IRJIEST 5 (2019) 23-29
Available online at http://irjiest.com.ph
International Research Journal on Innovations in Engineering, Science and Technology

A novel multi-scale crowd counting algorithm using python (PyMCCA)

Janice Peralta, Ian Benedict O. Aguila, Paul C. Atienza, Joseph Jeremy B. Botardo

College of Engineering, Architecture, and Fine Arts, Electronics-Instrumentation and Control-Mechatronics Engineering Department, Batangas State University

ABSTRACT

This research was conducted to create a Multi-Scale Crowd Counting Algorithm. This algorithm implements a patch-based inference that is able to give the number of people in an image whether the image contain a small or large crowd size. The algorithm was designed to perform within the standard of the existing small scale and large- scale crowd counting algorithms using Convolutional Neural Network Architectures such as *Inception V3* for the crowd classifier, *Resnet101* for the small-scale model, and *VGG-16* for the large-scale model. This algorithm was implemented along with a user interface using PyQt5 GUI designer to make it more convenient to use. The algorithm works by identifying whether an image contains a small or large crowd, then implements a model best suited for the image. If the image is determined to contain small amount crowd, the image will be divided into 4 patches then a Faster R-CNN model trained on human head and body annotations will be implemented, while if the image contains a large number of crowd, the image will be divide into 9 patches then an inference algorithm based on CSR model will be implemented.

Keywords: Convolutional Neural Networks, Crowd Counting, Faster R-CNN, CSR model

1. Introduction

Crowds occur every day in situations like rallies, concerts, political speeches, stadiums, and even marathons. They are an important aspect and need to be considered in terms of public safety, public transportation, and the correct establishment of news or data for media. Counting every individual in the crowd manually would need a lot of time and effort, and it could almost be impossible. Although the number of people at an event can be estimated by some experienced personnel, it would still give an inaccurate result. To reduce the time consumed in crowd counting and improve its accuracy, computer vision solutions can be implemented.

Crowd detection and counting is an essential tool for safety and crowd surveillance such as the detection of unusual crowd behavior and effective deployment of police officers. For crowd monitoring and public transportation, crowd counting can be an informative resource of data in infrastructure development which would help a lot in improving the pedestrian flow, traffic congestion, and crowd flow in shopping malls and certain events. Also, this solution can be further developed to analyze complexities that involve the crowd by tracking and detection of unusual behavior. By this, dangerous situations (stampedes, suffocation, death) brought by overcrowding in events like parades, concerts, and rallies can be prevented.

Crowd-counting algorithms, techniques, and methods were already contributed by lots of researchers from around the world. They are applied in different computer programs through image processing and are implemented in various applications. Traditional methods and algorithms, multiple local feature-based algorithms, and background segmentation techniques are some of the notable solutions for small-scale crowd counting.

Traditional methods and algorithms that include head and face detection for crowd counting performs well in small scale to medium scale crowd. When it comes to a large-scale crowd, their accuracy drops significantly. These methods/ algorithms use multiple local features and segmentation techniques to count the number of people in a single image.

Poor segmentation performance, reduced image resolution, and erroneous ground truth labeling. [1] Background segmentation focuses on the foreground and removes the background of an image to extract the target or blob and be counted. On the other hand, foreground segmentation focuses on the background and removes the foreground of a specific image. The problem with these techniques is that if applied to a larger crowd, they were unable to detect every single sample in an extremely compressed crowd. They cannot undergo processing for sample overlapping.

Crowd counting is an important research topic in the field of computer vision but the accuracy of crowd counting based still needs to be improved. [2] The problem lies within the accuracy, scalability, and practicality of these algorithms when dealing with a larger crowd scope. Researchers have already identified the problem in terms of large-scale crowd counting on still images. The notable challenges to a large-scale counting are: Low Resolution (few pixels per target) which makes the analysis difficult, Severe Occlusion wherein body parts are not visible, and Perspective Effects or the change in scale.

Throughout the years, researchers have found a way to gradually improve large-scale crowd counting methods. Idrees et. al [3] have proposed a method that includes interest point based counting, Fourier analysis, head detection, and Markov random field. They were successful in accurately estimating crowd counts at densities up to about 1,280 people using the UCF_CC_50 image dataset but, unfortunately, their design gave inaccurate results in counting small-scale crowd. Since then, the scalability of the crowd is still a challenge up until today. Due to the improvement of crowd counting studies nowadays, present methods were now using Deep Learning to address the dense crowd counting challenges. Zhang et. al [4], Boomingthan et. al [5], and Sam et. al [6] have used deep learning through convolutional neural networks to provide better accuracy but focused on detection and counting in large crowd densities.

A notable strength of deep learning is that it can be trained well so it can be able to detect unseen humans in a crowd. If trained well, desired results can be obtained. A downside of this technique is that it will require a lot of

*Corresponding author
Email address: janice.peralta@g.batstate-u.edu.ph

patience and time since a single training will require lots of hours and even days before it can be used for image processing. Even so, reliable results can be obtained by this method which is better and feasible than conventional ways.

