Real-Time Hand Pose Detection as a Computer Interface

Mukesh Yadav, ID: 19069792, Bogdan Secara, ID: 20108933

Students of Computer and Information Sciences at Auckland University of Technology

***Abstract*—Hands play an important role on how we interact today with computers. As intelligent devices become ubiquitous, humans need to find better ways to interact with and to control computers, outside the traditional hand touching interfaces such as keyboard or interactive screens. The paper explores practical considerations, such as the artificial intelligence model selection, the training challenges, including the prediction model into a real-life application, tunning the hyperparameters, for a real-time hand detection solution to control computer actions. To demonstrate the viability of the solutions proposed, a prototype was designed, built and tested, in the form of a web-based game in which the user controls the game player through the real-time hand images captured by the computer webcam. The outcomes from this paper may be applied to limit the effects of a pandemic transmission, as interacting with a device though touching could be replaced by real-time hand pose detection even in less sophisticated computers.**

***Index Terms*—artificial intelligence (AI), browser game, deep learning, Ego Hands, exergame, human computer interaction (HCI), object detection, PhaserJS, real-time image classification, MobileNet, TensorFlow, transfer learning.**

I. Introduction

THE hand has been the first tool ever used by humans, which has helped us across the millennia to build other tools which helped us to learn and to evolve. And hands still continue to play an important role in humans' daily lives, as one of the today's most common modern tools are the computers, which continue to be operated mainly through the human hands.

As hand actions are a rather slow communication interface between our thoughts and the rapidly evolving intelligent machines, other interfaces might replace typing of the computer keyboard. Artificial Intelligence (AI) enables us today to use sound or images to extract more meaning from sources which were until recently, understood only by us, humans. Computer vocal interfaces are a reality today and new venues for using images, including real time video streams might soon enhance the way we interact with the world.

Applications such as detecting human pose in real time could be used as a way to limit COVID-19 effects. Detecting handshakes, or using hand pose as a human computer interface in those public places might prevent the spread of the disease, by avoiding touching surfaces compromised by viruses. Other applications may include promoting physical activities, through games played on regular computers, which would require users to control game players through hands movement. This is why finding today practical ways to implement a real-time hand pose detection, even in the ordinary simple computers, could help us making a better world.

The next sections introduce the problem definition, followed by the approach taken in this paper, details of the hand feature computation, the experiments performed, discussions, conclusion and future work. In the Appendix are listed the link to the code developed and the various libraries with information of the specific software versions which were used during development and experiments.

II. Problem definition

There are several ways to perform a hand pose detection which have been researched and developed. Among them are specialized equipment which could be wear such as PowerGlove [1], dedicated cameras for detecting and fingers pose such as Leap Motion [2], or even specialized body poses detection devices such as Microsoft Kinect [3]. There are also proprietary software packages which have been developed

Our aim in this paper is to explore how a regular, readily available computer camera could be turned into a hand pose detection device, through the advancements in deep learning neural networks, as a way to reduce the customer costs by repurposing existent devices for more advanced tasks.

The main drawback for the real-time hand detection through images is related to the complexity of performing numerous computations in a very short period of time, in order to keep up with the change of an image from one frame to another. Including a hand pose detection inside another application aggravated the computational cost, by the preprocessing as well as the post processing which tax even more the resources needed from the device to allocate for the real-time image detection processes.

On one hand, deep neural networks are well suited to solve intricate problems such as hand pose detection, but on the other hand, they are also complex, as they require abundant computational resources, and sometimes specialized processors such as graphical Processing Units (GPUs). As a consequence, deep neural networks are slow compared to specialized tracking-only algorithms. Neural networks also require large labelled datasets, and data preprocessing which

are quite expensive to assemble and to compute.

To find answers for the above challenges, authors formulated three questions:

Research Question RQ1. Could a standard, CPU only computer be controlled through images of hand gesture using a deep neural network?

Research Question RQ2. Which strategy could be used to compensate for not having an adequate hand detection training / testing dataset?

