CHAPTER 1
INTRODUCTION

Eye gaze estimation is the process of detecting eye gaze direction. Here it is estimated by Convolutional Neural Network. gaze estimation remains challenging due to the individuality of eyes, occlusion, variability in scale, location, and light conditions.

1.1 What is Eye Gaze Estimation

In simplest terms, it is the measurement of eye activity. Where do we look? What do we ignore? When do we blink? How does the pupil react to different stimuli? The concept is basic, but the process and interpretation can be quite complex. Most of the traditional techniques for gaze detection (e.g., Pupil Centre Corneal Reflection) were intrusive. Recently non-intrusive gaze detection techniques based on image or video processing are being given much emphasis for obvious advantages. The process of eye gaze direction classification has two major steps. These are eye localization and eye gaze detection. Eye localization are usually done in two major approaches: traditional approach (appearance-based methods and pattern-based methods) and neural network approach. For eye gaze detection, three approaches have been proposed. They are- appearance based technique, shape based approach and component separation method. In some recent works, deep neural network based approach has been proposed for eye gaze detection and real time eye gaze direction classification system has been developed using a two stage Convolutional Neural Network (CNN).

1.2 Why Eye Gaze Estimation

For last many years, estimation of eye gaze has drawn attention from researchers because of its wide range of applications. According to a study, there are nearly 1 in 50 people living with paralysis [1].In south Asia the number of paralyzed people is about 124 million [2]. Paralysis can be triggered by a wide range of diseases and conditions such as Stroke, Spinal cord injury, Cerebral palsy, Traumatic brain injury, Birth defects and Multiple sclerosis. Paralyzed people having nerve and muscle damages suffer in communicating and depend on others for even basic day to day activities. Eye movement may become the only way of communication for them. For road traffic safety, eye movement of drivers is being investigated to detect drowsiness and right direction of car movement [3][4][5]. Pilot training [6], ergonomics [7], market research [8] and mood disorder [9] are some of the area where eye gaze detection can have huge impacts. With very few exceptions, anything with a visual component can be eye tracked. We use our eyes almost constantly, and understanding how we use them has become an extremely important consideration in research and design. The automotive, medical and defence industries have applied eye tracking technology to make us safer. The fields of advertising, entertainment, packaging and web design have all benefited significantly from studying the visual behaviour of the consumer. Research with special populations has generated exciting breakthroughs in psychology and physiology. Eye gaze information is used in a variety of user platforms. The main use cases may be broadly classified into (i) desktop computers (ii) TV panels (iii) head mounted (iv) automotive setups (v) handheld devices. Applications based on desktop platforms involve using eye gaze for computer communication and text entry, computer control and entering gaze based passwords. Remote eye tracking has recently been used on TV panels to achieve gaze controlled functions, for example selecting and navigating menus and switching channels. Head-mounted gaze tracking setups usually comprise of two or more cameras mounted on a support framework worn by the user. Such systems have been extensively employed in user attention and cognitive studies, psychoanalysis, occulo-motor measurements, virtual and augmented reality applications. Real time gaze and eye state tracking on automotive platforms is used in driver support systems to evaluate driver vigilance and drowsiness levels. These use eye tracking setups mounted on a car's dashboard along with computing hardware running machine vision algorithms. In handheld devices such as smartphones or tablets, the front camera is used to track user gaze to activate functions such as locking/unlocking phones, interactive displays, dimming backlights or suspending sensors. Every day, as eye tracking is used in creative new ways, the list of applications grows.

CHAPTER 2
SYSTEM ANALYSIS

System analysis is the Knick knack description of the system modules.

2.1 Theoretical Background

2.1.1 What is Convolutional Neural Network (CNN)

In the last few decades, artificial intelligence (AI) has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Researchers and enthusiasts alike, work on numerous aspects of the field to make amazing things happen. One of many such areas is the domain of computer vision. The agenda for this field is to enable machines to view the world as humans do, perceive it in a similar manner and even use the knowledge for a multitude of tasks such as image & video recognition, image analysis & classification, media recreation, recommendation systems, natural language processing, etc. The advancements in computer vision with deep learning has been constructed and perfected with time, primarily over one particular algorithm—a Convolutional Neural Network.

A Convolutional Neural Network (CNN) is a deep learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these filters/characteristics. The architecture of a CNN is analogous to that of the connectivity pattern of neurons in the human brain and was inspired by the organization of the visual cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. A collection of such fields overlap to cover the entire visual area.

2.1.2 Convolutional Neural Network (CNN) Architecture

CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function and all the tips and tricks that we developed for neural networks still apply on CNNs. A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume (e.g. holding the class scores) through a differentiable function. A few distinct types of layers are commonly used. These are further discussed below.

- Convolutional Layer
- Pooling Layer
- Rectified Linear Unit (ReLU) Layer
- Fully Connected Layer (Full Connectedness)

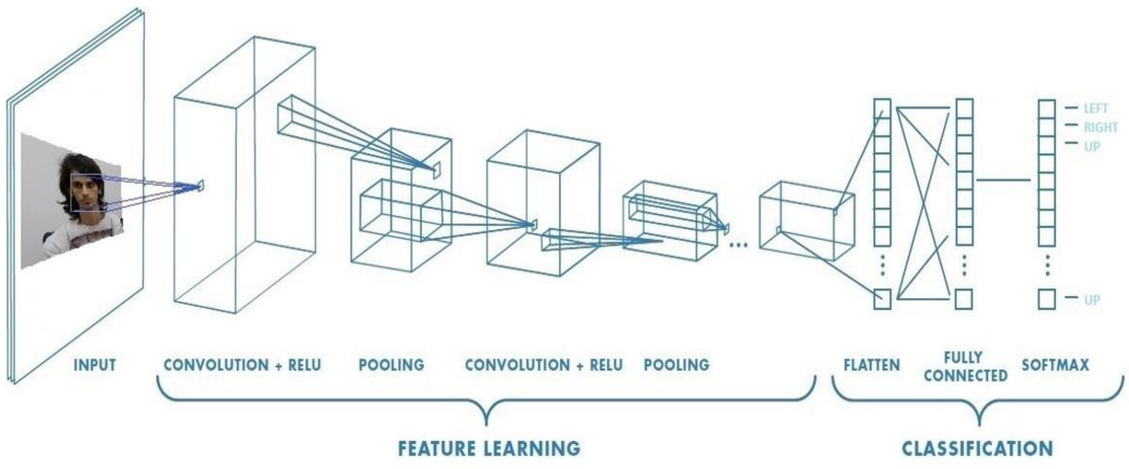


Figure 2.1: Typical Convolutional Neural Network Architecture

➢ Convolutional Layer

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each kernel is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input. Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map. Convolutional layer applies a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. In this way, it resolves the vanishing or exploding gradients problem in training traditional multi-layer neural networks with many layers by using backpropagation.

In Fig. 2 and Fig. 3, the green section presents a 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix.

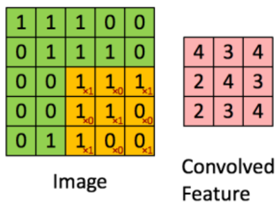


Figure 1.2 Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

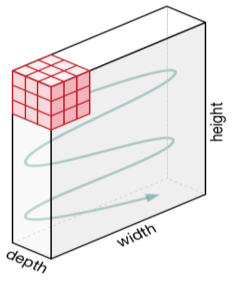


Figure 2.3: Movement of the Kernel

The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering. The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

In the case of images with multiple channels (e.g. RGB) as shown in Fig. 4, the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

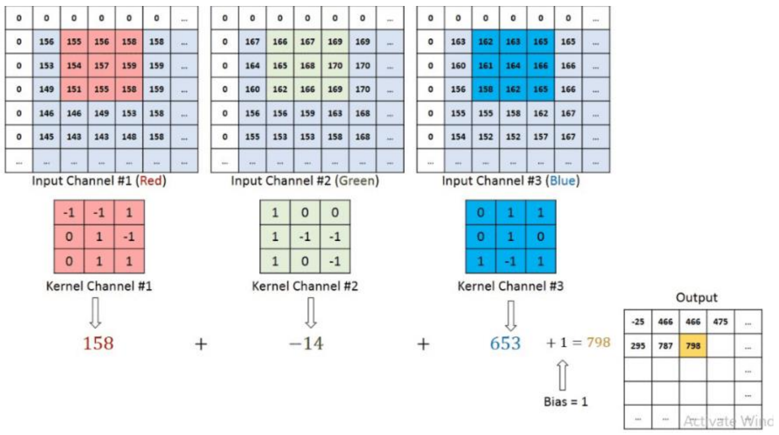


Figure 2.2: Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. CNN need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

➢ Pooling Layer:

The function of the pooling layer is to progressively reduce the spatial size of the convolved feature to reduce the amount of parameters and computation in the network. This layer decreases the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. Pooling layer operates on each feature map independently. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. Max Pooling also performs as a noise suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.

On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, it can be said that Max Pooling performs a lot better than Average Pooling. It is the most common approach used in pooling.

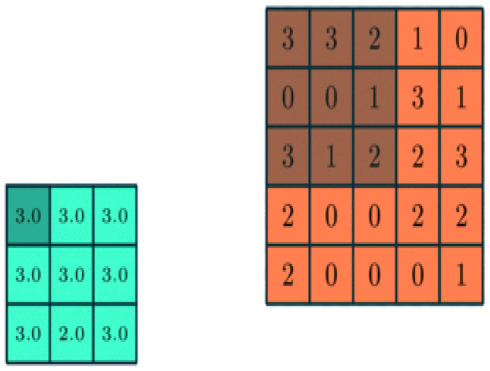


Figure 2.3: 3x3 pooling over 5x5 convolved feature

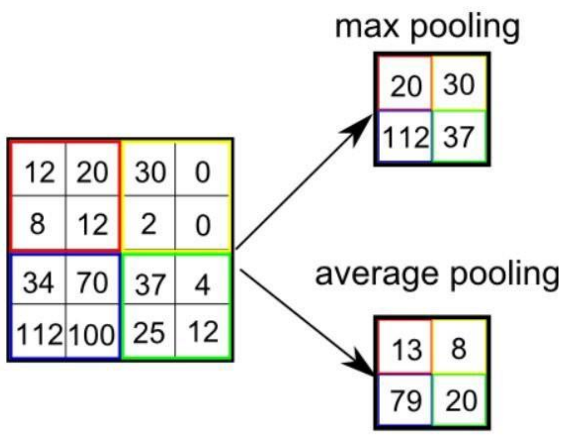


Figure 2.4: Types of Pooling

➢ Rectified Linear Unit (ReLU) Layer

ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function. ReLU transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable. It effectively removes negative values from an activation map by setting them to zero. It effectively removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent function and the sigmoid function. ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy.

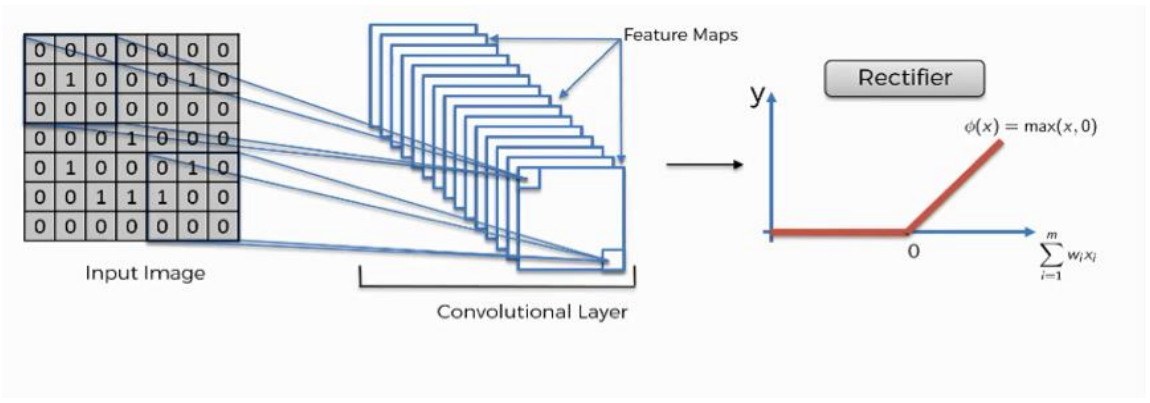


Figure 2.5: Rectified Linear Unit (ReLU) Layer

➢ **Fully Connected Layer (Full Connectedness)**

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (vector addition of a learned or fixed bias term). Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

The fully connected (FC) layer in the CNN represents the feature vector for the input. This feature vector/tensor/layer holds information that is vital to the input. When the network gets trained, this feature vector is further used for classification, regression, or as input into other network like RNN for translating into other type of output, etc.

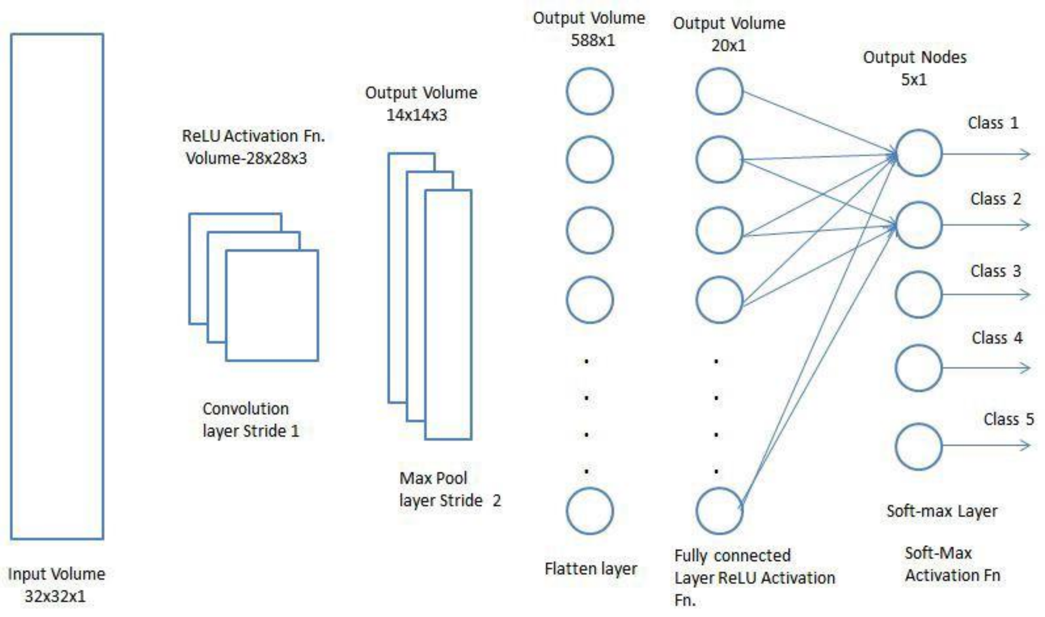


Figure 2. 6: Fully Connected Layer

During training, this feature vector is used to determine the loss, and it helps the network to get trained. The convolution layers before the FC layer(s) hold information regarding local features in the input image such as edges, blobs, shapes, etc. Each convolution layer holds several filters that represent one of the local features. The FC layer holds composite and aggregated information from all the convolution layers that matters the most.

After converting the input image into a suitable form for the Multi-Level Perceptron, the images are flattened into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation is applied to every iteration of training. Over a series of epochs, the model becomes able to distinguish between dominating and certain low-level features in images and then, the model can classify the images using the Softmax Classification technique.

There are various architectures of CNNs available which have been key in building algorithms which power and shall power Artificial Intelligence as a whole in the foreseeable future. Some of them have been listed below:

- LeNet
- AlexNet
- VGGNet
- GoogLeNet
- ResNet
- ZFNet

2.1.3 Data Sets in Machine Learning

Data sets used in machine learning is classified as the following:

1. Training Dataset
2. Validation Dataset
3. Test Dataset

(a) Training Dataset: The sample of data used to fit the model is known as training dataset. The model sees and learns from this data. It is the actual dataset that we use to train the model (weights and biases in the case of Neural Network).

