Capstone 2: Convolutional Neural Network - Crowd Counting

Contents

Preface

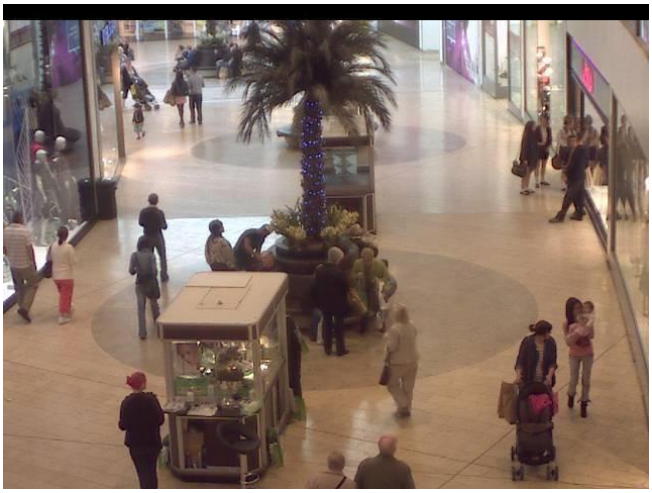
Convolutional networks have existed for over 30 years. First proposed in 1979 by Kunihiko Fukushima, a neocognitron system (hierarchical, multilayered artificial neural network) would become the basis for modern day convolutional neural networks.

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]
```

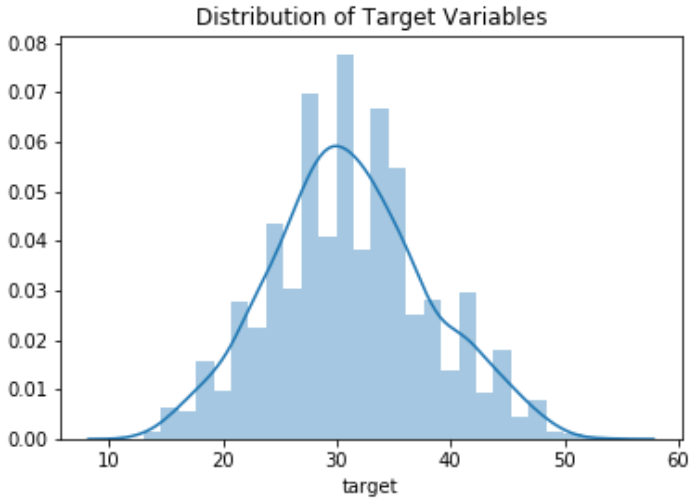

```
[19]
```

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

Benchmarking

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

Plotting the target variable results in the following distribution:



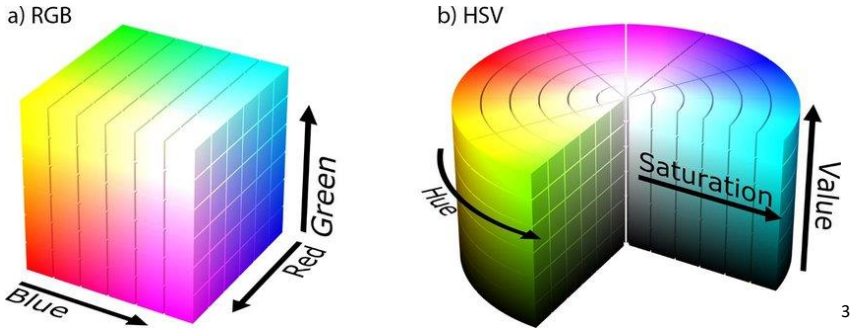
To get a general idea of model performance, a benchmark was created using the average target variable (where target variable is equal to the number of people in a given frame).

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.

