Real-Time Hand Gesture to Emoji Recognition

Mr.Liam Bugeja Douglas
Department of Artificial Intelligence
University of Malta
+356 7944 4155
liam.bugeja-douglas.20@um.edu.mt

Mr.Kristian Guillaumier
Department of Artificial Intelligence
University of Malta
+356 2340 2542
kristian.guillaumier@um.edu.mt

1.Overview

1.1 Problem

In this practical task the problem was to create a model to detect hand gestures and correspond an emoji to them in real time. The model would be able to detect up to eight different types of hand gestures (fist, okay, peace, point, point sideways, rock, stop, thumbs up). This project required to build a model from scratch by defining a custom-made neural network and train it using a dataset of images. Finally, the model would be able to display the corresponding emoji, when it detects a hand gesture using a webcam or video and it must be able to give a prediction as fast as possible.

2. Literature Review

The intention of this literature review is to better understand the methodology of creating a CNN for real time hand gesture to emoji recognition. Whilst other projects mainly use pre-trained models or a modified version of them, this would still help to better understand what is required to build a CNN for hand gesture detection.

In research paper [1] the goal of the researchers was to advance human interaction with smart appliances by capturing hand gestures and identifying them to allow communication between the two. To allow this the researchers first captured images through a webcam and segmented the hand from the rest of the frame by using skin colour detection, this will be known as the region of interest (ROI). Skin colour detection is the process of separating the skin with the non-skin area of the image, this is done by analysing the pixel values of the image.

However, in [2] the authors discuss that it is difficult to have a uniform method to segment human skin with the rest of the frame due to variance in different skin pigment. It concluded that using the YCbCr colour space will give the best results. Furthermore, to remove background noise from the ROI the authors used a kernelized correlation filter (KCF) algorithm to be able to track the ROI in real time. KCFs are correlation filters which have improved computation speed by using kernel tricks and circulant matrices as described in [3].

Afterwards, the ROI is resized before entering the CNN. In this case the authors decided to resize the ROI to a value of 100 by 120. The researchers in this study developed two different CNN architectures which are a modified version of two popular models, AlexNet and VGGNet. AlexNet is a deep CNN mainly designed by Alex Krizhevsky which contains 60 million parameters and 500,000 neurons. It contains five convolution layers which are followed with some max-pooling layers and two final layers [4]. The CNN was trained on the ImageNet [5] which is a dataset containing around 1.3 million images with 1000 different classes. VGGNet developed by Karen Simonyan and Andrew Zisserman is a CNN which has a total of 19 layers and is one of the best architectures due to its ability to generalise well with different datasets [6]. This CNN was also trained on the ImageNet [5] dataset.

The training data in this research was able to reach a recognition rate of 99.9% and the for the test data the model was able to reach a recognition rate of 95.6%. Finally, the process of tracking and recognition is repeated in real time and stops when the hand leaves the range of the webcam.

Input 100x120x3
Conv1(5x5x32)stride1/ReLU
Maxpool1(2x2)stride2
Norm1(LRN)
Conv2(3x3x64)stride1/ReLU
Maxpool2(2x2)stride2
Norm2(LRN)
Conv3(3x3x64)stride1/ReLU
Maxpool3(2x2)stride2
Norm3(LRN)
Conv4(3x3x128)stride1/ReLU
Maxpool4(2x2)stride2
Norm4(LRN)
Dropout
Full1(1024)
Full2(1024)
Output (6 categories)

Fig.1 [1]

