ARPIT CHAUHAN
MSC DATASCIENCE &AI

SELF-DRIVING CAR

Exploring Autonomous Driving through Udacity's Unity-Based Simulation Environment

AIM :

- The aim of this presentation is to explore and demonstrate the capabilities of the Unity-based Self-Driving Car Simulation Software provided by Udacity.
- Specifically, the presentation will highlight how this simulation environment serves as a critical tool for understanding and developing autonomous driving technologies.
- It will showcase the practical applications of the software in testing algorithms, simulating real-world driving scenarios, and enhancing the learning experience for those pursuing expertise in the field of autonomous vehicles.

DATA COLLECTION

- Simulating self driving car on a video game .Initially user runs through the road using his/her mouse and collects two kinds of data :
- IMAGE(shot from camera mounted on car)
- Steering Angle

DATA COLLECTION DETAILS

- **Input Data:** The car's camera takes a picture of the road every second.
- **Labeled Data:** The steering angle, which is a number between -10 and +10, is saved in a CSV file every second.

Steering Angle = 0: The car is going straight.

Negative Steering Angle: The car is turning left. A larger negative number means a sharper turn.

Positive Steering Angle: The car is turning right. A larger positive number means a sharper turn.

LEFT :

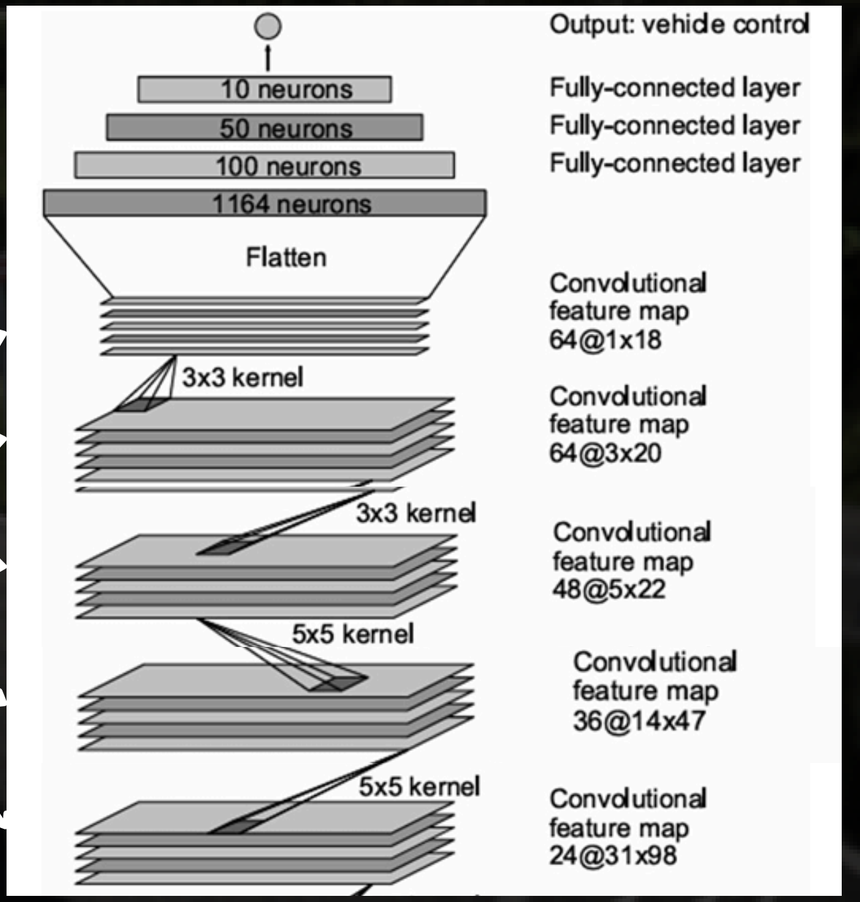
STRAIGHT :

RIGHT :

```
def nvidia_model():
    model = Sequential()
```

THE **SEQUENTIAL** MODEL IN KERAS (A HIGH-LEVEL NEURAL NETWORKS API, PART OF TENSORFLOW) IS A LINEAR STACK OF LAYERS. YOU USE THE SEQUENTIAL CLASS TO DEFINE A MODEL WHERE YOU ADD LAYERS ONE AFTER THE OTHER IN A SEQUENCE, WHICH MEANS THE DATA FLOWS THROUGH EACH LAYER IN ORDER.

```
def nvidia_model():
    model = Sequential()
    model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200, 3)))
    model.add(Conv2D(36, (5, 5), strides=(2, 2), activation='elu'))
    model.add(Conv2D(48, (5, 5), strides=(2, 2), activation='elu'))
    #model.add(Conv2D(64, (3, 3), activation='elu'))
    #model.add(Conv2D(64, (3, 3), activation='elu'))
```

