Report on Research at

Indian Institute of Technology, Bombay (IIT)

May 2016 to June 2016

Facial Expression Recognition using Deep Learning

Rheeya Uppaal

3rd Year, Computer Science

Table of Contents

1. Introduction

The history of artificial intelligence (AI) dates back into antiquity – intelligent robots appear in the myths of many ancient societies, including Greek, Arabic, Egyptian and Chinese. Today, the field of artificial intelligence is more vibrant than ever and some believe that we're on the threshold of discoveries that could change human society irreversibly. Apart from the world of computer science, AI has been used in a wide range of fields including medical diagnosis, stock trading, robot control, law, remote sensing, scientific discovery and toys, among many more.

AI is exhibited by machines which are flexible rational agents that perceive their environment and take actions that maximize the chances of success at some goal.

The term "artificial intelligence" was coined at the 1956 Dartmouth conference. It is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving". As machines become increasingly capable, facilities once thought to require intelligence are removed from the definition. For example, optical character recognition is no longer perceived as an exemplar of "artificial intelligence" having become a routine technology. Capabilities still classified as AI include advanced Chess and Go systems and self-driving cars.

The central goals of AI research include reasoning, knowledge, planning, learning, natural language processing, perception and the ability to move and manipulate objects. Approaches include statistical methods, computational intelligence, machine learning and traditional symbolic AI. Many tools are used in AI, including versions of search and mathematical optimization, logic, methods based on probability and economics. The AI field draws upon computer science, mathematics, psychology, linguistics, neuroscience and artificial psychology.

1.1 Machine Learning: An Overview

Machine learning (ML) is a subfield of artificial intelligence that evolved from the study of pattern recognition and computational learning theory. In 1959, Arthur

Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed". Machine learning is a method of data analysis that automates analytical model building. Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look. It explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from an example *training set* of input observations in order to make data-driven predictions or decisions expressed as outputs, rather than following strictly static program instructions.

The process of machine learning is similar to that of data mining. Both systems search through data to look for patterns. However, instead of extracting data for human comprehension -- as is the case in data mining applications -- machine learning uses that data to detect patterns in data and adjust program actions accordingly. Machine learning algorithms are often categorized as being supervised or unsupervised. Supervised algorithms can apply what has been learned in the past to new data. Unsupervised algorithms can draw inferences from datasets

Over the past two decades Machine Learning has become one of the mainstays of information technology and with that, a rather central, albeit usually hidden, part of our life. With the ever increasing amounts of data becoming available there is good reason to believe that smart data analysis will become even more pervasive as a necessary ingredient for technological progress. Much of the art of machine learning is to reduce a range of fairly disparate problems to a set of fairly narrow prototypes. Much of the science of machine learning is then to solve those problems and provide good guarantees for the solutions.

Many of our day-to-day activities are powered by machine learning algorithms, including: Fraud detection, Web search results, Real-time ads on web pages and mobile devices, Text-based sentiment analysis, Credit scoring and next-best offers, Prediction of equipment failures, New pricing models, Network intrusion detection, Pattern and image recognition, Email spam filtering, to name a few.

1.1.1 Types

Two of the most widely adopted machine learning methods are supervised learning and unsupervised learning. About 70 percent of machine learning models follow supervised learning. Unsupervised learning accounts for 10 to 20 percent. Semi-supervised and reinforcement learning are two other technologies that are gaining popularity.

Supervised learning algorithms are trained using labelled examples, such as an input where the desired output is known. For example, a piece of equipment could have data points labelled either “F” (failed) or “R” (runs). The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors. It then modifies the model accordingly. Through methods like classification, regression, prediction and gradient boosting, supervised learning uses patterns to predict the values of the label on additional unlabelled data. Supervised learning is commonly used in applications where historical data predicts likely future events. For example, it can anticipate when credit card transactions are likely to be fraudulent or which insurance customer is likely to file a claim.

Unsupervised learning is used against data that has no historical labels. The system is not told the "right answer." The algorithm must figure out what is being shown. The goal is to explore the data and find some structure within. Unsupervised learning works well on transactional data. For example, it can identify segments of customers with similar attributes who can then be treated similarly in marketing campaigns. Or it can find the main attributes that separate customer segments from each other. Popular techniques include self-organizing maps, nearest-neighbour mapping, k-means clustering and singular value decomposition. These algorithms are also used to segment text topics, recommend items and identify data outliers.

Semi-supervised learning is used for the same applications as supervised learning. But it uses both labelled and unlabelled data for training – typically a small amount of labelled data with a large amount of unlabelled data (because unlabelled data is less expensive and takes less effort to acquire). This type of learning can be used with methods such as classification, regression and prediction. Semi-supervised learning is useful when the cost associated with labelling is too high to allow for a fully labelled training process. Early examples of this include identifying a person's face on a web cam.

