu đầu vào và nhãn. Với tập training, mô hình có thể nhìn thấy cả dữ liệu và nhãn của ảnh. Nó sử dụng dữ liệu này để tối ưu loss function thông qua việc điều chỉnh parameter. tập validation cũng có dữ liệu giống như tập training. Nhưng mô hình không hề nhìn thấy nhãn. Mô hình đơn thuần dùng dữ liệu đầu vào của tập validation để tính toán ra output. Sau đó nó so sánh với nhãn để tính loss function. Parameter hoàn toàn không được điều chỉnh ở bước này. Tập validation là bộ dữ liệu để chúng ta giám sát mô hình. Ta sử dụng kết quả của mô hình ở tập training và tập validation để đưa ra các quyết định như điều chỉnh hyperparameter, bổ sung thêm dữ liệu... Mô hình cần phải dự đoán tốt ở tập validation. Tức là nó phải làm tốt với những dữ liệu ảnh và góc lái mà nó chưa từng nhìn thấy.

## 4.4 Huấn luyện với CNN

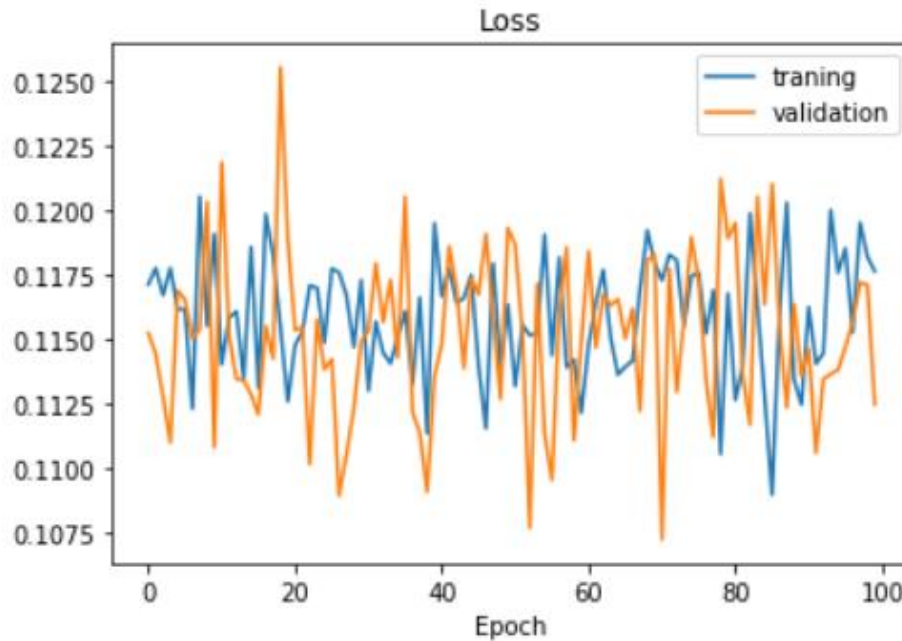
Như đã đề cập rất nhiều ở các phần trước em sẽ dùng mô hình huấn luyện của NVIDIA, gồm 27 000 000 kết nối và 250 000 tham số.



**Hình 52** Kiến trúc CNN của NVIDIA paper [3]

Tất nhiên là khi ta có được một mô hình CNN tốt thì chưa phải là xong, ta cần phải chọn các hàm kích hoạt và cải thiện nó để nó phù hợp với dữ liệu đầu vào và phải chọn các hàm kích hoạt lẫn các kỹ thuật để tránh Overfitting. Dưới đây là các thử nghiệm để chọn ra một mô hình huấn luyện tốt nhất mà em đã thử nghiệm và nghiên cứu được.