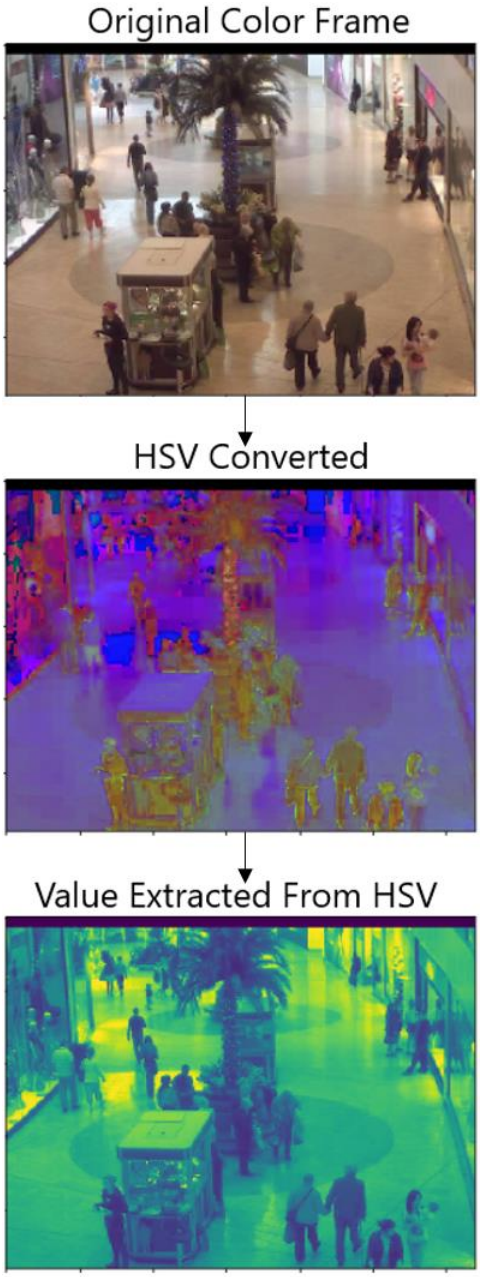


[3] Images by Michael Horvath, available under Creative Commons Attribution-Share Alike 3.0 Unported license.

The change in format to HSV was done due to the marginal model performance increase between a 3-dimensional RGB tensor and a 1-dimentional HSV “value” tensor.

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.

6 simple convolutional networks were tested. In addition, 2 variations AlexNet[4] were trained as 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

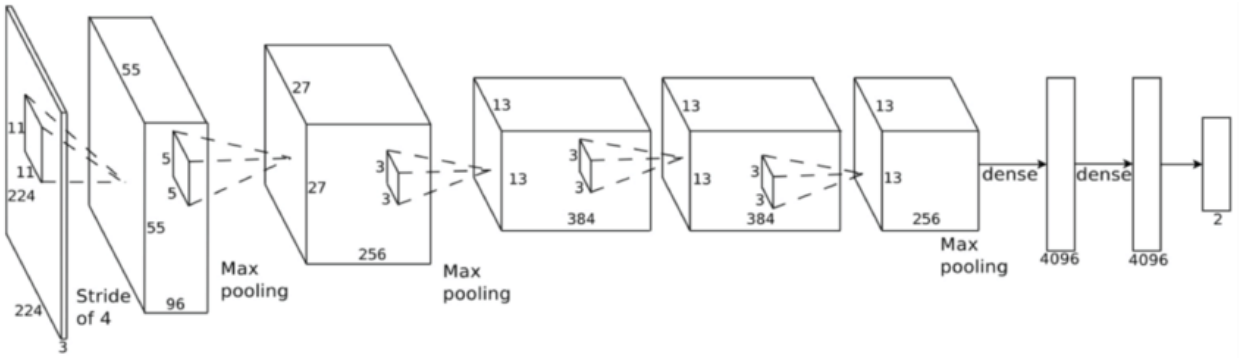
The modification to the AlexNet architecture was removing a dense layer with 4096 perceptrons after the convolutional filters were flattened. This was done due to the computer not having enough memory to apply the tensor to the model. Removing the dense layer drastically reduces the number of trained 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

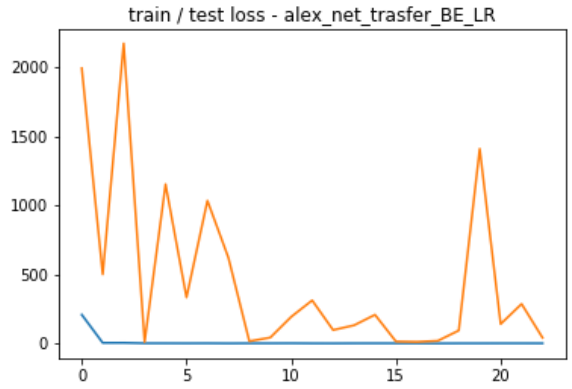
AlexNet Training

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 performed the best in terms of it's ability to generalize to new data.

Training Part 3 – Augmenting the training sample

Keras Image Generator

In order to boost the number of samples in the dataset, the Keras ImageDataGenerator class was used to duplicate the original dataset with minor changes.