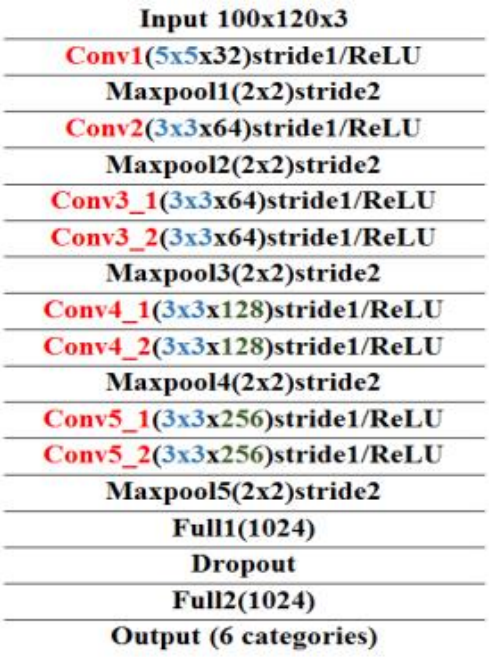


Fig.2 [1]

Above are the two architectures used in paper [1], from them I implemented some of the ideas to my model. The first one is that after each convolution layer I added a max-pooling layer, this was done since this layer outputs a feature map containing the most prominent features found from the images. Secondly, I changed the size of the input image, when I built the first model the input images where 500x500 and the training was taking a long time to process. From this study I concluded that having smaller images will not affect the results of the prediction and therefore opted to use a 254x254 size. Finally, before the output layer both architectures added two dense layers. After some research I found that adding a few dense layers helped the model to classify which features are relevant to the class. Finally, after implementing all of these techniques used in research paper [1] the validation accuracy was significantly higher.

3. Design of Solution

3.1 Environment

To be able to build the model, create the dataset and capture video the following software was needed:

1. TensorFlow
2. OpenCV
3. Keras
4. CUDA Toolkit
5. NVIDIA cuDNN

3.2 TensorFlow

TensorFlow is a free and open-source library which is built for machine learning and artificial intelligence. It offers multiple functions to be able to build your own neural network and save it, also the TensorFlow community offers multiple pre-trained models which can used for free. The most popular pre-trained models are created by Model Garden and each model has a different use such as lightweight models for mobile applications or fast models which can be used in autonomous driving.

3.3 OpenCV

OpenCV is a free and open-source library which is mainly built for real time computer vision. It offers a variety of pre-trained models for object detection however, in this project, OpenCV was only used to create a script to capture real time video and import the trained model to detect the hand gestures.

3.4 Keras

Keras is a free and open-source library which provides an interface to artificial neural networks using the Python language. It mainly supports many software backends such as TensorFlow but with the release of Keras 2.4 it became an independent software. Keras gives the functionality of defining the size and weights of the neural network and therefore it is able to customize the model to the needed solution.

3.5 CUDA Toolkit

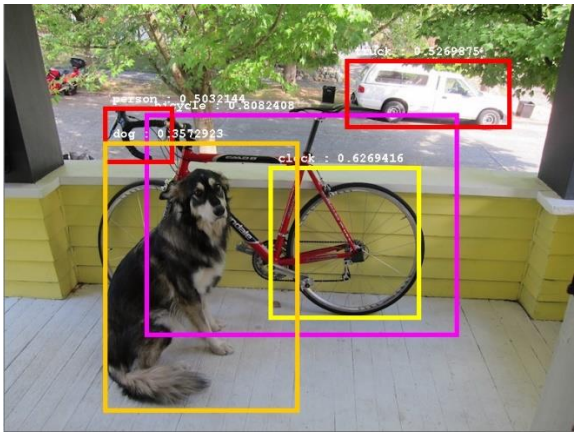
The CUDA toolkit software gives the ability for programs to be deployed on a GPU giving faster results. This is only available on NVIDIA hardware since only NVIDIA GPUs contain CUDA cores. CUDA cores are only able to perform floating point math calculations. In this project a GTX 1060 was used which on average contains around 1280 CUDA cores.

3.6 NVIDIA cuDNN

An extension to the CUDA Toolkit which allows access to libraries for deep neural networks to have access to GPU-acceleration.

4. Convolutional Neural Network

A convolutional neural network can be defined as a deep learning algorithm which given an input of an image, can assign the different objects in the image to their respective classes. For example, figure 1 shows an image which through a CNN has been identified with multiple objects which are different from each other. Another upside of CNNs is that the pre-processing stage requires much less compared to other classification algorithms [7]. CNNs have been used in multiple real-life applications spanning from entertainment to medicine.