#### 4.4.1 Thử nghiệm 1



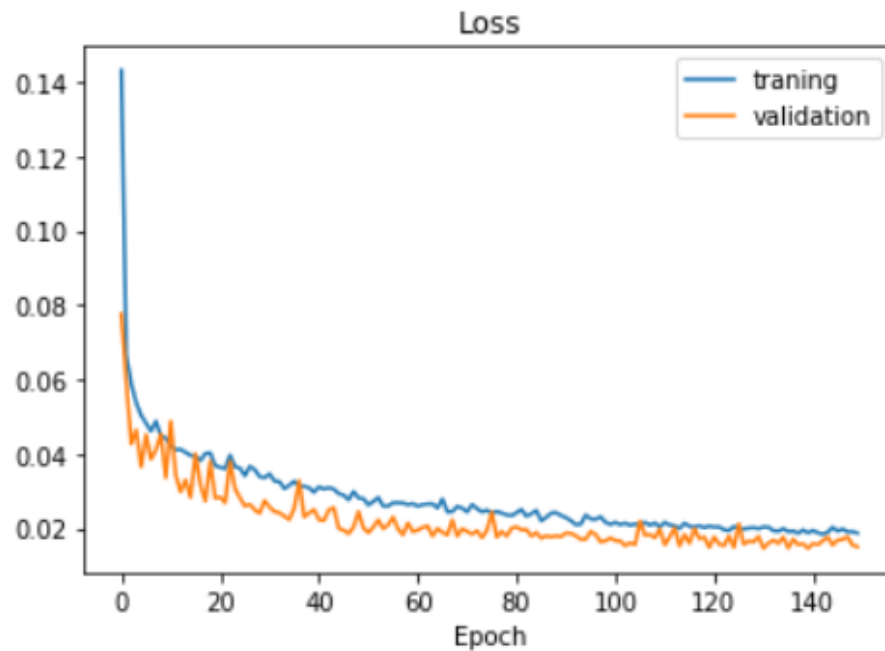
**Hình 53** Kết quả mô hình thử nghiệm một

- Lambda- Conv- Conv- Dropout- Conv- Dropout- Conv- Dropout- Conv- Flatten- FC- FC- FC

Như ta thấy ở trên mô hình này thực sự quá tệ khi ta đặt Dropout vào các lớp Conv, mô hình bị Overfitting và không ổn định. Ngoài ra trong trường hợp này hàm kích hoạt ReLU thực sự không thích hợp bởi vì dữ liệu góc lải của ta trong khoảng từ  $[-1,1]$ , đối với ReLU thì nó sẽ làm tròn những giá trị nhỏ hơn 0 về 0 và gây ra tình trạng “dying ReLU”. Và từ mô hình này em đã rút kinh nghiệm sử dụng hàm kích hoạt ELU thay cho ReLU.

#### 4.4.2 Thử nghiệm 2

Trong trường hợp này ta sẽ dùng hàm kích hoạt ELU và sử dụng thêm hàm kích hoạt Tanh ở Dense cuối cùng thay vì mặc định là Linear

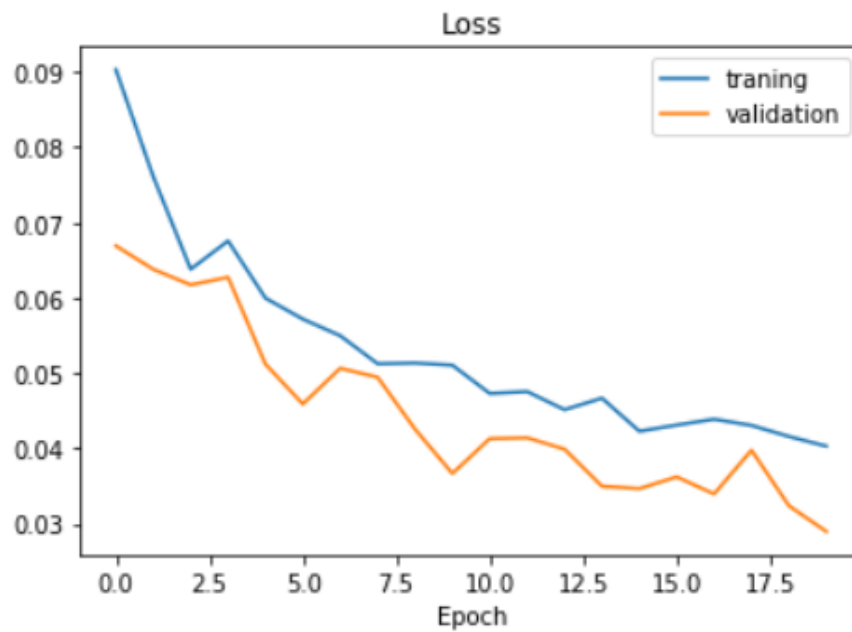


**Hình 54** Kết quả mô hình thử nghiệm hai

- Conv- BatchNorm- Conv- BatchNorm- Conv- BatchNorm- Conv- BatchNorm- Conv- BatchNorm- Flatten- FC –FC –FC – FC

Mô hình được trên Batch Normalization để tăng hiệu quả trong quá trình học và sử dụng hàm tanh ở Dense cuối cùng. Nhìn ở trên thì mô hình hoạt động khá ổn định giữa validation loss và training loss. Nhưng khi cho vào simulator test thì khi thử nghiệm ở mô hình ở địa hình mới thì xe đã đâm vào thanh chắn đường.