Based on these studies, the researchers realized the possibility of solving the crowd detection and counting scalability problem by training models through deep learning and the utilization of convolutional neural networks added with head detection, feature detection and extraction, image patch division, and other crowd counting methods as a feasible method to detect and count individuals in a crowd by a trained model in Python programming language. It's been a privilege of Batangas State University – BS Electronics Engineering researchers to design, build, and use powerful open-source tools for image processing to create a crowd counting algorithm which is named "A Novel Multi-Scale Crowd Counting Algorithm Using Python (PyMCCA)".

2. Materials and methods

This section presents the various procedures that were initially done for the development of a new algorithm and optimally achieve the research objectives. This portion serves as the researchers' guide throughout the entire process of developing a multi-scale head counting and detection in a still image.

2.1. Predesign stage

Through extensive research, the researchers confirmed the use of Python as a feasible programming language tool to do image processing/computer vision techniques and as the most optimal alternative to MATLAB considering the constraints and advantages. Image processing techniques can also be done on alternative software with their own language such as ImageJ, Scilab, Microsoft Visual Studio, Amira, etc. Their distinction lies within its cost, ease of use, minimum hardware specifications, and portability feature. The researchers came up to decide to use Python with Spyder as its IDE and extend its libraries to perform this study based on its favorable advantages when it comes to computer vision algorithm development, low hardware specification requirement, cross-platform feature, and future microcontroller embedding.

The researchers believe that the optimal hardware requirement for the system will solely depend upon the application. When it comes to deep learning, having a computer of the best specifications will truly beneficial for deep learning tasks. According to Huntlaptop.com, an Intel Core i7 is highly recommended for machine learning or deep learning applications. Moreover, a dedicated GPU would be highly preferred to render well on projects or jobs. The preferred dedicated GPU series is the NVIDIA GTX 10 series or above. RAM Memory of 16 GB Capacity would be sufficient but 32 GB could also be considered if budget is not considered as a hindrance.

The researchers used four different datasets to evaluate the performance of the algorithm. These datasets include two small-scale crowd datasets and two large-scale crowd datasets. For small-scale datasets, the researchers used the UCSD dataset which contains 2000 grayscale images, and SmartCity dataset which include 50 colored images. For large -scale datasets, the researchers used UCF_CC_50 dataset which include 50 grayscale images, and ShanghaiTech Part A which contains 482 colored images.

Working with Spyder as the main IDE is chosen due to its MATLAB-like environment. Moreover, exploring the variables will be at much ease. When further debugging the code, Jupyter Notebook will be the best alternative. A combination of Jupyter Notebook and Spyder is considered the best IDEs by the researchers since it will work well for the development of the research study.

The researchers adhered to the use of Python libraries such as Tensorflow, Keras, pandas, matplotlib, Cython, Glob, lxml, numpy, open-cv, pillow, scikit-learn, and sci-py for the development of this study.

2.2. Design stage

The researchers wanted to have a crowd counting software (algorithm with GUI) with the capability to count from 0 to a maximum of 1000 headcounts from the chosen image datasets. In order to do that, the researchers studied image processing techniques such as CNN-based methods for head detection, crowd classification, image normalization, bilinear interpolation, image compression, density map generation, image patch division, one-hot encoding, GUI design, and counting through Tensorflow Object Detection API, Python programming language together with essential libraries.

The study adheres to the challenge of counting at extreme density. The problem lies with counting every head. This is almost impossible due to the loss of the necessary pixels to detect a person in a very dense crowd. The researchers will tackle this problem by estimating the count of people in dense quantities. The method will be based on crowd analysis methods such as convolutional neural network, patch division, filtering, and final count as the final process.

When dealing with procedures and processes, devising a flowchart is a helpful way to portray a logical and systematic approach. This part includes the algorithm flow considering the general methods for crowd detection.

The devised flowchart will undergo a series of processes after a single input image is selected. A binary crowd classifier will be first initiated by the algorithm as an input image is selected and will implement a small-scale algorithm or large-scale algorithm according to its string output. Since the image is crowded with head samples, the image will be

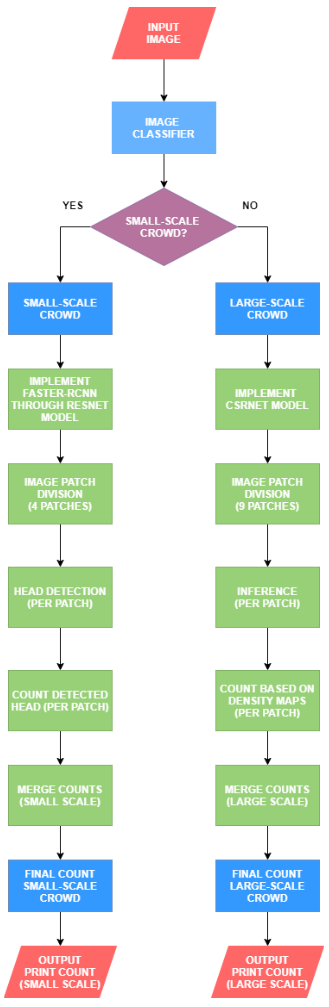

INPUT IMAGE
IMAGE CLASSIFIER
YES
SMALL-SCALE CROWD?
NO
SMALL-SCALE CROWD
LARGE-SCALE CROWD
IMPLEMENT FASTER-RCNN THROUGH RESNET MODEL
IMPLEMENT CSRNET MODEL
IMAGE PATCH DIVISION (4 PATCHES)
IMAGE PATCH DIVISION (9 PATCHES)
HEAD DETECTION (PER PATCH)
INFERENCE (PER PATCH)
COUNT DETECTED HEAD (PER PATCH)
COUNT BASED ON DENSITY MAPS (PER PATCH)
MERGE COUNTS (SMALL SCALE)
MERGE COUNTS (LARGE SCALE)
FINAL COUNT SMALL-SCALE CROWD
FINAL COUNT LARGE-SCALE CROWD
OUTPUT PRINT COUNT (SMALL SCALE)
OUTPUT PRINT COUNT (LARGE SCALE)