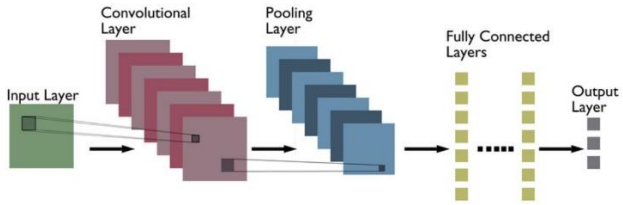


[8]

Fig.1

In [9] the research article details how a 3D Convolutional Network was used to detect cancer cells in patients' lungs, in which not only did they get positive results but in the conclusion, the researchers also stated that "In the future, it could be possible to extend our current model to not only determine whether or not the patient has cancer, but also determine the exact location of the cancerous nodules".

A CNN typically is split up into three main layers called: the convolutional layer, the pooling layer and the fully connected layer as shown in figure 2.



[10]

Fig.2

The convolutional layer applies filters on the input image, this creates a matrix for each colour channel (red, green, blue) called the feature map. Convolutional layers can be stacked to create complex feature maps which can retrieve more information from the image, thus in theory having a better learning rate [10].

The pooling layer takes care of reducing the spatial size of the input after the convolutional layer so that the fully connected layer processes the data easier and would require less memory power to work with. Thus, this makes the training faster of the CNN. Generally, two pooling functions are used max and average pooling. Max pooling takes the maximum value from each feature map, whilst the average pooling takes an average of the value in the feature map [10].

The fully connected layer is the last layer in the CNN, it is called fully connected since each neuron int this layer is connected to each neuron in the pooling layer. Each neuron represents a class, so if the CNN is trained to distinguish between a dog, a cat, a horse and a cow, the fully connected layer must have 4 neurons. Each neuron has a weight identifying the probability that the object in the image represents the class [10].

Even though there are many other neural networks such as Radial Based Networks and Recurrent Neural Networks, CNNs often give the best results for image classification and have been used in many different applications such as facial recognition, image search and augmented reality.

5. Description of the Implementation

5.1 Dataset

The dataset was split into two files called 'training' and 'validation'. The training folder was used to train the CNN whilst the validation folder was used to calculate the accuracy during training of the CNN. Each folder contains a sub-folder with the names of the eight different hand gestures, in total each hand gesture has around fifty images, with forty of them inputted in the training folder and ten of them in the validation folder. This 80-20 split was chosen since it is commonly used in the industry.

The images were captured on a personal smartphone with a plain background (white background) so that the CNN only captures the hand gesture.

5.2 create_data.py

This python script handles the image acquisition, builds the model and trains the model.

Firstly, the images are acquired by using the flow.from.directory() function in which the following are specified: the file path for the images, the target size, batch_size and class_mode. The file path is where the images are stored in the directory, the target size requires two integer values which specify the image height and width required, if these are not met, they are changed to the ones specified. The batch_size is given an integer value which refers to the number of training examples used in one iteration. The class_mode refers to the label type, in our case since we have more than 1 label name it must be "categorical".

Next the CNN is build using the keras.Sequential() function which contains multiple functions to build the CNN as needed for the problem.

The first three layers: layers.RandomZoom(), layers.RandomRotation(), and layers.RandomFlip() augment the images when entering the CNN, this is done since the model can be over trainedwith the given dataset therefore to counteract this image augmentation is used. Next the hidden layers are built using the layers.Conv2D() function and for each of these layers, the layers.MaxPool2D() function was used. This was done so that only the important features are extracted from the images. Finally the data is flattened using the layers.Flatten() function and the output layer is created using the layer.Dense() and specified that only 8 neurons are needed for the 8 different classes.

