Academic Journal of Nawroz University (AJNU), Vol.10, No.4, 2021
This is an open access article distributed under the Creative Commons Attribution License
Copyright ©2017. e-ISSN: 2520-789X
https://doi.org/10.25007/ajnu.v10n4a1328

Keras Deep Learning for Pupil Detection Method

Renas R. Asaad
Department of Computer Science, Nawroz University, Duhok, Kurdistan Region – Iraq

ABSTRACT

Nowadays, deep learning is the most common field in artificial intelligence, it is a technique invented by scientist trying to imitate the way the human mind works. Deep learning tries to simulate the human mind in all its capabilities, which include; Seeing, understanding speech, composing it, hearing, and other powerful abilities that our human mind possesses and is not rivaled by anything else, or so it was. The matter did not stop at that only, but that scientists have studied the human brain and how it works in order to design algorithms and programs capable of simulating it, and for this reason we find that these algorithms are inspired by medical and neurological studies of humans and try as much as possible to imitate them, but by computer methods. Not biological. In this paper studied and practiced the multi-task deep convolution neural network(MTCNN) for face detection after detecting the faces its tries to extract eyes from an image, finally pupil detector works by using Keras model in python. The result of this paper shows the power of the deep learning field biologically comparative with human thinks.

Keywords: Machine Learning, Deep Learning, MTCNN Keras Model, AI.

1. Introduction

Deep learning is a largely new science, but it has roots in human knowledge, and it has stages through which it has developed from the 1940s to the present day, and according to an article on the history of the development of deep learning on the popular website Towards Data Science, these stages are:

The stage of cybernetics: which extended from the forties of the twentieth century until the nineties.

The stage of Connectionism, which was in the nineties and eighties of the last century.

The stage of Deep Learning: It is what we now know as deep learning, and it started from 2006 and extended until now.[1]

1.1 The First Stage (Nervous Stage)

It was characterized by an attempt to understand how the human mind works, in which computer scientists (who were then mathematicians), psychologists and neuroscientists participated, who worked together to create an equation or a mathematical system that simulates the way the brain works, in order to provide the same inputs to it, and in return Obtaining the same judgments and outputs as what the human mind produces.

The artificial neural unit or perceptron, which we talked about a while ago, is one of the most prominent discoveries of this stage, and was reached by the famous American psychologist Frank Rosenblatt, who is considered one of the most prominent scientists who contributed to the sciences of artificial intelligence.

1.2 The Second Stage (Communicative Stage)

It is the stage that began after the discoveries and developments accompanying the increased interest in cognitive science, which was carried out by some brilliant psychologists to try to understand how the mind and senses work in a more abstract way. Computer scientists and mathematicians have benefited a lot from this stage, and used it to produce the first simple forms of what we know today. In the name of deep learning.

One of the most famous productions of this important period in the history of global technology is Artificial Neural Networks (ANNs), or Artificial Neural Networks. They are neural networks that simulate how the natural neural networks that operate in the human mind work. Scientists at this stage also came up with the back propagation model, which has been used a lot in natural language processing, computer vision, and many other branches of artificial intelligence.

1.3 The Third Stage (Deep Learning Stage)

It was also due to a development made by Canadian cognitive psychologist and computer scientist Geoffrey Hinton in 2006; Where he did DBN or Deep Belief Networks; Which is a neural network that contains a lot of hidden layers under it, which works without human supervision, and from that time until now the era of artificial intelligence and deep learning began.

2. Literature Review

Initially, the concept of machine learning refers to the discipline concerned with the study of algorithms based on making a machine learn and evolve on its own without the need for specific software commands. A "data set" is fed into these algorithms, which they analyze, learn from, and then build predictions based on what they've learned. The machine learning major represents the intersection of computer science and statistics.

A simple example of an application of machine learning algorithms is "Suggestions". When you listen to music on a popular listening platform, on your next visit you will find more similar music and recommendations related to your listening history. Machine learning algorithms analyze your listening history and preference data, find patterns within them and then build predictions for many other music that suits your taste.[1]

As for the concept of deep learning, it is a specialty of machine learning, which in turn is a specialty of artificial intelligence. As shown in below figure.

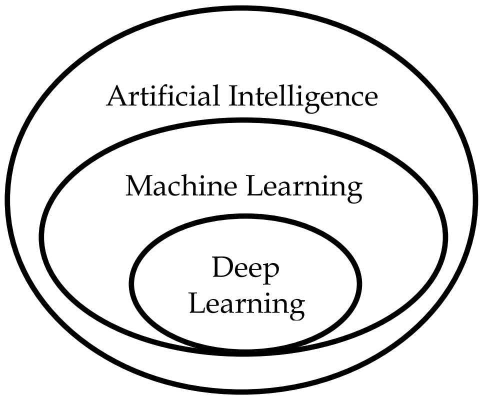


Fig 1: Concept of Deep Learning

It is the specialization concerned with the study of "artificial neural networks" that simulate the neural networks in the human brain.