Research Question RQ3. Which model and architecture to use in a real-life application, which should include a real-time hand pose detection on a small computing device?

In this paper authors propose a lightweight neural network approach which is designed to be implemented with the specifics of a real-life context, on low computing power devices with the goal of extracting dynamic hand real-time information. To demonstrate the viability of their research, authors conducted experiments which involved building a functional prototype of hand-detection.

At the core, the problem to solve is a combination of resulted from:

1. object detection, detecting a hand in an image:
   1.1. a list of bounding boxes, surrounding each hand in an image
   1.2. the hand label associated with each bounding box
   1.3. the confidence score associated with each bounding box
2. feature extraction, analyzing only the detected image
3. correctly classifying the poses of the hand

To tackle the issue of performing hands detection from images sourced by a live video stream *S*, individual images which compose the video sequence must be analyzed, frame by frame. The density of the images is given by the framerate, measured in number of frames produced every second. The higher the framerate, more images are needed to be processed. The framerate for webcams usually can be set up, the default setting for older devices being 30, which translates into a feed of 30 images for every second. Another characteristic which impacts the computer processing needs is the resolution of the images. The higher image resolution is, meaning the number of pixels, given by the image height *h* and image width *w*, the more computing power is needed for image processing. For a color image, there are three layers of pixels to be processed. Therefore, for a stream *St*, of a duration *t* in seconds we have a number of *n* images where:

$$S_t = (I_1, I_2, \ldots, I_x, \ldots, I_n) \text{ where } I_x \in \mathbb{R}^{h \times w \times 3} \text{ and } n = 30 \times t$$

Each of the intensity in the three layers for every pixel requires, for every image, a computation. That is why with larger resolutions, faster framerates, there are more computations needed, and therefore more computing power is required. As the problem of controlling a computer interface using images of the hand would require analyzing a sequence of events, the inference time for the model should be at least 0.03 per second, therefore model inference time was identified as a critical factor. Adding time needed for image preprocessing and post model processing would add important delays which could drive the controlling of the computer interface not to be as responsive as current interfaces, such as touchscreens or keyboards.

Another challenge is the lack of hand images trained models. While authors identified what most models are trained with general images such as COCO, ImageNet or Pascal VOC, no evidence of an existent hand image trained model could have been retrieved.

Finally, building a prototype to assess the implementation of a deep neural network on a low computing power device presents its own challenges, both related to software development, but also in correctly measuring the effects on the experiments performed.

III. Related work and Approach

Accuracy is one of the main success indicators for any prediction model. The authors searched for the state-of-the-art deep learning neural network which provided the best accuracy. Convolutional neural networks (CNN) have been developed and successfully used in real-time object detection [4]. Among several factors which drive the fast evolution of object detection, the most notable is rapid development of computing power through the use of graphic profession units (GPUs). In the absence of the powerful GPUs, the state-of-the-art prediction models have little applications for analyzing real-time video content [5], as the inference time for VGG16 on a CPU only machine is of one prediction every 3.6 seconds, which is 100 times slower than what is required for a 30 frames per second camera, as noted in Table 1. The more parameters the neural network has, the longer it takes to make a prediction, given all the other remain unchanged. Besides making slow but accurate predictions, authors considered the size of the model when deployed on an embedded device, as VGG16 might take half Giga Bytes just for itself, without any other applications or APIs.

Table 1. CNN state-of-the-art-image performance on CPU enabled devices: accuracy (mAP), inference time (s), number of parameters, model size

Model	mAP	Inference(s)	Parameters	Model size
VGG16	93.75	3.6	138 million	528 MBytes
ResNet50	99.05	2.1	26 million	102 Mbytes

Given the challenges identified in the Problem Definition, an ideal object detection model is accurate and fast, and lean in order to run on a CPU only enabled device. Authors investigated other deep learning models which produce similar accuracies, but much faster than the VGG16.