### 4.4.3 Thử nghiệm 3



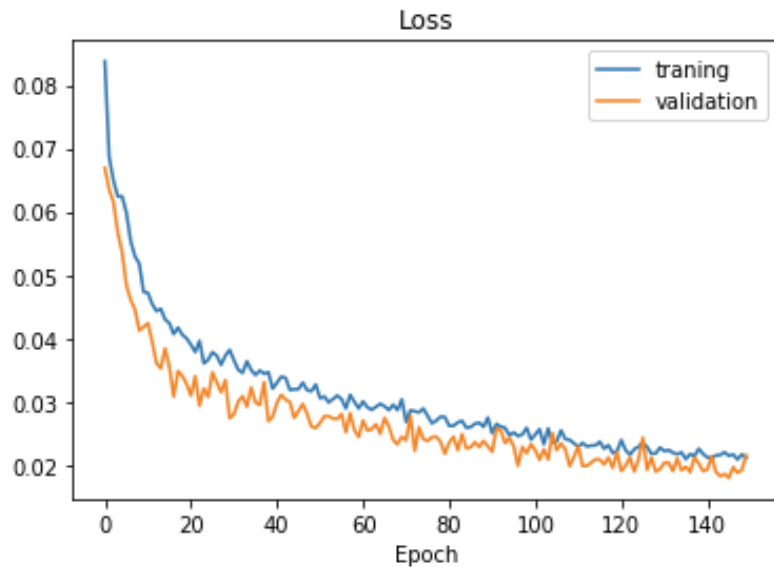
**Hình 55** Kết quả mô hình thử nghiệm 3

- Conv- Conv- Conv- Conv- Conv- Flatten- FC- FC- FC- FC

Trong mô hình này ta sẽ train 20 Epochs để thử ,và có vẻ như những lần huấn luyện ở 20 Epochs này cho ra kết quả khá ổn, mô hình này khá tương đương với mô hình gốc bởi và cho ra kết quả xe có thể chạy ổn định trên cả hai làn đường trong simulator. Nhưng ta cần training nhiều hơn để có kết quả tốt nhất.

### 4.4.4 Mô hình cuối

Đây là mô hình em thử nghiệm được cho ra kết quả chạy tốt nhất trên cả hai khung đường so với các lần thử nghiệm trước.



**Hình 56** Kết quả mô hình cuối

Mô hình thử nghiệm này giống mô hình thứ ba nhưng được huấn luyện thêm 130 lần nữa là 150 lần huấn luyện và sử dụng hàm kích hoạt ELU với các layers

Mô hình thử nghiệm này giống mô hình thứ ba nhưng được huấn luyện thêm 130 lần nữa là 150 lần huấn luyện , giảm learning rate xuống còn 0.0001 và sử dụng hàm kích hoạt ELU với các layers

