A General Purpose Intelligent Surveillance System For Mobile Devices using Deep Learning

Antreas Antoniou
School of Computing and Communications
Lancaster University, Lancaster, UK
a.antoniou@lancaster.ac.uk

Plamen Angelov, FIEEE
School of Computing and Communication
Lancaster University, Lancaster, UK
p.angelov@lancaster.ac.uk

***Abstract*—In this paper the design, implementation, and evaluation of a general purpose smartphone based intelligent surveillance system is presented. It has two main elements; i) a detection module, and ii) a classification module. The detection module is based on the recently introduced approach that combines the well-known background subtraction method with the optical flow and recursively estimated density. The classification module is based on a neural network using Deep Learning methodology. Firstly, the architecture design of the convolutional neural network is presented and analyzed in the context of the four selected architectures (two of them recent successful types) and two custom modifications specifically made for the problem at hand. The results are carefully evaluated, and the best one is selected to be used within the proposed system. In addition, the system is implemented on both a PC (using Linux type OS) and on a smartphone (using Android). In addition to the compatibility with all modern Android-based devices, most GPU-powered platforms such as Raspberry Pi, Nvidia Tegra X1 and Jetson run on Linux. The proposed system can easily be installed on any such device benefiting from the advantage of parallelisation for faster execution. The proposed system achieved a performance which surpasses that of a human (classification accuracy of the top 1 class >95.9% for automatic recognition of a detected object into one of the seven selected categories. For the top-2 classes, the accuracy is even higher (99.85%). That means, at least, one of the two top classes suggested by the system is correct. Finally, a number of visual examples are showcased of the system in use in both PC and Android devices.**

I. Introduction

Surveillance systems are widespread (in the UK there are reported 4.2 Million surveillance cameras in use ten years ago [1]; now this number is obviously much higher), and most of them are being computerised. However, the dramatic increase of the power and lowering the cost of the devices such as smartphones, GPU (graphic processing units) as well as the developments in the area of computational intelligence and neural networks, in particular, open new horizons for the confluence of the advanced platforms and advanced algorithms.

In this paper, we present a general purpose surveillance system that has at its core two main elements: i) a detection module, and ii) a classification module. The detection module is based on the recently developed and published SARIVA method [2] and WhatMovesApp [3] application for detecting objects that move using smart-phone devices. The detection module was a subject of a recent publication [2] and will only briefly be outlined in section II. The main novelty of this paper is the classification module which is based on convolutional neural network (CNN) using Deep Learning methodology [4]. This module classifies the object into one of the seven pre-defined categories using a pre-trained CNN based on the DL methodology. These categories include "*human*", "*cat*", "*dog*", "*car/4-wheeled vehicle*", "*motorbike*", "*bird*", and "*trees/forest*". The reason only seven categories were chosen is because maximum accuracy was desired on categories that are valuable for a general purpose home or pet security system. The system is general purpose as the user can select which categories should trigger notifications and what actions to be taken after a category has been identified such as raise an alarm, send an SMS, email and/or a photograph using GSM or Wifi, etc. Having a smartphone as a platform makes it possible to get a two-way communication with the remote user who may ask specific type of requests following a detection of object of a certain class at a certain time instant. For example, if an object is being autonomously being detected and this has been classified as being a *human*, the system can send a message to the remote user. The user may or may not request a photo snapshot. The user may also request that all further images where objects of type *human* are being detected are also sent to him/her and/or saved.

The proposed system and smartphone application can be very useful for general use, for social science research, traffic studies and control, robotics-based image recognition applications and video surveillance.

Studying different architectures and their suitability for the specific task, the following recent benchmark deep neural network architectures were investigated: VGG-Net [4], GoogLeNet [5], AlexNet [6]. In this paper, we present the results of two custom made architectures which borrow some elements of the above in comparison with the original designs. They proved to be very competitive and particularly suitable for the low memory and processing capacity requirements of a contemporary smart phone as well as ensuring no latency so that a real-time applications is possible. The accuracy of the proposed system is ***95.9%*** in the Top-1 category. This level of accuracy is comparable and surpassing human level object recognition ability [7]. Therefore, the system can rightfully be called intelligent, and it is also very powerful.

The system is efficient to be used on currently available high-end mobile devices, as well as embedded system platforms such as Android-based smartphones, Raspberry Pi [8] and Tegra X1 [9].

Finally, we compare and contrast all the architectures investigated and identify future work and possible additions to the system. To test in a real world scenario, we implemented the system as a PC application as well as an Android app.