A trade-off should be obtained, between the accuracy of the model, the speed it takes for making a prediction on a certain device, the energy consumed for making the prediction. A model suitable to run on a powerful gaming, GPU enable machine is not a good option for a mobile, or even for smaller devices, as it takes out storage space and, even more, it quickly drains the battery. A series of attempts were made to develop architectures of reduced size, [6, 7], and in the last three years more than 16 light models were assembled [8].

A number of literature reviews [8, 9], were consulted on how different models performed given implementation on CPU only devices, in order to get an answer for Research Question 1. With the aim to implement a model which is fast and efficient, here are the potential candidates:

I. MobileNet is using less resources by replacing replace

computational expensive spatial convolutions with depthwise convolutions

II. TinyYOLO, with DarkNet19 architecture, is definitely slower than MobileNet on CPU only enabled devices, but trumps MobileNet on GPU enabled devices, such as iPhone X

III. FBNet-C work great on neural architecture search, implemented in mobile devices like iPhone XS, Samsung S8, or newer.

Authors gathered evidence from literature review on the performance of light deep learning neural networks. Table 2 contains the mean Average Precision (mAP) for the top prediction, inference time to predict the class of an image, measured in seconds, number of neural network parameters, and the size of the model for the top three performing models, all assessed on the ImageNet dataset:

Table 2. Light CNN models on CPU enabled devices: accuracy (mAP), inference time (s), number of parameters, model size

Model	mAP	Inference (s)	Parameters	Model size
MobileNet	70.6	0.015	4.3 million	17 MBytes
TinyYolo	74.1	0.039	8.6 million	5.3 MBytes
FBNet-C	74.9	0.029	5.5 million	9.3 MBytes

As all the other computer applications perform better only if GPUs or if the CPU has a special neural network architecture built in, authors selected MobileNet model to be implemented. The model selected was MobileNet as it runs well on CPU only enabled devices, with a mean average precision mAP of 72% for the top prediction, inference time of 0.015 sec for an image, resulting a maximum push for the stream at 66 framerate.

The MobileNet v1 comes trained with 224x244, 160x160 or 128x128 ImageNet datasets. ImageNet a large dataset containing over 14 million images [10], each image being labelled by multiple words, or phrases, called synsets. Although there are 6,978 synsets related to persons, there are no synsets indicating hands, even less classifying hand poses or hand gestures. That is why the MobileNet model, loaded with the weights obtained from training ImageNet dataset will identify correctly different classes: humans, birds, flowers, however it will not identify hands.

In order to cope with the limitation, identified as Research Question 2, authors decided to expose MobileNet to a dedicated hands image dataset. Two alternatives were considered on the strategy:

a) training from scratch MobileNet with the images from a hands image dataset, or

b) using transfer learning to tweak already trained ImageNet weights, thus enhancing the existent model with hand images.

As the number of images in the specialized hand pose datasets are 1,000 times smaller than the one contained in ImageNet, authors considered using transfer learning as their prime option. In transfer learning, a neural network which is already trained on a particular task and gain knowledge transfer its knowledge to domain, without the need of retraining a new model from the starch. Inductive transfer learning [11] that is used for transfer knowledge between same domains, even though task could be different such as, images classification as source task and hand pose classification as target task. More details on the specifics of how transfer learning was achieved are details in the experiments section.

In the next section papers explores the architecture used to tweak the original MobileNet model, by adding a classifier in order to distinguish among images which may be learned by the model, on the fly.

IV. Hand Feature Computation

As most of other models, MobileNet is mainly an image classifier. However, in this task, we are going to use the model as object detector, for pinpointing the hand different poses.

MobileNet is an architecture based on depthwise separable convolutions to construct a lighter deep convolutional network. Depthwise separable convolution filters are replacing the traditional convolution filters with depthwise convolution filters and 1x1 pointwise convolution filters. In a regular convolution network, inputs and filters are combined into output in a single step, however, in depthwise convolution, it is done in two steps, first layer finds filters on input and then second layer combines result of filters into the output. The differences between the types of convolutions, are pictured in Fig. 1, below.