- **Input:** 66x200x3

**Layer 1 :** Conv 24 , bộ lọc 5x5, bước sải 2x2, hàm kích hoạt ELU

**Layer 2 :** Conv 36 , bộ lọc 5x5, bước sải 2x2, hàm kích hoạt ELU

**Layer 3 :** Conv 48 , bộ lọc 5x5, bước sải 2x2, hàm kích hoạt ELU

**Layer 4 :** Conv 64 , bộ lọc 5x5, bước sải 2x2, hàm kích hoạt ELU

**Layer 5 :** Conv 64 , bộ lọc 5x5, bước sải 2x2, hàm kích hoạt ELU

**Layer 6 :** Conv 64 , bộ lọc 5x5, bước sải 2x2, hàm kích hoạt ELU

**Layer 7 :** Flatten layer.

**Layer 8 :** FC 100 , Hàm kích hoạt ELU

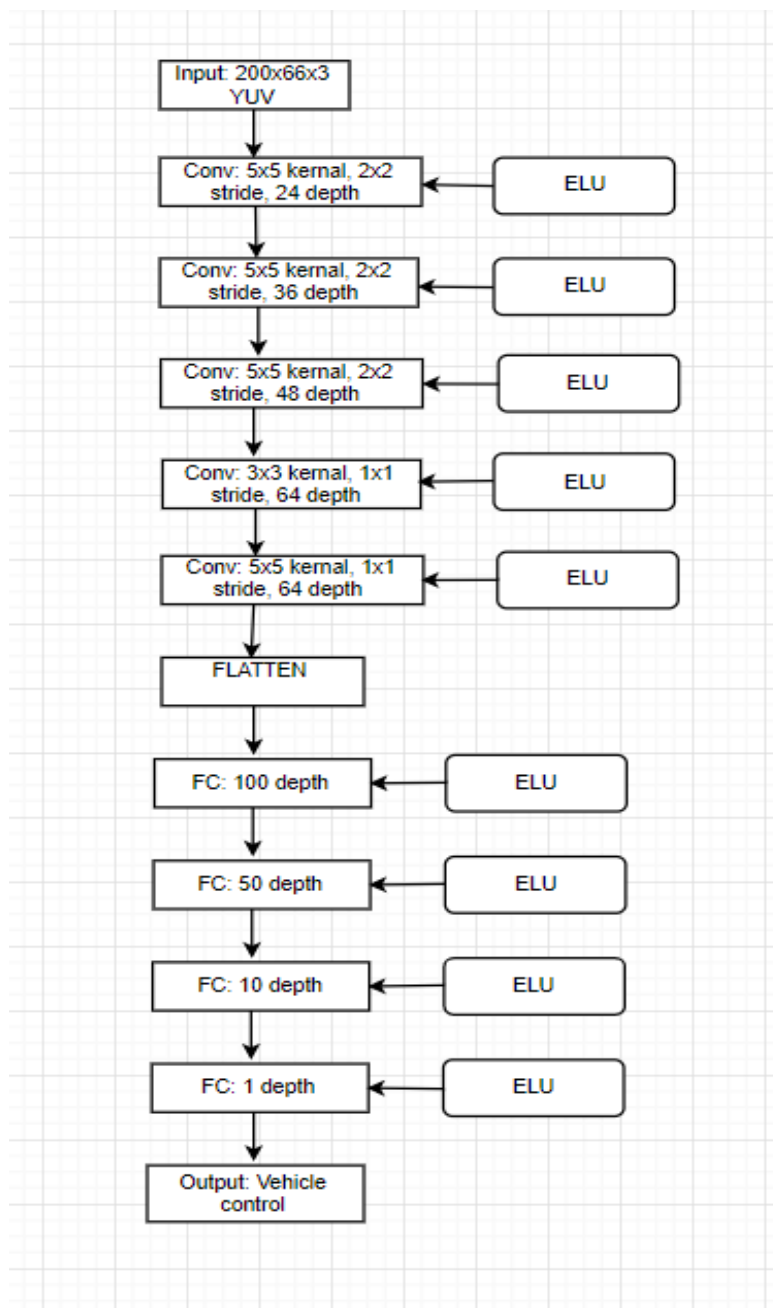
**Layer 9 :** FC 50 , Hàm kích hoạt ELU

**Layer 10:** FC 10 , Hàm kích hoạt ELU

**Layer 11:** FC 1 , Hàm kích hoạt ELU hoặc Linear

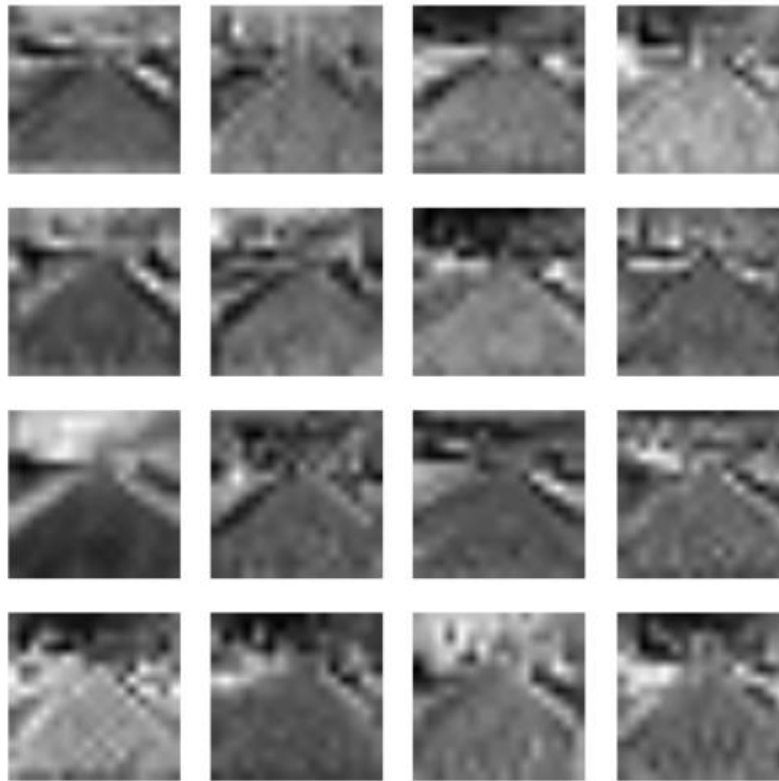
**Output :** Vehicle Control

- 150 epochs, lr = 1e-4, Optimizer Adam, loss MSE



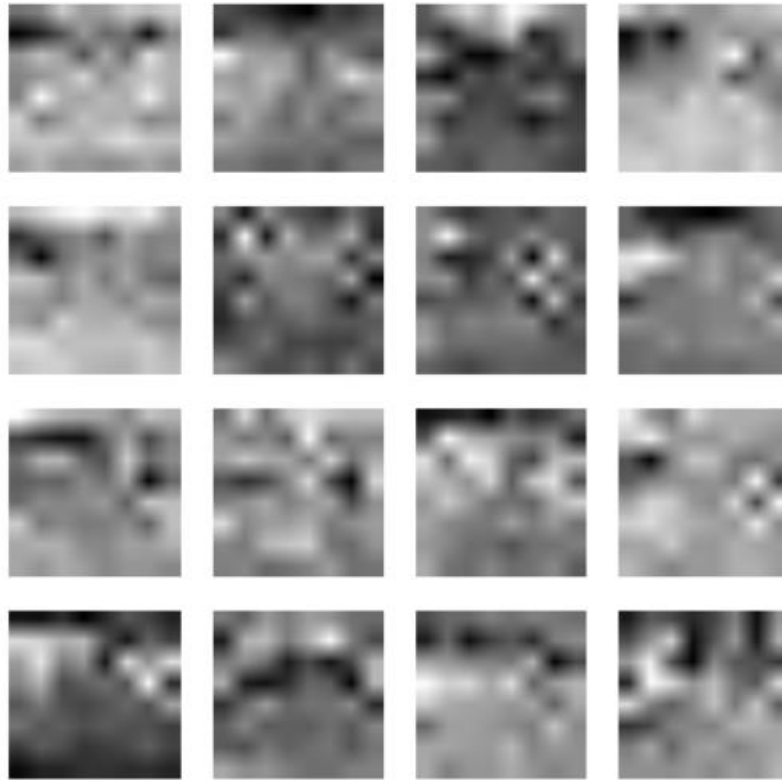
**Hình 57** Kiến trúc mô hình cuối

**Một số hình ảnh mà cách mạng model hiểu cách học điều khiển xe:**



**Hình 58** Đặc trưng khi mạng học tìm đường lớp Conv thứ nhất

Đây là một số đặc trưng sau lần học đầu tiên. Như ta có thể thấy model đã