The remainder of the paper is organized as follows. Section II briefly outlines the SARIVA method used to autonomously detect any moving object. Further details on SARIVA method can be found in [2]. Section III describes the image classification module in general. Section IV then goes into the architectures of the proposed two custom Deep Learning classifiers as well as of two benchmark classifiers (AlexNet and GoogLeNet). Section V describes the data pre-processing and augmentation. Section VI provides details of the Experimental Results and Analysis. Section VII describes the specifics and demonstrates the computer and smartphone applications. Finally, the conclusions and the outline of the further work are given in Section VIII.

II. Moving Object Detection Module

The detection module of the proposed system is based on the recently introduced SARIVA approach [2] which is also implemented as an Android app and available on Google play Store [3]. SARIVA overcomes the main issue related with the use of a moving camera to detect moving objects, namely that the prior information about the objects to be tracked is assumed to be available. Other existing approaches also suffer from high computational complexity. In SARIVA this is overcome by the use of recursive calculation of the similarity between different images using data density [10]. The high computational complexity combined with the limited (although growing in recent years dramatically) computational, memory and energy resources of a handheld device mean that traditional approaches will not run real time on a smartphone. This leads to a reduced detection rate, restrictions on the movement of the camera/phone and makes the whole system not viable. The recently proposed SARIVA approach allows real-time object detection by introducing a new method called Optical ORB [2]. It has the ability to detect multiple objects without the need of prior knowledge about these objects. The main advantage of Optical ORB used in SARIVA and ,respectively, in the proposed system and also implemented and available on Google Play Store under the name WhatMoves app [3] is that there is no need for so-called "image stitching" to eliminate the effect of moving camera and get the ego-motion [2]. This operation used in traditional approaches as well as in the recently introduced ARTOT method [10] is most computationally costly within the object detection module. At its first stage, Optical ORB method used and implemented within SARIVA extracts the features from the previous image frame using ORB feature detector which is a pyramidal approach to the FAST feature detection algorithm aiming to detect stable keypoints [2]. *The ORB* detector can be tuned to detect a lot of features in a short amount of time compared to the standard methods [2] without degrading the quality of the detected points. Once the features have been selected, the Lucas-Kanade optical flow algorithm [11] is used to locate each feature's position in the current frame and, therefore, the optical flow displacement vector which describes the movement of the feature. Then the detection module applies an evolving clustering algorithm called ELM [12] that was also recently introduced by the authors which groups the features and thus removes the influence of the number and velocity of the moving objects. That is to say, on a scene there may be more than one object moving with different velocity vectors (speed, direction) and ideally, we would like to group the feature points that were detected into clusters or groups that correspond to these objects. Moreover, the cluster with the majority of the optical flow vectors is considered to represent the background. The features that relate to the background are then removed. More details on SARIVA and Optical ORB are provided in [2]. At the end of the detection module an area around each detected object is being cropped and each of these regions containing a single physical object is passed to the classification module which is described in the next section.

III. Image Classification Module

In this module, the proposed system classifies the cropped part of the image frame with detected object into one of the seven pre-defined categories, namely: "*human*", "*cat*", "*dog*", "*bird*", "*motorbike*", "*car/vehicle*" or "*tree/forest*". The architecture of the neural network and the training methodology used including the feature extraction falls within the so called Deep Convolutional Neural Networks [13] (CNN). A Deep CNN is composed of stacked convolutional layers that are used for feature training and extraction, followed by additional fully connected MLPs at the end to classify the features to each category and, thus, successfully classify the images. The deep learning framework "Caffe" [14] was used in combination with the tools provided by Nvidia called "Digits" [15] to train and visualize our models. The existing state of the art architectures that were studied as a starting point were:

1. GoogLeNet [5]
2. AlexNet [6]
3. VGG Net [4]

To allow a proper comparison all of the above three state of the art Deep CNN were build and trained as described in [5]-[6]. The only exception was with VGG Net because of its massive size and number of parameters (180 million parameters) the system (running on a 960 GTX) would need about a week to train it. Such an amount of time was not acceptable due to time-frame constraints and as such smaller networks inspired by it were trained instead as described in the next section. In addition, such an amount of parameters will consume too much memory, and access to them will not allow real-time application on a smart phone platform which is the ultimate aim of this study.