(b) Validation Dataset: Validation dataset is the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyper-parameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.

The validation set is used to evaluate a given model, but this is for frequent evaluation. Machine learning engineers use this data to fine-tune the model hyper-parameters. Hence the model occasionally sees this data, but never does it "Learn" from this. The validation set results are then used and higher level hyper-parameters are updated. So the validation set in a way affects a model, but indirectly.

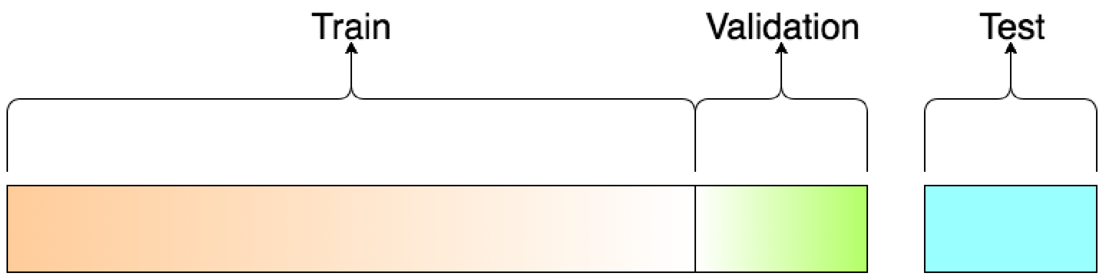


Figure 2.7: A visualization of the splits of dataset

(c) Test Dataset: Test dataset is the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained (using the train and validation sets). The test set is generally what is used to evaluate competing models (For example on many Kaggle competitions, the validation set is released initially along with the training set and the actual test set is only released when the competition is about to close, and it is the result of the model on the Test set that decides the winner). Many a times the validation set is used as the test set, but it is not good practice. The test set is generally well curated. It contains carefully sampled data that spans the various classes that the model would face, when used in the real world.

2.1.4 Parameters of Machine Learning:

1. **Validation Loss & Validation Accuracy:** The training is running in a well-defined manner or not is determined by the validation loss. If the validation loss is more than the system is learning in a wrong manner or extra nose in the data set.
2. **Training Loss & Training Accuracy:** At the time of training using the full dataset is known as accurate training. But due to the noise of the training data the error comes and this is called Training Loss and the accuracy is known as the opposite term of loss.
3. **Learning Rate:** It is the parameter that describes the rate how much well the model is learning. There are three types of them very high learning rate, low learning rate good learning rate.

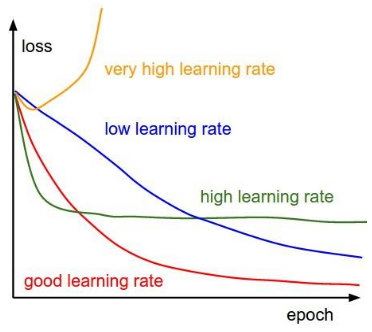


Figure 2.10: Parameters of Machine Learning

4. **Epoch:** A batch is the total number of training examples present in a single batch and an iteration is the number of batches need to complete in one epoch. If there are 2000 training and then there are 500 batches then need 4 iterations in one epoch.

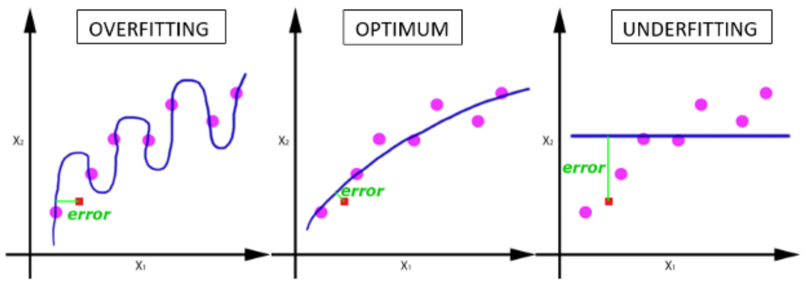


Figure 2. 8: Curve Fitting in Machine Learning

Confusion Matrix: A confusion matrix is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in dataset. Calculating a confusion matrix can give us a better idea of what your classification model is getting right and what types of errors it is making.

2.2 Proposed-Algorithm:

Total eye gaze direction detection process involves two major sequential processes: face and eye detection and eye gaze classification. This section describes proposed algorithms for these two processes.

2.2.1 Face Detection and Eye Localization:

Before feeding image in CNN, at first face and 64 facial landmark points are detected using Dlib-ml. Later, using facial landmarks, left eye and right eye regions are identified and separated. In 64 facial landmark systems, 37th and 40th landmark points indicate two corners of left eye and 43th and 46th landmarks points indicate two corner points of right eye. An extended rectangle covering these end points along y-axis is considered eye region.

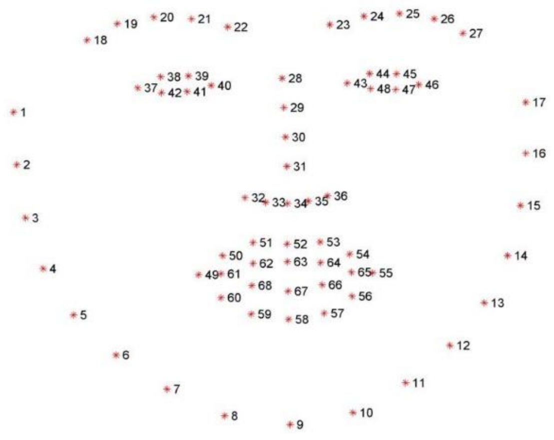
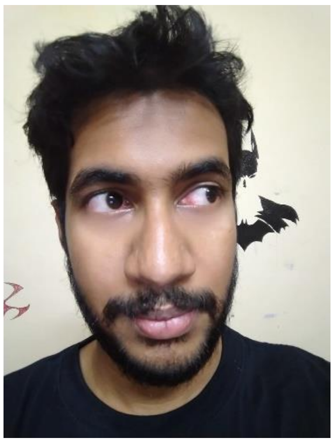


Figure 2.9: Positions of 64 facial landmark co-ordinates



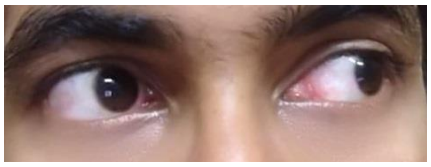
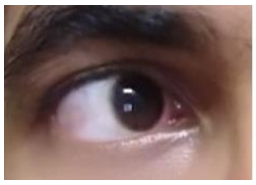


Figure 2.14: Detected Eye Region



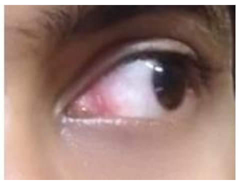


Figure 2.13: Full Image

(a) Right Eye

(b) Left Eye

Figure 2.10: Detected Eyes

2.2.2 Eye Gaze Classification:

a) Proposed CNN Architecture: The proposed network has two identical streams for left and right eye image. Each stream has three identical stages followed by a fully connected layer. Each stage starts with a convolution layer which is followed by a ReLU layer, a batch normalization layer and a max-pooling layer. The output $X_1^{(l)}$ can be described as

$X_1^{(l)}$ =MP (BN (max (0, $X_0^{(l)} * W_1^{(l)} + b_1^{(l)}$)))

Here, X_0 represents input image and superscript *(l)* indicates left eye. MP (.) and BN (.) denote max-pooling and batch normalization operation respectively. W and b indicate the weight matrix and bias term.

Similarly $X_2^{(l)}$ and $X_3^{(l)}$ can be written as

$X_2^{(l)}$ =MP (BN (max (0, $X_1^{(l)} * W_2^{(l)} + b_2^{(l)}$)))

$X_3^{(l)}$ =MP (BN (max (0, $X_2^{(l)} * W_3^{(l)} + b_3^{(l)}$)))

For another steam for right eyes,

$X_1^{(r)}$=MP(BN(max(0, $X_0^{(r)} * W_1^{(r)}$+$b_1^{(r)}$)))

$X_2^{(r)}$=MP(BN(max(0, $X_2^{(r)} * W_2^{(r)}$+$b_2^{(r)}$)))

$X_3^{(r)}$=MP(BN(max(0, $X_3^{(r)} * W_3^{(r)}$+ $b_3^{(r)}$)))

Flattened version of$X_3^{(l)}$ and$X_3^{(r)}$ are $x_3^{(l)}$ and$x_3^{(r)}$.

Concatenation layer concatenates$x_3^{(l)}$ and$x_3^{(r)}$ and gives$x_3^{(c)}$

$x_3^{(c)}$ = $x_3^{(l)}$ $||x_3^{(r)}$

A softmax operation acts on$x_3^{(c)}$ and finds out the class with maximum probability.

x=$W_4^{(c)T} x_3^{(c)} \underset{T}{+} b_4(c)$

$$P(y=j|x)=\frac{e^{x\quad w_j}}{\sum_{k=1}^{K} e^{x^T w_k}}$$

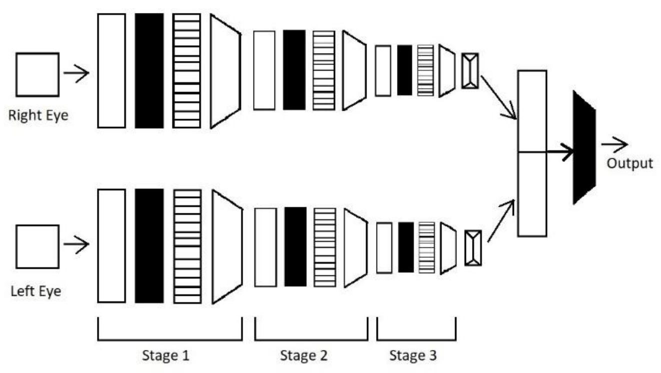


Figure 2.11: Proposed CNN Model for eye gaze direction classification. Convolution layer, ReLU layer, Batch Normalization layer, Max-Pooling layer, flattened layer, Concatenation layer, Softmax layer are represented by rectangle, solid line, striped rectangle

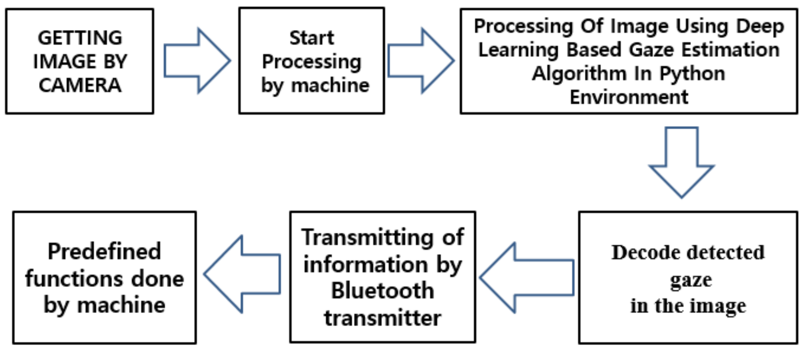

GETTING IMAGE BY CAMERA
Start Processing by machine
Processing Of Image Using Deep Learning Based Gaze Estimation Algorithm In Python Environment
Decode detected gaze in the image
Transmitting of information by Bluetooth transmitter
Predefined functions done by machine

Figure 2.12: Flowchart of actions

b) Training Scheme*:* Cross entropy loss is used as loss function. CNN model is trained using Adam algorithm with constant learning rate .0001. Weights are initialized with a normal distribution of standard deviation .01.

2.3 Experimental Details

2.3.1 Database

(a) Eye-Chimera: Eye-Chimera (Eye part from the Cognitive process Interference by the Mutual use of the Eye and expression Analysis) has altogether 1135 frames classified into 7 classes according to eye gaze directions. 7 classes are labelled as: center, downright, downleft, left, right, upleft, upright.

(b) HPEG: HPEG (Head Pose and Eye Gaze) dataset has 2 sections. Second section is for eye gaze estimation. It has videos of 10 subjects and contains 6 eye gaze position for each subject. Altogether it has 3187 frames and these frames are classified into 6 classes according to Yaw and Pitch angles.

TABLE I: HPEG Database

Labels	Head Position	Eye Gaze Direction
EYES C	Center	Center
EYES L	Center	Left
EYES R	Center	Right
HEAD R	Right	Center
EYES RL	Right	Left
EYES RR	Right	Right

(c) EGDC: This paper introduces new dataset named EGDC [Eye Gaze Direction Classification]. The dataset has two sections. Section A contains 1759 images of 31 subjects classified into directions. Nine classes are center, down, downleft, downright, left, right, up, upleft and upright. Eye gaze are labelled for different head positions. Section B has 3684 images of a single person classified into five classes (center, down, left, right and up).

TABLE II: Overview of Databases

Database Name	Image No.	Class No.	Person No
Eye Chimera	1135	7	40
HPEG	3187	6	10
EGDC(Section A)	1759	9	31
EGDC(Section B)	3684	5	1

2.3.2 Model Setup

64*64 size gray scale image is used as input image for each steam. For convolution layers in first three stages for each steam, number of filters are set to 32, 64,128 respectively and kernel size is 7*7 for all convolution layer with no stride. For max-pooling layers, kernel size is set to 3*3 with 2*2 stride. After these three stages, 128 features are extracted for each steam. The merged layer has total 256 features. In proposed CNN model there are total 1009061 parameters out of which 1008725 parameters are trainable.

TABLE III: Performance on Different Datasets for Proposed CNN model

Dataset Name	Eye Localization Accuracy	Gaze Direction Detection Accuracy	Eye Gaze Direction Classification Accuracy
Eye Chimera	.9815	.9192	.9021
HPEG	.9950	.9969	.9919
EGDC(Section A)	.9829	.8844	.8693
EGDC(Section B)	.9991	1	.9991

TABLE VI: Evaluated precision, recall and F1 score for HPE dataset

Class name	Precision	Recall	F1 score
EYES _C	1	1	1
EYES _L	1	1	1
EYES _R	1	1	1
EYES _RL	1	1	1
EYES_RR	.98	1	.99
HEAD_R	1.	.98	.99
Avg	.99	.99	.99

2.4 System Modules

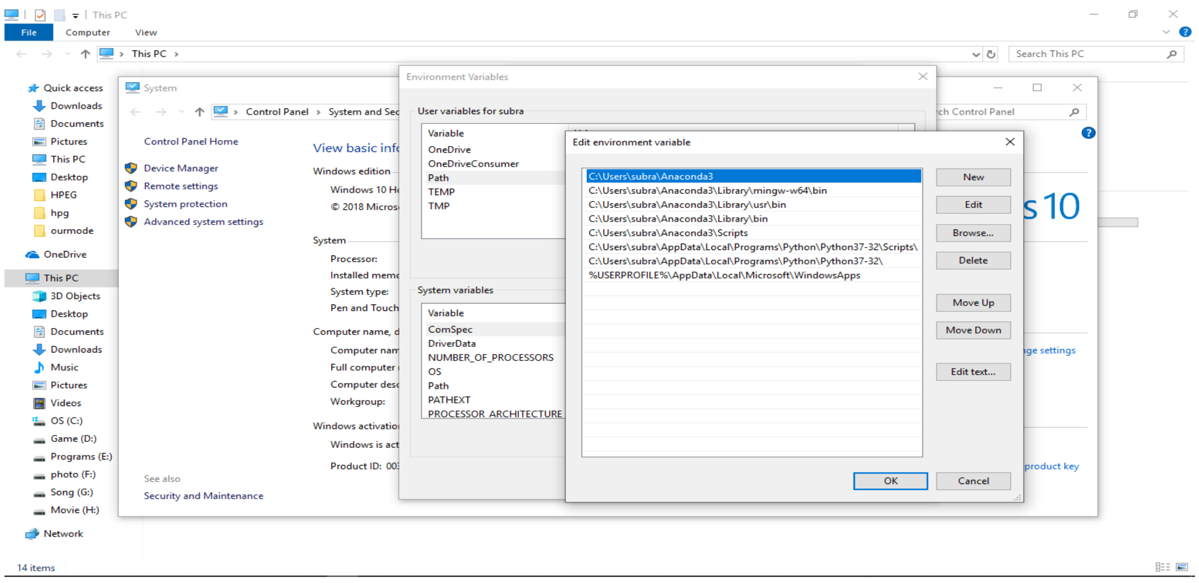
2.4.1 Python 3.7

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python Features:

1) Easy to Learn and Use
2) Expressive Language
3) Interpreted Language
4) Cross-platform Language
5) Free and Open Source
6) Object-Oriented Language
7) Extensible
8) Large Standard Library
9) GUI Programming Support
10) Integrated

Python has a huge predefined library that can easily accessible by any compiler. But the system should be modified by selecting the ENVIRONMENT VARIABLE as the path of python. Once the path is recognized then all the codes those will be developed in the compiler can use the python path.