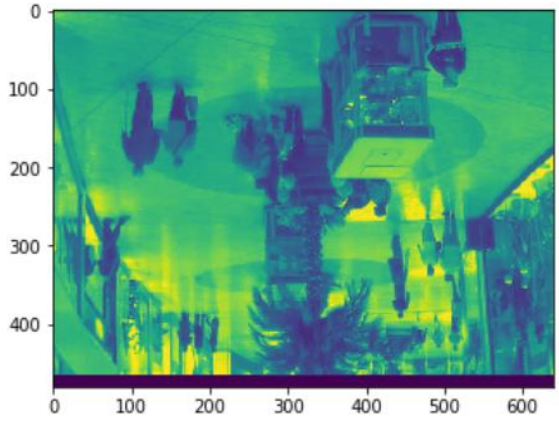
The augmented sample is the 2,000 original frames but flipped randomly on the X and/or Y axis. No transformations were made to the pixels of the images themselves (ex. skew, rotation, slice).

Augmenting the sample size in such a manner will hopefully help with overfitting as well as improve the models ability to generalize.

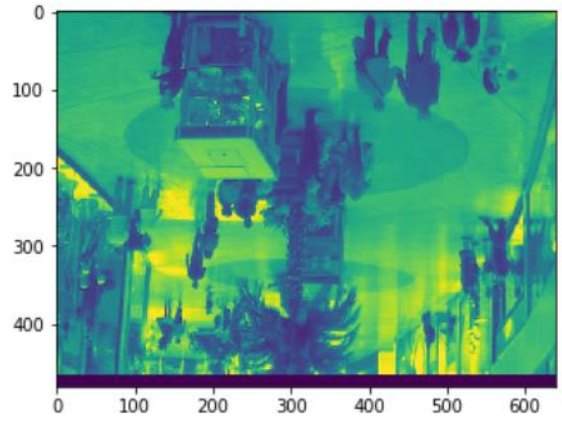
Keras Image Generator Examples

The following samples were generated by the Keras ImageDataGenerator class:

26



35



Training the Augmented Sample

Using the augmented sample, the following 5 models were trained:

Model	Model Name
5	Basic CNN v2 DO LR
6	Basic CNN v2 BN LR
7	AlexNet Transfer Learning[6]
8	AlexNet Transfer Learning LR
11	AlexNet transfer BE
12	AlexNet transfer BE LR

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

The results were not as expected. The models with more parameters were unable to better capture the complexity in the flipped images. This would suggest that there is room for improvement in terms of feature extraction in the filter layers.

Training Part 3 – Augmented Sample

Model	Model Name	# of layers	DO	BN	Activation	Parameters	Original	Aug[7]
5	Basic CNN v2 DO LR	2	X		LeakyRelu	62,891,665	10.56	14.00
6	Basic CNN v2 BN LR	2		X	LeakyRelu	62,893,073	10.66	27.24
7	AlexNet Transfer Learning[6]	7	X	X	Relu	13,011,313	244.97	328.51
8	AlexNet Transfer Learning LR	7	X	X	LeakyRelu	13,014,257	19.53	20.40
11	AlexNet transfer BE	8		X	Relu	110,887,769	15.69	24.46
12	AlexNet transfer BE LR	8		X	LeakyRelu	110,887,769	12.86	42.72

Training Part 4 – Mean Normalization

Adjustments to the sample dataset

In an attempt to make the objects in the frames more sensitive in the neural network, the mean image from the dataset was subtracted from each frame.

Subtracting the mean image from a frame causes the static background objects to be removed. However, in doing so the model becomes ungeneralizable. For example, the model would not work on other webcams in the same mall without applying the same mean normalization operation to a different sample of images.

In short, the model would remain the same however, a new dataset for each webcam would be required.

Math behind the adjustments

The normalization process was performed by taking the absolute value of the difference between each pixel value in each sample with the mean of each pixel in the dataset.