Afterwards the model.compile() function is used to configure the model for training by specifying the optimizer, the loss function and the metric function. The following were used for the project:

optimizer=keras.optimizers.Adam(learning_rate=0.0001)
loss=keras.losses.CategoricalCrossentropy()
metrics=['accuracy']

Finally the model.fit() function was used to train the data by specifying the dataset and the number of steps it should take called epochs. The model is saved after training by using the model.save() function.

5.3 webcam.py

This python script loads the model from the directory and uses OpenCV to capture real time video through a webcam.

The model is loaded using the keras function, keras.models.load_model() and specifying the model name. Next a list is created containing all the emoji names which must be in the same order as in the images folders.

A while loop is used to keep video capture going until the 'q' button is pressed on the keyboard by the user. Afterwards the current frame is taken and pre-processed so that the model can be able to evaluate it, the weight values of the final layer are saved in a variable called value. The value variable contains a list of 8 different numbers each representing the weight that the image contains with the specified hand gesture. This was done by using the model.predict() function. Since we want the highest probability of what the object is, np.argmax is used to get the index value of the highest value, this will be used to get the corresponding name of the hand gesture from the list we created in the beginning. Finally, a text is displayed using the cv2.putText() function to output the name of the emoji.

6. Evaluation

6.1 Results

The model which I created is able to detect six out of the eight hand gestures correctly, the images below show the prediction for each hand gesture in real time.

Okay gesture:

Fig.3

Peace gesture:

Fig.4

Fist gesture:

Fig.5

Point sideways gesture:

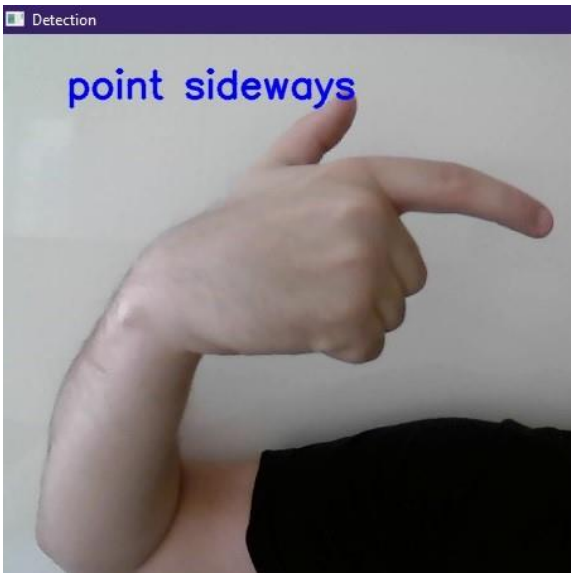


Fig.6

Point gesture:

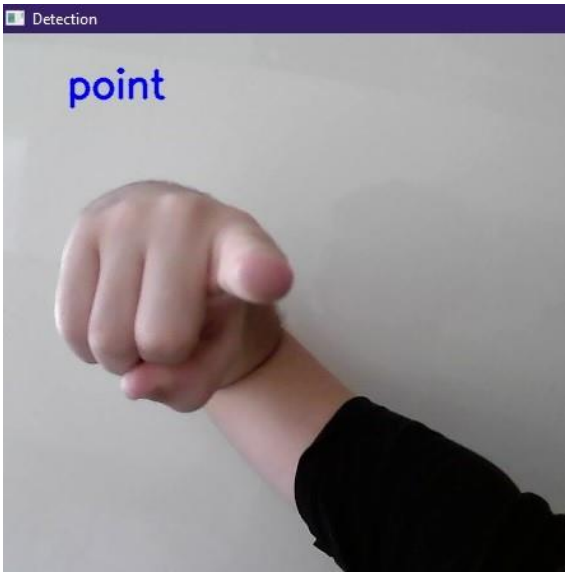


Fig.7

Rock gesture:

Fig.8

Stop gesture:

Fig.9

Thumbs up gesture:

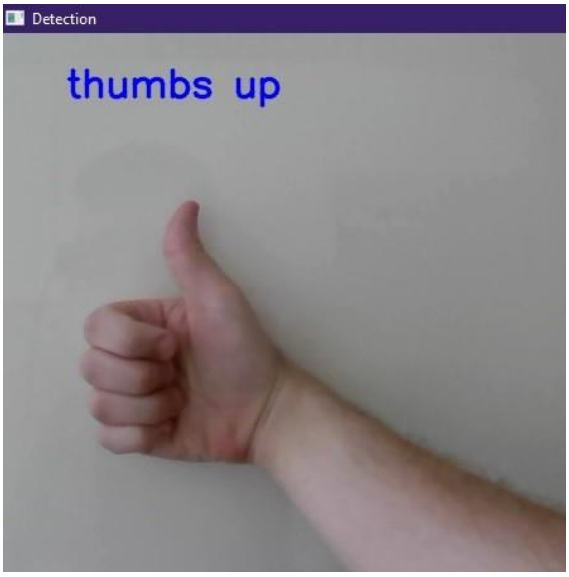


Fig.10

Figures 3 and 4 show the two hand gestures that the model cannot detect properly, even after improving the lighting of the room and trying multiple-camera angles the model still could not detect the okay and peace hand gesture. On the other hand, the other gestures shown in figures 5-10 were detected properly meaning that the output was correct.

6.2 Limitations

For the model to give correct predictions for the hand gestures, the hand must be positioned in the middle of the frame and must have adequate lighting whilst also being around two to three meters away from the camera lens. If the hand moves too close to the camera, random predictions would be given and this would obviously be incorrect. Moreover, the background of the video must be of a static colour or the model would give incorrect predictions. I believe this happens since the images captured were taken with a static white background. Finally, the frame must not contain any other objects than the hand since this would interfere with the prediction accuracy.

6.3 Improvements

The model can improve in two factors to be able to detect the hand gesture more accurately in different lighting environments and different camera angles.

1. *Having a larger and more accurate dataset*

   Data is the foundation of model training, it is common to hear that the larger the dataset the more accurate the model prediction is, however simply having a large dataset does not always lean to better results. If the quality of the data is mediocre the model would not have the ideal training scenario, therefore the quality and quantity of the data go hand in hand and must be discussed before being used [11]. In this case I believe that having around 50 images for each hand gesture was too low of a dataset even though it achieved adequate results. With a larger dataset, it could have detected the hand gesture in different angles and lighting. Moreover, it is my belief that the quality of my dataset is not great. Some images are blurred and others do not have the whole hand gesture in frame. With these combined the model would be more accurate in different scenarios.

2. *Adding more layers to the model*

   By adding more layers, the model can extract much more information from the dataset and is able to detect more differences between the classes. For example, GoogLeNet is a CNN developed by researchers at Google, and has an average accuracy of 93% and has 22 different layers [12]. In this project the model has 14 layers, therefore by increasing the layers in the CNN, the model would have better data to predict the hand gestures. However, not all problems require large amounts of layers. For example, if the model has to distinguish between two totally different objects such as a chair and a table a few layers would be enough since not much features can be extracted and the extra layers would slow the model and be unnecessary [13].

6.4 Model Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 random_zoom (RandomZoom)    (None, None, None, None)  0

 random_rotation (RandomRota  (None, None, None, None)  0
 tion)

 random_flip (RandomFlip)    (None, None, None, None)  0

 random_contrast (RandomCont  (None, None, None, None)  0
 rast)

 conv2d (Conv2D)             (None, None, None, 32)    896

 max_pooling2d (MaxPooling2D  (None, None, None, 32)   0
 )

 conv2d_1 (Conv2D)           (None, None, None, 64)    18496

 max_pooling2d_1 (MaxPooling  (None, None, None, 64)   0
 2D)

 conv2d_2 (Conv2D)           (None, None, None, 64)    36928

 max_pooling2d_2 (MaxPooling  (None, None, None, 64)   0
 2D)