IV. Deep Neural Network Architecture Designs

As it is well known [13], the term *Deep Learning* was coined recently, although the problem with the amount of neurons, structure and architecture of the neural networks, so called "vanishing gradient", recurrent type networks and links with the memory [16]-[17] exist and were a subject of analysis much earlier (as early as 1990s) and all of them found a new angle of interest within the *Deep Learning* paradigm. Some define formally the Deep Learning (DL) type NN as ones in which there is a large number of layers, millions or billions of neurons and parameters. However, there is more to DL NN than just the size and quantitative factors. Within DL NN it is now considered also an automatic feature selection using CNN which found numerous applications to image [18]-[19] and speech processing [16]-[17] recently grabbing some of the headlines even outside of the scientific circles. In what follows, we briefly analyze the two existing popular architectures that we used as a starting point as well as the two proposed custom designs which borrow from the first two but go further.

A. AlexNet (Original)

AlexNet [6] was the network architecture that won the ImageNet competition in 2012, in which a computer system was required to classify 1000 different classes of animals and objects. AlexNet improved the state-of-the-art that year, bringing the error rate down astonishingly almost twice (to about 15% from the previous 26.5%). The AlexNet architecture is composed of cascading convolutional, Max-Pooling layers of size *3x3*, local response normalization (LRN) layers and *3* fully connected multilayer perceptron (MLP) layers to synthesize the features and classify the objects.

More specifically, a data layer is used that prepares the mini-batches and feeds them into the input layer. The reason a dedicated data layer is used is that by asynchronously building the next mini-batch while the network is training then a speed increase is achieved as no time will be spent building the batches in between iterations. The next component is the input layer that receives an image of size *224x224* in red-green-blue (RGB) format, thus, having a total size of *224x224x3*. The next component is the first convolutional layer which convolves patches of the image of size *11x11* with stride *4* and creates a total of *96* feature maps. Then the next component is the linear normalization layer that normalizes all values before sending them to the Max pool layer of size *3x3* and stride *2* that selects the dominant feature neurons to send to the next layer. After that the data is sent to the second convolutional layer that uses a patch of *5x5* with stride *1* and creates a total of *256* feature maps, followed by a Max pool and a normalization layer. Finally, the extracted features are sent to 3 cascading convolutional layers of filter sizes *512*, *1024* and *256* with patch size *3x3*, stride *1* and pad *1*. Then, the high-level features obtained from the final convolutional layer are then sent to a Max pool layer that selects the strongest activations to send to the fully connected layer of size *4096*, which then connects to a dropout layer with parameter *0.5*. In other words, this layer will randomly drop half its connections in the training session to prevent overfitting (this is a regularization instrument). The next module is another fully connected MLP of the same size with a dropout layer with a parameter of 0.5 [20]. Finally, the last component is a fully connected MLP of size *7* which sends out activations into a SoftMax Layer to output the result.

B. GoogLeNet (2015 BN Revision) with PReLU

GoogLeNet [5] was Google's submission to the ImageNet competition in 2014, which also won the competition with an impressive error rate of 6.65%. GoogLeNet is a convolutional type network that uses an advanced architecture layer called Inception Layer; the inception layer takes advantage of the speed of parallel execution to process huge convolutional layers in smaller bits and then concatenate them back together. This approach reduces the number of parameters by a factor of *10* and increases execution speed by a factor of *2*. In 2015, Google also presented a new layer architecture called Batch Normalization, then by making slight changes to GoogLeNet and adding batch normalization [20] it was able to achieve the astonishing 4.82% error rate which surpasses the human performance at 5.01% error rate [21]. GoogLeNet takes advantage of parallelization of multiple smaller convolutional layers in parallel rather than in series. This results in a network with a much smaller amount of parameters with a performance that exceeded all the other networks (for comparison, GoogLeNet has 13.6 M parameters vs. 180 M parameters for the VGG-Net). In our implementation, the ReLU (rectified linear unit) activation type functions used in the original architecture were replaced by parametric ReLU or PReLU in an attempt to improve the performance. The number of parameters in our implementation is approximately $13.6\ x\ 10^6$, therefore, making the file which contains the weights and has to be stored and accessed in real time on the smart phone only 13.6 MB which is not much and is especially important since a mobile app should be compact.

