Capstone 2 Milestone Report:
Convolutional Neural Network - Crowd Counting

Contents

Preface

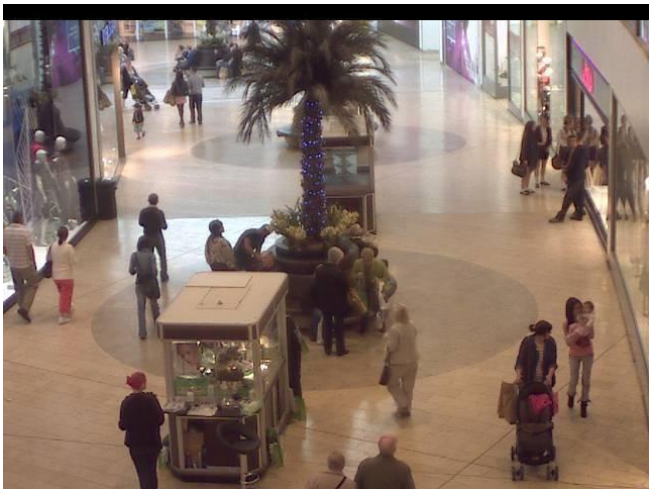
Convolutional networks have existed for over 30 years. First proposed in 1979 by Kunihiko Fukushima, a neocognitron system (hierarchical, multilayered artificial neural network) would become the basis for modern day convolutional neural networks.

Today, convolutional neural networks play an important role in many image classification problems from cancer detection to self driving cars.

Problem Statement

Given an image from a mall webcam, can a convolutional neural network count the number of people in the frame?

For example, can a CNN model count the 29 people in the frame below:



Use case for crowd counting

There are many use cases for crowd counting in the context of a mall images. To name a few:

- Counting crowds could act as a proxy for overall retail economic conditions.
- Crowd counting for specific retailers could provide the basis for long/short stock purchases[1]
- A mall could offer high traffic areas to specific retailers and charge more (ex. Apple)
- The feature extraction process for counting a crowd could be useful for other applications through transfer learning (ex. Crowd counting at a political rally, estimations of animal populations using a webcam in the forest etc.)
- Further advances in a crowd counting context could provide additional details on people in the image (demographics, time of day, etc.)

[1] https://www.theatlantic.com/magazine/archive/2019/05/stock-value-satellite-images-investing/586009/

Explanatory Data Analysis

Data Collection

The data was sourced from associate professor Chen Change Loy's personal website[2].

The process to extract the images involved processing a series of .mat files with matlab.

The output of the matlab processing was:

- 2,000 640x480 RGB frames from a mall webcam.
- an associated target variable which represents the number of people in each frame.

Data Validation

Prior to training, several frames were evaluated to confirm the target value was valid. Below are two example frames and their associated target values:

```
[41]
```


```
[19]
```

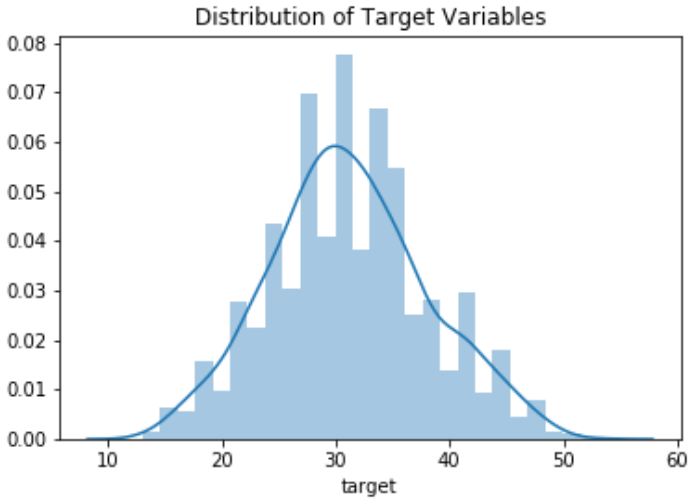
[2] http://personal.ie.cuhk.edu.hk/~ccloy/downloads_mall_dataset.html

Benchmarking

To get a general idea of model performance, a benchmark was created using the average target variable (where target variable is equal to the number of people in a given frame).

