Behavioral Cloning

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the Udacity simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- **model.py** containing the script to create and train the model (utilizes Generators)
- **drive.py** for driving the car in autonomous mode
- **model_nvidia.h5** containing a trained convolution neural network
- **visualize.py** script to visualize the results from the different stages of the network
- **writeup.md or writeup_report.pdf** summarizing the results

2. Submission includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model_nvidia.h5
```

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Data Collection and Augmentation

There were several approaches discussed in the course to help with data augmentation, which in turn will help with generalization. Some of which are:

- Use all three camera images
- Flip the image (since this track is biased towards left turns)

For my experiments, I relied on these recommendations with small changes. I did use all three camera angles leveraging ths suggested correction factor of .3.

I did not flip the images, rather, I tried to collect image data (by driving)under the following scenarios:

- [x] Drive forward focusing on the center (ended up using the data that was provided as a part of the project)
- [x] Turn the car around and drive several rounds in the opposite direction (was very beneficial)
- [ ] Sway from side to side without going off the road
  - The hope here was that the algorithm would recognize the corrective action to take when veering off course.
  - This did not have the intended effect, cause jittery actions.
- [ ] Take the vehicle off course and show how to come back to the center of the road.
  - Did not help
- [ ] Focus on curves (smooth turns)
  - Little to no impact
- [ ] High and Low speed driving
  - Little to no impact
- [ ] Hug left
  - NO impact

Data Import

model.py The starting point to the application is a list of image data files:

```
imgdatafiles = ['..\\..\\data\\driving_log.csv',
                '..\\..\\Track1\\New\\Reverse2\\driving_log.csv',
                ]
```

The python csv model is used to read these data files. Data import is deffered to help optimize memory usage. Rather a mapped list of image locations with measurements is curated as the data set. This dataset is shuffled as they are initially in the order in which they were recorded. (model.py line 62 and 83) The dataset is then split (80-20) intro training and validation. (model.py line 66)

`create_generator` : This function creates a python generator for a given data set and batch size. This is used in the code to generate the training and validation generators.

Model Architecture and Training Strategy

1. The Model

A Keras Sequential Model lends itself beautifully to this use case. The goal here (as seen below) is to replicate the NVIDIA CNN architecture, which is just a stack of layers with exactly one input and one output tensor at each layer.

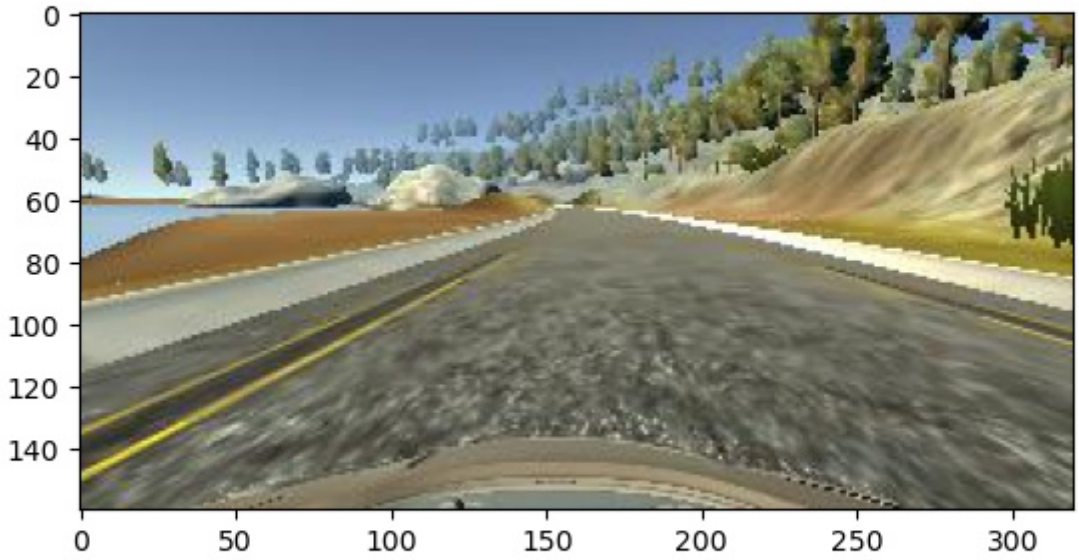
1. Pre-Processing

The first two layer of the model are pre-processing layers.

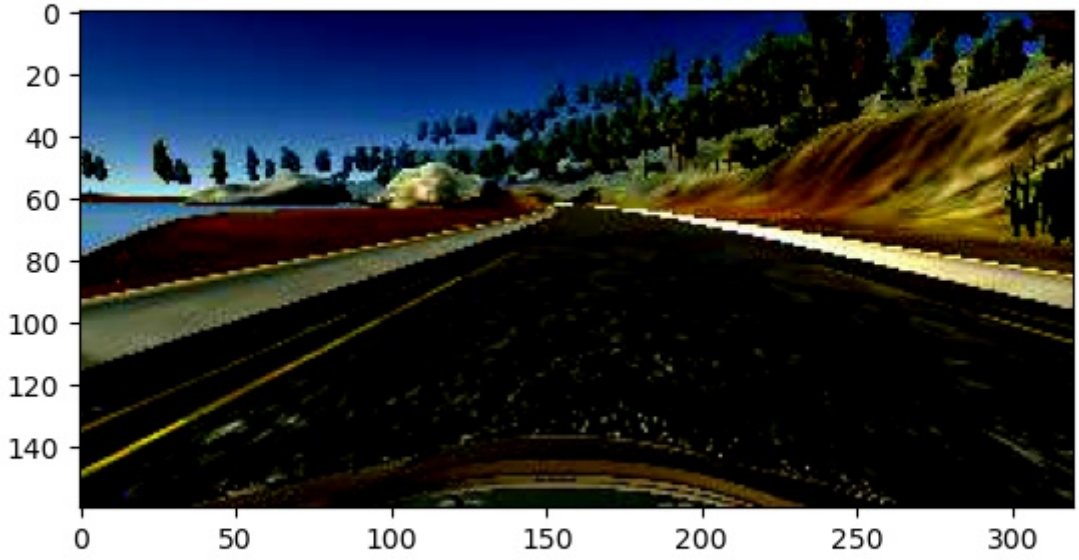
- Layer 1: Normalization of the image, centered around zero