3. Artificial Neural Network

An artificial neural network is an information processing system, the concept is taken from biological neurons. this is human kind's best effort to emulate a biological network. The computing method is based on multiple connected artificial neurons that process information in a distributed matter across the computing neurons.[1]

Artificial neural networks are best suited to solve complex problems that require generalization and pattern recognition. As shown in below figure.

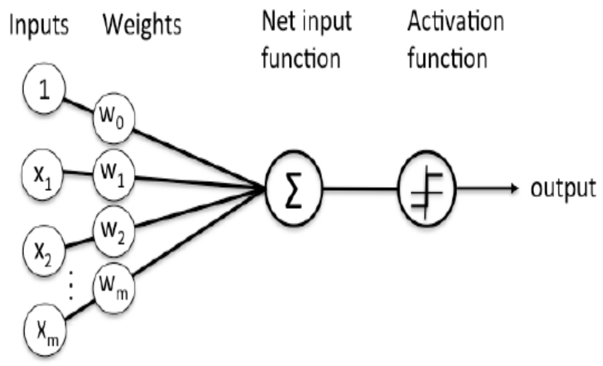


Fig 2: Perceptron Neurons

Properties of the Nervous System

- Parallel, distributed information processing.
- High degree of connectivity between basic processing units.
- Connections are modified based of experience.
- Learning is a constant process.
- Learning is based on local information.

Biological Neuron

A biological neuron is most basic information processing unit in the nervous system. a biological neuron consists of the following parts:

1. Dendrites (input)

2. Cell body

3. Axon (output)

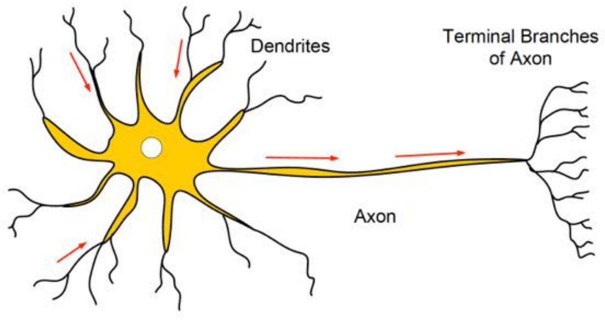


Fig 3: Biological Neuron

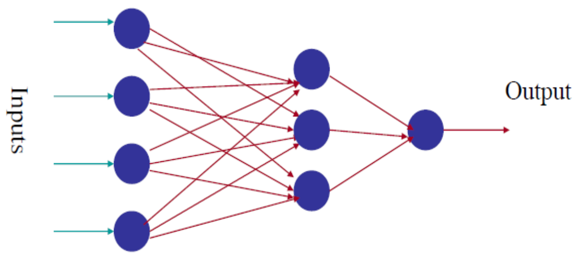
Figure 3 shows an biological neuron takes signals from its dendrites and processes the signal and outputs a signal from its axon based on the input signal.[2]

4. Deep Learning

The discipline of deep learning emerged as an extension and evolution of machine learning when traditional machine learning algorithms were unable to perform some complex tasks. For example, traditional machine learning algorithms require a simplified and tidy data set to learn from. But it is not able to learn from large and complex data sets such as different sound waves, image dimensions and the number of pixels inside, so deep learning algorithms are used to handle such complex data as in the "voice recognition" applications that Siri uses to help Google recognize The voices of its creators and the "image recognition" applications used by Facebook to identify people's faces in photos.[3]

4.1 Configure a Deep Learning Algorithm

The algorithm consists of a multi-layer structure of artificial neural networks. Where the layer on the left end is the input layer, the layer on the right is the output layer, and in the middle are several hidden layers responsible for processing. The architecture of stratified deep learning algorithms enables better data handling and better performance. As shown in below figure.[3]



$$y_1^1 = f(x_1, w_1^1) \quad y_2^1 = f(x_2, w_2^1) \quad y_3^1 = f(x_3, w_3^1) \quad y_4^1 = f(x_4, w_4^1)$$

$$y^1 = \begin{pmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \\ y_4^1 \end{pmatrix} \quad y_1^2 = f(y^1, w_1^2) \quad y_2^2 = f(y^1, w_2^2) \quad y_3^2 = f(y^1, w_3^2) \quad y^2 = \begin{pmatrix} y_3^2 \\ y_3^2 \\ y_3^2 \end{pmatrix} \quad y_{Out} = f(y^2, w_1^3)$$

Fig 4: Multi-Layer ANN

Accordingly, the deep learning algorithm requires a lower level of human intervention in optimizing the algorithm's results. The algorithm learns and improves from its mistakes on its own. Unlike the traditional machine learning algorithm that requires a lot of human intervention for modification and optimization.[3]

However, deep learning algorithms are still under development. Despite its progress and the remarkable improvement in its results and performance, it requires a lot of time and high computing power to learn from the huge data set and build a viable model.