The mean value for the target variable is **31.16** for the 2,000 frames.

Plotting the target variable results in the following distribution:

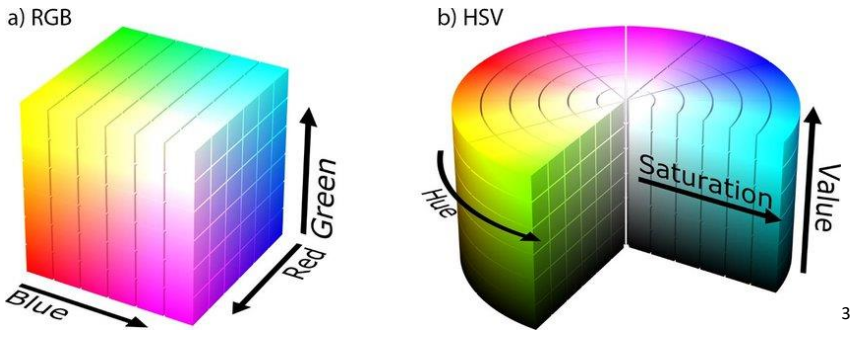


To create a benchmark MSE, the MSE was calculated assuming we choose the mean for every frame in the dataset.

Calculating the benchmark MSE in this manner results in a value of **48.21**. This value will serve as a starting comparison for our trained model.

Data Preprocessing – RGB to HSV conversion

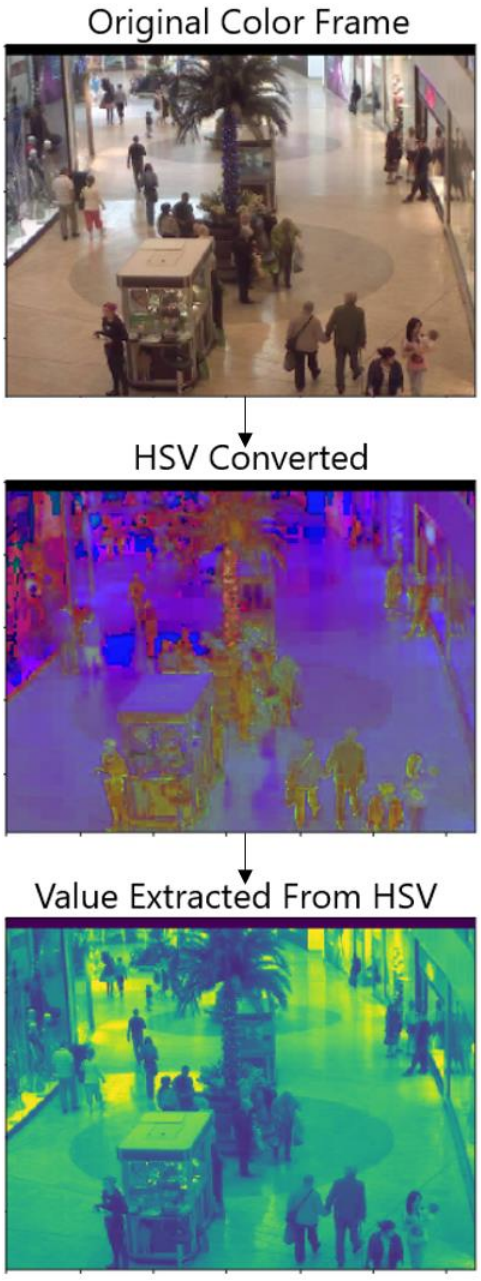
Prior to training the model, the RGB images were transformed to an HSV ("hue, saturation, value") format. From the HSV format, the "value" was extracted to form 2,000 samples with the tensor shape of 640x480x1.