Environment Variables

User variables for subra

Variable
OneDrive
OneDriveConsumer
Path
TEMP
TMP

System variables

Variable
ComSpec
DriverData
NUMBER_OF_PROCESSORS
OS
Path
PATHEXT
PROCESSOR_ARCHITECTURE

Edit environment variable

```
C:\Users\subra\Anaconda3
C:\Users\subra\Anaconda3\Library\mingw-w64\bin
C:\Users\subra\Anaconda3\Library\usr\bin
C:\Users\subra\Anaconda3\Library\bin
C:\Users\subra\Anaconda3\Scripts
C:\Users\subra\AppData\Local\Programs\Python\Python37-32\Scripts\
C:\Users\subra\AppData\Local\Programs\Python\Python37-32\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
```

Figure 2.13: Environment Variable Path

2.4.2 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It is available for Windows, macOS, and Linux.

To get Navigator, get the Navigator Cheat Sheet and install Anaconda.

2.4.3 Spyder

Spyder, the Scientific Python Development Environment, is a free integrated development environment (IDE) that is included with Anaconda. It includes editing, interactive testing, debugging and introspection features.After you have installed Anaconda, start Spyder on Windows, macOS or Linux by running the command spyder.Spyder is also pre-installed in Anaconda Navigator, included in Anaconda. On the Navigator Home tab, click the Spyder icon.

2.4.4 CUDA (NVIDIA)

CUDA Python is just one of the ways you can create massively parallel applications with CUDA. It lets you use the powerful Python programming language to develop high performance algorithms accelerated by thousands of parallel threads running on GPUs. Many developers have accelerated their computation- and bandwidth-hungry applications this way, including the libraries and frameworks that underpin the ongoing revolution in artificial intelligence known as Deep Learning.

So, you've heard about CUDA and you are interested in learning how to use it in your own applications. If you are a C or Python programmer, this blog post should give you a good start. To follow along, you'll need a computer with an CUDA-capable GPU (Windows, Mac, or Linux, and any NVIDIA GPU should do), or a cloud instance with GPUs (AWS, Azure, IBM SoftLayer, and other cloud service providers have them). You'll also need the free CUDA Toolkit installed.

2.4.5 TensorFlow Library

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It has been created by the Google Brain team.

Features:

TensorFlow provides stable Python and C APIs; and without API backwards compatibility guarantee: C++, Go, Java, JavaScript and Swift (early release). Third party packages are available for C#, Haskell, Julia, R,Scala, Rust, OCaml and Crystal. TensorFlow bundles together a slew of machine learning and deep learning (i.e. neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++. TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

How TensorFlow works

TensorFlow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.

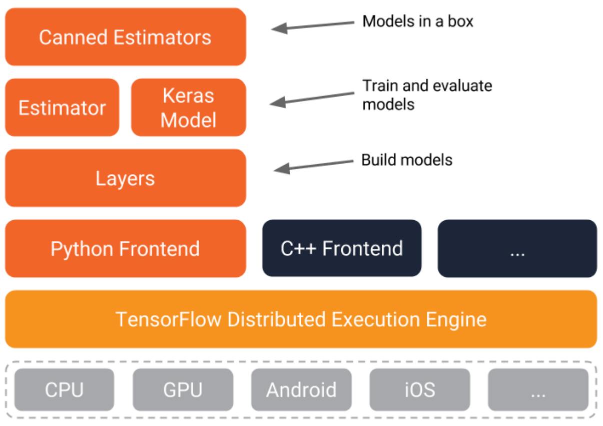


Figure 2.14: Tensorflow

TensorFlow provides all of this for the programmer by way of the Python language. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions can be coupled together. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.

TensorFlow applications can be run on most any target that’s convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If Google’s own cloud is used, TensorFlow can be run on Google’s custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

2.4.6 Keras Library

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.Use Keras if you need a deep learning library that:

Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).

Supports both convolutional networks and recurrent networks, as well as combinations of the two.

- Runs seamlessly on CPU and GPU.
- Read the documentation at Keras.io.

Why use Keras?

There are countless deep learning frameworks available today. Why use Keras rather than any other? Here are some of the areas in which Keras compares favorably to existing alternatives.

Keras prioritizes developer experience:

- Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- This makes Keras easy to learn and easy to use. As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn helps you win machine learning competitions.
- This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as tf.keras, the Keras API integrates seamlessly with your TensorFlow workflows.

Keras supports multiple backend engines and does not lock you into one ecosystem:

Keras models can be developed with a range of different deep learning backends. Importantly, any Keras model that only leverages built-in layers will be portable across all these backends: you can train a model with one backend, and load it with another (e.g. for deployment). Available backends include:

- The TensorFlow backend (from Google)
- The CNTK backend (from Microsoft)
- The Theano backend

About Keras models:

There are two main types of models available in Keras: the Sequential model, and the Model class used with the functional API.

These models have a number of methods and attributes in common:

model.layers is a flattened list of the layers comprising the model.

model.inputs is the list of input tensors of the model.

model.outputs is the list of output tensors of the model.

model.summary() prints a summary representation of your model. Shortcut for utils.print_summary model.get_config() returns a dictionary containing the configuration of the model. The model can be reinstantiated from its config via:

config = model.get_config()

model = Model.from_config(config)

# or, for Sequential:

model = Sequential.from_config(config)

model.get_weights() returns a list of all weight tensors in the model, as Numpy arrays.

model.set_weights(weights) sets the values of the weights of the model, from a list of Numpy arrays. The arrays in the list should have the same shape as those returned by get_weights().

model.to_json() returns a representation of the model as a JSON string. Note that the representation does not include the weights, only the architecture. You can reinstantiate the same model (with reinitialized weights) from the JSON string via.

HDF5 for Python:

The h5py package is a Python interface to the HDF5 binary data format.HDF5 lets you store huge amounts of numerical data, and easily manipulate that data from NumPy. For example, you can slice into multi-terabyte datasets stored on disk, as if they were real NumPy arrays. Thousands of datasets can be stored in a single file, categorized and tagged however you want. HDF5 is a unique technology suite that makes possible the management of extremely large and complex data collections.

The HDF5 technology suite includes:

- A versatile data model that can represent very complex data objects and a wide variety of metadata.
- A completely portable file format with no limit on the number or size of data objects in the collection.

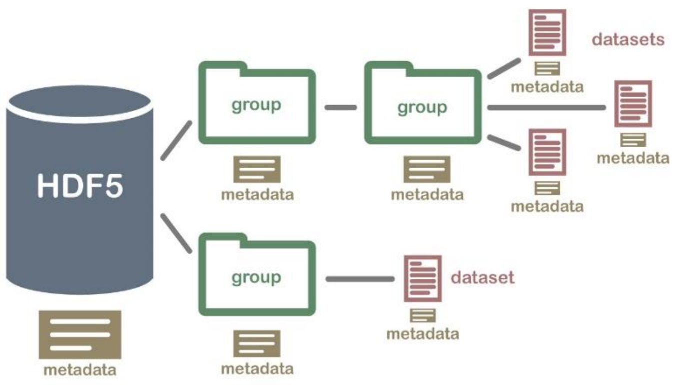


Figure 2.15: HDF5

- A software library that runs on a range of computational platforms, from laptops to massively parallel systems, and implements a high-level API with C, C++, Fortran 90, and Java interfaces.
- A rich set of integrated performance features that allow for access time and storage space optimizations. Tools and applications for managing, manipulating, viewing, and analysing the data in the collection.

CHAPTER 3
SYSTEM IMPLEMENTATION

System should be implemented in a consistence way and the functionality of the modules should be as defined in the system requirement.

3.1 Syntax Analysis

3.1.1 ImageDataGenerator class:

keras.preprocessing.image.ImageDataGenerator(featurewise_center=False, samplewise_center=False,featurewise_std_normalization=False,samplewise_std_normalization=False ,zca_whitening=False,zca_epsilon=1e06 ,rotation_range=0,width_shift_range=0.0,height_shift

_range=0.0,brightness_range=None,shear_range=0.0,zoom_range=0.0,channel_shift_range=0.0, fill_mode='nearest', cval=0.0, horizontal_flip=False, vertical_flip=False, rescale=None, preprocessing_function=None, data_format=None, validation_split=0.0, dtype=None)

Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches).

Arguments:

featurewise_center: Boolean. Set input mean to 0 over the dataset, feature-wise.

samplewise_center: Boolean. Set each sample mean to 0.

featurewise_std_normalization: Boolean. Divide inputs by std of the dataset, feature-wise.

samplewise_std_normalization: Boolean. Divide each input by its std.

zca_epsilon: epsilon for ZCA whitening. Default is 1e-6.

zca_whitening: Boolean. Apply ZCA whitening.

rotation_range: Int. Degree range for random rotations.

width_shift_range: Float, 1-D array-like or int float: fraction of total width, if < 1, or pixels if >= 1.

1-D array-like: random elements from the array.

int: integer number of pixels from interval (-width_shift_range, +width_shift_range)

With width_shift_range=2 possible values are integers [-1, 0, +1], same as with width_shift_range=[-1, 0, +1], while with width_shift_range=1.0 possible values are floats in the half-open interval [-1.0, +1.0[.

height_shift_range: Float, 1-D array-like or int

float: fraction of total height, if < 1, or pixels if >= 1.

1-D array-like: random elements from the array.

int: integer number of pixels from interval (-height_shift_range, +height_shift_range)

With height_shift_range=2 possible values are integers [-1, 0, +1], same as with height_shift_range=[-1, 0, +1], while with height_shift_range=1.0 possible values are floats in the half-open interval [-1.0, +1.0[.

brightness_range: Tuple or list of two floats. Range for picking a brightness shift value from.

shear_range: Float. Shear Intensity (Shear angle in counter-clockwise direction in degrees)

zoom_range: Float or [lower, upper]. Range for random zoom. If a float, [lower, upper] =

[1-zoom_range, 1+zoom_range]**.**

channel_shift_range: Float. Range for random channel shifts.

fill_mode: One of {"constant", "nearest", "reflect" or "wrap"}. Default is 'nearest'. Points outside the boundaries of the input are filled according to the given mode:

'constant': kkkkkkkk|abcd|kkkkkkkk (cval=k)

'nearest': aaaaaaaa|abcd|dddddddd

'reflect': abcddcba|abcd|dcbaabcd

'wrap': abcdabcd|abcd|abcdabcd

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data ()
y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)

datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True
,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)
datagen.fit(x_train)
model.fit_generator(datagen.flow(x_train, y_train, batch_size=32),
                    steps_per_epoch=len(x_train) / 32, epochs=epochs)
```

```
# here's a more "manual" example
for e in range(epochs):
    print ('Epoch', e)
    batches = 0
    for x_batch, y_batch in datagen.flow(x_train, y_train, batch_size=32):
        model.fit(x_batch, y_batch)
        batches += 1
        if batches >= len(x_train) / 32:
            # we need to break the loop by hand because
            # the generator loops indefinitely
            break
```

- cval: Float or Int. Value used for points outside the boundaries when fill_mode = "constant".
- horizontal_flip: Boolean. Randomly flip inputs horizontally.
- vertical_flip: Boolean. Randomly flip inputs vertically.

Rescale: rescaling factor. Defaults to None. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (after applying all other transformations).

Preprocessing_Function: function that will be implied on each input. The function will run after the image is resized and augmented. The function should take one argument: one image (Numpy tensor with rank 3), and should output a Numpy tensor with the same shape.

Data_Format: Image data format, either "channels_first" or "channels_last". "channels_last" mode means that the images should have shape (samples, height, width, channels), "channels_first" mode means that the images should have shape (samples, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Validation_split: Float. Fraction of images reserved for validation (strictly between 0 and 1).

Dtype: Dtype to use for the generated arrays.

Fit:

```
fit (x, augment=False, rounds=1, seed=None)
```

Fits the data generator to some sample data.This computes the internal data stats related to the data-dependent transformations, based on an array of sample data.Only required if featurewise_center or featurewise_std_normalization or zca_whitening are set to True.

Arguments:

x: Sample data. Should have rank 4. In case of grayscale data, the channels axis should have value 1, in case of RGB data, it should have value 3, and in case of RGBA data, it should have value 4.

Augment: Boolean (default: False). Whether to fit on randomly augmented samples.

Rounds: Int (default: 1). If using data augmentation (augment=True), this is how many augmentation passes over the data to use.

Seed: Int (default: None). Random seed.

Standardize:

standardize(x)

Applies the normalization configuration in-place to a batch of inputs. x is changed in-place since the function is mainly used internally to standarize images and feed them to your network. If a copy of x would be created instead it would have a significant performance cost. If you want to apply this method without changing the input in-place you can call the method creating a copy before:

```
standarize(np.copy(x))
```

Arguments:

x: Batch of inputs to be normalized.

Returns:

The inputs, normalized.

flow_from_dataframe:

flow_from_dataframe(dataframe, directory=None, x_col='filename', y_col='class', target_size=(256, 256), color_mode='rgb', classes=None, class_mode='categorical', batch_size=32, shuffle=True, seed=None, save_to_dir=None, save_prefix='', save_format='png', subset=None, interpolation='nearest', drop_duplicates=True)

Takes the data frame and the path to a directory and generates batches of augmented/normalized data. A simple tutorial can be found here.

Arguments:

dataframe: Pandas dataframe containing the filepaths relative to directory (or absolute paths if directory is None) of the images in a string column. It should include other column/s depending on the class_mode: - if class_mode is "categorical" (default value) it must include the y_col column with the class/es of each image. Values in column can be string/list/tuple if a single class or list/tuple if multiple classes. - if class_mode is "binary" or "sparse" it must include the given y_col column with class values as strings. - if class_mode is "other" it should contain the columns specified in y_col. - if class_mode is "input" or None no extra column is needed.

directory: string, path to the directory to read images from. If None, data in x_col column should be absolute paths.

x_col: string, column in dataframe that contains the filenames (or absolute paths if directory is None).

y_col: string or list, column/s in dataframe that has the target data.

target_size: tuple of integers (height, width), default: (256, 256). The dimensions to which all images found will be resized.

color_mode: one of "grayscale", "rgb". Default: "rgb". Whether the images will be converted to have 1 or 3 color channels.

classes: optional list of classes (e.g. ['dogs', 'cats']). Default: None. If not provided, the list of classes will be automatically inferred from the y_col, which will map to the label indices, will be alphanumeric). The dictionary containing the mapping from class names to class indices can be obtained via the attribute class_indices.

class_mode: one of "categorical", "binary", "sparse", "input", "other" or None. Default: "categorical". Mode for yielding the targets:

"binary": 1D numpy array of binary labels,

"categorical": 2D numpy array of one-hot encoded labels. Supports multi-label output.

"sparse": 1D numpy array of integer labels,

"input": images identical to input images (mainly used to work with autoencoders),

"other": numpy array of y_col data,

None, no targets are returned (the generator will only yield batches of image data, which is useful to use in model.predict_generator()).

batch_size: size of the batches of data (default: 32).

shuffle: whether to shuffle the data (default: True)

seed: optional random seed for shuffling and transformations.

save_to_dir: None or str (default: None). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).

save_prefix: str. Prefix to use for filenames of saved pictures (only relevant if save_to_dir is set).

save_format: one of "png", "jpeg" (only relevant if save_to_dir is set). Default: "png".

follow_links: whether to follow symlinks inside class subdirectories (default: False).

subset: Subset of data ("training" or "validation") if validation_split is set in ImageDataGenerator.

Interpolation: Interpolation method used to resample the image if the target size is different from that of the loaded image. Supported methods are "nearest", "bilinear", and "bicubic". If PIL version 1.1.3 or newer is installed, "lanczos" is also supported. If PIL version 3.4.0 or newer is installed, "box" and "hamming" are also supported. By default, "nearest" is used. drop_duplicates: Boolean, whether to drop duplicate rows based on filename.

Returns:

A DataFrameIterator yielding tuples of (x, y) where x is a numpy array containing a batch of images with shape (batch_size, *target_size, channels) and y is a numpy array of corresponding labels.

flow_from_directory:

flow_from_directory(directory, target_size=(256, 256), color_mode='rgb', classes=None, class_mode='categorical', batch_size=32, shuffle=True, seed=None, save_to_dir=None, save_prefix='', save_format='png', follow_links=False, subset=None, interpolation='nearest') .Takes the path to a directory & generates batches of augmented data.

Arguments:

directory: string, path to the target directory. It should contain one subdirectory per class. Any PNG, JPG, BMP, PPM or TIF images inside each of the subdirectories directory tree will be included in the generator. See this script for more details.

target_size: Tuple of integers (height, width), default: (256, 256). The dimensions to which all images found will be resized.

color_mode: One of "grayscale", "rgb", "rgba". Default: "rgb". Whether the images will be converted to have 1, 3, or 4 channels.

classes: Optional list of class subdirectories (e.g. ['dogs', 'cats']). Default: None. If not provided, the list of classes will be automatically inferred from the subdirectory names/structure under directory, where each subdirectory will be treated as a different class (and the order of the classes, which will map to the label indices, will be alphanumeric). The dictionary containing the mapping from class names to class indices can be obtained via the attribute class_indices.

class_mode: One of "categorical", "binary", "sparse", "input", or None. Default: "categorical". Determines the type of label arrays that are returned:

"categorical" will be 2D one-hot encoded labels, "binary" will be 1D binary labels, "sparse" will be 1D integer labels,

"input" will be images identical to input images (mainly used to work with autoencoders).

If None, no labels are returned (the generator will only yield batches of image data, which is useful to use with model.predict_generator()). Please note that in case of class_mode None, the data still needs to reside in a subdirectory of directory for it to work correctly.

batch_size: Size of the batches of data (default: 32).

shuffle: Whether to shuffle the data (default: True) If set to False, sorts the data in alphanumeric order.

seed: Optional random seed for shuffling and transformations.

save_to_dir: None or str (default: None). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).

save_prefix: Str. Prefix to use for filenames of saved pictures (only relevant if save_to_dir is set).

save_format: One of "png", "jpeg" (only relevant if save_to_dir is set). Default: "png".

follow_links: Whether to follow symlinks inside class subdirectories (default: False).

subset: Subset of data ("training" or "validation") if validation_split is set in ImageDataGenerator.

Interpolation method used to resample the image if the target size is different from that of the loaded image. Supported methods are "nearest", "bilinear", and "bicubic". If PIL version 1.1.3 or newer is installed, "lanczos" is also supported. If PIL version 3.4.0 or newer is installed, “box" and "hamming" are also supported. By default, "nearest" is used.

Returns:

A Directory Iterator yielding tuples of (x, y) where x is a numpy array containing a batch of images with shape (batch_size, *target_size, channels) and y is a numpy array of corresponding labels.

get_random_transform:

get_random_transform(img_shape, seed=None)

Generates random parameters for a transformation.

Arguments:

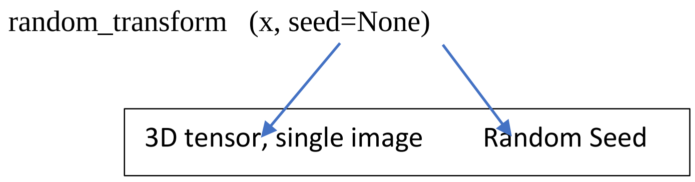
seed: Random seed.

img_shape: Tuple of integers. Shape of the image that is transformed.

Returns:

A dictionary containing randomly chosen parameters describing the transformation.

random_transform:



Applies a random transformation to an image.

Arguments:

x: 3D tensor, single image.

seed: Random seed.

Returns:

A randomly transformed version of the input (same shape).

3.1.2 Conv1D, Conv2D, ZeroPadding1D, ZeroPadding2D by Keras:

Conv1D:

keras.layers.Conv1D(filters,kernel_size,strides=1,padding='valid', data_format='

channels_last',dilation_rate=1,activation=None,use_bias=True, kernel_initializer='glorot_uniform

',bias_initializer='zeros',kernel_regularizer=None,bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)

1D convolution layer (e.g. temporal convolution).

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If use_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide an input_shape argument (tuple of integers or None, does not include the batch axis), e.g. input_shape=(10, 128) for time series sequences of 10 time steps with 128 features per step in data_format="channels_last", or (None, 128) for variable-length sequences with 128 features per step.

Arguments:

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

kernel_size: An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.

strides: An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value! = 1 is incompatible with specifying any dilation_rate value! = 1.

padding: One of "valid", "causal" or "same" (case-insensitive). "valid" means "no padding". "same" results in padding the input such that the output has the same length as the original input. "causal" results in causal (dilated) convolutions, e.g. output[t] does not depend on input [t + 1:]. A zero padding is used such that the output has the same length as the original input. Useful when modelling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.

data_format: A string, one of "channels_last" (default) or "channels_first". The ordering of the dimensions in the inputs. "channels_last" corresponds to inputs with shape (batch, steps, channels) (default format for temporal data in Keras) while "channels_first" corresponds to inputs with shape (batch, channels, steps).

dilation_rate: an integer or tuple/list of a single integer, specifying the dilation rate to use for dilated convolution. Currently, specifying any dilation_rate value! = 1 is incompatible with specifying any strides value! = 1.

activation: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).

use_bias: Boolean, whether the layer uses a bias vector.

kernel_initializer: Initializer for the kernel weights matrix (see initializers).

bias_initializer: Initializer for the bias vector (see initializers).

kernel_regularizer: Regularizer function applied to the kernel weights matrix (see regularizer).

bias_regularizer: Regularizer function applied to the bias vector (see regularizer).

activity_regularizer: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).

kernel_constraint: Constraint function applied to the kernel matrix (see constraints).

bias_constraint: Constraint function applied to the bias vector (see constraints).

Input shape

3D tensor with shape: (batch, steps, channels)

Output shape:

3D tensor with shape: (batch, new_steps, filters) steps value might have changed due to padding or strides.

Conv2D:

keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)

2D convolution layer (e.g. spatial convolution over images).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument input_shape (tuple of integers, does not include the batch axis), e.g. input_shape=(128, 128, 3) for 128x128 RGB pictures in data_format="channels_last".

Arguments:

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value! = 1 is incompatible with specifying any dilation_rate value! = 1.

padding: one of "valid" or "same" (case-insensitive). Note that "same" is slightly inconsistent across backends with strides! = 1, as described here

data_format: A string, one of "channels_last" or "channels_first". The ordering of the dimensions in the inputs. "channels_last" corresponds to inputs with shape (batch, height, width, channels) while "channels_first" corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

dilation_rate: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value! = 1 is incompatible with specifying any stride value! = 1.

activation: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).

use_bias: Boolean, whether the layer uses a bias vector.

kernel_initializer: Initializer for the kernel weights matrix (see initializers).

bias_initializer: Initializer for the bias vector (see initializers).

kernel_regularizer: Regularizer function applied to the kernel weights matrix (see regularizer).

bias_regularizer: Regularizer function applied to the bias vector (see regularizer).

activity_regularizer: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).

kernel_constraint: Constraint function applied to the kernel matrix (see constraints).

bias_constraint: Constraint function applied to the bias vector (see constraints).

Input shape:

4D tensor with shape: (batch, channels, rows, cols) if data_format is "channels_first" or 4D tensor with shape: (batch, rows, cols, channels) if data_format is "channels_last".

Output shape:

4D tensor with shape: (batch, filters, new_rows, new_cols) if data_format is "channels_first" or 4D tensor with shape: (batch, new_rows, new_cols, filters) if data_format is "channels_last". rows and cols values might have changed due to padding.

ZeroPadding1D:

keras.layers.ZeroPadding1D(padding=1)

Zero-padding layer for 1D input (e.g. temporal sequence).

Arguments:

padding: int, or tuple of int (length 2), or dictionary.

If int: How many zeros to add at the beginning and end of the padding dimension (axis 1).

If tuple of int (length 2):

How many zeros to add at the beginning and at the end of the padding dimension ((left_pad, right_pad)).

Input shape:

3D tensor with shape (batch, axis_to_pad, features)

Output shape

3D tensor with shape (batch, padded_axis, features)

ZeroPadding2D

keras.layers.ZeroPadding2D(padding=(1, 1), data_format=None)

Zero-padding layer for 2D input (e.g. picture).

This layer can add rows and columns of zeros at the top, bottom, left and right side of an image tensor.

Arguments:

padding: int, or tuple of 2 ints, or tuple of 2 tuples of 2 ints.

If int: the same symmetric padding is applied to height and width.

If tuple of 2 ints: interpreted as two different symmetric padding values for height and width: (symmetric_height_pad, symmetric_width_pad).

If tuple of 2 tuples of 2 ints: interpreted as ((top_pad, bottom_pad), (left_pad, right_pad))

data_format: A string, one of "channels_last" or "channels_first". The ordering of the dimensions in the inputs. "channels_last" corresponds to inputs with shape (batch, height, width, channels) while "channels_first" corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Input shape:

4D tensor with shape: - If data_format is "channels_last": (batch, rows, cols, channels) - If data_format is "channels_first": (batch, channels, rows, cols)

Output shape:

4D tensor with shape: - If data_format is "channels_last": (batch, padded_rows, padded_cols, channels) **-** If data_format is "channels_first": (batch, channels, padded_rows, padded_cols)

3.1.3 MaxPooling1D

keras.layers.MaxPooling1D(pool_size=2, strides=None, padding='valid', data_format='channels_last')

Max pooling operation for temporal data.

Arguments

pool_size: Integer, size of the max pooling windows.

strides: Integer, or None. Factor by which to downscale. E.g. 2 will halve the input. If None, it will default to pool_size.

padding: One of "valid" or "same" (case-insensitive).

data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, steps, features) while channels_first corresponds to inputs with shape (batch, features, steps).

Input shape

If data_format='channels_last': 3D tensor with shape: (batch_size, steps, features)

If data_format='channels_first': 3D tensor with shape: (batch_size, features, steps)

Output shape

If data_format='channels_last': 3D tensor with shape: (batch_size, downsampled_steps, features)

If data_format='channels_first': 3D tensor with shape: (batch_size, features, downsampled_steps)

3.1.4 MaxPooling2D

keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)

Max pooling operation for spatial data.

Arguments

pool_size: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.

strides: Integer, tuple of 2 integers, or None. Strides values. If None, it will default to pool_size.

padding: One of "valid" or "same" (case-insensitive).

data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Input shape

If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

Output shape

If data_format='channels_last': 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, pooled_rows, pooled_cols)

3.1.5 AveragePooling2D

keras.layers.AveragePooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)

Average pooling operation for spatial data.

Arguments

pool_size: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.

strides: Integer, tuple of 2 integers, or None. Strides values. If None, it will default to pool_size.

padding: One of "valid" or "same" (case-insensitive).

data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Input shape

If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

Output shape

If data_format='channels_last': 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, pooled_rows, pooled_cols)

Usage of loss functions

A loss function (or objective function, or optimization score function) is one of the two parameters required to compile a model:

model.compile(loss='mean_squared_error', optimizer='sgd') from keras import losses

model.compile(loss=losses.mean_squared_error, optimizer='sgd')

An existing loss function, or pass a TensorFlow/Theano symbolic function that returns a scalar for each data-point and takes the following two arguments:

y_true: True labels. TensorFlow/Theano tensor.

y_pred: Predictions. TensorFlow/Theano tensor of the same shape as y_true.

The actual optimized objective is the mean of the output array across all datapoints.

Arguments

y_true: tensor of true targets.

y_pred: tensor of predicted targets.

Returns

Tensor with one scalar loss entry per sample.

categorical_crossentropy

keras.losses.categorical_crossentropy(y_true, y_pred)

sparse_categorical_crossentropy

keras.losses.sparse_categorical_crossentropy(y_true, y_pred)

binary_crossentropy

keras.losses.binary_crossentropy(y_true, y_pred)

kullback_leibler_divergence

keras.losses.kullback_leibler_divergence(y_true, y_pred)

poisson

keras.losses.poisson(y_true, y_pred)

cosine_proximity

keras.losses.cosine_proximity(y_true, y_pred)

from keras.utils import to_categorical

categorical_labels = to_categorical(int_labels, num_classes=None)

When using the sparse_categorical_crossentropy loss, your targets should be integer targets. If you have categorical targets, you should use categorical_crossentropy.

Usage of metrics

A metric is a function that is used to judge the performance of your model. Metric functions are to be supplied in the metrics parameter when a model is compiled.A metric function is similar to a loss function, except that the results from evaluating a metric are not used when training the model. You may use any of the loss functions as a metric function.

Arguments

y_true: True labels. Theano/TensorFlow tensor.

y_pred: Predictions. Theano/TensorFlow tensor of the same shape as y_true.

Returns

Single tensor value representing the mean of the output array across all datapoints.

Available metrics

binary_accuracy

keras.metrics.binary_accuracy(y_true, y_pred)

categorical_accuracy

keras.metrics.categorical_accuracy(y_true, y_pred)

sparse_categorical_accuracy

keras.metrics.sparse_categorical_accuracy(y_true, y_pred)

top_k_categorical_accuracy

keras.metrics.top_k_categorical_accuracy(y_true, y_pred, k=5)

sparse_top_k_categorical_accuracy

keras.metrics.sparse_top_k_categorical_accuracy(y_true, y_pred, k=5)

In addition to the metrics above, you may use any of the loss functions described in the loss function page as metrics.

Custom metrics:

Custom metrics can be passed at the compilation step. The function would need to take (y_true, y_pred) as arguments and return a single tensor value.