Figure 1. Proposed algorithm.

divided into certain divisions for counting and run inference for head detection for each patch.

The head detection algorithm will then be executed by executing Tensorflow Object Detection API or Congested Scene Recognition Method. As the head samples are detected, the count output for every patch will be merged and will be printed on the designed GUI. The input image, image patches, and final image output with detections or density maps will be portrayed on the GUI.

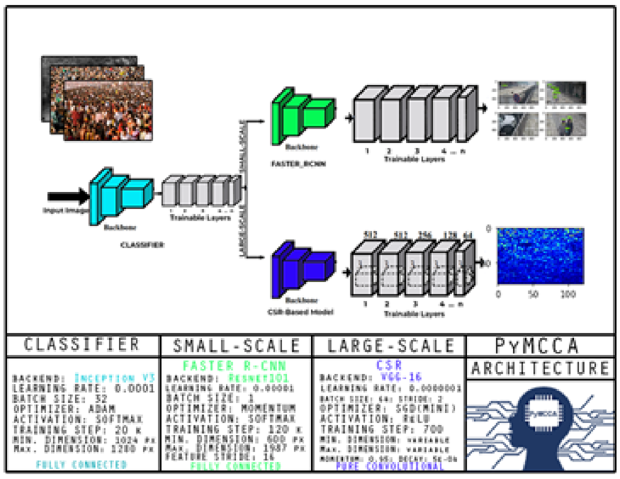


Figure 2. Design architecture.

The proposed algorithm deploys the final architecture as shown on the figure. The Backbone that was used on the classifier is an Inception V3 Model with its trainable layers trained to perform classification with two crowd scales. For small-scale, the Faster R-CNN backend that was used is a Resnet101 Model with its trainable layers trained to detect heads for small-scale. For Large-scale, the backend model that was implemented is a VGG-16 Model together with its trainable layers with 3x3 filters and a 1x1 filter at the last layer for the detection and counting of heads.

2.3. Development preparation stage

Along with the creation of the algorithm, the researchers also created a functional GUI. For the creation of the Graphical User Interface (GUI), the Spyder Integrated Development was used alongside with Qt Designer (PyQt5). Spyder IDE is a powerful scientific environment written in Python. It features a unique combination of editing, analysis, profiling, and debugging functionality of a comprehensive development tool with the data exploration, deep inspection, interactive execution, and visualization capabilities of a scientific package. It offers built-in integration with many scientific packages including Numpy, Matplotlib, Pandas, Scipy, QtConsole, and many more. Furthermore, it can also be used as a PyQt5 extension library, which in return allows us to build upon its functionality and embed its components. Qt Designer (PyQt5) is a Qt tool for building and designing Graphical User Interfaces (GUIs) with Qt Widgets. In this module, composing and customizing windows or dialog boxes in a what-you-see-is-what-you-get manner and testing them using different resolutions and styles are possible. Forms and Widgets created with Qt Designer (PyQt5) integrate smoothly with programmed code, using Qt's slots mechanisms and signals so that the assignment of behavior to graphical elements will be easy. Furthermore, all properties set in PyQt5 can be changed dynamically within the code.

The Anaconda package manager also includes Jupyter Notebook as a part of its package. The researchers also made use of this IDE since programming can be more manageable by its cell division run time, consuming RAM Memory at a time, and early detection of errors by debugging per cell.

LabelImg is an annotation tool for graphical images. It is written in Python and uses Qt Designer for its graphical interface. It was used to label object bounding boxes in images. This annotation tool was used because it is easy to use and it is portable to any operating system.

2.4. Testing stage

The proposed crowd counting algorithm will be based on the chosen methods as provided on the initial flowchart. The tools for testing are (a) Spyder, Jupyter Notebook IDE (b) A computer/laptop.

The researchers made use of density map generation will be essential when it comes to training the pre-trained VGG-16 Model. The process includes the input image being selected at first. The ground truth will be extracted from .MAT files. After extraction, the extracted ground truth which is now in the form of an array will then be passed through one-hot encoding to properly place the exact locations of the heads. The output will then be an h5 file that is compatible with Keras library.

Another important image processing technique that the researchers used for their methodology is normalizing the images for Congested Scene Recognition. It is deemed to be essential especially for VGG-16 Model to have normalization be initialized for training. The process includes the input image to be subtracted with the dataset mean of VGG-16 together with its standard deviation per channel (RGB). The output image would then be used for running inference prediction.