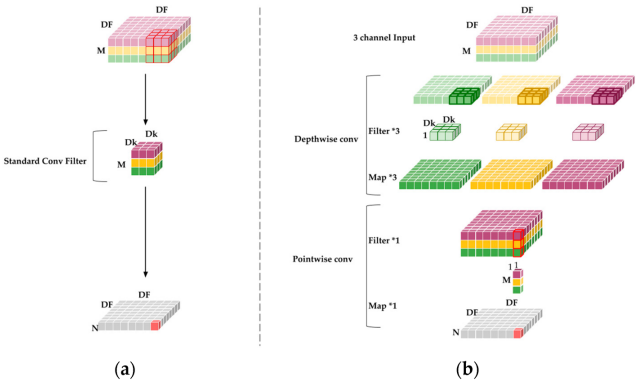

Fig 1. Comparison between standard convolution diagram (a) and the depthwise and pointwise convolution of MobileNet (b). [12]

In order to assess the computational costs, main parameters are names as D_F representing the input picture side length, D_K the convolution kernel side length, M representing the input channels, N the output channels, K the convolution kernel, and F the feature map. The depthwise separable convolutions are:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

Total computation depthwise cost is:

$$\frac{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

which is $\frac{1}{N} + \frac{1}{D_K^2}$ smaller than a standard similar convolutional model, thus making this model having less latency, and requiring less computational power. Still, there are 4.3 million parameters which make up a 28 layers architecture, as pictured in Fig. 2.

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
5× Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

Fig. 2. MobileNet architecture with 28 layers [13].

Image features may be extracted through HOG or SIFT; however, these are designed manually. Knowing that in this classification model there are only two classes: up and down, K-Nearest Neighbors (KNN) algorithm was used, for classification. KNN is a supervised machine learning algorithm, used to solve both classification and regression problems. The reason why KNN was chosen is that it produces robust classification, especially for different images, such as an open palm versus a closed fist. KNN algorithm looks for K centroids points that minimizes total distances from each point in the space.

V. Experiments

A. Transfer Learning

An important step in performing the transfer learning was to identify and to select the annotated hand image dataset. As with any other deep learning neural networks, the larger the dataset, the better, as it offers the model to be trained and tested the opportunity to encounter images of hands captures on different background, from different angles, and in different light conditions.

Authors considered the Oxford Hands Dataset, containing 11,194 labeled hand images [14], and the Egohands Dataset, containing 15,053 ground truth labeled hands [15]. As suggested by [16], MobileNet training was done in several steps:

a. downloading the Egohands dataset
b. splitting dataset into train test partitions
c. convert files into TensorFlow tfrecods
d. train MobileNet starting with the pre-train weights
e. test the accuracy, loss
f. assess the mean Average Precision (mAP) every 50,000 iterations

Compared to the initial MobileNet mAP of 70.6%, the mAP obtained after 150,000 epochs was of 92.3% which is an important increase in the model performance.

B. Game Implementation

In order to check how the model performs in real-life, a web-based demo game was implemented for testing how hand poses classification makes predictions in real-time.

The authors imagined a game in which the user should be able to control the vertical position of the game player, through the input of two different images which, upon successful detection, should act as up (↑) and down (↓) commands of the game player. Even more, the user should be able to customize these images by uploading his own palm pose images, to act either as an up or down command.