```
import keras.backend as K

def mean_pred(y_true, y_pred):

    return K.mean(y_pred)

model.compile(optimizer='rmsprop',

              loss='binary_crossentropy',

              metrics=['accuracy', mean_pred])
```

Usage of activations:

Activations can either be used through an Activation layer, or through the activation argument supported by all forward layers:

```
from keras.layers import Activation, Dense

model.add(Dense(64))

model.add(Activation('tanh'))
```

This is equivalent to:

```
model.add(Dense(64, activation='tanh'))
```

You can also pass an element-wise TensorFlow/Theano/CNTK function as an activation:

```
from keras import backend as K

model.add(Dense(64, activation=K.tanh))
```

Available activations

softmax

keras.activations.softmax(x, axis=-1)

Softmax activation function.

Arguments:

x: Input tensor.

axis: Integer, axis along which the softmax normalization is applied.

Returns:

Tensor, output of softmax transformation.

Raises

ValueError: In case dim(x) == 1.

Elu

keras.activations.elu(x, alpha=1.0)

Exponential linear unit.

Arguments

x: Input tensor.

alpha: A scalar, slope of negative section.

Returns

The exponential linear activation: x if x > 0 and alpha * (exp(x)-1) if x < 0.

With default values, it returns element-wise max(x, 0).

Otherwise, it follows: f(x) = max_value for x >= max_value, f(x) = x for threshold <= x < max_value, f(x) = alpha * (x - threshold) otherwise.

Arguments

x: Input tensor.

alpha: float. Slope of the negative part. Defaults to zero.

max_value: float. Saturation threshold.

threshold: float. Threshold value for thresholded activation.

Returns

A tensor.

tanh

keras.activations.tanh(x)

Hyperbolic tangent activation function.

sigmoid

keras.activations.sigmoid(x)

Wrappers for the Scikit-Learn API

You can use Sequential Keras models (single-input only) as part of your Scikit-Learn workflow via the wrappers found at keras.wrappers.scikit_learn.py.

There are two wrappers available:

keras.wrappers.scikit_learn.KerasClassifier(build_fn=None, **sk_params), which implements the Scikit-Learn classifier interface,

keras.wrappers.scikit_learn.KerasRegressor(build_fn=None, **sk_params), which implements the Scikit-Learn regressor interface.

Arguments

build_fn: callable function or class instance

sk_params: model parameters & fitting parameters

build_fn should construct, compile and return a Keras model, which will then be used to fit/predict. One of the following three values could be passed to build_fn:

A function

An instance of a class that implements the __call__ method

None. This means you implement a class that inherits from either KerasClassifier or KerasRegressor. The __call__ method of the present class will then be treated as the default build_fn.

sk_params takes both model parameters and fitting parameters. Legal model parameters are the arguments of build_fn. Note that like all other estimators in scikit-learn, build_fn should provide default values for its arguments, so that you could create the estimator without passing any values to sk_params.

sk_params could also accept parameters for calling fit, predict, predict_proba, and score methods (e.g., epochs, batch_size). fitting (predicting) parameters are selected in the following order:

Values passed to the dictionary arguments of fit, predict, predict_proba, and score methods Values passed to sk_params.The default values of the keras.models.Sequential fit, predict, predict_proba and score methods

When using scikit-learn's grid_search API, legal tunable parameters are those you could pass to sk_params, including fitting parameters. In other words, you could use grid_search to search for the best batch_size or epochs as well as the model parameters.

3.1.6 Getting started with the Keras functional API:

The Keras functional API is the way to go for defining complex models, such as multi-output models, directed acyclic graphs, or models with shared layers.

First example: a densely-connected network.The Sequential model is probably a better choice to implement such a network, but it helps to start with something really simple.

A layer instance is callable (on a tensor), and it returns a tensor.

Input tensor(s) and output tensor(s) can then be used to define a Model

Such a model can be trained just like Keras Sequential models.

```
from keras.layers import Input, Dense

from keras.models import Model

# This returns a tensor

    inputs = Input(shape=(784,))

    # a layer instance is callable on a tensor, and returns a tensor
    x = Dense(64, activation='relu')(inputs)
    x = Dense(64, activation='relu')(x)
    predictions = Dense(10, activation='softmax')(x)

    # This creates a model that includes
    # the Input layer and three Dense layers
```

```
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

All models are callable, just like layers, with the functional API, it is easy to reuse trained models: you can treat any model as if it were a layer, by calling it on a tensor. Note that by calling a model you aren't just reusing the architecture of the model,

```
x = Input(shape=(784,))

# This works, and returns the 10-way softmax we defined above.
y = model(x)
```

This can allow, for instance, to quickly create models that can process sequences of inputs.

```
from keras.layers import TimeDistributed

# Input tensor for sequences of 20 timesteps,

# each containing a 784-dimensional vector
input_sequences = Input(shape=(20, 784))

# This applies our previous model to every timestep in the input sequences.

# the output of the previous model was a 10-way softmax,
# so the output of the layer below will be a sequence of 20 vectors of size 10.
processed_sequences = TimeDistributed(model)(input_sequences)
```

Multi-input and multi-output models

Here's a good use case for the functional API: models with multiple inputs and outputs. The functional API makes it easy to manipulate a large number of intertwined datastreams.

Example:

Let's consider the following model. We seek to predict how many retweets and likes a news headline will receive on Twitter. The main input to the model will be the headline itself, as a sequence of words, but to spice things up, our model will also have an auxiliary input, receiving extra data such as the time of day when the headline was posted, etc. The model will also be supervised via two loss functions. Using the main loss function earlier in a model is a good regularization mechanism for deep models.

Here's what our model looks like:

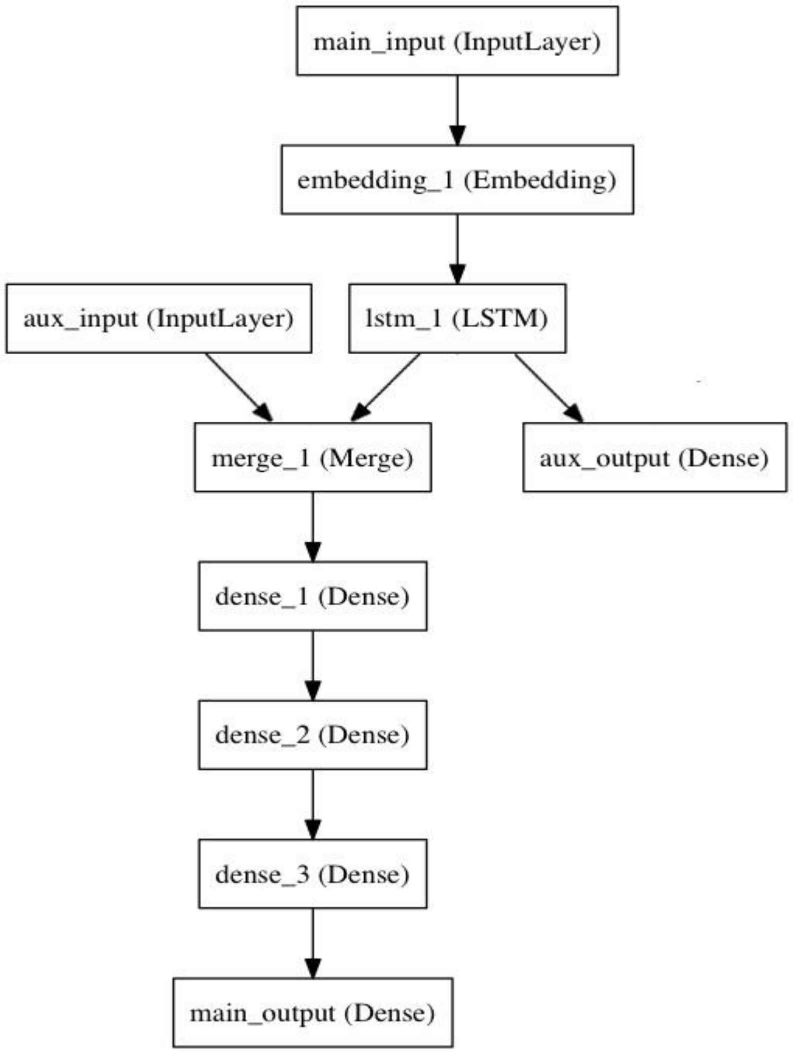


Figure 16: Functional API

Shared layers:

Another good use for the functional API are models that use shared layers.

Example:

To build a model that can tell whether two tweets are from the same person or not (this can allow us to compare users by the similarity of their tweets, for instance).

One way to achieve this is to build a model that encodes two tweets into two vectors, concatenates the vectors and then adds a logistic regression; this outputs a probability that the two tweets share the same author. The model would then be trained on positive tweet pairs and negative tweet pairs.Because the problem is symmetric, the mechanism that encodes the first tweet should be reused (weights and all) to encode the second tweet. Here we use a shared LSTM layer to encode the tweets to Take as input for a tweet a binary matrix of shape (280, 256), i.e. a sequence of 280 vectors of size 256, where each dimension in the 256-dimensional vector encodes the presence/absence of a character (out of an alphabet of 256 frequent characters).

```
import keras

from keras.layers import Input, LSTM, Dense

from keras.models import Model

tweet_a = Input(shape=(280, 256))

tweet_b = Input(shape=(280, 256))
```

To share a layer across different inputs, simply instantiate the layer once, then call it on as many inputs as you want:

```
# This layer can take as input a matrix

# and will return a vector of size 64
shared_lstm = LSTM(64)
# When we reuse the same layer instance  multiple times, the weights of the layer are also being reused# (it is effectively *the same* layer)
encoded_a = shared_lstm(tweet_a)
encoded_b = shared_lstm(tweet_b)

# We can then concatenate the two vectors:

merged_vector = keras.layers.concatenate([encoded_a, encoded_b], axis=-1)

# And add a logistic regression on top

predictions = Dense(1, activation='sigmoid')(merged_vector)

# We define a trainable model linking the

# tweet inputs to the predictions
model = Model(inputs=[tweet_a, tweet_b], outputs=predictions)
model.compile(optimizer='rmsprop',
          loss='binary_crossentropy',
        metrics=['accuracy'])
model.fit([data_a, data_b], labels, epochs=10)
```

Usage of optimizers:

An optimizer is one of the two arguments required for compiling a Keras model:

```
from keras import optimizers :
model = Sequential()
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(Activation('softmax'))
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
#As in the above example, or you can call it by its name.
#In the latter case, the default #parameters for the optimizer will be used.
# pass optimizer by name: default parameters will be used
model.compile(loss='mean_squared_error', optimizer='sgd')
```

Parameters common to all Keras optimizers:

The parameters clipnorm and clipvalue can be used with all optimizers to control gradient clipping: from keras import optimizers

```
# All parameter gradients will be clipped to

sgd = optimizers.SGD(lr=0.01, clipnorm=1.)
from keras import optimizers

# All parameter gradients will be clipped to# a maximum value of 0.5 and

# a minimum value of -0.5.

sgd = optimizers.SGD(lr=0.01, clipvalue=0.5)
```

3.2 Code Analysis

Our whole project has three parts for programming.

1. Programming for Training.
2. Programming for Classification.
3. Programming through Arduino

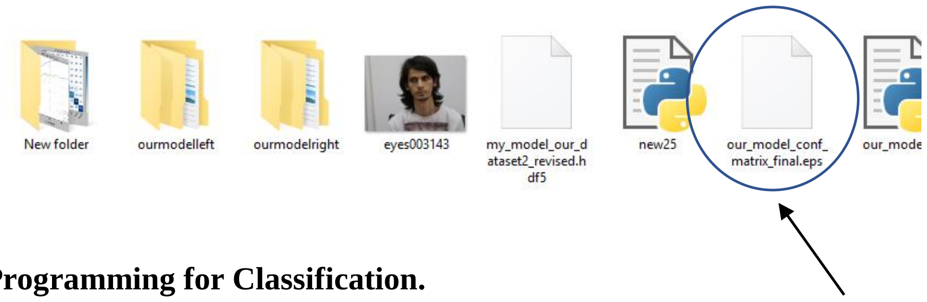
1. **Programming for Training:**

   After importing all the library by import library name, the dataset is provided to the neural net.

```
import math
import numpy as np
……………
from keras.models import Model
from…………
………………………
img_rows=64
img_cols=64
num_channel=3
# Define the number of classes
num_classes = 9
names = [’center’,’down’,’left’,’right’,’down’]
data_path = ’ourmodelleft’
data_dir_list = os.listdir(data_path)
img_data_list=[]
for dataset in data_dir_list:
        img_list=os.listdir(data_path+’/’+ dataset)
        print (’Loaded the images of dataset-’+’{}\n’.format(dataset))
        for img in img_list:
                input_img=cv2.imread(data_path + ’/’+ dataset + ’/’+ img )
                input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
                input_img_resize=cv2.resize(input_img,(img_rows,img_cols))
                img_data_list.append(input_img_resize)
```

After that we have broken the data set into small parts to feed the neuron. While the neuron will manipulate the data then a classified weighted graph file will be created in the same directory.

```
num_channel=1

if num_channel==1:

       if K.image_dim_ordering()=='th':

              img_data= np.expand_dims(img_data, axis=1)

              print (img_data.shape)

       else:

              img_data= np.expand_dims(img_data, axis=4)

              print (img_data.shape)

else:

       if K.image_dim_ordering()=='th':

              img_data=np.rollaxis(img_data,1,1)

              print (img_data.shape)

right_train=img_data.astype('float32')

right_train=right_train/255

right_train=right_train.reshape(right_train.shape[0],1,img_rows,img_cols)

right_train1, right_test, Y_train1, Y_test = train_test_split(right_train,Y_train,
test_size=0.2)

left_train1,left_test,Y_train1, Y_test
=train_test_split(left_train,Y_train,test_size=.2)

input_shape_video=(1,img_rows,img_cols)

input2=Input(shape=input_shape_video)

input3=Input(shape=input_shape_video)

x = Conv2D(32, (7,7), strides=(1, 1),
kernel_initializer=initializers.random_normal(stddev=0.01))(input2)

x = Activation('relu')(x)

x = BatchNormalization()(x)

x = MaxPooling2D((3, 3), strides=(2, 2))(x)
```


New folder
ourmodelleft
ourmodelright
eyes003143
my_model_our_dataset2_revised.hdf5
new25
our_model_conf_matrix_final.eps
our_mode

2. Programming for Classification.

my_model_our_dataset2_revised. hdf5 calculated weight at the training is used as an input in the classification at the place of weighted “W”.

```
if name=="left_eye":
        (x, y)=shape[42]
        left_left_x=x
        left_left_y=y
        (x,  y)=shape[45]
left=left.reshape(1,1,64,64)
right=right.astype('float32')
prediction=[]
prediction = loaded_merged_model.predict([left,right],verbose=0)
#print(prediction)
pred_classes = np.argmax(prediction,axis=1)
if pred_classes==0:  count[pred_classes]=count[pred_classes]+1
                print('center')
elif pred_classes==2: count[pred_classes]=count[pred_classes]+1
                print('left')
elif pred_classes==3:  count[pred_classes]=count[pred_classes]+1
                print('right')
elif pred_classes==4:  count[pred_classes]=count[pred_classes]+1
                print('up')
elif pred_classes==1:
                count[pred_classes]=count[pred_classes]+1
                print('down')
```

Here we have used a shape_predictor_68_face_landmarks.dat where the predefined object type will identify the shape of our eye positon. After that it will predict the right left up down center.

```
while(1):   url='http://192.168.0.102:8080/photo.jpg'

    imgResp=urllib.request.urlopen(url)

    imgResp=cv2.imread('eyes003143',0)

    imgNp=np.array(bytearray(imgResp.read()))

    img=cv2.imdecode(imgNp,-1)

# load the input image, resize it, and convert it to grayscale

# define a dictionary that maps the indexes of the facial

# landmarks to specific face regions

FACIAL_LANDMARKS_IDXS = collections.OrderedDict([("right_eye",(36,
42)),("left_eye",(42, 48)),])

image = imutils.resize((image, width=500)gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY))

     # detect faces in the grayscale image

rects = detector(gray, 1)

# loop over the face detections

      # loop over the face detections

for(i, rect) in enumerate(rects):

      # determine the facial landmarks for the face region, then

             shape = predictor(gray, rect)

             shape = face_utils.shape_to_np(shape)

      # loop over the face parts individually

             for (name, (i, j)) in face_utils.FACIAL_LANDMARKS_IDXS.items():

             # clone the original image so we can draw on it, then

                    clone = img.copy()
```

3. Programming through Arduino:

The output will be thrown by the laptop Bluetooth module to the car module and in the car module the following code will analyze the input and the car will start moving according to the Eye Gaze.

```
import serial

#port="COM4"

#bluetooth=serial.Serial(port,9600)

port="COM3"

bluetooth=serial.Serial(port,9600)

while(1):

        print("connected")

        pred_classes=1234567

        bluetooth.flushInput()

        bluetooth.write(str(pred_classes).encode())

        #input_data=bluetooth.readline()

        #print(input_data.decode())
```

CHAPTER 4

RESULTS & APPLICATION

To evaluate the performance and the working procedure the results are described below.