[3] Images by Michael Horvath, available under Creative Commons Attribution-Share Alike 3.0 Unported license.

The change in format to HSV was done due to the model likely not requiring a 3-dimensional tensor to filter out shapes form the image. If for example, we were interested in counting people who wore red shirts, then the use of RGB channels would be necessary.

Converting to HSV simplifies the data by encoding information from 3 dimensions to 1 dimension. Through the conversion, there is a non-zero loss in the overall information that each frame provides. However, the benefits of the conversion are a simplified tensor which reduces overall model complexity and improves train time.



Training Part 1 – Initial Training

Preface - Training

- Training was completed on a Nvidia GeForce GTX 1660 TI graphics card.
- All pixel tensor inputs were normalized to values between 0 and 1. This was done to avoid gradient exploding and to improve overall train performance.
- Training was done through a Keras generator to reduce the overall memory overhead of running the models.

Baseline Training

To complete the training, a Jupyter Notebook was created that referenced different models for testing. Abstracting the models away allowed for further small adjustments to models that performed well.

During the initial training, the following models were tested:

Model	Model Name	# of layers	DO	BN	Activation	Parameters	Val MSE
1	Basic CNN	1			Relu	130,515,833	15.77
2	Basic CNN v2	2			Relu	62,891,665	11.35
3	Basic CNN v2 DO	2	X		Relu	62,891,665	44.41
4	Basic CNN v2 BN	2		X	Relu	62,893,073	11.01
5	Basic CNN v2 DO LR	2	X		LeakyRelu	62,891,665	10.56
6	Basic CNN v2 BN LR	2		X	LeakyRelu	62,893,073	10.66
7	AlexNet Transfer Learning[4]	7	X	X	Relu	13,011,313	244.97
8	AlexNet Transfer Learning LR	7	X	X	LeakyRelu	13,014,257	19.53

DO = Dropout | BN = Batch Normalization

Best Performance

The best performing model was surprisingly model 5 (Basic CNN v2 DO LR) which used dropout and LeakyRelu activations.

We also see model improvements between model 2 and model 5&6 by adding a LeakyRelu activation.

Worst Performance

Interestingly, model 7 (adjusted AlexNet) model had the worst performance. Adding LeakyRelu as an activation improved the performance of model 7 ten fold. This would suggest an issue with a vanishing gradient.

A note on models 7 & 8 – AlexNet variations

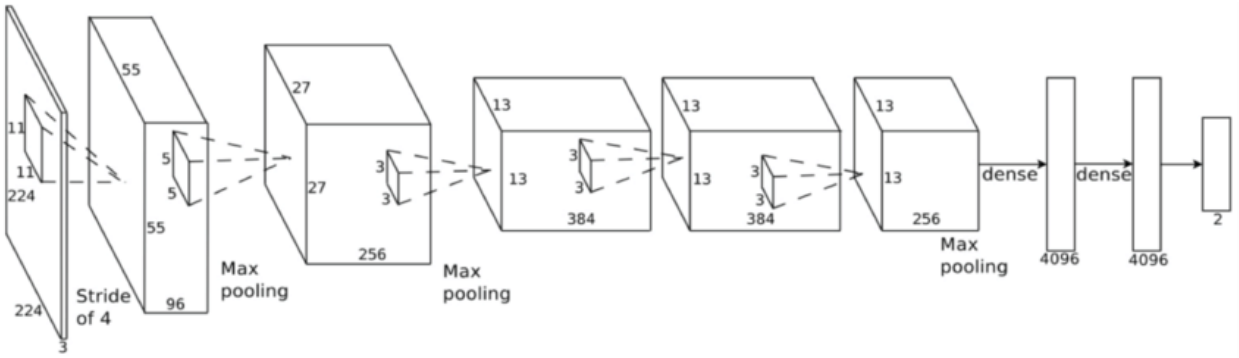
The modification to the AlexNet architecture was removing a dense layer with 4096 perceptrons after the convolutional filters were flattened. This was done due to the computer not having enough memory to apply the tensor to the model. Removing the dense layer drastically reduces the number of trained parameters significantly and creates a simplified AlexNet model. In future training, the dense layer will

[4] Transfer learning model – Variations made to the Alexnet architecture: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

be added back in an attempt to encapsulate more complexity from the features since the model did not perform well.