Despite the advancement of algorithms and techniques in the world of artificial intelligence and the promising results that their applications carry, they are still in their early stages, and there are many advances and opportunities ahead.[4]

5. Face Detector

Speaking of face detection, we first think of using the Harr extraction feature and the Adaboost face detection classifier, the detection effect is also good, but the current application scenarios for face detection are gradually developing from Indoor to outdoor, from one limited scene to scenes like squares, stations, subway entrances, etc. higher and higher. For example, faces vary in size, number is large, and poses are varied, including top shots of faces, masks wearing hats and masks, exaggerated expressions, camouflaging makeup, poor lighting conditions, low resolution, and even difficult to distinguish with the naked eye. In such a complex environment, the performance of face detection based on Haar features is unsatisfactory. With the development of deep learning, face detection technology based on deep learning has achieved great success. This section introduce the MTCNN algorithm, which is a high-precision face detection and real-time detection algorithm based on convolutional neural networks.[5]

The first step in building a face recognition system is face detection, which is to find the position of the face in the image. In this process, the input is an image containing human faces, and the output is a rectangular frame of all faces. In general, face detection should be able to detect all the faces in the image, and there should be no missed detections, not to mention false detections.[5]

After getting the face, the second step we need to do is align the face. Since the face in the original image may have differences in position and position, for subsequent unified processing, we have to "straighten" the face. To this end, it is necessary to detect the main points of the face, such as the position of the eyes, the position of the nose, the position of the mouth, and the points of the facial contour. According to these key points, the affine transformation can be used to uniformly calibrate the face to eliminate errors caused by different situations.[5]

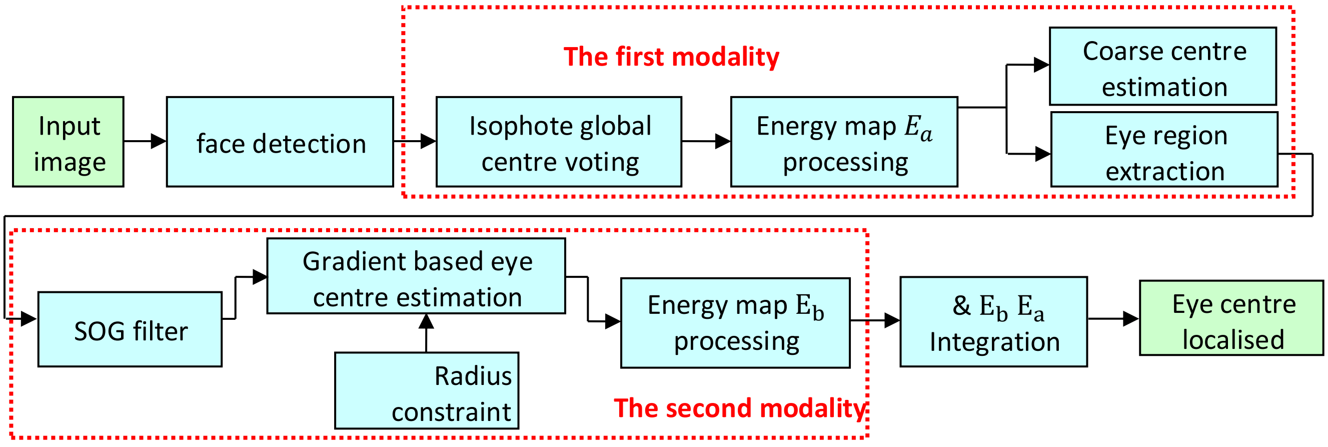


Fig 5: Eye Centre Localization

5.1 MTCNN Algorithm Structure

The MTCNN algorithm(multi-task deep convolutional neural network) is a deep learning-based face detection and face alignment method, which can complete face detection and face alignment tasks at the same time. Compared with traditional algorithms, it has better performance and faster detection speed.