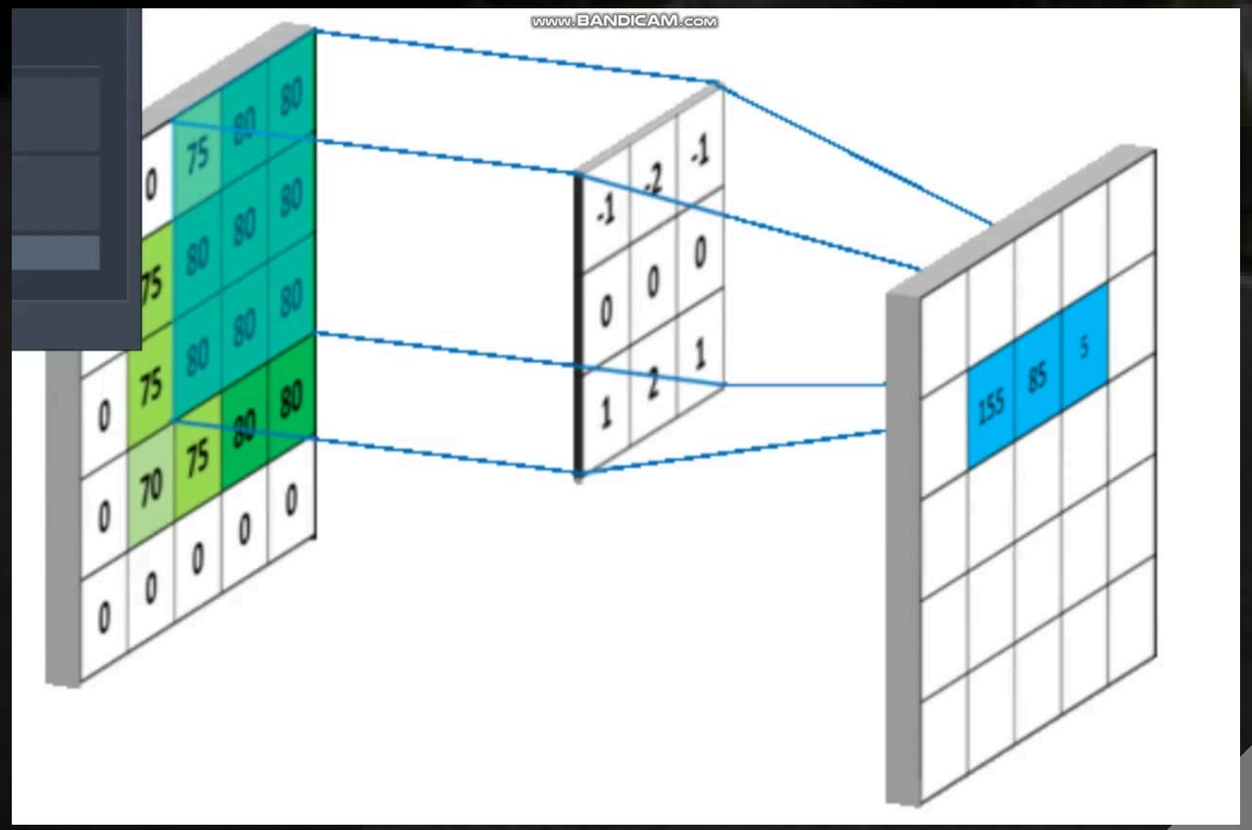


```
model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
```

- **24: NUMBER OF FILTERS (FEATURE DETECTORS)**

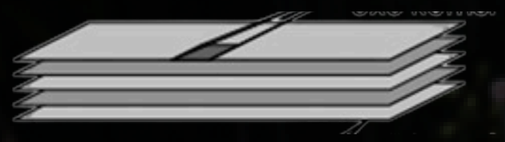
FILTERS :

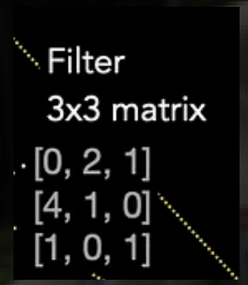
A FILTER IN A CONVOLUTIONAL NEURAL NETWORK (CNN) IS A SMALL MATRIX THAT SLIDES OVER AN INPUT (LIKE AN IMAGE) TO DETECT SPECIFIC FEATURES, SUCH AS EDGES OR TEXTURES



```
model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
```