phát hiện các cạnh và đường thẳng phía trước trong các đặc trưng trích xuất được và từ đó mạng sẽ tự học được qua những đặc trưng này và phát hiện được đâu là đường đi.



**Hình 59** Đặc trưng khi mạng học tìm đường lớp Conv thứ 2

Các đặc trưng ở lớp thứ hai khá khó nhìn hơn lớp thứ nhất, tại thời điểm này đã rất khó để xem được mạng đang làm gì.

### Tổng kết:

- Bước đầu tiên để đánh giá mô hình hoạt động tốt như thế nào, em đã chia dữ liệu hình ảnh và góc lái của em thành training set và validation set. Em thấy rằng mô hình đầu tiên MSE của validation set và training set không ổn định và bị overfitting quá mức.
- Để xử lý tình trạng overfitting em đã thêm một số lớp dropout và batch normalization vào các lớp Conv và FC thì kết quả đã khá hơn rất nhiều và cho ra một mô hình khá tốt nhưng khi cho vào để xe chạy thì đã bị đâm. Để khắc phục tình trạng em đã thử huấn luyện lại từ 100 lên 150 epochs nhưng kết quả vẫn vậy.
- Sau đó em đã loại bỏ hết tất cả các hàm ReLU vào và thay bằng ELU và chạy thử trên 20 epochs đầu với ( $lr = 0.001$ ) và đã cho ra mô hình của **Hình 56**.