- Layer 2: Crop the image to focus on the lane to be navigated. Removed 70 rows from the top (landscape) and 20 from the bottom (hood)

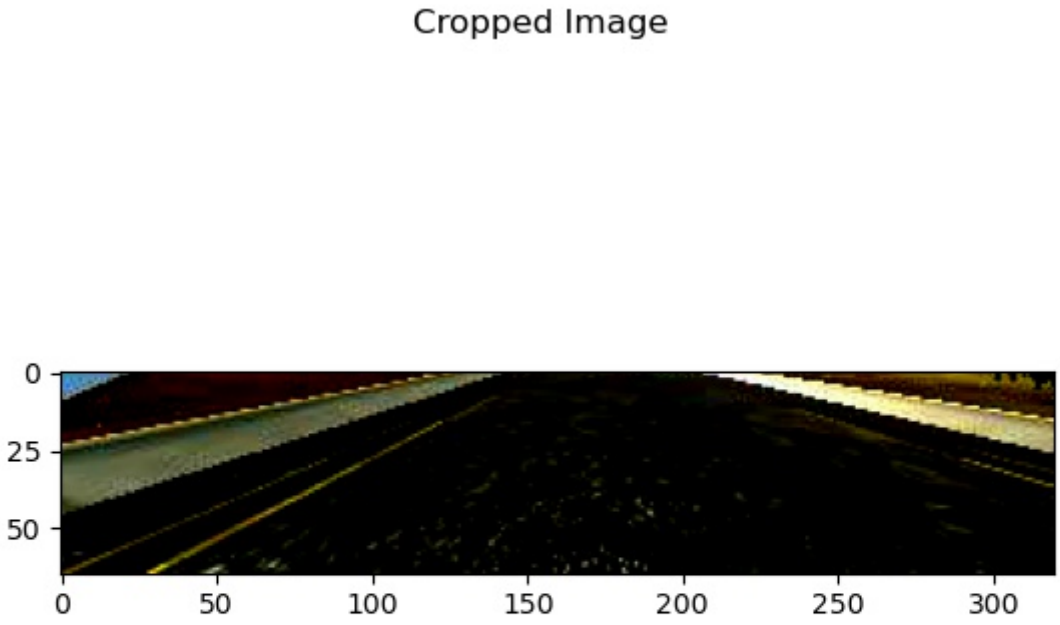
Here are the outputs from these layers for an example image:

Original Image



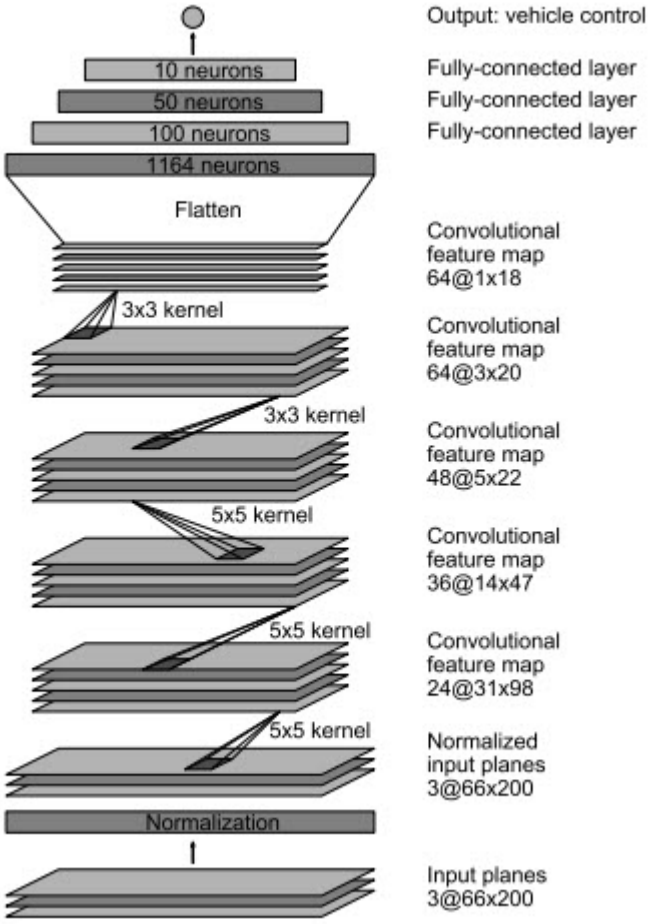
Normalized Image





2. NVIDIA CNN Architecture

I have adopted the NVIDIA CNN, that has the architecture shown below.



Keras model summary:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 160, 320, 3)       0
_________________________________________________________________
cropping2d_1 (Cropping2D)    (None, 65, 320, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 31, 158, 24)       1824
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 77, 36)        21636
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 37, 48)         43248
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 35, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 33, 64)         36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 2112)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               211300
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 348,219
Trainable params: 348,219
Non-trainable params: 0
```