The MTCNN algorithm contains three subnets: Proposal Network (P-Net), Refine Network (R-Net), and Output Network (O-Net). These three grids process faces from coarse to fine see figure 5.[8]

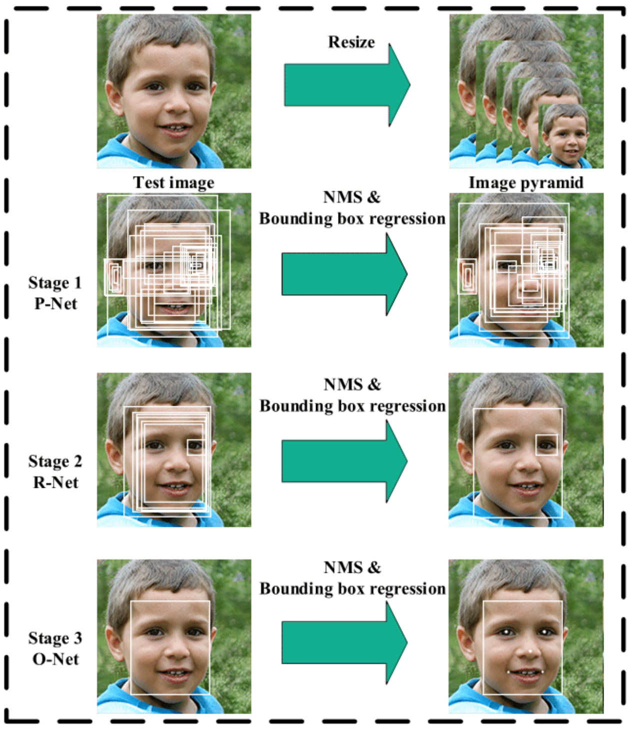


Fig 5: MTCNN stages

6. Keras Model

Keras is one of the most popular deep learning libraries, and its artificial intelligence has made significant contributions. It's simple use, just a few lines of code can build a powerful neural network. In this article, you will learn how to build a neural network using KERAS.[9]

Keras is an open source Python library that allows you to easily build a neural network. This library can run on Tensorflow, Microsoft Cognitive Toolkit, Theano and MXNet. Tensorflow and Theano are the most common scientific computing platforms used to build deep learning algorithms in Python, but they can sometimes be very complex and difficult to use. In contrast, Keras provides a simple and convenient way to build an in-depth learning model. Its creator, Françoischollet, developed it to enable people to build a neural network as quickly and easily as possible. Focus on scalability, maintenance, simplicity and support for the python. Keras can use both the GPU and the CPU, which supports Python 2 and 3. Google Keras makes great contributions to the field of deep learning and artificial intelligence, because it has been product-embedded in a modern learning algorithm, these algorithms are not only accessible, but also unusable.

In 2017, the TenserFlow team from Google decided to add Keras support in the TenserFlow core library. He also explained that Keras was designed as a programming interface rather than a standalone machine learning framework. Keras offers a higher and more intuitive set of abstractions that make it easy to develop deep learning models regardless of the background used to perform the calculations (be it Tenserflow or otherwise). Microsoft has also added a backend based on Microsoft Cognitive Tools CNTK to Keras.[9]

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from six import string_types,
iteritems
import numpy as np
import tensorflow as tf
import cv2
import os
```

```
def layer(op):
    """Decorator for composable
network layers."""

    def layer_decorated(self, *args,
**kwargs):
        name =
kwargs.setdefault('name',
self.get_unique_name(op.__name__))
        if len(self.terminals) == 0:
            raise RuntimeError('No
input variables found for layer %s.' %
name)
        elif len(self.terminals) == 1:
            layer_input =
self.terminals[0]
        else:
            layer_input =
list(self.terminals)
        layer_output = op(self,
layer_input, *args, **kwargs)
        self.layers[name] =
layer_output
        self.feed(layer_output)
        return self
    return layer_decorated
class PNet(Network):
    def setup(self):
        (self.feed('data')
             .conv(3, 3, 10, 1, 1,
padding='VALID', relu=False,
name='conv1')
             .prelu(name='PReLU1')
             .max_pool(2, 2, 2, 2,
name='pool1')
             .conv(3, 3, 16, 1, 1,
padding='VALID', relu=False,
name='conv2')
             .prelu(name='PReLU2')
             .conv(3, 3, 32, 1, 1,
padding='VALID', relu=False,
name='conv3')
             .prelu(name='PReLU3')
             .conv(1, 1, 2, 1, 1,
relu=False, name='conv4-1')
             .softmax(3,name='prob1'))
        (self.feed('PReLU3')
             .conv(1, 1, 4, 1, 1,
relu=False, name='conv4-2'))
class RNet(Network):
    def setup(self):
        (self.feed('data')
             .conv(3, 3, 28, 1, 1,
padding='VALID', relu=False,
name='conv1')
             .prelu(name='prelu1')
             .max_pool(3, 3, 2, 2,
name='pool1')
             .conv(3, 3, 48, 1, 1,
padding='VALID', relu=False,
name='conv2')
             .prelu(name='prelu2')
             .max_pool(3, 3, 2, 2,
padding='VALID', name='pool2')
             .conv(2, 2, 64, 1, 1,
padding='VALID', relu=False,
name='conv3')
             .prelu(name='prelu3')
             .fc(128, relu=False,
name='conv4')
             .prelu(name='prelu4')
             .fc(2, relu=False,
name='conv5-1')
             .softmax(1,name='prob1'))