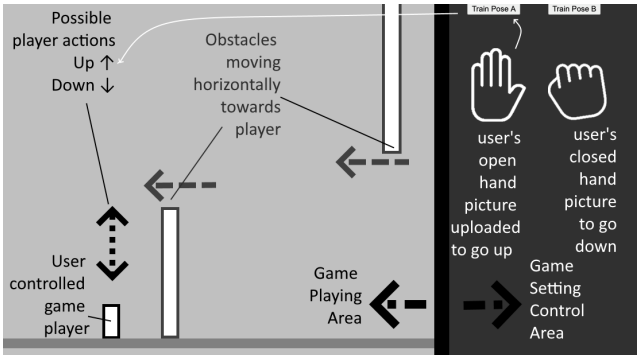

Fig. 3. Demo Game Interface for MobileNet

As pictured in Fig. 3, the demo game is a simple arcade game, in which the game player is presented with obstacles towing towards it, and the aim of the game is for the game player to avoid the horizontally moving obstacles. The user controlling the game player can move the player up and down, however instead of pressing the button up (↑) on a keyboard, the user shows computer webcam an open palm, and if the model successfully classifies the hand palm as the image assigned to be the up command, then the game player moves up. In a similar way, if the user presents the computer webcam a closed hand, as a fist, and if the model correctly classifies the hand as a closed hand, the game player moves down. The game screen is divided in two areas: the left part of the screen is where the game takes place, while in the right hand side, is the game control setting panels, in which the user may be able to assign, before the game begins, his own image for the up button, in "Train Pose A", and the customized button for "Train Pose B" for the down movement.

PhaserJS [17] was used as a game engine library for rendering the graphics and Arcade Physics Engine [18] for implementing physics in the game and collision among objects. As with all browser-based games, HTML, CSS and JavaScript code were written for the design of game, and its source is available in the URL mentioned in Appendix 1.

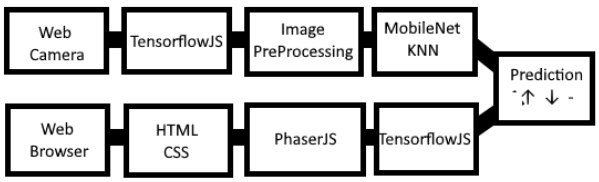

Fig 4. The web-based game architecture

The code for TensorFlowJS [19, 20, 21] can be accessed from Appendix 4. As shown in Fig. 6, there are two buttons labelled "Train Pose A", for the up command, and "Train Pose B" for down command. On pressing the any of the buttons captures image from the webcam the image corresponding to label of that particular button, which is fed into the MobileNet model [22] to get intermediate activation of the model, in the form of a probabilistic value of the object predicted in the image. The code for getting used for activation is:

net.infer(img, 'conv_preds')

where "img" is the image that is fed to the model and "conv_preds" is the parameter to get the probabilistic value. KNN classifier to classify hand pose images, therefore, an image is then fed into this classifier get labelled. For the KNN, the one-shot learning approach was applied. Using this method, the model learns how to classify an image, after being exposed to it only once. The code for this is:

classifier.addExample(activation, classId)

where classifier is the instance of algorithm, activation is the probabilistic value from the MobileNet model [23] and classId is the label for the hand pose and code for KNN can be accessed form Appendix 3.

After that, a game loop waits for an image to come from video stream and it is passed to the MobileNet model and get its activation value which is then passed to KNN classifier to get its class. With this class, gameplay of the game is controlled, through the code:

```
const activation = net.infer(img,'conv_preds');
const result = await
classifier.predictClass(activation);
```

To run a demo game whose source code can be found from appendix 4, the file should be served from an http server.

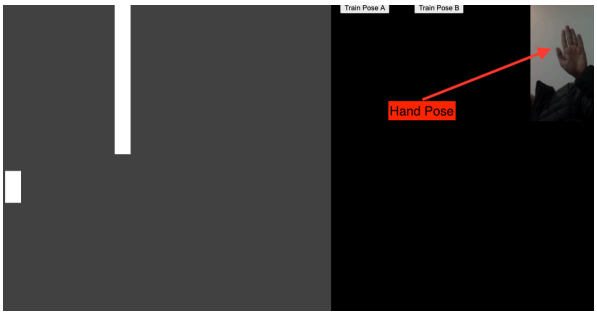

Fig. 5. Screen shot taken during one of the authors playing the game, Hand Pose Classification using MobileNet

After all files are loaded as shown in figure 5, a game can be played after training the model. After training the MobileNet model, demo game ran at the speed of 58 frames per second on MacBook 2.3 Hz core i7 with 16GB RAM.