C. Custom Network 1

The first of the two newly proposed architectures was inspired by both the AlexNet and VGG. The Local Response Normalization [20] layer contributions to error rate minimization have been proven in practice to be very low if

Layer Type	Feature Maps	Other Parameters	Activation Function	Patch Size/Stride
Convolutional	128	-	PReLU	7x7/2
Convolutional	128	-	PReLU	3x3/2
Convolutional	128	-	PReLU	3x3/1
Maxpool	-	-	-	3x3/2
Fully Connected MLP	-	512 Neurons	PReLU	-
Dropout	-	Dropout Parameter: 0.5	-	-
Fully Connected MLP	-	512 Neurons	PReLU	-

TABLE 1: PARALAYER DETAILS

any. Therefore, it was decided not to use an LRN layer in the proposed network architecture. In addition, the new PReLU activation function has proven to be better on average than the ReLU type activation function used in the above-mentioned benchmark architectures. In addition, because in this particular application of a general purpose intelligent surveillance system only seven classes are considered (as opposed to the *1000* classes considered by AlexNet and VGG) the fully connected layers were with a significantly reduced size (only *1024x1024x7* as opposed to *4096x4096x1000* used in AlexNet and VGG. This leads to a reduction in the memory usage and speeds up the training and deployment. An increase in the feature maps in convolutional layers *3, 4,* and *5* from *384, 384, 256* to *512, 1024, 512* was experimentally proven to be contributing to improving the accuracy. The proposed network architecture is using much less memory than AlexNet and produces better accuracy results for the problem at hand we have. The larger feature maps provide richer results though can be a reason for over-fitting if the training is not stopped on time.

D. Custom Network 2 – ParaNet (VGG-Inspired)

The second network architecture that is proposed in this paper is *ParaNet*. It draws ideas from a variety of networks; first, the parallelism of GoogLeNet; the smaller receptive fields of *VGG* and the idea of Multi-Task learning [19]. In Multi-Task learning, a network can achieve multiple tasks, through the use of dedicated parallel pipelines where each one of them is responsible for carrying out a specific task. For example, one pipeline can be responsible for image classification; another pipeline for caption generation, etc. In our case, we used separate pipelines for each class allowing the network to have dedicated convolutional and MLP layers for each category/class. Parallel layers were designed with identical architecture and called "ParaLayers" for simplicity. In Table 1 the specific details for each one of those layers are tabulated. The architecture of the network is composed initially of *2* cascading convolutional layers plus a Max-Pooling and then of seven ParaLayers (one for each class, e.g. one for "*human*"; another one for "*cat*", etc).

V. Data Pre-Processing and Augmentation

One of the most important steps in training a model using supervised learning and especially for DL-NN architectures and image processing tasks is the data pre-processing and augmentation. The level of accuracy for the state of the art techniques can be greatly impacted by it, as it has been experimentally shown that 10-15% higher accuracy rates can be reached. In our particular problem, this step includes:

A. Resizing

Each and every one of the images was transformed into *256x256* size using the method of "squashing" or, in other words, resizing without having any considerations about the previous aspect ratio, see Figure 4, for example.

B. Mean Subtraction

For better results subtracting the mean of all the images from each image is recommended, as it prevents saturation and makes the differences between images more apparent.

C. Data Augmentation

Because large neural networks need massive amounts of data, it was decided that applying data augmentation to our dataset was appropriate. By generating new samples based on the existing ones in a way that preserves the objects to be recognized allows the training data set to increase multiple times which leads to a much better-performing network and much higher generalization results. By increasing our data, over-fitting is significantly reduced which allows the accuracy of the network to increase. The following techniques were used to augment the dataset we considered.

D. Random Rotations from 0 to 360 degrees

Rotating the images to random angles ranging from 0 to 360 degrees not only increases the data but also allows for better generalizations of the model.

E. Random Cropping

Randomly crop *224x224* pixel size patches from the original images and add these to the training data set, thus producing new images in each mini-batch.

F. Mirroring

Creating mirrored versions of the images.

VI. Experimental Results and Analysis

A. Model Training and Evaluation

The models were trained using stochastic gradient descent method and error back-propagation. The initial learning rate was 0.1 and the learning rate decay was 0.9. The batch sizes varied with the models (for AlexNet and CustomNet 1 a batch size of *128* was used whilst for GoogLeNet the batch size was *48* and in the case of CustomNet 2 the batch size was *12*). The cross-entropy type loss function was considered (see figure 1). Finally, for each epoch the accuracy of correct predictions was also calculated. The dataset was split into 85% of the data for training and 15% for validation. Naturally, in the validation data set (the 15%) only original images were used, not augmented data. Finally, the system was trained for *10* epochs in order to select the best model out of the *4* and then additional *20* epochs on the best model to allow it to reach a much better score. This took approximately *4* to *12* hours on a laptop depending on the network.