In this approach, the researches intended to use patch-based inference for better detection of heads. The whole image-based predictions sometimes fail to count some heads in the image. To counter this, the researchers divided the images and run prediction inferences for each patch. For congested scene recognition, the researchers implemented a 3x3 or 9 patches in total.

Head and upper body features data was manually labeled by the researchers using LabelImg software. The annotated images will generate an XML file containing the position of heads and was converted to a CSV file to be ready for training.

As the object detection algorithm generates the scores array, it would be squeezed to reduce its dimension. By this, the squeezed array will contain every detected head considering its threshold. The final count will be extracted.

For the CSR Algorithm, the generated density map for the image input will be then converted to a numpy array. It will then be passed to a numpy sum method to sum the array of elements that will give out the final count.

The datasets that will be used are packaged with their real headcounts. To determine the accuracy of the algorithm developed in python language, this will be compared with the real crowd count. To test this, input images will be extracted from the datasets and percent difference calculation will be developed to show the amount of difference with respect to accuracy.

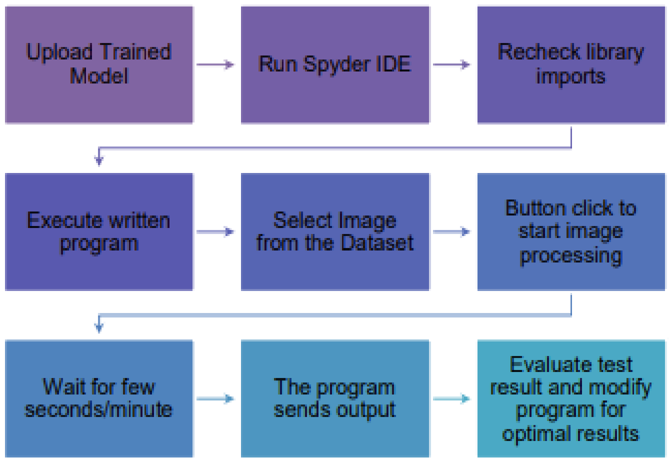

Upload Trained Model
Run Spyder IDE
Recheck library imports
Execute written program
Select Image from the Dataset
Button click to start image processing
Wait for few seconds/minute
The program sends output
Evaluate test result and modify program for optimal results

Figure 3. Algorithm and GUI testing plan.

The step-by-step procedure above for the algorithm testing process will be executed by the researchers. By this, the researchers will be able to learn more by actual prototype testing. Problems not anticipated in the early stage will also be resolved and addressed in this process.

3.5. Evaluation

The count output is the most significant data in this study since it will provide accuracy with the provided real headcounts. To evaluate the output of the designed algorithm, graphs and charts will be used to visually observe the algorithm's output with respect to varying scales.

As provided on the objectives of this study, the researchers will be using MAE for the evaluation of the performance of the final algorithm with regards to two scales. There are accepted MAE values for every dataset included on this study as provided.

Small-scale - Accepted Mean Absolute Error (MAE) of

- UCSD Image Dataset – **2.86**
- SmartCity Image Dataset – **8.6**

Large-Scale – Accepted Mean Absolute Error (MAE) of

- UCF_CC_50 Image Dataset – ***468***
- ShanghaiTech Part A Image Dataset – ***181.8***

4. Results and discussion

The researches initially tried with the OS readily available for them to install Anaconda. In the initial phase of development, the researchers faced a slow run time of software pre-installed by Anaconda. Moreover, the researchers observed that good internet speed is needed when preparing well for the installation of the needed libraries. The researchers faced problems with using pip for the installation of packages together with conda installed packages. Also, they found out that it is better to install python libraries through conda since it also includes additional installation of necessary packages to install a specific library. The researchers then tried to install a Linux-based OS through dual-boot which is Ubuntu 18.04 Bionic Beaver. Installations were way better and observed better software run times with their laptops used for developing the algorithm.

The researchers have chosen FasterR-CNN (Faster Region-based Convolutional Neural Network) to be used for detecting heads for a small-scale crowd. This algorithm performs very well at a high accuracy but has a tradeoff when it comes to speed of detection. Detection accuracy was simply chosen over speed to get lower MAE results. The datasets that were trained on this algorithm include the following (Table 1.)

Table 1. Faster R-CNN training setup.

Dataset	Number of Images	Image Size (px)	Head Annotations
UCSD	**2000** 800 – *Train* 1200 - *Test*	238 x 158	11-46 per image
SCUT-HEAD Part B	**2405** 1905 - Train 500 - Test	Varying, approx. 640 x 245 to 1024 x 900	43,940

The researchers considered to use UCSD's training set to be trained for the small-scale algorithm. It contains images that were challenging to be detected by computer vision. Also, it contains small heads at about px by px. This would be desirable in counting small-scale crowd at a far or top-view perspective. It contains 1,200 images for training and 800 images for testing. For transfer learning, there is a rule of thumb of 1,000 needed images (more or less) only. This is a big advantage when it comes to training time than training from scratch since ImageNet pre-trained models as backend already gives 90% and above top 1 and top 5 accuracy.

Before the training that will be done by using the Faster-RCNN algorithm, the researchers first annotated the small-scale crowd images that will undergo training. Using the LabelImg annotation tool, the researchers created bounding boxes for each individual on the cro wd images and labeled each of them as "head".