        (self.feed('prelu4')
             .fc(4, relu=False,
name='conv5-2'))
class ONet(Network):
    def setup(self):
        (self.feed('data'
             .conv(3, 3, 32, 1, 1,
padding='VALID', relu=False,
name='conv1')
             .prelu(name='prelu1')
             .max_pool(3, 3, 2, 2,
name='pool1')
             .conv(3, 3, 64, 1, 1,
padding='VALID', relu=False,
name='conv2')
             .prelu(name='prelu2')
             .max_pool(3, 3, 2, 2,
padding='VALID', name='pool2')
             .conv(3, 3, 64, 1, 1,
padding='VALID', relu=False,
name='conv3')
             .prelu(name='prelu3')
             .max_pool(2, 2, 2, 2,
name='pool3')
             .conv(2, 2, 128, 1, 1,
padding='VALID', relu=False,
name='conv4')
             .prelu(name='prelu4')
             .fc(256, relu=False,
name='conv5')
             .prelu(name='prelu5')
             .fc(2, relu=False,
name='conv6-1')
             .softmax(1,
name='prob1'))

        (self.feed('prelu5')
             .fc(4, relu=False,
name='conv6-2'))

        (self.feed('prelu5'
             .fc(10, relu=False,
name='conv6-3'))
```

6.Results

This model groups the layers into an objects with training and inference features and contain three arguments (Inputs, Outputs and Name):

- **Inputs**: The input(s) of the model: a keras.Input object.
- **Outputs**: The output(s) of the model. See Functional API example below.
- **Name**: String, the name of the model.

With the "Functional API", where you start from Input, you chain layer calls to specify the model's forward pass, and finally you create your model from inputs and outputs:

A new Functional API model can also be created by using the intermediate tensors. This enables you to quickly extract sub-components of the model. The MTCNN algorithm, is a high-precision face detection and real-time detection algorithm based on convolutional neural networks. The MTCNN algorithm contains three subnets: Proposal Network

(P-Net), Refine Network (R-Net), and Output Network (O-Net). These three grids process faces from coarse to fine. Before using these three subnets, you need to use the image hierarchy to scale the original image at different scales, and then send the images at different scales to these three subnets for training. The purpose is to detect different sizes of faces • Achieve multi-band target detection. The finalizing and the outputs are plotting eyes, eyes circle and pupil of eyes as shown in below figure.

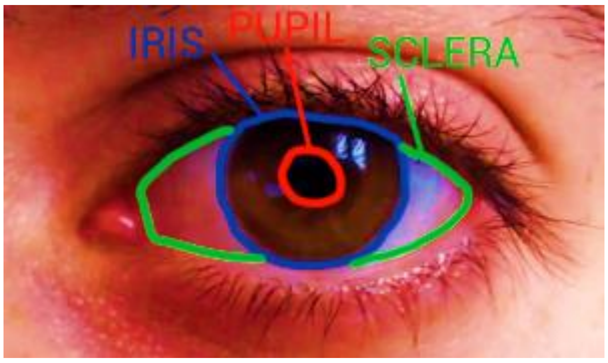


Fig 6: Three steps of detection

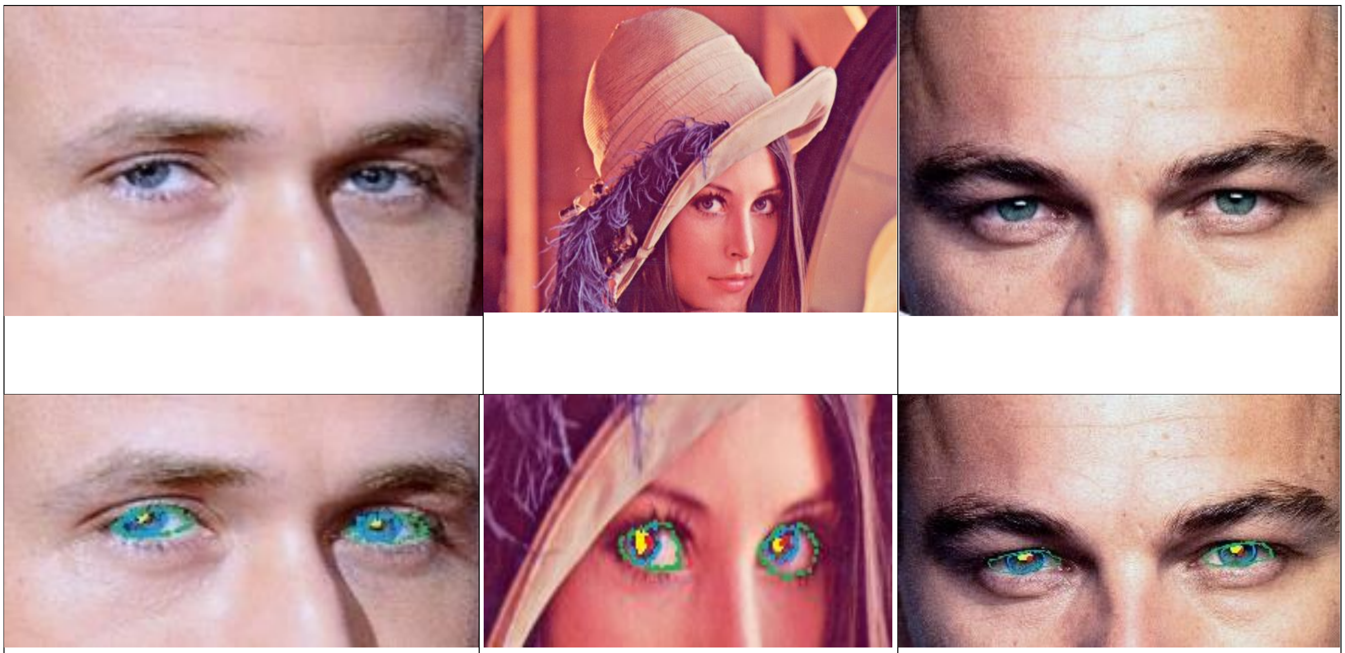


Fig 7: Results by plotting of pupil detection

7.Conclusion