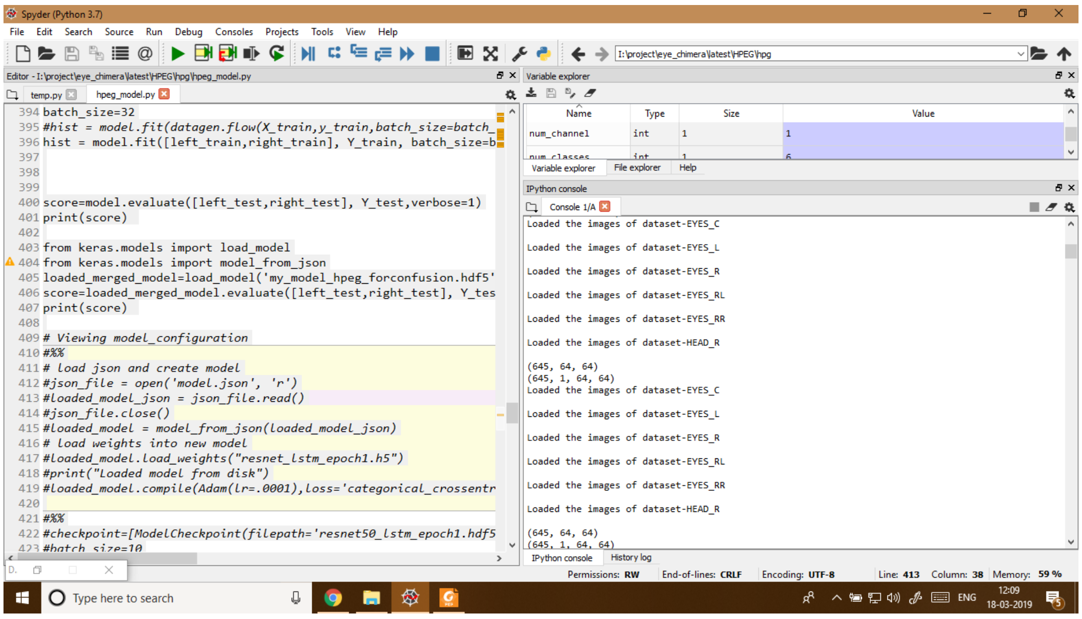
4.1 Results

By the Training we can get the three types of graph for analysis…….

- Confusion Matrix (True Label vs Prediction Label)
- Training Loss (Loss vs num of epochs)
- Training accuracy (accuracy vs num of epochs)

As we have used 3 different datasets we will get total 9 analysis graph for the training field.

While we run the code the code will automatically start generate the values. How the processes are going on in the spyder IDE is shown below.


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
I:\project\eye_chimera\latest\HPEG\hpg
Editor - I:\project\eye_chimera\latest\HPEG\hpg\hpeg_model.py
temp.py
hpeg_model.py
Variable explorer
File explorer
Help
IPython console
Console 1/A
History log
Type here to search

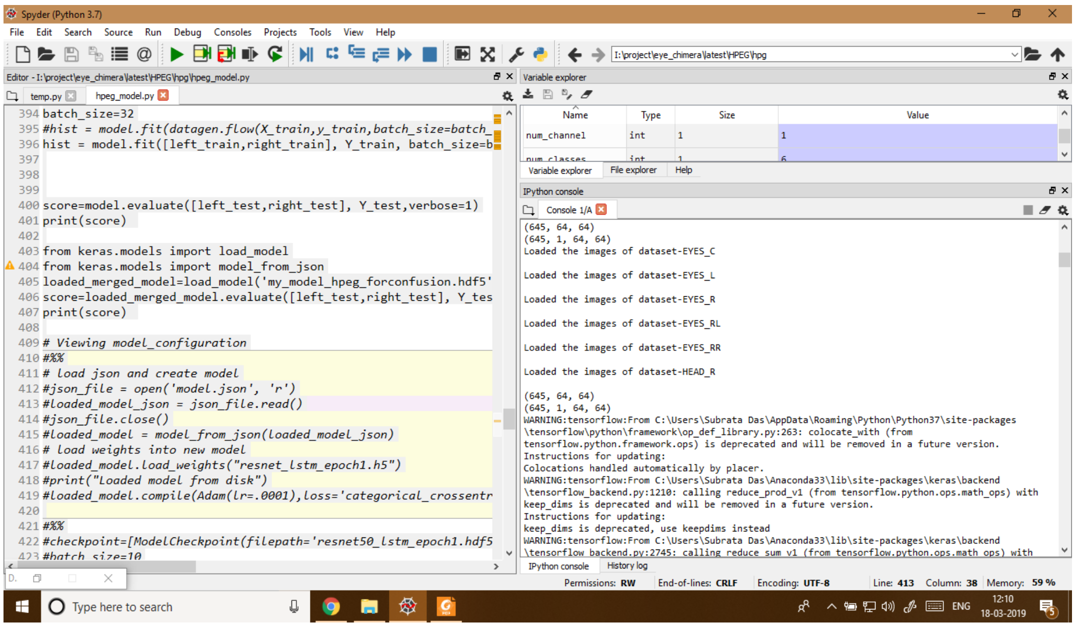
```
batch_size=32
#hist = model.fit(datagen.flow(X_train,y_train,batch_size=batch_
hist = model.fit([left_train,right_train], Y_train, batch_size=b



score=model.evaluate([left_test,right_test], Y_test,verbose=1)
print(score)

from keras.models import load_model
from keras.models import model_from_json
loaded_merged_model=load_model('my_model_hpeg_forconfusion.hdf5'
score=loaded_merged_model.evaluate([left_test,right_test], Y_tes
print(score)

# Viewing model_configuration
#%%
# load json and create model
#json_file = open('model.json', 'r')
#loaded_model_json = json_file.read()
#json_file.close()
#loaded_model = model_from_json(loaded_model_json)
# load weights into new model
#loaded_model.load_weights("resnet_lstm_epoch1.h5")
#print("Loaded model from disk")
#loaded_model.compile(Adam(lr=.0001),loss='categorical_crossentr

#%%
#checkpoint=[ModelCheckpoint(filepath='resnet50_lstm_epoch1.hdf5
#batch_size=10
```

Name	Type	Size	Value
num_channel	int	1	1
num_classes	int	1	6

```
Loaded the images of dataset-EYES_C

Loaded the images of dataset-EYES_L

Loaded the images of dataset-EYES_R

Loaded the images of dataset-EYES_RL

Loaded the images of dataset-EYES_RR

Loaded the images of dataset-HEAD_R

(645, 64, 64)
(645, 1, 64, 64)
Loaded the images of dataset-EYES_C

Loaded the images of dataset-EYES_L

Loaded the images of dataset-EYES_R

Loaded the images of dataset-EYES_RL

Loaded the images of dataset-EYES_RR

Loaded the images of dataset-HEAD_R

(645, 64, 64)
(645, 1, 64, 64)
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 413 Column: 38 Memory: 59 %

12:09 18-03-2019



Editor - I:\project\eye_chimera\latest\HPEG\hpg\hpeg_model.py

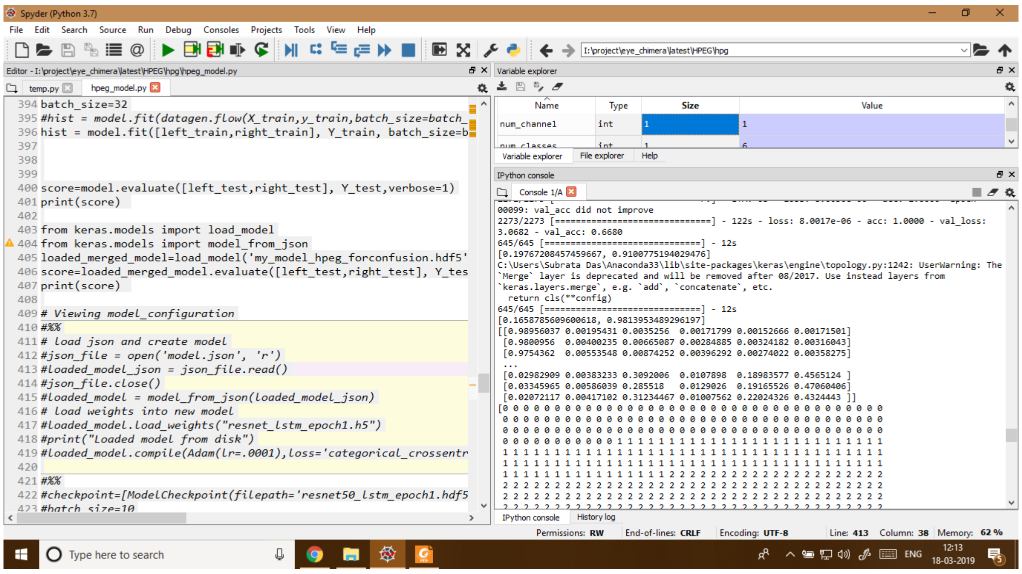
```
batch_size=32
#hist = model.fit(datagen.flow(X_train,y_train,batch_size=batch_
hist = model.fit([left_train,right_train], Y_train, batch_size=b



score=model.evaluate([left_test,right_test], Y_test,verbose=1)
print(score)

from keras.models import load_model
from keras.models import model_from_json
loaded_merged_model=load_model('my_model_hpeg_forconfusion.hdf5'
score=loaded_merged_model.evaluate([left_test,right_test], Y_tes
print(score)

# Viewing model_configuration
#%%
# load json and create model
#json_file = open('model.json', 'r')
#loaded_model_json = json_file.read()
#json_file.close()
#loaded_model = model_from_json(loaded_model_json)
# load weights into new model
#loaded_model.load_weights("resnet_lstm_epoch1.h5")
#print("Loaded model from disk")
#loaded_model.compile(Adam(lr=.0001),loss='categorical_crossentr

#%%
#checkpoint=[ModelCheckpoint(filepath='resnet50_lstm_epoch1.hdf5
#batch_size=10
```

Name	Type	Size	Value
num_channel	int	1	1
num_classes	int	1	6

```
(645, 64, 64)
(645, 1, 64, 64)
Loaded the images of dataset-EYES_C

Loaded the images of dataset-EYES_L

Loaded the images of dataset-EYES_R

Loaded the images of dataset-EYES_RL

Loaded the images of dataset-EYES_RR

Loaded the images of dataset-HEAD_R

(645, 64, 64)
(645, 1, 64, 64)
WARNING:tensorflow:From C:\Users\Subrata Das\AppData\Roaming\Python\Python37\site-packages
\tensorflow\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\Subrata Das\Anaconda33\lib\site-packages\keras\backend
\tensorflow_backend.py:1210: calling reduce_prod_v1 (from tensorflow.python.ops.math_ops) with
keep_dims is deprecated and will be removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
WARNING:tensorflow:From C:\Users\Subrata Das\Anaconda33\lib\site-packages\keras\backend
\tensorflow_backend.py:2745: calling reduce_sum_v1 (from tensorflow.python.ops.math_ops) with
```

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **413** Column: **38** Memory: **59 %**

12:10 18-03-2019



Editor - I:\project\eye_chimera\latest\HPEG\hpg\hpeg_model.py

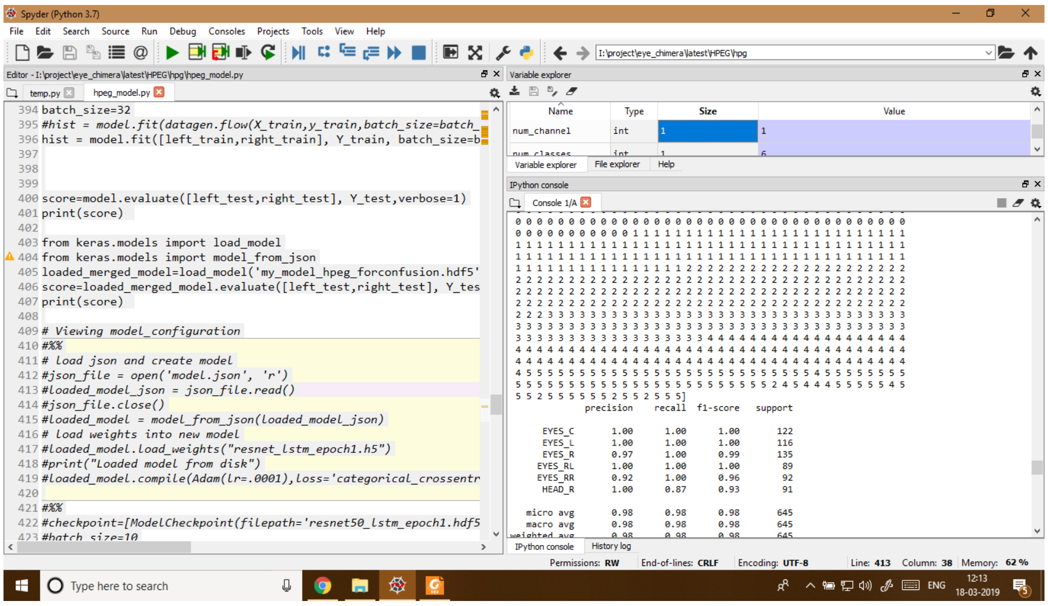
```
batch_size=32
#hist = model.fit(datagen.flow(X_train,y_train,batch_size=batch_
hist = model.fit([left_train,right_train], Y_train, batch_size=b



score=model.evaluate([left_test,right_test], Y_test,verbose=1)
print(score)

from keras.models import load_model
from keras.models import model_from_json
loaded_merged_model=load_model('my_model_hpeg_forconfusion.hdf5'
score=loaded_merged_model.evaluate([left_test,right_test], Y_tes
print(score)

# Viewing model_configuration
#%%
# load json and create model
#json_file = open('model.json', 'r')
#loaded_model_json = json_file.read()
#json_file.close()
#loaded_model = model_from_json(loaded_model_json)
# load weights into new model
#loaded_model.load_weights("resnet_lstm_epoch1.h5")
#print("Loaded model from disk")
#loaded_model.compile(Adam(lr=.0001),loss='categorical_crossentr

#%%
#checkpoint=[ModelCheckpoint(filepath='resnet50_lstm_epoch1.hdf5
#batch_size=10
```

Name	Type	Size	Value
num_channel	int	1	1
num_classes	int	1	6

```
00099: val_acc did not improve
2273/2273 [==============================] - 122s - loss: 8.0017e-06 - acc: 1.0000 - val_loss:
3.0682 - val_acc: 0.6680
645/645 [==============================] - 12s
[0.19767208457459667, 0.9100775194029476]
C:\Users\Subrata Das\Anaconda33\lib\site-packages\keras\engine\topology.py:1242: UserWarning: The
`Merge` layer is deprecated and will be removed after 08/2017. Use instead layers from
`keras.layers.merge`, e.g. `add`, `concatenate`, etc.
  return cls(**config)
645/645 [==============================] - 12s
[0.165878560960618, 0.9813953489296197]
[[0.98956037 0.00195431 0.0035256  0.00171799 0.00152666 0.00171501]
 [0.9800956  0.00400235 0.00665087 0.00284885 0.00324182 0.00316043]
 [0.9754362  0.00553548 0.00874252 0.00396292 0.00274022 0.00358275]
 ...
 [0.02982909 0.00383233 0.3092006  0.0107898  0.18983577 0.4565124 ]
 [0.03345965 0.00586039 0.285518   0.0129026  0.19165526 0.47060406]
 [0.02072117 0.00417102 0.31234467 0.01007562 0.22024326 0.4324443 ]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **413** Column: **38** Memory: **62 %**