Expressed mathematically, the following formula below was applied:

$$X_{n,i,j} = \left| \left[x_{n,i,j} - \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{I} \sum_{j=1}^{J} v_{n,i,j} \right] \right|$$

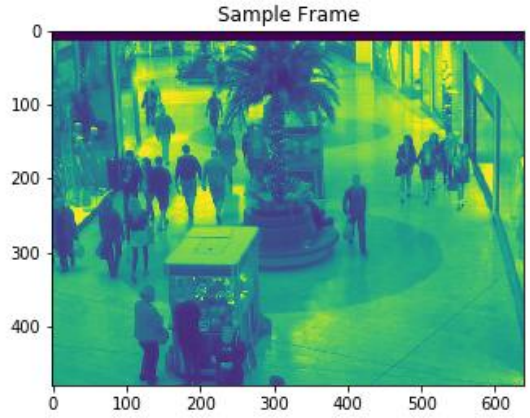
Where,
v = an individual sample
n = number of samples
i = pixel width (number of pixels in the x axis)
j = pixel height (number of pixels in the y axis)

The goal is to capture the change in value between the mean and samples.

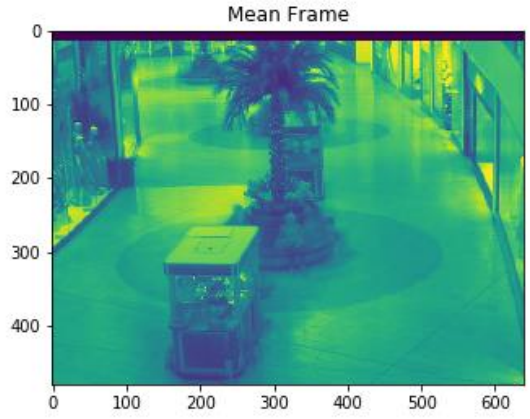
[7] Trained on augmented sample

Example of transformation in frames

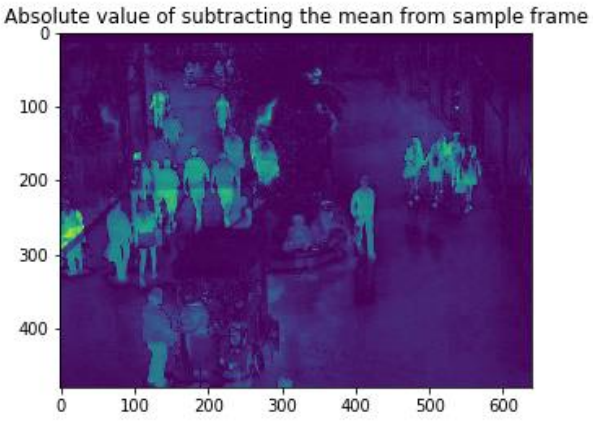
Below is an example of a transformation to a frame:



Taking the mean of all the images results in the following image:



Taking the absolute value of the difference between the sample frame and the mean results in the following image:



Issues with the normalization process

The main benefit to this normalization process is to accentuate any changes in value in each frame. However, in doing so, there are a couple of drawbacks:

- Instances where an individual in the frame has a similar value to the mean image. That is, the clothing the person is wearing might make it difficult for the model to pick up on.

- Removes **any** possibility of generalizing the model to other applications. For example, using the model on a different webcam in the mall would not be viable unless a sample of the mean image was provided for the new webcam.

Training on the mean normalized images

Using the same 5 models from part 3, the training was performed on the new set of

Training Part 4 – Mean Normalized Images

Model	Model Name	# of layers	DO	BN	Activation	Parameters	Original	Aug[8]	MN[9]
5	Basic CNN v2 DO LR	2	X		LeakyRelu	62,891,665	10.56	14.00	16.69
6	Basic CNN v2 BN LR	2		X	LeakyRelu	62,893,073	10.66	27.24	78.53
7	**AlexNet Transfer Learning[6]**	**7**	**X**	**X**	**Relu**	**13,011,313**	**244.97**	**328.51**	**28.36**
8	AlexNet Transfer Learning LR	7	X	X	LeakyRelu	13,014,257	19.53	20.40	34.48
11	AlexNet transfer BE	8		X	Relu	110,887,769	15.69	24.46	16.20
12	AlexNet transfer BE LR	8		X	LeakyRelu	110,887,769	12.86	42.72	14.57

Training on the mean normalized frames worsens performance for 5 out of 6 of the models. One hypothesis for the decrease in performance across the 5 models is that the models are already capable of picking up the small differences in the frames. Accentuating the major changes through the normalization process causes certain features to be lost in the background of the frame.