B. Results

In Figure 1 one can clearly see that GoogLeNet is by far the best of all four architectures. However, it is worth noting that CustomNet 1 had very similar performance to GoogLeNet and needed only 3 hours to train compared to GoogLeNet that needed 10 hours to train as shown in Table 2. This is an important factor to consider when choosing an architecture as decreased training time can save both time and resources.

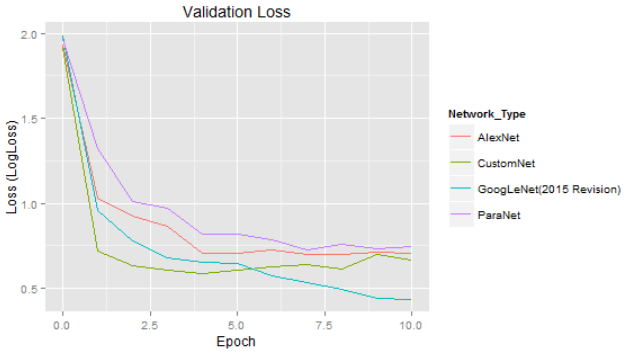

FIGURE 1: LOSS FUNCTION ON THE VALIDATION DATA

For that reason, it was chosen as the architecture to be further trained with the available limited GPU handheld device.

Validation Loss	Validation Accuracy	Network Type	Total Time to needed for training
0.436948	**0.839887**	GoogLeNet (2015 Revision)	10 hours
0.665369	0.81925	CustomNet 1	**3 hours**
0.745423	0.773935	CustomNet 2 - ParaNet	9 hours
0.708225	0.769173	AlexNet	**2 hours**

TABLE 2: LOSS AND ACCURACY OF ALL ARCHITECTURES AFTER 10 EPOCHS

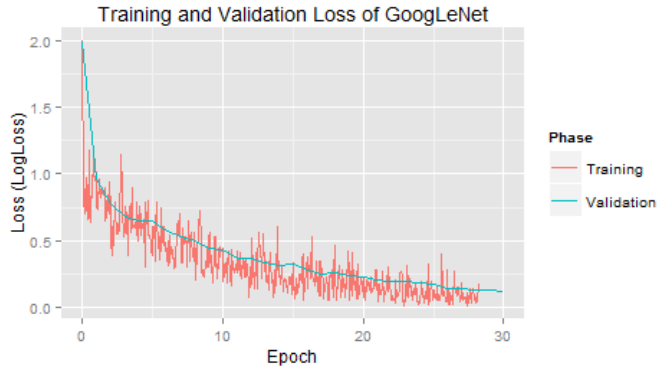

FIGURE 2: TRAINING AND VALIDATION LOSS OF GOOGLENET

In Figure 2 the performance of the GoogLeNet over 30 epochs can be seen. Its final validation loss is 0.122971 which is very small and translates to a very high-performance network.

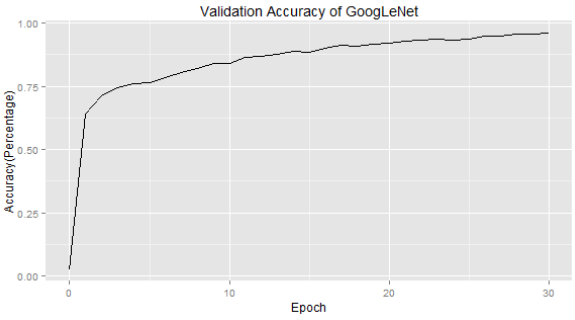

FIGURE 3: GOOGLENET VALIDATION ACCURACY

In Figure 3 one can see the performance in terms of accuracy of the GoogLeNet over 30 epochs. The final Top-1 accuracy is **95.9924%**. In other words, the system can predict the correct label in **its top-1 choice at a level that is on par or better than a human would do it [7], [21]**. This is extremely important for the overall system as it relies on the model being very reliable and accurate in its predictions. Also, the **top-2 error is 99.85%.** In other words by using a top-2 prediction, one can have an extremely high accuracy rate. The table below shows the performance of all the networks. Additionally, given that some images may have *2* or more objects in them, we can deduce that some of the erroneous classifications were, in fact, not real errors.