 flatten (Flatten)           (None, None)              0

 dense (Dense)               (None, 128)               7372928

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 8)                 1032

=================================================================
Total params: 7,446,792
Trainable params: 7,446,792
Non-trainable params: 0
_________________________________________________________________
None
```

Fig.11

Figure 11 shows a summary of the model describing each layer with the number of parameters trained for each layer and the total number of parameters of the model. As described previously the model contains 4 layers for image augmentation, 9 layers for feature detection and 1 output layer.

6.5 Confusion Matrix

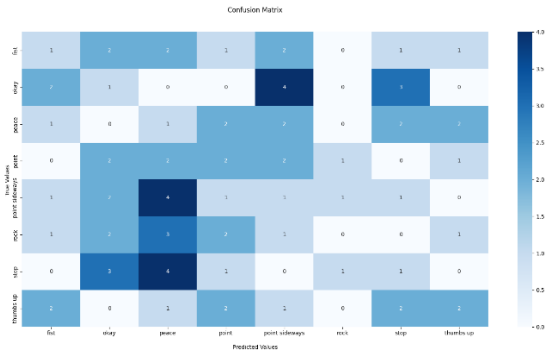


Fig.12

Figure 12 shows the confusion matrix of the model which defines the performance of the CNN. As can be seen the predicted and true values are extremely in-accurate due to the fact that in the results tab the model was able to predict six out of the eight emojis correctly. I believe that the confusion matrix that I built is coded poorly to achieve these different results.

7. Conclusion

Overall, I believe that in this project I achieved a good result given that the model had to be created from scratch. I understand that the model can improve in many ways as previously described however the model can be used as a stepping stone for myself for future model detection projects. During the research phase of this project, I have learnt a lot about the industry of computer vision and how it is used in many applications of today. Moreover, I have also learnt how time-consuming the field of computer vision is, changing a small value in model or also changing a bit the dataset can have an extreme difference in results, both good and bad. Finally, I also have gained more respect to programmer who dedicate their time to this field, especially to researchers who after many months of building a model publish it and let anyone use it for free.

8. References

[1] H.-Y. Chung, Y.-L. Chung, and W.-F. Tsai, "An efficient hand gesture recognition system based on deep CNN," in *2019 IEEE International Conference on Industrial Technology (ICIT)*, 2019, pp. 853–858.
[2] K. B. Shaik, P. Ganesan, V. Kalist, B. S. Sathish, and J. M. M. Jenitha, "Comparative study of skin color detection and segmentation in HSV and YCbCr color space," *Procedia Comput. Sci.*, vol. 57, pp. 41–48, 2015.
[3] S. Yadav and S. Payandeh, "Understanding tracking methodology of kernelized correlation filter," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2018, pp. 1330–1336.
[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.
[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv [cs.CV]*, 2014.
[7] S. Saha, "A comprehensive guide to convolutional neural networks — the ELI5 way," *Towards Data Science*, 15-Dec-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 12-Jun-2022].
[8] [Online]. Available: http://]https://blog.jetbrains.com/kotlin/2022/01/object-detection-with-kotlindl-and-ktor/. [Accessed: 12-Jun-2022].
[9] W. Alakwaa, M. Nassef, and A. Badr, "Lung cancer detection and classification with 3D convolutional neural network (3D-CNN)," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 8, 2017.
[10] A. Kumar, "Different types of CNN Architectures explained: Examples," *Data Analytics*, 12-Apr-2022. [Online]. Available: https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/. [Accessed: 12-Jun-2022].
[11] V. Chawla, "Is more data always better for building analytics models?," *Analytics India Magazine*, 16-Sep-2020. [Online]. Available: https://analyticsindiamag.com/is-more-data-always-better-for-building-analytics-models/. [Accessed: 12-Jun-2022].
[12] C. Szegedy *et al.*, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.
[13] J. Dsouza, "How to improve the accuracy of your image recognition models," *freeCodeCamp.org*, 29-Nov-2021. [Online]. Available: https://www.freecodecamp.org/news/improve-image-recognition-model-accuracy-with-these-hacks/. [Accessed: 12-Jun-2022].