As the labeled images with their corresponding XML files were prepared by the researchers, the conversion of XML to CSV to generate its TFRecord was then initiated.

This CSR model was used for the counting of people (based on density maps) on large-scale crowd images. The datasets that were trained on this model include the following:

Table 2. CSR-based model training set-up.

Dataset	Number of Images	Image Size (px)	Head Annotations
Shanghai Tech Part A	**482** 182 – Train 340- Test	Varying: average of 1024 x 760	241,667

The researchers simply chose part A of the said dataset since it contains most of the annotations containing 241,667 on the total 330,000 heads of the whole dataset. Moreover, the crowd is much more congested at Part A than Part B.

For the preparation of density maps, the process implemented can be seen in Figure 4.

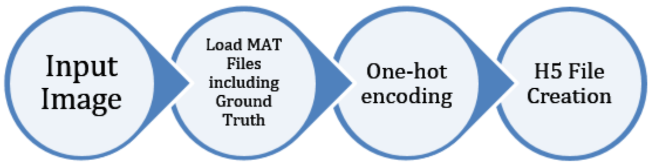


Figure 4. Density map generation process.

Table 3 shows the information about the ground truth generation for each dataset. Both UCSD and largescale dataset was included for this process since the researchers intend to extract every ground truth value for MAE evaluation later on. Also, these files were needed when training the model to be trained on CSR.

Table 3. Density map generation.

Dataset	Number of Images	Approximate Time Consumed
UCSD	**2000**	2 hours and 14 minutes
`ShanghaiTech	**1198**	5 hours
UCF_CC_50	**50**	2 hours and 15 minutes

VGG-16 was the chosen backend by the researchers for Large-scale algorithm. VGG-16 has also been used by other researchers for analyzing and counting large-scale crowds due to its strong transfer learning ability. Before applying transfer learning and retraining VGG-16, it is needed to perform image normalization before deploying the model. The following Mean and Standard Deviation were given on the Table 4.

Table 4. VGG-16 mean and standard deviation values.

Color Channel	Mean	Standard Deviation
Red	0.485	0.229
Green	0.456	0.224
Blue	0.406	0.225

The normalization function is called whenever input images are being trained. Also, it would be used when running the Inference prediction. This is necessary to get the desired prediction. If the normalization process was bypassed, the training for CSR-based model would considerably take more steps to get the desired Euclidean Distance Loss. On the other way around, if the normalization function was not included for the inference prediction of count, it would give poor prediction results.

The researchers proceed to train the labeled images for Faster R-CNN algorithm for different models. Different backend models were trained for evaluation as to which backend model will be the best fit for our training data. These backend models were pre-trained on COCO dataset. COCO dataset consists of 330K images with 80 object and 91 stuff categories. The good thing about this dataset is that it contains 250,000 people with key points which is ideal for this study. Kitti dataset is another considerable dataset since it contains at about 30 people per image. The researchers tried to implement training with models at the highest maP but it results to resource allocation error. With this in consideration, the researchers were only limited to the use of backend models. The following Faster R-CNN backends were chosen due to its training feasibility.

Training a new model from scratch would usually need 10,000 images per class. This would consume a lot of time on training. Transfer learning is a chosen technique which enables the researchers to harness a neural network from a previous specific task and apply it on small crowds. By transfer learning, images or annotations could be reduced up to 1,000 samples or less.

For the first setup, the researchers initially tried to train SCUT-HEAD Part B dataset and followed its training and testing set on Faster_RCNN_Inception_V2consideringits training speed. For 50,000 - 200,000 steps which is the sufficient limit of most of the Tensorflow Object Detection API models, the researchers initially trained as far as stated. The training result is visually analyzed through the given loss values using Tensorboard.

For the first training setup, the researchers still need to perform more steps for training. When the model reaches about 0.04-0.05 loss at 647k steps, another test was initiated.

SmartCity dataset was labeled beforehand but its image resolution is too big for Faster R-CNN architectures which has a limit mostly at 1024 x 600 image resolution. To counter this, the researchers tried to configure the configuration file of the model to increase the input size limit but it does not go well positively due to *Resource Exhausted Error* thrown during training. Due to this, researchers went along with images compatible for training Faster R-CNN model architectures.

For the UCSD dataset, the researchers initially picked 140 images with approximately 1,800-2,000 head annotations with training set as 112 images and testing set as 28 images.

Provided below are the graphs of selected feasible models that the researchers used. Their analysis graphs are provided correspondingly. The values of the loss are downloaded from local web server localhost:6006 and their plots were generated using Microsoft Excel.

The researchers have chosen ResNet101- Kitti as their model backend for Faster R-CNN algorithm for small-scale crowd detection and counting based on the training results.

For training the CSRNet the researchers did the following configuration as advised on the CSRNet paper by Yuhong Li et. Al [7]. The dataset used for the Congested Scene Recognition method uses ShanghaiTech Part A which consists of 182 images for testing and 300 images for training. The corresponding configurations for training together with other parameters such as optimizer and activation function that was given by CSRNet was also initiated for replicating the model.