- **(5, 5): SIZE OF EACH FILTER (KERNEL)**

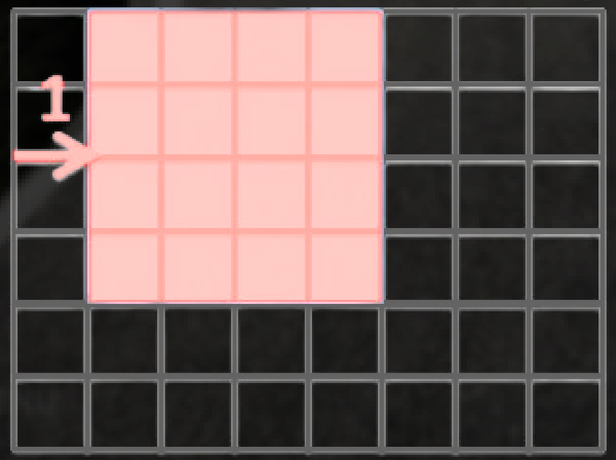




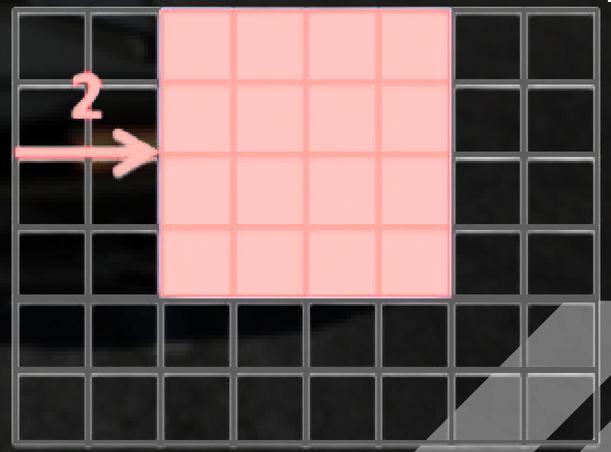

```
model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
```

- **STRIDES=(2, 2): THE STRIDE OF THE CONVOLUTION (HOW MUCH THE FILTER MOVES)**

If stride = 1, the filter will move one pixel



If stride = 2, the filter will move Two pixel



```
model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
```

- **INPUT_SHAPE=(66, 200, 3):**

  SHAPE OF THE INPUT IMAGE (HEIGHT, WIDTH, CHANNELS). THIS LAYER EXPECTS IMAGES OF **HEIGHT** 66 IN PIXEL, **WIDTH** 200 IN PIXEL , AND **3 COLOR CHANNELS (RGB)**


```
model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
```


- **ACTIVATION='ELU':**

  EXPONENTIAL LINEAR UNIT (ELU) ACTIVATION FUNCTION

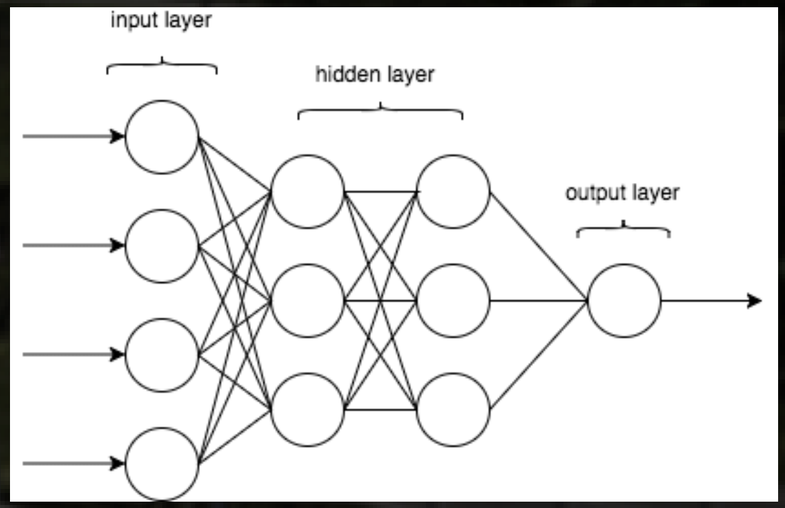
```
model.add(Flatten())
```

Intuition behind **flattening layer** is to converts data into 1-dimentional array for feeding next layer
This is typically done before feeding the output of the convolutional layers to the fully connected (dense) layers in the network.

```
model.add(Dense(100, activation='elu'))
model.add(Dense(50, activation='elu'))
model.add(Dense(10, activation='elu'))
model.add(Dense(1))
```

DENSE LAYER

In any neural network, a dense layer is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to every neuron of its preceding layer.



Feature Combination:

After the convolutional layers extract features from the input images, the dense layers take these features (flattened into a 1D vector by the Flatten layer) and combine them to form higher-level representations. This process helps the model learn complex patterns and relationships between the features.

Non-linear Transformations:

Each dense layer applies a non-linear activation function (in this case, ELU) to the input, enabling the model to learn non-linear relationships. This is crucial for the model's ability to generalize and make accurate predictions.

Decision Making:

The dense layers, especially the final one with a single neuron, are responsible for making the final decision or prediction. In your model, the output is a single value, which is typical for regression tasks like steering angle prediction in autonomous driving.