The architecture has 5 hidden convolution layers , followed by 3 fully connected dense layers.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py line 99).

The dropout layer is configured to drop 20% of the samples.

3. Model parameter tuning

- The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 113).
- The loss function was MSE (Mean Squared Error) (model.py line 113).
- Batch size per step was 32 (memory requirements were too high when this was increased)
- Number of epochs 3 (tried early stopping but this was more straightforward approach for this case)

Model Architecture and Training Strategy

1. Solution Design Approach

The initial focus was on creating the generators and defining appropriate preprocessing steps.

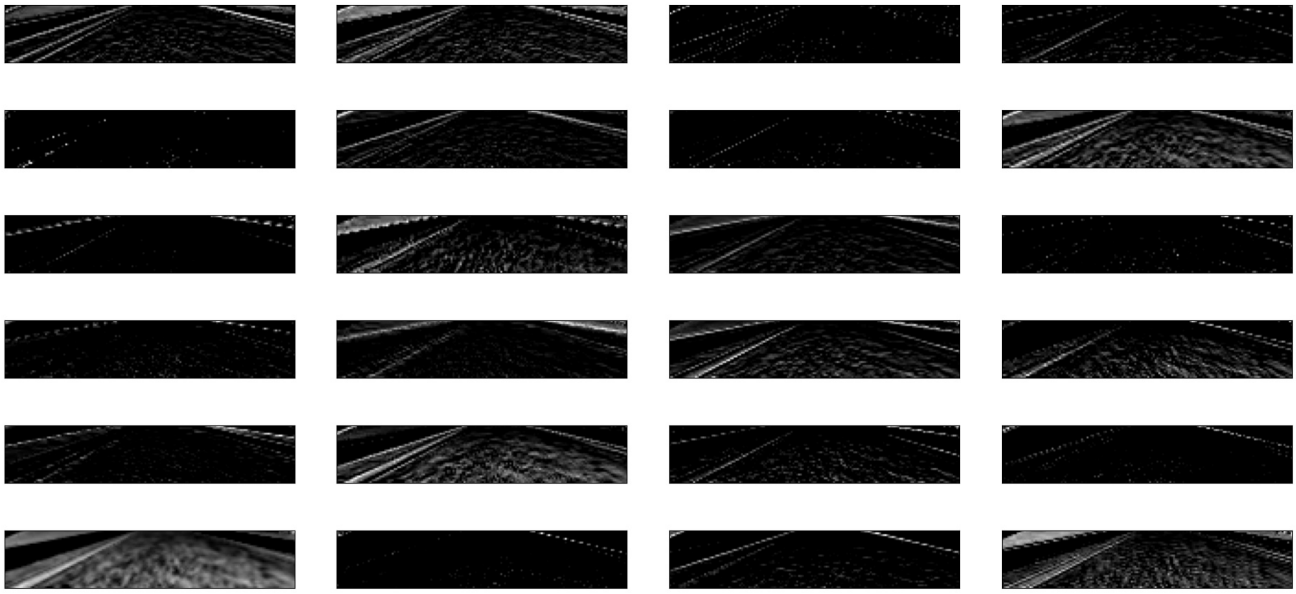
I tried several other architectures that yielded suboptimal results:

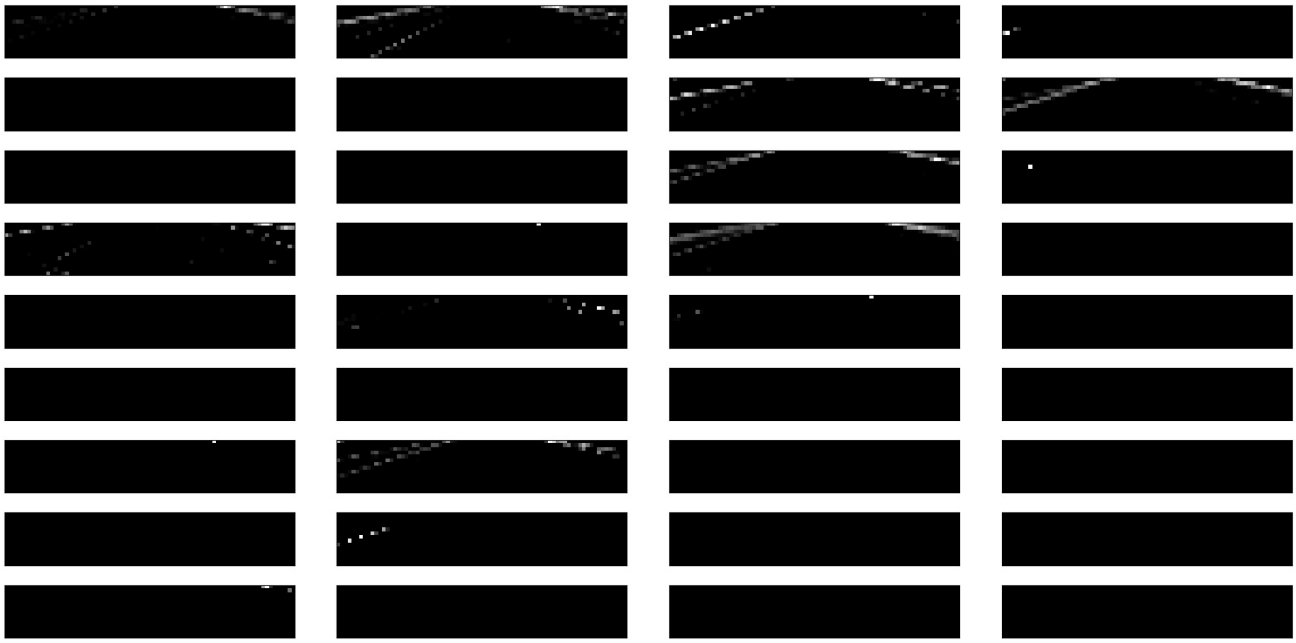
- Basic flat layer
- LeNet
- NVIDIA CNN with Max Pooling