VGG-16 was chosen due to its strong transfer learning ability for congested scene crowd according to [7] Another notable mention for this implementation is that it uses Dilated Convolutional Layers rather than Max Pooling Layers. The researchers followed this setup for training. The comparison result below. Shows how dilated convolution gives better details than the pooling procedure.

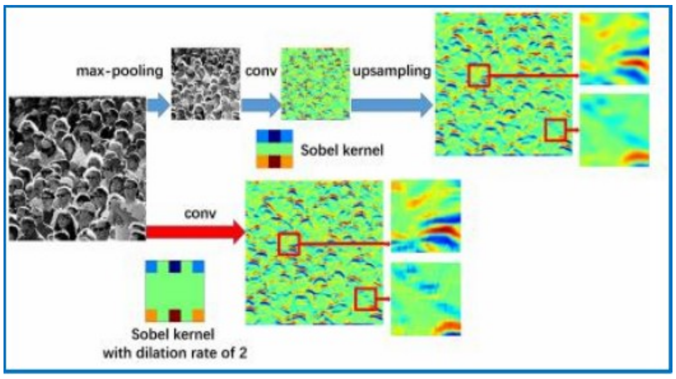
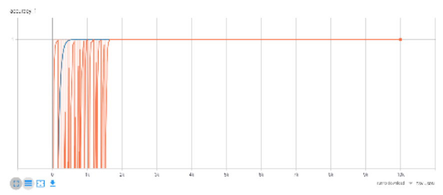


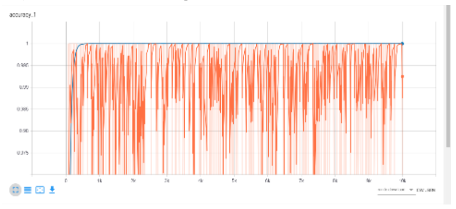
Figure 5. Max pooling and upsampling vs. dilation.

For the training, the researchers need to make sure that the normalization code was also included. The final code implementation for training CSR-based model was then executed. The expected training loss for Euclidean distance should be at 0.07.

After 700 steps, the researchers got the model saved as 'model_A_weights.h5' which is in hdf5 format compatible with Keras API. A loss value of 0.0787 was also generated. It is worth mentioning here that on the small-scale implementation, a loss of 0.05 was considered for training FasterR-CNN on a total loss graph. For the CSR-based model, Euclidean distance loss was chosen to measure its performance with the generated ground truth density maps during image pre-processing.



a) Adam training



b) Gradient descent training

Figure 6. Training using a) Adam and b) gradient descent.

For the Multi-scale algorithm, there should be a classifier that would predict what image input is being fed into the algorithm. To implement this, the researchers made use of a Tensorflow Image Classifier based on the Inception V3 model. The implementation was based on Tensorflow for Poets guide but the researchers made a lot of fine-tuning to get desirable classification results. For the training images, the researchers trained 182 Test Images from ShanghaiTech together with its training set which sums up to 482 Images from ShanghaiTech, 15 Images from UCF Dataset. For the Small-scale Image set, the researchers also trained the model with 1,200 Test Images from UCSD Dataset with additional augmentation and all SmartCity Images. The Training, testing, and validation set was randomly chosen by the code implementation by setting 80% for Training and another 20% for Validation. The researchers got favorable results on testing using this scheme which resulted to correct implementation of the right algorithm for each scale which was presented on the MAE graphs later on. The evaluation shows the following graphs:

Better training accuracy and validation accuracy results were observed using Adam Optimizer for the Crowd Classifier. This was then the chosen setup for the Crowd Classifier model.

Each dataset was subjected to the MAE testing as stated on the objectives of this study. The following result shows the summary of garnered MAE values for each dataset. It is important to note here that the final algorithm was initiated together with the image classifier on top of both Scale algorithm.

The graph above shows how well the Small-scale algorithm performs with SmartCity Dataset with a total of 0.64 Mean Absolute Error.

For UCSD Image Dataset the researchers got an MAE of 2.4 which is also in-lined with the objectives. The Prediction (blue line) shows that the evaluation had given a desirable plot graph with respect to the ground truth. Some minor misalignment of both lines was due to occlusions that are present on the UCSD Image Dataset which can be vividly visualized starting from 631 Image count.

Since the CSR-based model was trained on ShanghaiTech dataset Part A with its respective train and test division, desirable results were expected. The prediction and ground truth curves show how well the whole algorithm correctly classified every test image for ShanghaiTech dataset which is considered crucial for the calculation of MAE. The best results were summarized in the next corresponding table.

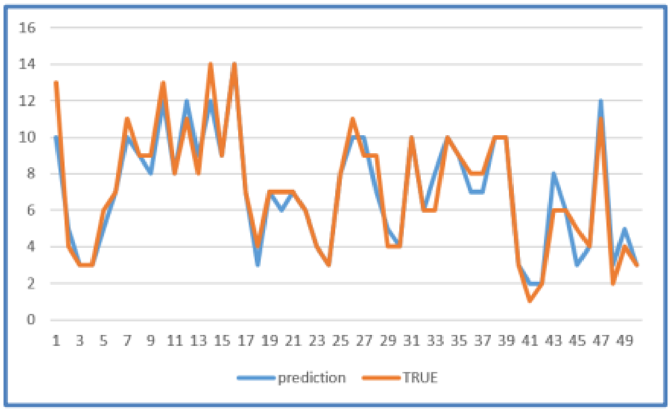


Figure 7. Smart city MAE result - 0.64.