12:13 18-03-2019



Editor - I:\project\eye_chimera\latest\HPEG\hpg\hpeg_model.py

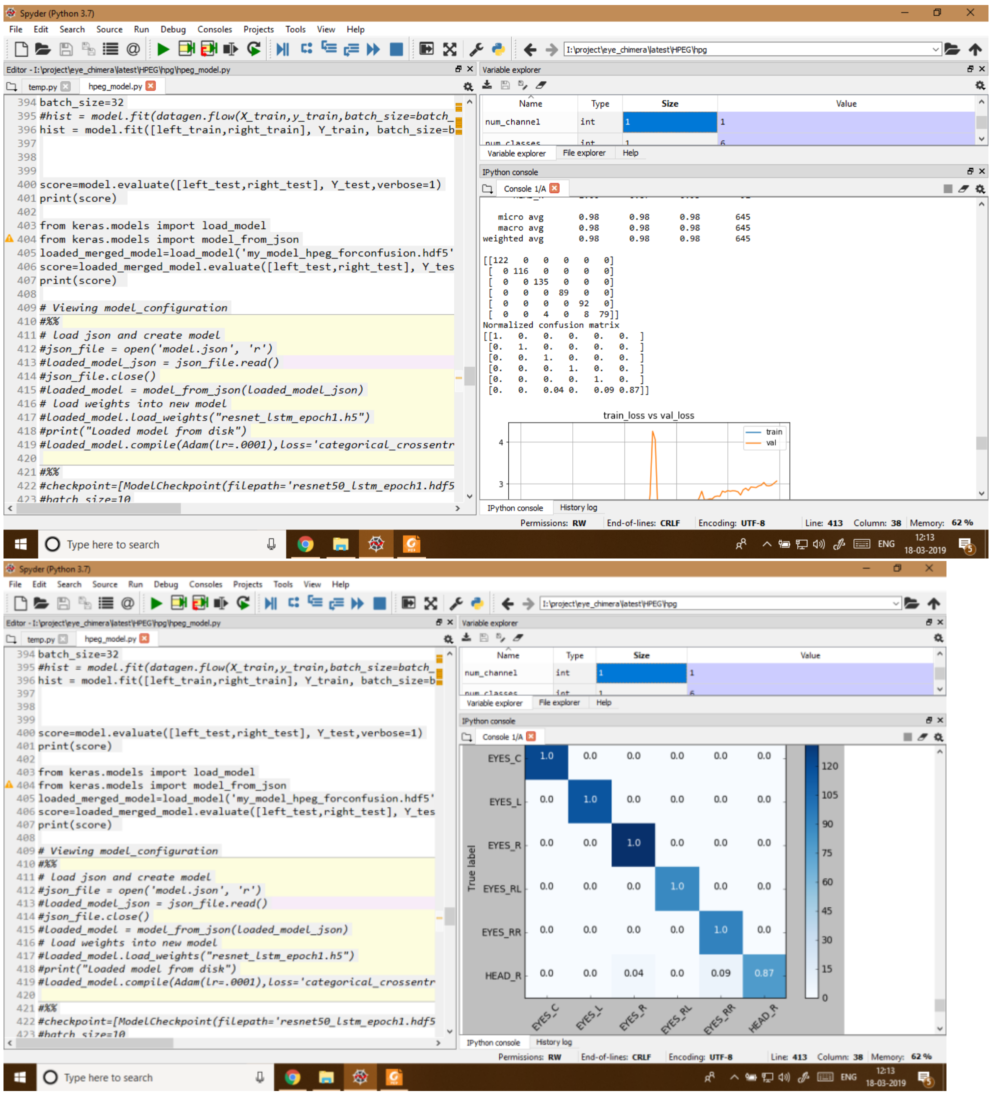
```
batch_size=32
#hist = model.fit(datagen.flow(X_train,y_train,batch_size=batch_
hist = model.fit([left_train,right_train], Y_train, batch_size=b



score=model.evaluate([left_test,right_test], Y_test,verbose=1)
print(score)

from keras.models import load_model
from keras.models import model_from_json
loaded_merged_model=load_model('my_model_hpeg_forconfusion.hdf5'
score=loaded_merged_model.evaluate([left_test,right_test], Y_tes
print(score)

# Viewing model_configuration
#%%
# load json and create model
#json_file = open('model.json', 'r')
#loaded_model_json = json_file.read()
#json_file.close()
#loaded_model = model_from_json(loaded_model_json)
# load weights into new model
#loaded_model.load_weights("resnet_lstm_epoch1.h5")
#print("Loaded model from disk")
#loaded_model.compile(Adam(lr=.0001),loss='categorical_crossentr

#%%
#checkpoint=[ModelCheckpoint(filepath='resnet50_lstm_epoch1.hdf5
#batch_size=10
```

Name	Type	Size	Value
num_channel	int	1	1
num_classes	int	1	6

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 4 5 4 4 4 5 5 5 5 5 4 5
 5 5 2 5 5 5 5 5 5 2 5 5 2 5 5 5]
              precision    recall  f1-score   support

      EYES_C       1.00      1.00      1.00       122
      EYES_L       1.00      1.00      1.00       116
      EYES_R       0.97      1.00      0.99       135
     EYES_RL       1.00      1.00      1.00        89
     EYES_RR       0.92      1.00      0.96        92
      HEAD_R       1.00      0.87      0.93        91

   micro avg       0.98      0.98      0.98       645
   macro avg       0.98      0.98      0.98       645
weighted avg       0.98      0.98      0.98       645
```

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **413** Column: **38** Memory: **62 %**

12:13 18-03-2019



```
batch_size=32
#hist = model.fit(datagen.flow(X_train,y_train,batch_size=batch_
hist = model.fit([left_train,right_train], Y_train, batch_size=b



score=model.evaluate([left_test,right_test], Y_test,verbose=1)
print(score)

from keras.models import load_model
from keras.models import model_from_json
loaded_merged_model=load_model('my_model_hpeg_forconfusion.hdf5'
score=loaded_merged_model.evaluate([left_test,right_test], Y_tes
print(score)

# Viewing model_configuration
#%%
# load json and create model
#json_file = open('model.json', 'r')
#loaded_model_json = json_file.read()
#json_file.close()
#loaded_model = model_from_json(loaded_model_json)
# load weights into new model
#loaded_model.load_weights("resnet_lstm_epoch1.h5")
#print("Loaded model from disk")
#loaded_model.compile(Adam(lr=.0001),loss='categorical_crossentr

#%%
#checkpoint=[ModelCheckpoint(filepath='resnet50_lstm_epoch1.hdf5
#batch_size=10
```

```
   micro avg       0.98      0.98      0.98       645
   macro avg       0.98      0.98      0.98       645
weighted avg       0.98      0.98      0.98       645

[[122   0   0   0   0   0]
 [  0 116   0   0   0   0]
 [  0   0 135   0   0   0]
 [  0   0   0  89   0   0]
 [  0   0   0   0  92   0]
 [  0   0   4   0   8  79]]
Normalized confusion matrix
[[1.   0.   0.   0.   0.   0.  ]
 [0.   1.   0.   0.   0.   0.  ]
 [0.   0.   1.   0.   0.   0.  ]
 [0.   0.   0.   1.   0.   0.  ]
 [0.   0.   0.   0.   1.   0.  ]
 [0.   0.   0.04 0.   0.09 0.87]]
```

Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
I:\project\eye_chimera\latest\HPEG\hpg
Editor - I:\project\eye_chimera\latest\HPEG\hpg\hpeg_model.py
temp.py
hpeg_model.py
Variable explorer
Name Type Size Value
num_channel int 1 1
num_classes int 1 6
Variable explorer File explorer Help
IPython console
Console 1/A
train_loss vs val_loss
train
val
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 413 Column: 38 Memory: 62 %
Type here to search
ENG 12:13 18-03-2019
True label
EYES_C
EYES_L
EYES_R
EYES_RL
EYES_RR
HEAD_R

```
batch_size=32
#hist = model.fit(datagen.flow(X_train,y_train,batch_size=batch_
hist = model.fit([left_train,right_train], Y_train, batch_size=b



score=model.evaluate([left_test,right_test], Y_test,verbose=1)
print(score)

from keras.models import load_model
from keras.models import model_from_json
loaded_merged_model=load_model('my_model_hpeg_forconfusion.hdf5'
score=loaded_merged_model.evaluate([left_test,right_test], Y_tes
print(score)

# Viewing model_configuration
#%%
# load json and create model
#json_file = open('model.json', 'r')
#loaded_model_json = json_file.read()
#json_file.close()
#loaded_model = model_from_json(loaded_model_json)
# load weights into new model
#loaded_model.load_weights("resnet_lstm_epoch1.h5")
#print("Loaded model from disk")
#loaded_model.compile(Adam(lr=.0001),loss='categorical_crossentr

#%%
#checkpoint=[ModelCheckpoint(filepath='resnet50_lstm_epoch1.hdf5
#batch_size=10
```

Figure 4.1: Spyder IDE

Thus the output is created as graphs and my_model_our_dataset2_revised. hdf5 for the next phase predictionGraph for Three datasets that we have used at the time of training.

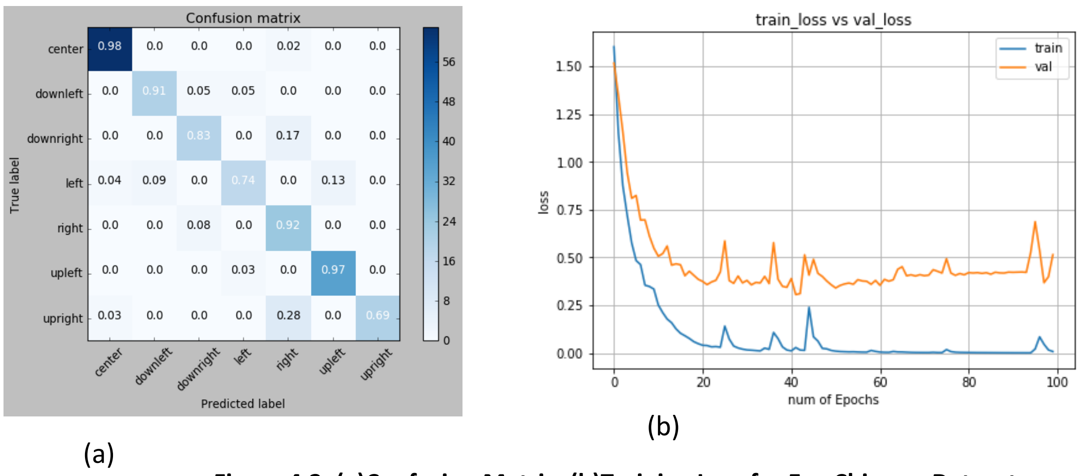


Figure 4.2: (a)Confusion Matrix, (b)Training Loss for Eye Chimera Dataset

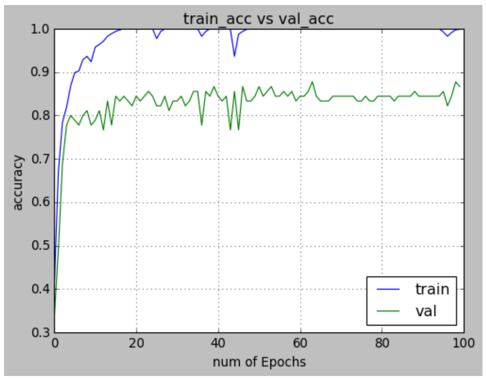
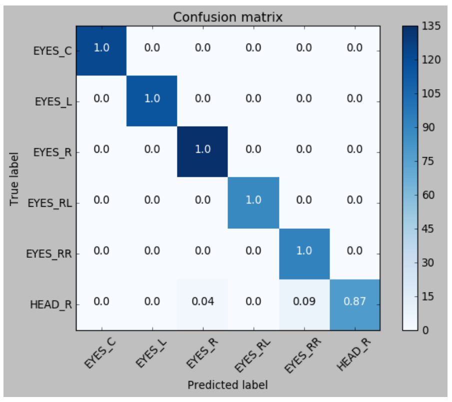
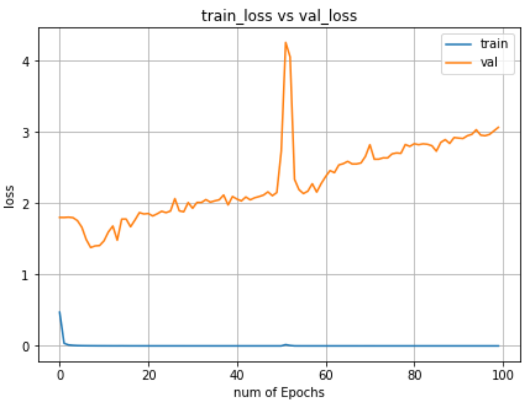


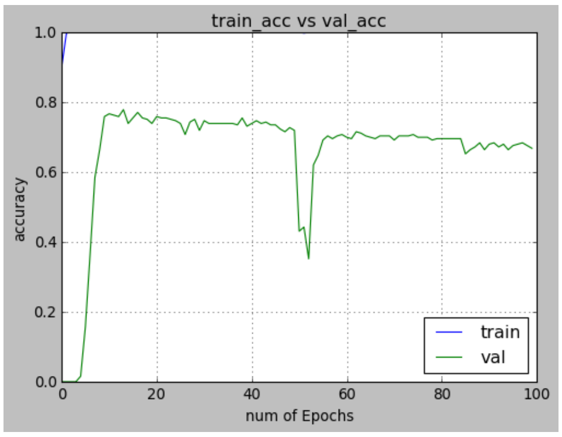
Figure 4.3: Training Accuracy for Eye Chimera Dataset



	EYES_C	EYES_L	EYES_R	EYES_RL	EYES_RR	HEAD_R
EYES_C	1.0	0.0	0.0	0.0	0.0	0.0
EYES_L	0.0	1.0	0.0	0.0	0.0	0.0
EYES_R	0.0	0.0	1.0	0.0	0.0	0.0
EYES_RL	0.0	0.0	0.0	1.0	0.0	0.0
EYES_RR	0.0	0.0	0.0	0.0	1.0	0.0
HEAD_R	0.0	0.0	0.04	0.0	0.09	0.87



(a) **(b)**

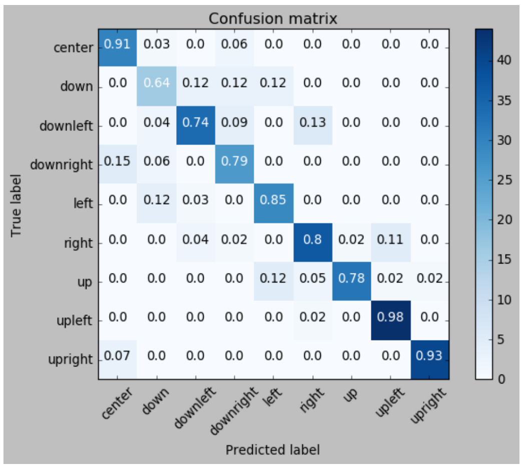


(c)

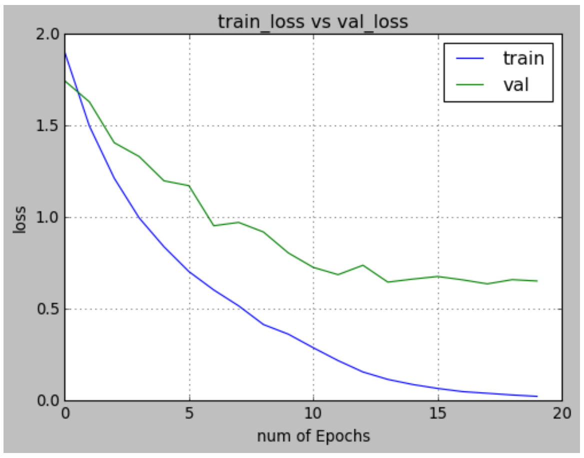
Figure 17: (a)Confusion Matrix (b) Training Loss and(c) Training Accuracy for HPEG Dataset

Testing Result:

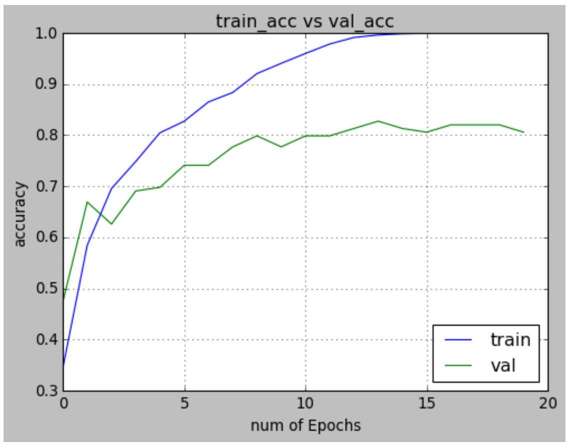
While we get the training output as a my_model_our_dataset2_revised. hdf5 we will use the weight as calculated weight and provide as a prediction weight.For a new image as input from the IP camera, the system will predict the output as a classified eye gazed direction.



	center	down	downleft	downright	left	right	up	upleft	upright
center	0.91	0.03	0.0	0.06	0.0	0.0	0.0	0.0	0.0
down	0.0	0.64	0.12	0.12	0.12	0.0	0.0	0.0	0.0
downleft	0.0	0.04	0.74	0.09	0.0	0.13	0.0	0.0	0.0
downright	0.15	0.06	0.0	0.79	0.0	0.0	0.0	0.0	0.0
left	0.0	0.12	0.03	0.0	0.85	0.0	0.0	0.0	0.0
right	0.0	0.0	0.04	0.02	0.0	0.8	0.02	0.11	0.0
up	0.0	0.0	0.0	0.0	0.12	0.05	0.78	0.02	0.02
upleft	0.0	0.0	0.0	0.0	0.0	0.02	0.0	0.98	0.0
upright	0.07	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.93

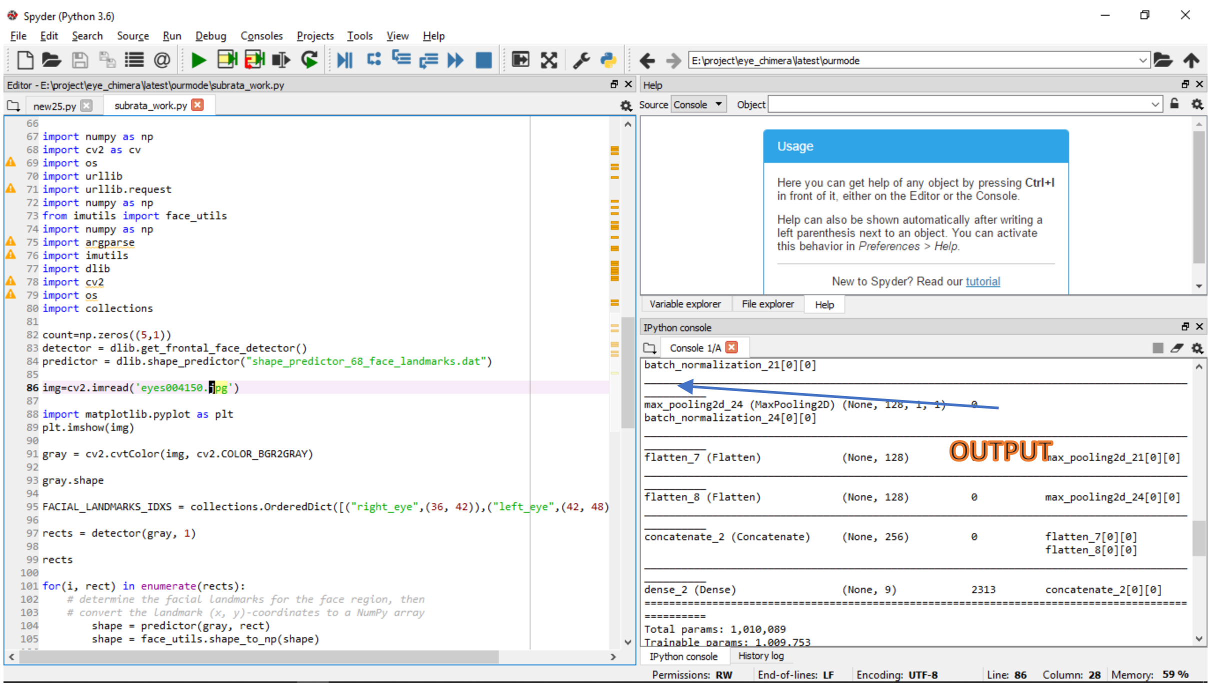


(a) (b)



(c)

Figure 18: (a)Confusion Matrix,(b) Training Loss (c)Training Accuracy for EGDC Dataset


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\project\eye_chimera\latest\ourmode
Editor - E:\project\eye_chimera\latest\ourmode\subrata_work.py
new25.py
subrata_work.py
Help
Source Console Object
Variable explorer File explorer Help
IPython console
Console 1/A
OUTPUT
IPython console History log