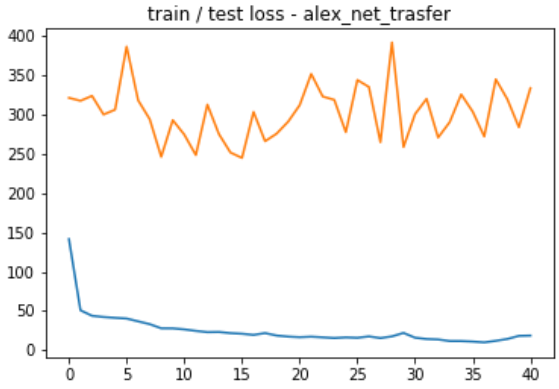
This idea is somewhat congruent with the major improvement in model 7. In normalizing the dataset, the Relu activation of model 7 is no longer extremely sensitive to minor changes in the frame. Thus, it's possible we no longer observe the huge validation mse from massive gradient spikes.

Comparison in model 7 training

Below are the two graphs between training model 7 on the standard dataset and the mean normalized dataset.

[8] Trained on augmented data
[9] Trained on mean normalized data

Trained on the standard dataset:



Trained on the mean normalized dataset:

It's apparent that through the normalizing process, model 7's improves it's ability to extract features from the frames. Although neither model performed particularly well, which could suggest that overly complex models are more sensitive to minor changes in the feature structure causing them to not converse.

Training Part 5 – Automating the model training process

Goals of automating the model training process

To this point, a simple model with two layers has been able to provide the best results in terms of validation MSE.

By making slight (automated) adjustments to this model, performance could possibly be improved by adding a small amount complexity to the model.

To do this, a class was created to automate the model building process. The class is abstracted away and allows for many different possible combinations to the basic model.

The automated model parameters will be generated by taking the cartesian product of a series of possible model parameters (filter size, kernel size, strides etc.)

Models trained

The training was performed by taking model 5 and adding two additional layers with various parameters. The cartesian product of **each** layer below was taken.

The cartesian product of both results was taken to generate 24 different combinations of layers.

First additional layer
filter_size = [128, 256]
kernel_size = [(11,11)]
stride_size = [(3,3), (2,2)]

4 different layers

Second additional layer
filter_size_2 = [128, 256]
kernel_size_2 = [(8,8), (4,4)]
stride_size_2 = [(2,2), (1,1)]

8 different layers

24 different models

The validation mse performance of each model was evaluated and the best model was written to a text file.

The best model performed at 15.65 MSE which is below the performance of the best model (10.56).

Efficiency of automating the model training and further improvements

One major drawback of automating the model training in this manner is that it is extremely inefficient in terms of generating a well generalized model. Generating models using the cartesian product is $O(n^n)$ in that each additional parameter added is exponentially adds to the overall training time.

However, one positive to generating models in this manner is it allows for minor changes to an existing models. In the above example, the model was not able to improve by adding additional layers. However, I am confident with further training, model performance could be improved using the automated.

Conclusion

Counting individuals in each frame proved to be more difficult than expected. The best model was off by roughly ~3.25 RMSE. (Selecting the mean target variable would result in a ~6.94 RMSE.)

Some key takaways from the project:

- One of the simplest models performed the best at a mean squared error of 10.56 MSE (2 CNN layers & 62,891,665 trainable parameters)
- LeakyRelu improved performance on models with gradient issues
- Drop out appears to be more effective in improving model performance than batch normalization
- The models are adequately capable of picking up on small differences in the images (ie. No mean normalization is required)
- Adding complexity to the models did not improve performance.
  Additional training with various model variations is required in order to improve overall performance of the model.

References

1. From Semi-Supervised to Transfer Counting of Crowds
   C. C. Loy, S. Gong, and T. Xiang
   in Proceedings of IEEE International Conference on Computer Vision, pp. 2256-2263, 2013 **(ICCV)**

2. Cumulative Attribute Space for Age and Crowd Density Estimation
   K. Chen, S. Gong, T. Xiang, and C. C. Loy
   in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 2467-2474, 2013 **(CVPR, Oral)**

3. Crowd Counting and Profiling: Methodology and Evaluation
   C. C. Loy, K. Chen, S. Gong, T. Xiang
   in S. Ali, K. Nishino, D. Manocha, and M. Shah (Eds.), Modeling, Simulation and Visual Analysis of Crowds, Springer, vol. 11, pp. 347-382, 2013

4. Feature Mining for Localised Crowd Counting
   K. Chen, C. C. Loy, S. Gong, and T. Xiang
   British Machine Vision Conference, 2012 **(BMVC)**