Training Part 2 – Improving AlexNet Transfer Learning

AlexNet Architecture Overview[5]



Current Issues with AlexNet Transfer Learning

Using the AlexNet architecture above comes with a couple of problems:

1. Firstly, the issue of vanishing gradient. Based on results from the initial training, adding a LeakyRelu activation to the layers will likely circumvent this issue.

2. Secondly, with input tensor of shape 640x480, after the convolutional layers have been applied, the flattening layer causes a memory overflow error. In order to circumvent this issue, one of three things could be done.
    a. Reduce the tensor size on convolution layers through increasing the kernel/strides.
    b. Reduce the tensor size on the pooling by increasing kernel/strides.
    c. Reduce the dense layer number of perceptrons after the tensor is flattened.

    To avoid the overflow error, **option c** was selected. This was done due to keep the overall filters outputs the same. However, this does reduce the models ability to pick up on non-linear relationships between output layer perceptrons.

    Option c was selected because of the models simplicity in that it is merely counting objects rather than combining a number of features to classify an image (as per the objective of the original paper). Put another way, people in the upper left corner of the frame have no relation to those in the lower right. As such, there is no need for such complexity or connectiveness between perceptrons that are activated in those corners.

[5] https://www.researchgate.net/figure/AlexNet-CNN-architecture-layers_fig1_318168077

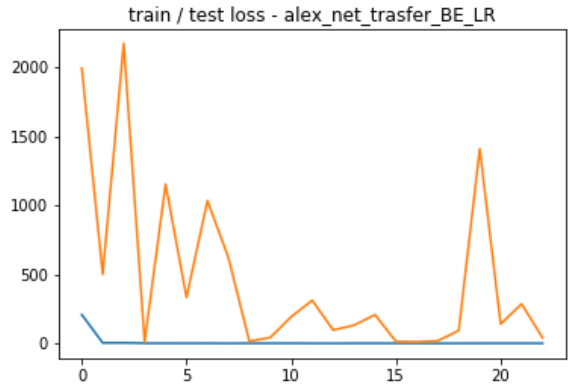
AlexNet Training

After making the adjustments described above, the following AlexNet models were trained:

Model	Model Name	# of layers	DO	BN	Activation	Parameters	Val MSE
9	AlexNet_trasfer actual	8	X	X	Relu	110,887,769	23.08
10	AlexNet transfer DO	8	X		Relu	110,882,265	28.80
11	AlexNet transfer BE	8		X	Relu	110,887,769	15.69
12	AlexNet transfer BE LR	8		X	LeakyRelu	110,887,769	12.86

DO = Dropout | BN = Batch Normalization

Model 12 had the best performance overall (12.85 MSE), however, the validation MSE varied substantially between epochs. This would suggest overfitting of the model. Adjustments to the dataset or the model are required to improve model performance.



Model 11 had the best out of the AlexNet variations in terms of it's ability to generalize to new data.

Next Steps

The AlexNet variations are more complex models, yet, they perform worse than the basic 2-layer networks.

This could suggest a number of issues such as:

- Overfitting
- Too small of sample
- Too much complexity in the network (too many layers)

To improve the AlexNet models, the following will be tested:

1. Increase sample size through image generation
2. Reduce the overall complexity of the models
3. Preprocessing on the original dataset to improve the models ability to pick up on changes in the image.