- Sau khi thấy một mô hình khá tốt em đã tăng số lần học lên là 150 lần và giảm lr xuống còn 0.0001 và được mô hình như **Hình 56** và build ra thành model cuối cùng để sử dụng.

Mô hình thử nghiệm	Thử nghiệm 1	Thử nghiệm 2	Thử nghiệm 3	Thử nghiệm cuối
Dung lượng	2.98	2.95	2.92	2.98
Số lần huấn luyện	100	100	20	150
Định dạng	.h5	.h5	.h5	.h5

**Bảng 2** Thông số các số lần thử nghiệm

## 4.1 Xây dựng demo

### 4.1.1 Thư viện và công cụ sử dụng

- **Thư viện Keras** : là một library được phát triển vào năm 2015 bởi François Chollet, là một kỹ sư nghiên cứu deep learning tại google. Nó là một open source cho neural network được viết bởi ngôn ngữ python. keras là một API bậc cao có thể sử dụng chung với các thư viện deep learning nổi tiếng như tensorflow(được phát triển bởi gg), CNTK(được phát triển bởi microsoft), theano(người phát triển chính Yoshua Bengio).
- **Thư viện OpenCV** : là một thư viện mã nguồn mở hàng đầu cho thị giác máy tính (computer vision), xử lý ảnh và máy học, và các tính năng tăng tốc GPU trong hoạt động thời gian thực.
- **Thư viện Scikit-learn**: Thư viện này chứa rất nhiều thuật toán học máy và các công cụ tiền xử lý dữ liệu hiệu quả.
- **Thư viện matplotlib**: Dùng để trực quan hóa dữ liệu qua biểu đồ
- **Thư viện Python Socket.io**: Module socket trong Python sẽ giúp chúng ta thực hiện các kết nối client server. để giao tiếp giữa các máy với nhau.

Mục đích	Công cụ	URL
Thiết kế giao diện	draw.io	<a href="https://www.draw.io/">https://www.draw.io/</a>
Môi trường	Python 3.6	<a href="https://www.python.org/">https://www.python.org/</a>
Lập trình và huấn luyện	Google Colab	<a href="https://colab.research.google.com">https://colab.research.google.com</a>
Kết nối simulator	Visual studio	<a href="https://visualstudio.microsoft.com/">https://visualstudio.microsoft.com/</a>
Giả lập xe tự lái	Udacity's Self-Driving Car Simulator	<a href="https://github.com/udacity/self-driving-car-sim">https://github.com/udacity/self-driving-car-sim</a>

**Bảng 3** Các công cụ sử dụng

#### 4.1.2 Kết quả đạt được

Với những kiến thức em đã học được và từ những thư viện và công cụ ở trên em đã xây dựng được một model có khả năng tự quyết định rẽ và lái, cải thiện từ những kinh nghiệm của người lái xe.

Tiêu chí thống kê	Số lượng
Số dòng code	~ 1125 dòng
Số lượng file (.py, .ipynb,.h5)	8
Số lượng mô hình	4
Dung lượng toàn bộ file	362 MB

**Bảng 4** Tiêu chí thống kê

## 4.2 Triển khai

Mô hình và giả lập được huấn luyện và sử dụng trên máy tính cá nhân có cấu hình và các phần mềm cần thiết sau:

- Ram: 8GB DDR4 Bus 2133Mhz (2 Slot, 4GB x 01)
- CPU: 7th Generation Intel® Kaby Lake Core™ i5 \_ 7200U (2.50 GHz, 3M Cache, up to 3.10 GHz)
- Hệ điều hành : Ubuntu 18.04
- Ngôn ngữ sử dụng : Python
- GPU huấn luyện: Nvidia tesla k80 của Google Colab

## Chương 5 Thực nghiệm và đánh giá kết quả

Với những mục tiêu đặt ra đầu tiên , để xây dựng được một model tốt , trong quá trình xây dựng đồ án, em đã lựa chọn những công nghệ và những phương pháp phù hợp nhất để đảm bảo cho ra sản phẩm demo cuối cùng là một model có khả năng học và cải thiện từ hành vi lái xe của người dùng .