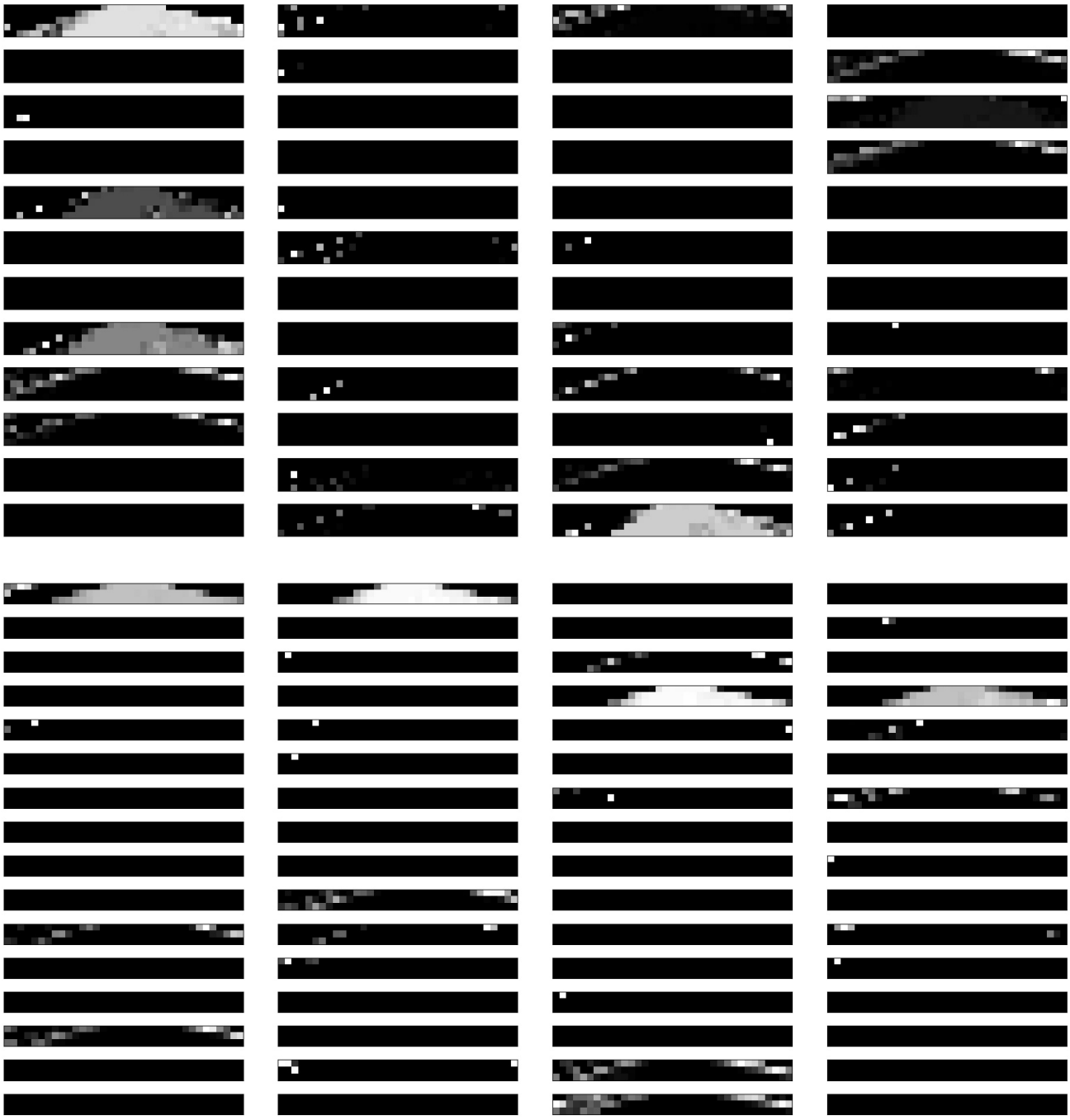
```
def addBasic(model):
    # Basic Network
    model.add(Flatten(input_shape=(160,320,3)))
    model.add(Dense(1))
    return model

def addLeNet(model):
    # LeNet
    model.add(Convolution2D(6,5,5, activation ='relu'))
    model.add(MaxPooling2D())
    model.add(Convolution2D(6,5,5, activation ='relu'))
    model.add(MaxPooling2D())
    model.add(Flatten())
    model.add(Dense(120))
    model.add(Dense(84))
    model.add(Dense(1))
    model.summary()
    return model
```