In this paper, pupil, iris and eyes detection allows researchers to study the movements of a participant's eyes during a range of activities. This gives insight into the cognitive processes underlying a wide variety of human behavior and can reveal things such as learning patters and social interaction methods. Big data is perhaps the most powerful factor in the process of developing deep learning, making it more relevant to our lives by providing the large amount of data it needs in order to learn and become smart on its own.

In this paper, keras deep learning technique was used to detect the pupil of the human eye. With these results, we can in the near future control computers by the eye or the pupil of the eye for ease of use or to extend artificial intelligence in the field of deep learning, and it is also possible for people with special needs and with disabilities to benefit from this experience.

References

[1] Hoang, A. T., Nižetić, S., Ong, H. C., Tarelko, W., Le, T. H., Chau, M. Q., & Nguyen, X. P. (2021). A review on application of artificial neural network (ANN) for performance and emission characteristics of diesel engine fueled with biodiesel-based fuels. *Sustainable Energy Technologies and Assessments*, *47*, 101416.

[2] Moldwin, T., & Segev, I. (2020). Perceptron learning

and classification in a modeled cortical pyramidal cell. *Frontiers in computational neuroscience*, *14*, 33.
[3] Ramasubramanian, K., & Singh, A. (2019). Deep learning using keras and tensorflow. In *Machine Learning Using R* (pp. 667-688). Apress, Berkeley, CA.
[4] Vasilev, I., Slater, D., Spacagna, G., Roelants, P., & Zocca, V. (2019). *Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow*. Packt Publishing Ltd.
[5] Ghofrani, A., Toroghi, R. M., & Ghanbari, S. (2019). Realtime face-detection and emotion recognition using mtcnn and minishufflenet v2. In *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)* (pp. 817-821). IEEE.
[6] Rajab Asaad, R. (2021). Review on Deep Learning and Neural Network Implementation for Emotions Recognition . Qubahan Academic Journal, 1(1), 1–4. https://doi.org/10.48161/qaj.v1n1a25
[7] Asaad, R. R., & Ali, R. I. (2019). Back Propagation Neural Network(BPNN) and Sigmoid Activation Function in Multi-Layer Networks. Academic Journal of Nawroz University, 8(4), 216–221. https://doi.org/10.25007/ajnu.v8n4a464
[8] Chen, X., Luo, X., Liu, X., & Fang, J. (2019, May). Eyes localization algorithm based on prior MTCNN face detection. In 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC) (pp. 1763-1767). IEEE.
[9] Manaswi, N. K. (2018). Understanding and working with Keras. In Deep Learning with Applications Using Python (pp. 31-43). Apress, Berkeley, CA.

Appendix