SPECIFIC ROLES IN YOUR MODEL

```
model.add(Dense(100, activation='elu'))
```

DENSE(100, ACTIVATION='ELU'):
Takes the flattened feature vector and processes it with 100 neurons to learn complex patterns.

```
model.add(Dense(50, activation='elu'))
```

DENSE(50, ACTIVATION='ELU'):
Further refines the learned patterns with 50 neurons.

```
model.add(Dense(10, activation='elu'))
```

DENSE(10, ACTIVATION='ELU'):
Narrows down the representation to 10 key features that are most relevant for the final decision.

```
model.add(Dense(1))
```

DENSE(1):
Outputs a single value, which could represent a regression target such as the predicted steering angle.

MODEL SUMMARY :

- Developed a self-driving car model using a Convolutional Neural Network (CNN), inspired by the NVIDIA architecture.
- Data collection involved capturing images from the car's center, left, and right cameras, paired with corresponding steering angles.
- Applied data augmentation techniques including zooming, shifting, brightness adjustment, and horizontal flipping to enhance model generalization.
- Preprocessing steps included cropping, resizing, and normalizing images to maintain consistency.
- CNN architecture featured multiple convolutional layers for feature extraction, followed by fully connected layers to map features to steering angles.
- Trained the model using the Adam optimizer, aiming to minimize mean squared error between predicted and actual steering angles.
- Conducted training over multiple epochs with batch processing to enable effective learning and accurate steering predictions.

MODEL SUMMARY

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 31, 98, 24)	1,824

BREAKDOWN OF (NONE, 31, 98, 24)

NONE:

This represents the batch size, which is not fixed in the model architecture.
It allows flexibility for the model to handle different batch sizes during training or inference.
The value None indicates that the batch size can vary depending on how many samples you feed into the model at once.

31,98 :

These are the spatial dimensions (height and width) of the output feature maps after the convolution operation has been applied. The original image size is reduced due to the filter size strides, and padding used in the Conv2D layer.

24 :

These are the Numbers of Filters

Layer 1: Conv2D (conv2d)

- **Input Shape**: (None, 66, 200, 3)
  - Height = 66
  - Width = 200
  - Channels = 3 (because it's an RGB image)
- **Filter Size**: 5x5
- **Number of Filters**: 24
- **Strides**: (2, 2)

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 31, 98, 24)	1,824
conv2d_1 (Conv2D)	(None, 14, 47, 36)	21,636
conv2d_2 (Conv2D)	(None, 5, 22, 48)	43,248
flatten (Flatten)	(None, 5280)	0
dense (Dense)	(None, 100)	528,100
dense_1 (Dense)	(None, 50)	5,050
dense_2 (Dense)	(None, 10)	510
dense_3 (Dense)	(None, 1)	11

Number of Parameters:

$$\text{Number of Parameters} = (5 \times 5 \times 3) \times 24 + 24 = 1,824$$

- The first part of the equation $(5 \times 5 \times 3)$ calculates the number of weights per filter.
- Then, you multiply by 24 (the number of filters) to get the total number of weights.
- Finally, add 24 for the bias terms (one for each filter).

flatten (Flatten)	(None, 5280)	0
dense (Dense)	(None, 100)	528,100
dense_1 (Dense)	(None, 50)	5,050
dense_2 (Dense)	(None, 10)	510
dense_3 (Dense)	(None, 1)	11

Layer 4: Flatten (flatten)

- **Input Shape**: (None, 5, 22, 48)

The **Flatten** layer is used to convert the multi-dimensional output from the previous Conv2D layer into a single vector that can be fed into a Dense (fully connected) layer.

Flattening Process:

- The output shape from the previous layer is (None, 5, 22, 48).
  - "5" is the height of the feature map.
  - "22" is the width of the feature map.
  - "48" is the number of channels or filters.
- Flattening combines all these into a single dimension. So:

$$5 \times 22 \times 48 = 5280$$

Output Shape:

- The output shape after flattening will be (None, 5280), where 5280 is the total number of elements per sample.

Since the **Flatten** layer just rearranges the data without learning any weights, **it has 0 parameters**.

Layer 5: Dense (dense)

- **Input Shape**: (None, 5280)
- **Number of Neurons**: 100

The **Dense** layer is a fully connected layer, meaning each input node is connected to every neuron in the layer.

Number of Parameters: The number of parameters in a Dense layer is given by:

$$\text{Number of Parameters} = (\text{Number of Input Units} \times \text{Number of Neurons}) + \text{Number c}$$

- **Number of Input Units**: 5280 (from the Flatten layer).
- **Number of Neurons**: 100.

Plugging in the values:

$$\text{Number of Parameters} = (5280 \times 100) + 100 = 528,000 + 100 = 528,100$$

So, this layer has **528,100 parameters**.

Output Shape:

- The output shape is (None, 100) because there are 100 neurons in this layer.

THANK YOU