```python
import numpy as np
import cv2 as cv
import os
import urllib
import urllib.request
import numpy as np
from imutils import face_utils
import numpy as np
import argparse
import imutils
import dlib
import cv2
import os
import collections

count=np.zeros((5,1))
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

img=cv2.imread('eyes004150.jpg')

import matplotlib.pyplot as plt
plt.imshow(img)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

gray.shape

FACIAL_LANDMARKS_IDXS = collections.OrderedDict([("right_eye",(36, 42)),("left_eye",(42, 48)

rects = detector(gray, 1)

rects

for(i, rect) in enumerate(rects):
    # determine the facial landmarks for the face region, then
    # convert the landmark (x, y)-coordinates to a NumPy array
        shape = predictor(gray, rect)
        shape = face_utils.shape_to_np(shape)
```

Usage

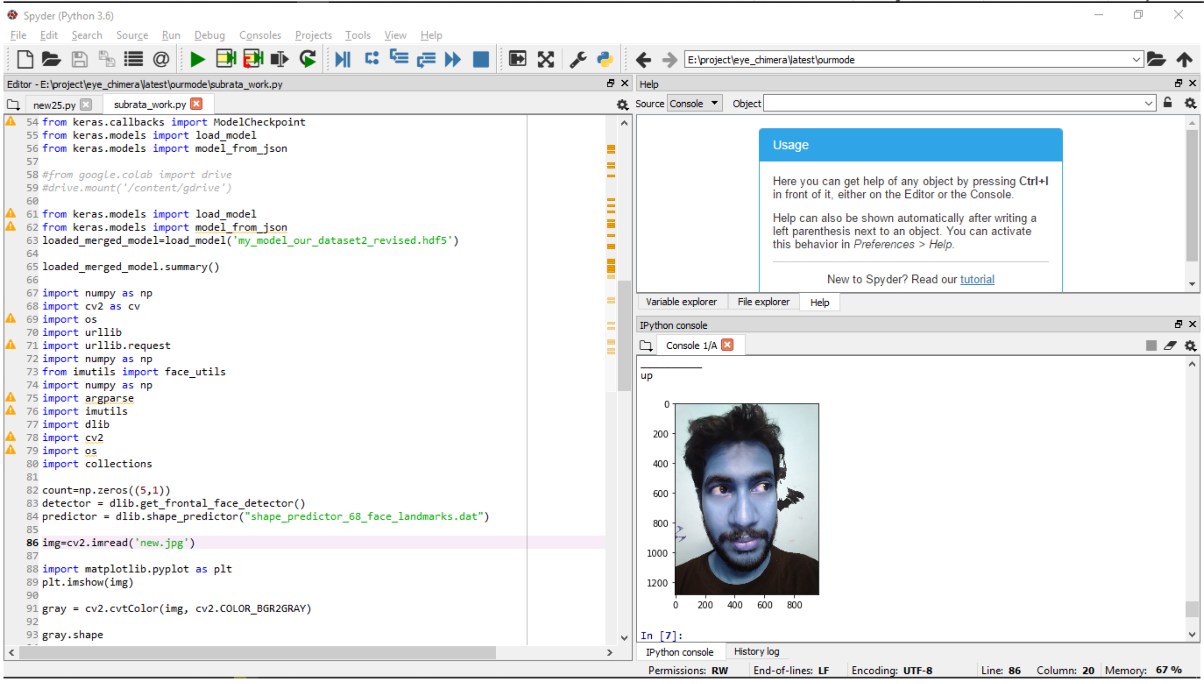
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
batch_normalization_21[0][0]
__________________________________________________________________________________________________
max_pooling2d_24 (MaxPooling2D) (None, 128, 1, 1)    0
batch_normalization_24[0][0]
__________________________________________________________________________________________________
flatten_7 (Flatten)             (None, 128)                      max_pooling2d_21[0][0]
__________________________________________________________________________________________________
flatten_8 (Flatten)             (None, 128)          0           max_pooling2d_24[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 256)          0           flatten_7[0][0]
                                                                 flatten_8[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 9)            2313        concatenate_2[0][0]
==================================================================================================
Total params: 1,010,089
Trainable params: 1,009,753
```

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 86 Column: 28 Memory: 59 %


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\project\eye_chimera\latest\ourmode
Editor - E:\project\eye_chimera\latest\ourmode\subrata_work.py
new25.py
subrata_work.py
Help
Source Console Object
Variable explorer File explorer Help
IPython console
Console 1/A
0
200
400
600
800
1000
1200
0
200
400
600
800
IPython console History log

```python
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from keras.models import model_from_json

#from google.colab import drive
#drive.mount('/content/gdrive')

from keras.models import load_model
from keras.models import model_from_json
loaded_merged_model=load_model('my_model_our_dataset2_revised.hdf5')

loaded_merged_model.summary()

import numpy as np
import cv2 as cv
import os
import urllib
import urllib.request
import numpy as np
from imutils import face_utils
import numpy as np
import argparse
import imutils
import dlib
import cv2
import os
import collections

count=np.zeros((5,1))
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

img=cv2.imread('new.jpg')

import matplotlib.pyplot as plt
plt.imshow(img)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

gray.shape
```

Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
up

In [7]:
```

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 86 Column: 20 Memory: 67 %

Figure 19: Outcome of Code

According to the image the image is classified into 5 classes and the output is shown in the iPython Console.

4.2 Application of the Project

As eye gaze detection is the identification of the eye movement, we can use the code for implementing an application for physically disable people. As they cannot move their legs or arms, they can give command, move their wheel chair, giving command to the computer. Paralysis can be triggered by a wide range of diseases and conditions such as Stroke, Spinal cord injury, Cerebral palsy, Traumatic brain injury, Birth defects and Multiple sclerosis. Paralyzed people having nerve and muscle damages suffer in communicating and depend on others for even basic day to day activities. Eye movement may become the only way of communication for them. For road traffic safety, eye movement of drivers is being investigated to detect drowsiness and right direction of car movement.

Pilot training, ergonomics, market research, and mood disorder are some of the area where eye gaze detection can have huge impacts.

For gamers, the eye detection can give a controlling system and easier way to play games. Automotive research has embraced eye tracking glasses for a long time to gauge driver's visual attention – both with respect to navigation and layout of dashboards. In the near future automobiles might be able to be responsive towards their driver's eye gaze, eye movements or the dilation of the pupil.

CONCLUSION

The conclusion that can be drawn from this project is that eye gaze can be estimated much more efficiently using deep learning based algorithm compared to conventional intrusive eye gaze detection techniques. In this project, we present a method for estimating eye gaze direction, which represents a departure from conventional eye gaze estimation methods, the majority of which are based on tracking specific optical phenomena like corneal reflection and the Purkinje images.

Our proposed deep learning based Gaze estimation algorithm has been proven considerably efficient. Further, efficacious and robust realization of this algorithm could be achieved by our developed prototype. In summary, this project has proposed an efficient model which has outperformed all state of the art results. However, the accuracy can be further increased using deeper network but that will make the system slower. That’s why accuracy has been traded off to system speed. The framework has been implemented with low cost mobile camera which ensures the systems robustness. In our device, we have overcome the difficulties and disadvantages of convention corneal reflection based EGT which caused discomfort to the users. The robotic car in the framework can be replaced with robotic arm or any other controllable devices which can make the framework useful in many other ways.

Eye gaze technology is perhaps the most exciting, innovative and important piece of assistive technology. In one way, eye gaze should be considered as one of the most potential access methods for those with physical disabilities.

REFERENCE

1. Paralysis Facts & Figures-Spinal Cord Injury-Paralysis Research Center. Christopherreeve.org. Retrieved 2013 02-19.
2. cureresearch.com, Statistics by Country for Paralysis,[Online].2005. Available: www: cureresearch: com =p = paralysis=stats-country printer:html:
3. M. Eizenman, T. Jares, and A. Smiley, A new methodology for the analysis of eye movements and visual scanning in drivers, in Proc. 31st An. Conf. Erg. & Safety, Hall, Quebec, Canada, 1999.
4. J. L. Harbluk, I. Y. Noy, and M. Eizenman, The impact of cognitive distraction on driver visual and vehicle control, in Proc. Transp. Res. Board 81st An. Meet., Washington, DC, USA, Jan. 2002.
5. D. Cleveland, Unobtrusive eyelid closure and visual point of regard measurement system, in Proc. Tech. Conf. on Ocular Measures of Driver Alertness, sponsored by The Federal Highway Administration Office of Motor Carrier and Highway Safety and The National Highway Traffic Safety Administration Office of Vehicle Safety Research, Herndon, VA, USA, 1999, pp. 57-74.
6. P. A. Wetzel, G. Krueger-Anderson, C. Poprik, and P. Bascom, An eye tracking system for analysis of pilots scan paths, United States Air Force Armstrong Laboratory, Tech. Rep. AL/HR-TR-1996-0145, Apr. 1997.
7. J. H. Goldberg and X. P. Kotval, Computer interface evaluation using eye movements: methods and constructs, Int. J. Ind. Erg., vol. 24, no. 6, pp. 631-645, Oct. 1999. Parana, Entre Rios, Argentina, in 2000.
8. Wedel, M.; Pieters, R. (2000). ”Eye fixations on advertisements and memory for brands: a model and findings”. Marketing Science.
9. M. Eizenman, L. H. Yu, L. Grupp, E. Eizenman, M. Ellenbogen, M. Gemar, and R. D. Levitan, A naturalistic visual scanning approach to assess selective attention in major depressive disorder, Psychiat. Res., vol. 118, no. 2, pp. 117-128, May 2003.
10. D. W. Hansen and Q. Ji. In the eye of the beholder: A survey of models for eyes and gaze. PAMI, 2010.
11. E. D. Guestrin and M. Eizenman, ”General theory of remote gaze estimation using the pupil center and corneal reflections,” in IEEE Transactions on Biomedical Engineering, vol. 53, no. 6, pp. 1124-1133, June 2006.
12. Huang J., Ii D., Shao X., Wechsler H. (1998) Pose Discriminiation and Eye Detection Using Support Vector Machines (SVM). In: Wechsler H., Phillips P.J., Bruce V., Souli F.F., Huang T.S. (eds) Face Recognition. NATO ASI Series (Series F: Computer and Systems Sciences), vol 163. Springer, Berlin, Heidelberg
13. Cozzi, M. Flickner, J. Mao, and S. Vaithyanathan, A comparison of classifiers for real-time eye detection, in Proceedings of the International Conference on Artificial Neural Networks, 2001, pp. 993”999.
14. Bin Li and Hong Fu, Real Time Eye Detector with Cascaded Convolutional Neural Networks, Applied Computational Intelligence and Soft Computing, vol. 2018, Article ID 1439312, 8 pages, 2018.
15. Ye L., Zhu M., Xia S., Pan H. (2014) Cascaded Convolutional Neural Network for Eye Detection Under Complex Scenarios. In: Sun Z., ShanS., Sang H., Zhou J., Wang Y., Yuan W. (eds) Biometric Recognition. CCBR 2014. Lecture Notes in Computer Science, vol 8833. Springer, Cham