```
#Keras Model
from keras.layers import *
from keras.models import Model
import tensorflow as tf
import numpy as np
class KerasELG():
  def __init__(self, first_layer_stride=3, hg_num_feature_maps=64, hg_num_modules=3):
    self._first_layer_stride = first_layer_stride
    self._hg_num_feature_maps = hg_num_feature_maps
    self._hg_num_modules = hg_num_modules
    self._hg_num_residual_blocks = 1
    self._hg_num_landmarks = 18
    self.net = self.build_elg_network()
  def build_elg_network(self):
    return self.elg()
    def elg(self):
    outputs = {}
    inp = Input((108, 180, 1))
    n = self._hg_num_feature_maps
    pre_conv1 = self._apply_conv(inp, n, k=7, s=self._first_layer_stride, name="hourglass_pre")
    pre_conv1 = self._apply_bn(pre_conv1, name="hourglass_pre_BatchNorm")
    pre_conv1 = Activation('relu')(pre_conv1)
    pre_res1 = self._build_residual_block(pre_conv1, 2*n, name="hourglass_pre_res1")
    pre_res2 = self._build_residual_block(pre_res1, n, name="hourglass_pre_res2")
    x = pre_res2
    x_prev = pre_res2
    for i in range(self._hg_num_modules):
      prefix = f"hourglass_hg_{str(i+1)}"
      x = self._build_hourglass(x, steps_to_go=4, f=self._hg_num_feature_maps, name=prefix)
      x, h = self._build_hourglass_after(
        x_prev,
        x,
        do_merge=(i<(self._hg_num_modules-1)),
        name=prefix)
      x_prev = x
```

```
        x = h
        outputs['heatmaps'] = x
        return Model(inp, outputs['heatmaps'])
    def _apply_conv(self, x, f, k=3, s=1, padding='same', name=None):
        return Conv2D(f, kernel_size=k, strides=s, use_bias=True, padding=padding, name=name)(x)
    def _apply_bn(self, x, name=None):
        return BatchNormalization(name=name)(x)
    def _apply_pool(self, x, k=2, s=2):
        return MaxPooling2D(pool_size=k, strides=s, padding="same")(x)
    def _build_residual_block(self, x, f, name="res_block"):
        num_in = x.shape.as_list()[-1]
        half_num_out = max(int(f/2), 1)
        c = x
        conv1 = self._apply_bn(c, name=name+"_conv1_BatchNorm")
        conv1 = Activation('relu')(conv1)
        conv1 = self._apply_conv(conv1, half_num_out, k=1, s=1, name=name+"_conv1")
        conv2 = self._apply_bn(conv1, name=name+"_conv2_BatchNorm")
        conv2 = Activation('relu')(conv2)
        conv2 = self._apply_conv(conv2, half_num_out, k=3, s=1, name=name+"_conv2")
        conv3 = self._apply_bn(conv2, name=name+"_conv3_BatchNorm")
        conv3 = Activation('relu')(conv3)
        conv3 = self._apply_conv(conv3, f, k=1, s=1, name=name+"_conv3")
        if num_in == f:
            s = x
        else:
            s = self._apply_conv(x, f, k=1, s=1, name=name+"_skip")
        out = Add()([conv3, s])
        return out
    def _build_hourglass(self, x, steps_to_go, f, depth=1, name=None):
        prefix_name = name + f"_depth{str(depth)}"
        up1 = x
        for i in range(self._hg_num_residual_blocks):
            up1 = self._build_residual_block(up1, f, name=prefix_name+f"_up1_{str(i+1)}")
        low1 = self._apply_pool(x, k=2, s=2)
        for i in range(self._hg_num_residual_blocks):
            low1 = self._build_residual_block(low1, f, name=prefix_name+f"_low1_{str(i+1)}")
        low2 = None
        if steps_to_go > 1:
            low2 = self._build_hourglass(low1, steps_to_go-1, f, depth=depth+1, name=prefix_name)
        else:
            low2 = low1
            for i in range(self._hg_num_residual_blocks):
                low2 = self._build_residual_block(low2, f, name=prefix_name+f"_low2_{str(i+1)}")
        low3 = low2
        for i in range(self._hg_num_residual_blocks):
            low3 = self._build_residual_block(low3, f, name=prefix_name+f"_low3_{str(i+1)}")
        up2 = Lambda(
            lambda x: tf.image.resize_bicubic(
                x[0],
                x[1].shape.as_list()[1:3],
                align_corners=True))([low3, up1]) # default resize_bilear
        out = Add()([up1, up2])
        return out
    def _build_hourglass_after(self, x_prev, x_now, do_merge=True, name=None):
        prefix_name = name+"_after"
        for j in range(self._hg_num_residual_blocks):
            x_now = self._build_residual_block(
                x_now,
                self._hg_num_feature_maps,
            name=prefix_name+f"_after_hg_{str(j+1)}")
        x_now = self._apply_conv(x_now, self._hg_num_feature_maps, k=1, s=1, name=prefix_name)
        x_now = self._apply_bn(x_now, name=prefix_name+"_BatchNorm")
        x_now = Activation('relu')(x_now)
        h = self._apply_conv(x_now, self._hg_num_landmarks, k=1, s=1, name=prefix_name+"_hmap")
        x_next = x_now
        if do_merge:
            prefix_name = name
            x_hmaps = self._apply_conv(
                h,
                self._hg_num_feature_maps,
                k=1,
                s=1,
                name=prefix_name+"_merge_h")
            x_now = self._apply_conv(
                x_now,
                self._hg_num_feature_maps,
                k=1,
                s=1,
                name=prefix_name+"_merge_x")
            x_add = Add()([x_prev, x_hmaps])
            x_next = Add()([x_next, x_add])
        return x_next, h
    def _calculate_landmarks(x, beta=5e1):
        def np_softmax(x, axis=1):
            t = np.exp(x)
            a = np.exp(x) / np.sum(t, axis=axis).reshape(-1,1)
            return a
        if len(x.shape) < 4:
            x = x[None, ...]
        h, w = x.shape[1:3]
        ref_xs, ref_ys = np.meshgrid(np.linspace(0, 1.0, num=w, endpoint=True),
                                     np.linspace(0, 1.0, num=h, endpoint=True),
                                     indexing='xy')
        ref_xs = np.reshape(ref_xs, [-1, h*w])
        ref_ys = np.reshape(ref_ys, [-1, h*w])
        beta = beta
        x = np.transpose(x, (0, 3, 1, 2))
        x = np.reshape(x, [-1, 18, h*w])
        x = np_softmax(beta * x, axis=-1)
        lmrk_xs = np.sum(ref_xs * x, axis=2)
        lmrk_ys = np.sum(ref_ys * x, axis=2)
        return np.stack([lmrk_xs * (w - 1.0) + 0.5, lmrk_ys * (h - 1.0) + 0.5], axis=2)  # N x 18 x 2
```