C. Action/Notification

Finally, after detecting a moving object and classifying it successfully, our proposed system also offers an intelligent action which may be a two-way communication process. For example, the system can enable an alarm autonomously or inform the remote user by SMS, email, etc. Moreover, it can get back the response from the user who may want specific image frames to be sent to him/her remotely or other action (e.g. call security firm/police, relatives etc.) to be taken automatically on his/her behalf. The system is general purpose and as such one can change its functionality from pet detector to theft detector or even car detector (for parking spaces).

FIGURE 4: EXAMPLES OF CLASS HUMAN USED FOR TRAINING

FIGURE 5: EXAMPLES OF TRAINING DATA FOR THE CLASS TREE

D. PC API and Android App Implementation

1) PC Application

After training, fine tuning and selecting the best model the implementation of the PC and Android apps was initiated. First, SARIVA, including Optical ORB, was implemented using the OpenCV API. The programming language for the PC app was Python as its prototyping speed is among the best available. Along with Python, we used the PyCharm IDE, which allowed even quicker iteration speed. After building the moving object detector, classification module was built using the Caffe Python API, which allows quick and easy deployment of models trained in Caffe. Finally, the action/notification module was built using Python. On the PC application, a GUI based solution was not implemented but instead a terminal based system that could quickly and efficiently find moving objects, classify, make a decision and act on it using email and SMS. Because of the speed of the Caffe framework which is written in C++, the system could easily be used on a Raspberry Pi [8].

2) Android Application

The tools used for implementing the Android app were Android Studio with the Android Development Kit (SDK) and Android Development Tools (ADT). Additionally, a "Caffe" port to Android was used that allowed the C++ code of "Caffe" to interface with Android. In addition, the Android Native Development Kit (NDK) was used to combine C++ and Java. Features of the app include a selection of activation labels as well as the availability of email and SMS messages. The object movement detector was first implemented in Python using OpenCV and then directly translated into Java OpenCV API. This was rather simple as the method names are consistent. Special care was taken to drive all module-tasks using multi-threading and, thus, maximize the performance and responsiveness of the app. The classification pipeline was implemented using the Caffe port and the trained model. The model that was already trained on the GPU was copied to the mobile device's hard drive and accessed when the app is launched. The trained DL CNN architecture required 15Mb of memory. Finally, the action/notification module was implemented in Java. The UI was built using the Android Development Tools. The size of the app was additionally reduced by the fact that the OpenCV library is pre-installed on the phone as a separate app that allows sharing of the API with multiple apps. Thus, the app only needed to access the already installed tools. Also, a prompt was added to automatically download OpenCV toolkit in case the user did not have them installed beforehand.

E. Real World Testing

The app was tested module by the module during production, and then system-level tests followed the full implementation. In this sub-section, only some of the tests carried out are presented.

1) Object Classification Tests in Real-Time:

In Figures 6 and seven we can see some real-time classifications that took place as we moved with the smart phone and the App enabled.

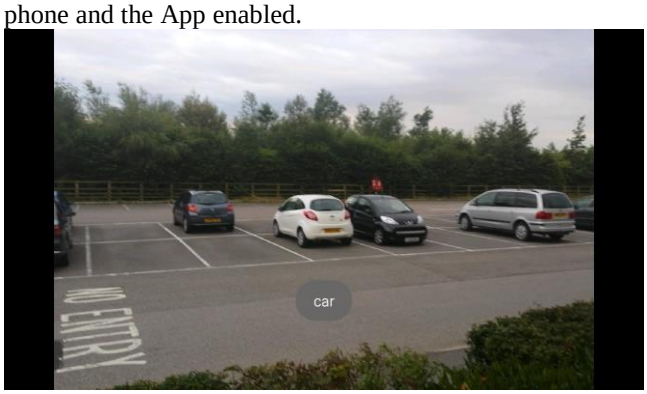

FIGURE 6: CLASSIFICATION OF CARS

FIGURE 7: CLASSIFICATION OF TREES

F. Movement Detection:

The movement detector was tested, and results showed adequate performance, see Figure 8.