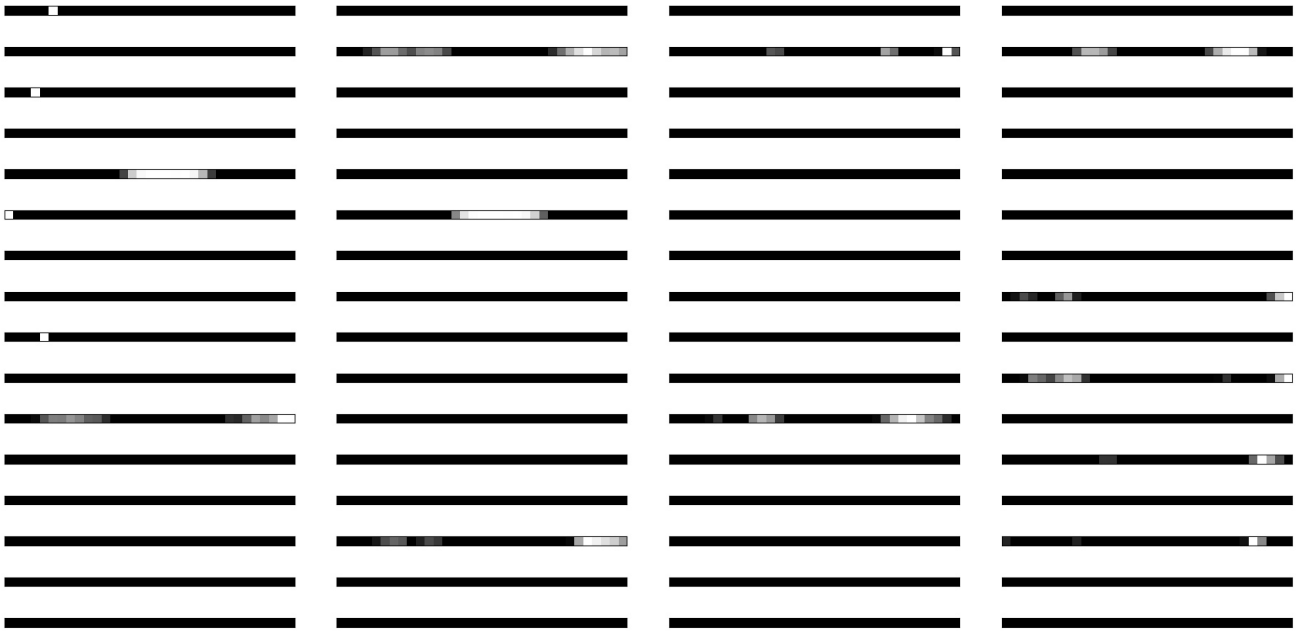
3. Visualizing Results

visualize.py was used to review the convolution results for an example image as shown:









4. Output Video

video

Issues Faced

- Often ran out of VRAM on my machine resuling in ***Error in CuDNN: CUDNN_STATUS_ALLOC_FAILED***