```
import mtcnn_detect_face
import tensorflow as tf
from keras import backend as K
import numpy as np
import cv2
import os
class MTCNNFaceDetector():
    def __init__(self, sess, model_path="./mtcnn_weights/"):
        self.pnet = None
        self.rnet = None
        self.onet = None
        self.create_mtcnn(sess, model_path)
            def create_mtcnn(self, sess, model_path):
        if not model_path:
            model_path, _ = os.path.split(os.path.realpath(__file__))
        with tf.variable_scope('pnet'):
            data = tf.placeholder(tf.float32, (None,None,None,3), 'input')
            pnet = mtcnn_detect_face.PNet({'data':data})
            pnet.load(os.path.join(model_path, 'det1.npy'), sess)
        with tf.variable_scope('rnet'):
            data = tf.placeholder(tf.float32, (None,24,24,3), 'input')
            rnet = mtcnn_detect_face.RNet({'data':data})
            rnet.load(os.path.join(model_path, 'det2.npy'), sess)
        with tf.variable_scope('onet'):
            data = tf.placeholder(tf.float32, (None,48,48,3), 'input')
            onet = mtcnn_detect_face.ONet({'data':data})
            onet.load(os.path.join(model_path, 'det3.npy'), sess)
        self.pnet = K.function([pnet.layers['data']], [pnet.layers['conv4-2'], pnet.layers['prob1']])
        self.rnet = K.function([rnet.layers['data']], [rnet.layers['conv5-2'], rnet.layers['prob1']])
        self.onet = K.function([onet.layers['data']], [onet.layers['conv6-2'], onet.layers['conv6-3'],
onet.layers['prob1']])
    def detect_face(self, image, minsize=20, threshold=0.7, factor=0.709, use_auto_downscaling=True,
min_face_area=25*25):
        if use_auto_downscaling:
            image, scale_factor = self.auto_downscale(image)
                faces, pnts = mtcnn_detect_face.detect_face(
            image, minsize,
            self.pnet, self.rnet, self.onet,
            [0.6, 0.7, threshold],
            factor)
        faces = self.process_mtcnn_bbox(faces, image.shape)
        faces, pnts = self.remove_small_faces(faces, pnts, min_face_area)
        if use_auto_downscaling:
            faces = self.calibrate_coord(faces, scale_factor)
            pnts = self.calibrate_landmarks(pnts, scale_factor)
        return faces, pnts
    def auto_downscale(self, image):
        if self.is_higher_than_1080p(image):
            scale_factor = 4
            resized_image = cv2.resize(image,
 (image.shape[1]//scale_factor,
image.shape[0]//scale_factor))
        elif self.is_higher_than_720p(image):
            scale_factor = 3
            resized_image = cv2.resize(image,
 (image.shape[1]//scale_factor,
image.shape[0]//scale_factor))
        elif self.is_higher_than_480p(image):
            scale_factor = 2
            resized_image = cv2.resize(image,
 (image.shape[1]//scale_factor,
image.shape[0]//scale_factor))
        else:
            scale_factor = 1
            resized_image = image.copy()
        return resized_image, scale_factor
```