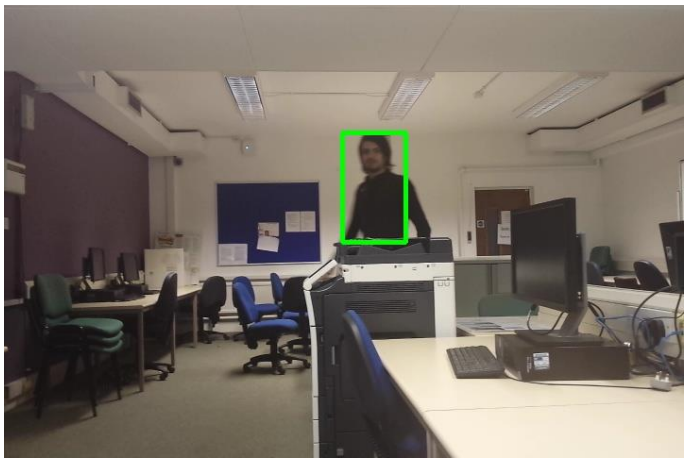

FIGURE 8 ANDROID MOVEMENT DETECTION

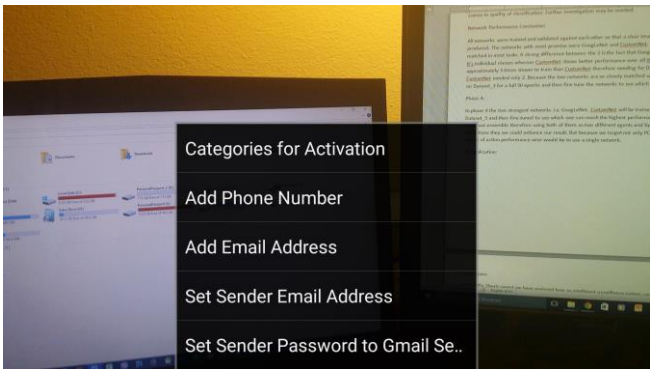

FIGURE 9: "MORE" MENU

Allows further options from the main menu

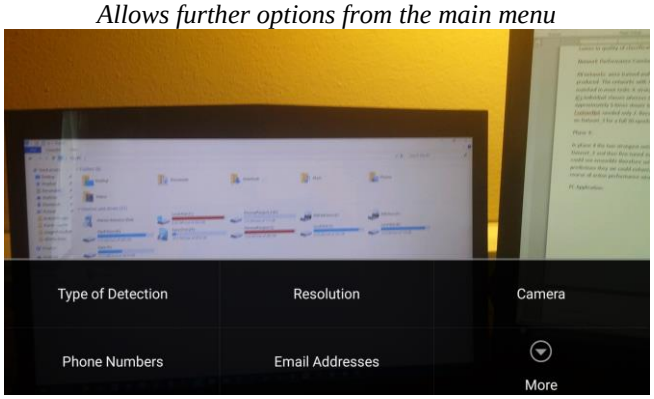

FIGURE 10: MAIN MENU

Allows selection of detection type, resolution, phone numbers, emails and more

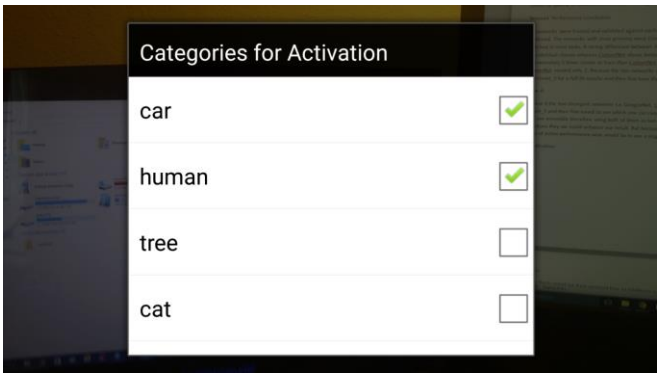

FIGURE 11: ACTIVATION MENU

Allows the selection of which objects should trigger a response (SMS, email, alarm) from the system.