For UCF Image dataset, the researchers were able to generate a total MAE of 432.21 which is in-lined with the research objectives. This dataset as not trained with the CSR-based model and still gives a result much better than the study of Idrees et. al that uses algorithms such as Fourier Analysis. UCF is a challenging dataset due to its maximum count of 4600 heads on a single image. This contributes a large error if maximum counts were far from the predicted values. This is then considered as a limit of this study.

By the graph results, the researchers found out that implementing Image Patch Divisions were indeed a big help for garnering better crowd count which leads to the accomplishment of the objectives of this study. The trade-off

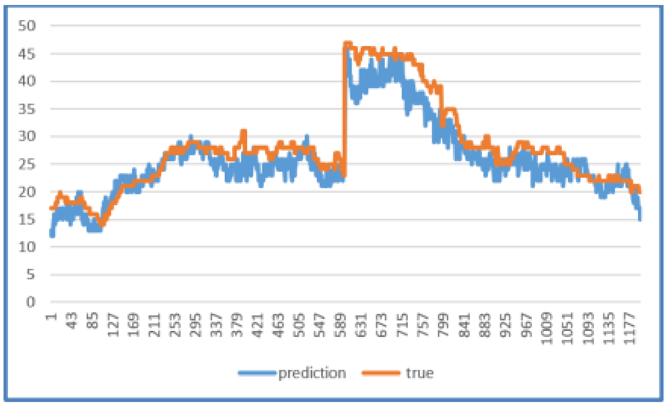


Figure 8. UCSD MAE Result – 2.4.

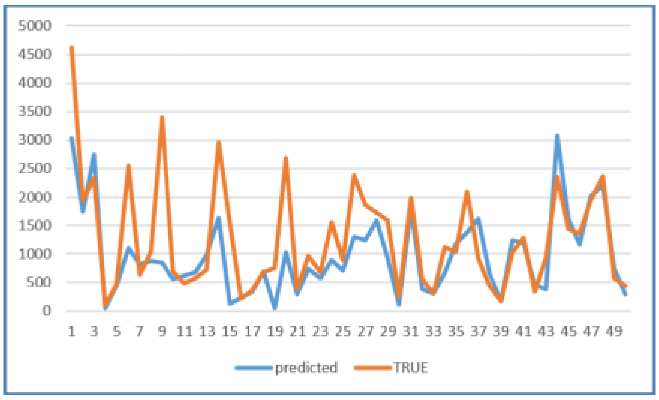


Figure 9. ShanghaiTech MAE result – 69.42.

for this image patches is that higher computing power is much more needed if it needs to be applied to real-time applications. This is considered as this study's limitation and it focuses on the study of giving acceptable MAE values for varying crowd scale which was then achieved.

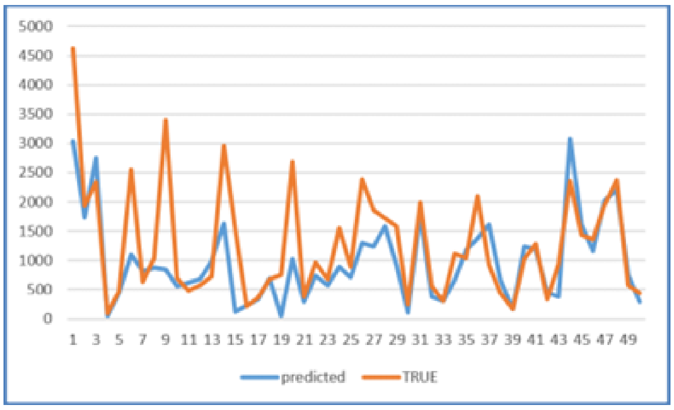


Figure 10. UCF MAE Result – 432.21.

For the creation of the GUI, the researchers first created a form composed of labels, pushbutton, and line edit using the Qt Designer (PyQt5). The objects used inside the form was placed using the drag-and-drop method. Furthermore, in order to change the properties of the objects used including text sizes and frames for labels, Qt designer Property Editor was used.

Now, in order to improve the visual appearance of the form, the researchers added additional labels for them to load some images and texts. For the new and final appearance of the form, a total of twelve (12) labels, one (1) push button, and one (1) line edit was used. This was also done through the Qt designer (PyQt5)

In order to put some functions on the objects used again, Spyder was used. The algorithm created by the researchers was also placed inside the Python code done through Spyder.