VI. Discussion

The mean Average Accuracy achieved at 30 frames per seconds is of 92.3%. For controlling a web-based game, or a computer interface, the accuracy would be enough for most of the tasks. However, the applicability might be limited in the case of analyzing real-time fast-moving hand gestures such as a martial arts strike. Also, not having 100% accuracy would limit the usage of this model in scenarios which a mislabeling might have a huge impact, such as performing robotic brain surgeries using only images from the surgeon hands. Authors noted in their experiments that some features are often not distinctive enough to differentiate between similar hand postures. Therefore, experimenting with several other classifiers such as support vector machines, multiple decision trees, and finally using ensemble methods to decide which classifier would most likely produce better results.

VII. Conclusions

The high computationally intensive deep learning models may be successfully replaced by the use of a lighter, depthwise convolution network like MobileNet, to classify hand poses on CPU only reliant devices, such as older laptops, mobiles or even small computing embedded devices. MobileNet was successfully trained in hand pose detection mainly due to transfer learning from EgoHands dataset, which increased the mean Average Precision from 70.6% to 92.3%. Furthermore, the ability to quickly make predictions and the small model size, allows the identified solution to be deployed in applications, such as web-based games, in video streams with above the 30 frames per second commonly used framerate, while running without glitches.

VIII. Future Work

The encouraging results from this paper might crosspollinate into other areas of interest, apart from computer interfaces. Think how the gestures of a car driver may be supervised by a computer system in order to prevent human errors, or how sign language, which is in many cases country specific, might open to automatic translation. Further investigation should be done by deploying Ego Hands image data on other light deep learning frameworks such as TinyYolo or FBNet-C, to observe which ones are more accurate, produce faster predictions on specific devices, such as mobile phones and small single-board computers such as Raspberry Pi. Also, to assess the impact of the transfer learning, other hand image datasets such as Oxford Hands Dataset might be used to re-tune the model. Another element to investigate is tuning the model hyperparameters, when the device CPU is heavily stress with other computing processes, which could be done by adding further gameplay elements and complexity to the existing demo. Authors used only two hand poses to control the computer interface, therefore a more sophisticated hand gesture library would require more investigation.

Appendix

1. *Code for the interface design:* https://github.com/mukeshyadav-cdac/mobileNetDemoGame/blob/main/exergame.js and https://github.com/mukeshyadav-cdac/mobileNetDemoGame/blob/main/index.html
2. *Code for pre-trained MobileNet:* https://cdn.jsdelivr.net/npm/@tensorflow-models/mobilenet
3. *Code for KNN algorithm:* https://cdn.jsdelivr.net/npm/@tensorflow-models/knn-classifier
4. *Source code of the demo game:* https://github.com/mukeshyadav-cdac/mobileNetDemoGame
5. *Code for TensorFlowJS:* https://cdn.jsdelivr.net/npm/@tensorflow/tfjs
6. *Code for PhaserJS library:* https://github.com/mukeshyadav-cdac/mobileNetDemoGame/blob/main/phaser.min.js
7. *Instructions for downloading the tensorflow EgoHands tfrecods:* https://github.com/beliverable/egoHandsTF
8. *Software and libraries versions used during development and model implementation: PhaserJS version:* 3.0.0, TensorFlowJS version: 2.4.0, MobileNet JavaScript port version: 2.0.5, KNN JavaScript port version: 1.2.2.

Acknowledgment

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. All the software tools involved in this paper are open source.

REFERENCES

[1] Kadous, M. (1996). Machine Recognition of Auslan Signs Using PowerGloves: Towards Large-Lexicon Recognition of Sign Language. Proceedings of the Workshop on the Integration of Gesture in Language and Speech, (165-174).

[2] Chophuk, P., Pattanaworapan, K., Chamnongthai, K. (2018). Fist american sign language recognition using leap motion sensor. In Proceedings of the 2018 International Workshop on Advanced Image Technology (IWAIT), Chiang Mai, Thailand, 7–10 January 2018; pp. 1–4.