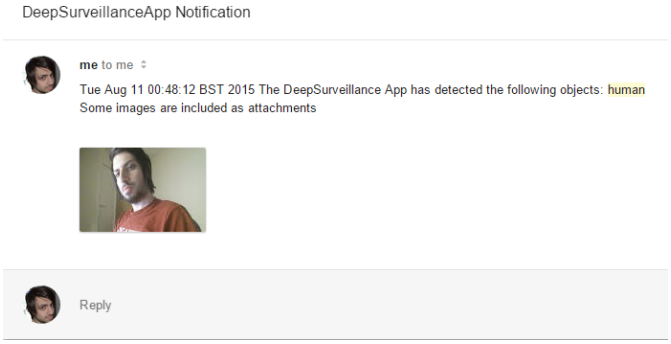

FIGURE 12: EMAIL SENT WITH OBJECT CATEGORY AND PICTURE

VII. CONCLUSION AND FURTHER WORK

A. *Conclusion*

In this paper, we present and analyze a novel general purpose intelligent video surveillance system portable on a smart phone. The procedure of the design and training a deep neural network capable of 95.9% accuracy in the top-1 category over seven categories is described. In addition, we proposed two new custom architectures capable of performance comparable to the best-known state of the art. Additionally, it is worth noting that CustomNet 1 had a very fast training of only 3 hours compared to GoogLeNet's 10 hours. In addition the GoogLeNet trained used PReLU instead of ReLU as the original architecture did. Furthermore, we present and outline the design and implementation of both a PC and Android application that can be used for the intelligent surveillance systems. We have demonstrated that deep neural networks can be used for classification on Android devices with high frame rate and human-like performance. The PC app can also be used on Raspberry Pi and other Linux-based embedded devices. Future applications could include real-time learning as well as classification through the use of combined unsupervised and supervised learning given that the mobile devices of the future have powerful GPUs like the Tegra X1 [9].

B. *Future Work:*

The system presented in this paper is only capable of using a pre-trained model for its classification. The truly game-changing capability is to allow a mobile device to learn as it collects samples of images. This is currently only possible through the use of Cloud GPU servers and combination of unsupervised learning with deep neural networks. As mobile devices become increasingly more powerful, we believe that GPUs such as Tegra X1 will become part of the mainstream smartphone hardware. Further ways to achieve learning in real time is the use of reinforcement learning in combination with a deep neural network [7]. In addition online learning can also be implemented through the use of self-learning and dynamically evolving systems [22] which, however, for the case of Deep Neural Networks are not yet developed. Further studies will be directed to this. When that is achieved new more powerful systems will be able to learn on the device itself without the need of the Internet or wireless communication.

REFERENCES

[1] K. Ball, D. Lyon, D. M. Wood, C. Norris, C. Raab, A report on the Surveillance Scoiety, Sept. 2006, available online at http://news.bbc.co.uk/1/hi/uk/6108496.stm, accessed on 18 January 2016.

[2] C. Clarke, P. Angelov, Y. Majid, P. Sadeghi-Tehran, SARIVA: Smartphone App for Real-time Intelligent Video Analytics, *Journal of Automation, Mobile Robotics and Intelligent Systems*, vol.8 (4), 2014, pp.15-19.

[3] WhatMovesApp, https://play.google.com/store/apps/details?id=lancs.free, accessed on 18 January 2016.

[4] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Sep. 2014.

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," Sep. 2014.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[8] Raspberry-Pi 2 [Online] https://www.adafruit.com/pdfs/raspberrypi2modelb.pdf

[9] "Tegra X1 Super Chip | NVIDIA Tegra | NVIDIA." [Online]. Available: http://www.nvidia.com/object/tegra-x1-processor.html. [Accessed: 11-Aug-2015].

[10] P. Angelov, C. Gude, P. Sadeghi-Tehran, and T. Ivanov, "ARTOT: Autonomous real-time object detection and tracking by a moving camera," in *2012 6th IEEE International Conference on Intelligent Systems*, pp. 446–452.

[11] J. Y. Bouguet, Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm, Intel Corp. Microprocessor Research Labs, 1999.

[12] R. D. Baruah, P. Angelov, Evolving Local Means Method for Clustering of Streaming Data, *Proc. IEEE World Congress on Computational Intelligence*, 2012, Brisbane, Australia, pp. 2161-2168.

[13] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Oct. 2014.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe," in *Proceedings of the ACM International Conference on Multimedia - MM '14*, 2014, pp. 675–678.

[15] NVIDIA DIGITS – Interactive Deep Learning GPU Training System NVIDIA Developer, available online at https://developer.nvidia.com/digits, accessed 18 January 2016.

[16] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," Feb. 2014.

[17] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig, and Y. Shi, "Spoken language understanding using long short-term memory neural networks," in *2014 IEEE Spoken Language Technology Workshop (SLT)*, 2014, pp. 189–194.

[18] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko, "Translating Videos to Natural Language Using Deep Recurrent Neural Networks," Dec. 2014.

[19] B. Yu and I. Lane, "Multi-task deep learning for image understanding," in *2014 6th International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, 2014, pp. 37–42.

[20] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Feb. 2015.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," Feb. 2015.

[22] P. Angelov, Autonomous Learning Systems: From Data Streams to Knowledge in Real time, John Willey and Sons, Dec.2012, ISBN: 978-1-1199-5152-0.