Sau khi nghiên cứu một bài báo của công ty NVIDIA em đã cảm thấy rất phấn khích và đã bắt tay vào học và thực hiện xây dựng một model cho xe tự lái trên giả lập của UDACITY cung cấp, sau đó tìm hiểu về mạng tích chập (CNNs) để trích chọn đặc trưng của một bức ảnh mà không có sự can thiệp của con người trong quá trình học của máy, trong quá trình thực hiện đồ án này khó khăn của em gặp phải là những kiến thức từ những lần huấn luyện và cho ra một model tốt nhất có thể, vì bài báo của NVIDIA chỉ đưa ra mô hình CNN đề xuất chứ không đưa ra các hướng dẫn cụ thể để xây dựng một mô hình tốt dựa trên mô hình đề xuất, em đã thử trên dưới 20 mô hình và qua 30 lần train với Google Colab để thay đổi các thông số , hàm kích hoạt, lr, hàm tối ưu hóa... để chọn ra được mô hình tốt nhất, Ở trên Chương 4 em chỉ nêu ra 4 mô hình điển hình từ tồi nhất đến tốt nhất em đã thử nghiệm được , và tất nhiên đây không phải là mô hình tốt nhất , nó chỉ tốt nhất trong phạm vi em nghiên cứu được. Vì Google đã cho sử dụng miễn phí GPU cho nên em đã huấn luyện các mô hình của mình trên Google Colab và đưa ra được model hoàn thiện nhất trong phạm vi mình có thể làm được.

## Chương 6 Kết luận và hướng phát triển

Từ những hướng và mục tiêu là xây dựng một file model tốt để giả lập hành vi, em đã cố gắng nghiên cứu và thực nghiệm để cho ra được kết quả tốt nhất dựa trên hành vi của người dùng.

Tuy nhiên em cũng nhận biết rằng do kiến thức của em còn hạn chế, cùng với đó là những kết quả còn chưa thực sự tốt. Đây cũng chỉ là một module nhỏ trong bài toán xe tự lái rất lớn, còn rất nhiều module riêng khác và khó hơn, Em đã cố hoàn thành đề án này với những kiến thức em thu lượm được trong quá trình học và thực hành để xây dựng lên một model tốt nhất trong những dữ liệu từ trình mô phỏng. Trong lúc quá trình chạy mô phỏng thì thỉnh thoảng xe vẫn hay tự rẽ sang trái và sang phải bất ngờ và điều đó chứng minh model này và các cách xử lý của em vẫn còn sai sót và đây cũng là hạn chế của em. Những mô hình mạng CNN em đã tự nghiên cứu và thử nghiệm chọn các thông số cần thiết hy vọng một phần nào đó góp phần nhỏ trong một bài toán lớn trong công nghệ xe tự lái thật trong cuộc sống.

# Tài liệu tham khảo

- [1] N. N. Vĩnh, "viblo," 28 10 2017. [Online]. Available: <https://viblo.asia/p/vanishing-exploding-gradients-problems-in-deep-neural-networks-part-2-ORNZqPEeK0n>. [Accessed 5 2019].
- [2] M. Tuấn, "dantri," 24 12 2015. [Online]. Available: <https://dantri.com.vn/o-to-xe-may/7-loi-ich-cua-xe-tu-lai-20151223162421801.htm>. [Accessed 5 2019].
- [3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, "End to End Learning for Self-Driving Cars," 25 6 2016. [Online]. Available: <https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf>.
- [4] D. M. Hai, "Hai's Blog," 23 4 2018. [Online]. Available: <https://dominhhai.github.io/vi/2018/04/nn-intro/>.
- [5] Andrew Ng, Kian Katanforoosh, Younes Bensouda Mourri, "Deep Learning Specialization," deeplearning.ai, [Online]. Available: <https://www.coursera.org/learn/convolutional-neural-networks>. [Accessed 16 5 2019].
- [6] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," [Online]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>. [Accessed 5 2019].
- [7] S. Ruder, "An overview of gradient descent optimization algorithms," [Online]. Available: <http://ruder.io/optimizing-gradient-descent/index.html>. [Accessed 5 2019].

- [8] D. H. Tiep, "Machine learning cơ bản," 24 2 2017. [Online]. Available: <https://machinelearningcoban.com/2017/02/24/mlp/#-activation-functions>. [Accessed 5 2019].
- [9] N. Quang, "autodaily," 22 2 2017. [Online]. Available: <https://autodaily.vn/2017/02/xe-khong-nguoi-lai-tu-phim-anh-den-doi-thuc>. [Accessed 5 2019].