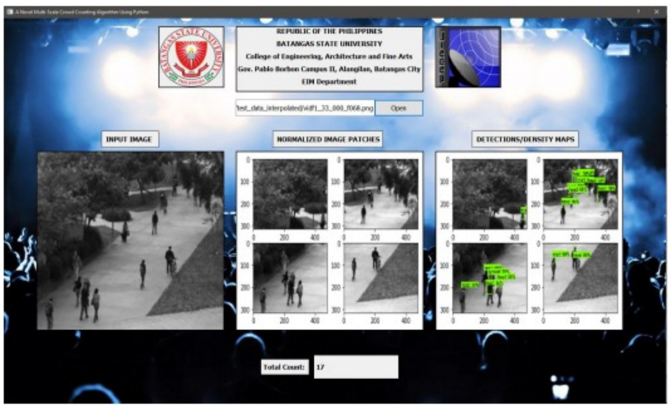

REPUBLIC OF THE PHILIPPINES
BATANGAS STATE UNIVERSITY
College of Engineering, Architecture and Fine Arts
Gov. Pablo Borbon Campus II, Alangilan, Batangas City
EIM Department
Open
INPUT IMAGE
NORMALIZED IMAGE PATCHES
DETECTIONS/DENSITY MAPS
Total Count:
17

Figure 11. Implementation on small-scale.

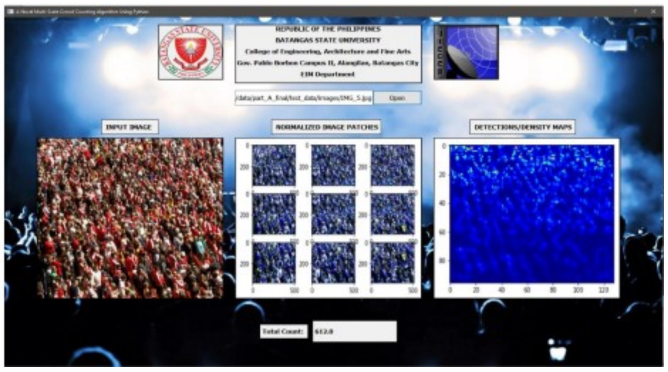

REPUBLIC OF THE PHILIPPINES
BATANGAS STATE UNIVERSITY
College of Engineering, Architecture and Fine Arts
Gov. Pablo Borbon Campus II, Alangilan, Batangas City
EIM Department
Open
INPUT IMAGE
NORMALIZED IMAGE PATCHES
DETECTIONS/DENSITY MAPS
Total Count:
612.0

Figure 12. Implementation on large-scale.

To make the algorithm executable, PyInstaller was used. PyInstaller takes the python code for the algorithm and byte-compiles it. It creates an executable application that basically loads and runs the algorithm and analyzes the code to discover every other module and library that the algorithm needs in order to execute.

4. Conclusion

The researchers were able to develop a multi-scale crowd counting algorithm by implementing three deep convolutional neural networks, namely: InceptionV3, ResNet101, and VGG-16. InceptionV3 was used as the image classifier, classifying whether the input is an image with a small-scale crowd or a large-scale crowd. ResNet101 was used for Faster-RCNN for object detection which the researchers have proven to be effective on small-scale crowds. Lastly, VGG-16 which was used as the backbone for the CSR-based model which was proven to be effective on large crowds. The researchers have proven this type of algorithm to be effective in counting crowds with different scales.

A Graphical User Interface was done by the researchers for easier implementation of the algorithm. The GUI provides less work on running the program and keeps the program codes safe from unintentional alterations. Implementing the program on a GUI does not vary the behavior of the algorithm.

The algorithm was able to perform on the standards set by the researchers based on the minimum accepted values of MAE in different datasets. The developed multi-scale crowd counting algorithm has achieved the following MAE upon evaluation:

PyMCCA Performance:
Small-Scale MAE:
UCSD Image Dataset – 2.4
SmartCity Image Dataset – 0.64
Large-Scale MAE:
UCF_CC_50 Dataset – 432.21
Shanghai Tech Part A Image Dataset – 69.42

References

[1] Taran, V., Gordienko, Y, et.al. (2018), *Impact of Ground Truth Annotation Quality on Performance of Semantic Image Segmentation of Traffic Conditions*.

[2] Z. Liu, Y. Chen, B. Chen, L. Zhu, D. Wu and G. Shen, *Crowd Counting Method Based on Convolutional Neural Network With Global Density Feature*, in IEEE Access, vol. 7, pp. 88789-88798, 2019, doi: 10.1109/ACCESS.2019.2926881.

[3] Idrees, H., Saleemi, I., Seibert, C., Shah, M. (2013). *Multi-Source Multi-Scale Counting in Extremely Dense Crowd Images.*

[4] Zhang, L., Shi, M., Chen, Q. (2018). *Crowd counting via scale-adaptive convolutional neural network.*

[5] Boominathan, L., et. al. (2016). *CrowdNet: A Deep Convolutional Network for Dense Crowd Counting*.

[6] Sam, D. B., Surya, S., Babu, R. V. (2017). *Switching Convolutional Neural Network for Crowd Counting.*

[7] Li, Y., Zhang, X., Chen, D. (2018). CSRNet: *Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes* (UnpubDoctoral Dissertation, University of Illinois).

Data Sets

Crowd Counting Data Set (UCF-CC-50). *https://www.crcv.ucf.edu/data/ucf-cc-50/*

USCD Crowd dataset. *http://visal.cs.cityu.edu.hk/downloads/*

Smartcity Datate. *https://pan.baidu.com/s/1pMuGyNp#list/path=%2F*

ShanghaiTech Part A. *https://www.kaggle.com/tthien/shanghaitech*