[3] Dong, C., Leu, M.C., Yin, Z. (2015). American Sign Language alphabet recognition using Microsoft Kinect. IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), (44-52). https://doi.org/10.1109/CVPRW.2015.7301347

[4] Jiao, L., Zhang, F., Liu, F. Yang, S., Li, L., Feng, Z., Qu, R. (2019). A Survey of Deep Learning-Based Object Detection. IEEE Access, vol. 7, (128837-128868). https://doi.org/10.1109/ACCESS.2019.2939201

[5] Kaseb, A.S., Fu, B., Mohan, A., Lu, Y., Reibman, A., Thiruvathukal, G.K. (2018). Analyzing Real-Time Multimedia Content from Network Cameras Using CPUs and GPUs in the Cloud. 2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), (69-74). https://doi.org/10.1109/MIPR.2018.00020

[6] Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J. (2016) Quantized Convolutional Neural Networks for Mobile Devices. http://arxiv.org/abs/1512.06473.

[7] Wang, M., Liu, B., Foroosh, H. (2017). Factorized Convolutional Neural Networks. 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), Venice, 2017, pp. 545-553, doi: 10.1109/ICCVW.2017.71

[8] Hollemans, M., (2020). New mobile neural network architectures. Published on MachineThink. Retrieved 30 Sept, 2020. https://machinethink.net/blog/mobile-architectures/

[9] Deng, Y., (2019). Deep Learning on Mobile Devices - A Review. ArXiv, abs/1904.09274, Retrieved 30 Sept, 2020. https://arxiv.org/pdf/1904.09274.pdf

[10] Bambach, A., Sven, L. (2015). Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions. Proceedings of the IEEE International Conference on Computer Vision. Retrieved from: http://vision.soic.indiana.edu/projects/egohands/

[11] Chen, L.C., Sheu, R.K.,Peng, W.Y., Wu, J.H., Tseng, C.H.(2020). Video-Based Parking Occupancy Detection for Smart Control System, Applied Sciences Vol. 10, Issue 3, (1079-1088). https://doi.org/10.3390/app10031079

[12] Sarkar, D. (2018). A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning. Medium, Nov. 17, 2018. https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[13] Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. ArXiv, abs/1704.04861

[14] Mittal, A., Zisserman, A., Torr, P. (2011). Hand detection using multiple proposals. British Machine Vision Conference. http://www.robots.ox.ac.uk/~vgg/data/hands/

[15] Bambach, A., Sven, L. (2015). Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions. Proceedings of the IEEE International Conference on Computer Vision. Retrieved from: http://vision.soic.indiana.edu/projects/egohands/

[16] Dibia, V. (2017). How to Build a Real-time Hand-Detector. Retrieved from: https://medium.com/how-to-build-a-real-time-hand-detector-using-neural-networks-ssd-on-tensorflow-d6bac0e4b2ce

[17] Phaser - A fast, fun and free open source HTML5 game framework. Retrieved 01 Oct 2020 from h https://phaser.io/

[18] Github. Phaser 3 API Documentation - Class: ArcadePhysics. Retrieved 02 Oct, 2020 https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Arcade.ArcadePhysics.html

[19] TensorFlow website. Retrieved 30 Sept, 2020. https://www.tensorflow.org/

[20] TensorFlow website. TensorFlow.js, Machine Learning for Javascript Developers. Retrieved 30 Sept, 2020. https://www.tensorflow.org/js

[21] Github website - tensorflow/tfjs-models. Retrieved 30 Sept, 2020. https://github.com/tensorflow/tfjs-models

[22] Google Artificial Intelligence website. MobileNets: Open-Source Models for Efficient On-Device Vision. Retrieved 21 Sept, 2020. https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html

[23] Github website. The MobileNet v1 Image Dataset – MobileNet. Retrieved 28 Sept, 2020. https://github.com/tensorflow/models/blob/master/research/slim/README.md