Reinforcement learning is often used for robotics, gaming and navigation. With reinforcement learning, the algorithm discovers through trial and error which actions yield the greatest rewards. This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do). The objective is for the agent to choose actions that maximize the expected reward over a given amount of time. The agent will reach the goal much faster by following a good policy. Thus, the goal in reinforcement learning is to learn the best policy.

1.1.2 Technical Definition

A popular definition of learning in the context of computer programs is "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E" (Mitchell, 1997).

The process of learning itself is not the task. Learning is our means of attaining the ability to perform the **task**. Some of the most common machine learning tasks include the following:

- *Classification*: In this type of task, the computer program is asked to specify which of k categories some input belongs to. To solve this task, the learning algorithm is usually asked to produce a function f: $R_n \rightarrow \{1,...,k\}$. When $y = f(x)$, the model assigns an input described by vector x to a category identified by numeric code y.
- *Classification with missing inputs*: Classification becomes more challenging if the computer program is not guaranteed that every measurement in its input vector will always be provided. In order to solve the classification task, the learning algorithm only has to define a *single* function mapping from a vector input to a categorical output. When some of the inputs may be missing, rather than providing a single classification function, the learning algorithm must learn a *set* of functions. Each function corresponds to classifying x with a different subset of its inputs missing. This kind of situation arises frequently in medical diagnosis, because many kinds of medical tests are expensive or invasive.
- *Regression*: In this type of task, the computer program is asked to predict a numerical value given some input. To solve this task, the learning algorithm is asked to output a function f: Rn → R. This type of task is similar to classification,

except that the format of output is different. An example of a regression task is the prediction of future prices of securities.

- *Transcription*: In this type of task, the machine learning system is asked to observe a relatively unstructured representation of some kind of data and transcribe it into discrete, textual form. For example, Google Street View uses deep learning to process address numbers in this way (Goodfellow *et al.*,2014). Another example is speech recognition, where the computer program is provided an audio waveform and emits a sequence of characters or word ID codes describing the words that were spoken in the audio recording. Deep learning is a crucial component of modern speech recognition systems (Hinton *et al.*,2012).
- *Machine translation*: In a machine translation task, the input already consists of a sequence of symbols in some language, and the computer program must convert this into a sequence of symbols in another language. Deep learning has recently begun to have an important impact on this kind of task (Sutskever *et al.*, 2014; Bahdanau *et al.*, 2014).
- *Structured output* tasks involve any task where the output is a vector (or other data structure containing multiple values) with important relationships between the different elements. These tasks are called structured output tasks because the program must output several values that are all tightly inter-related. An example is parsing—mapping a natural language sentence into a tree that describes its grammatical structure and the relative role of its constituents.
- *Anomaly detection*: In this type of task, the computer program sifts through a set of events or objects, and flags some of them as being unusual or atypical. An example of an anomaly detection task is credit card fraud detection. If a thief steals your credit card or credit card information, the thief's purchases will often come from a different probability distribution over purchase types than your own.
- *Synthesis and sampling*: In this type of task, the machine learning algorithm is asked to generate new examples that are similar to those in the training data. This can be useful for media applications where it can be expensive or boring for an artist to generate large volumes of content by hand. For example, video games can automatically generate textures for large objects or landscapes, rather than requiring an artist to manually label each pixel (Luo *et al.*, 2013).

- *Imputation of missing values*: In this type of task, the machine learning algorithm is given a new example $x \in$ Rn, but with some entries xi of x missing. The algorithm must provide a prediction of the values of the missing entries.
- *Denoising*: In this type of task, the machine learning algorithm is given as input a *corrupted example* $\tilde{x} \in$ Rn. The learner must predict the clean example x from its corrupted version $\tilde{x}$.
- *Density estimation* or *probability mass function estimation*: The machine learning algorithm is asked to learn a function which can be interpreted as a probability density function on the space that the examples were drawn from. The algorithm needs to learn the structure of the data it has seen. It must know where examples cluster tightly and where they are unlikely to occur.

The **performance measure *P*** is specific to the task. For tasks such as classification, classification with missing inputs, and transcription, the *accuracy* of the model is measured. Accuracy is the proportion of examples for which the model produces the correct output. Equivalent information by measuring the *error rate* can also be obtained. For tasks such as density estimation, a different performance metric that gives the model a continuous-valued score for each example is used. The choice of performance measure may seem straightforward and objective, but it is often difficult to choose a performance measure that corresponds well to the desired behaviour of the system, because, it is difficult to decide what should be measured or measuring a known quantity is impractical.

A dataset is a collection of many objects called *examples* (or *data points*), with each example containing many *features* that have been quantitatively measured.

Unsupervised learning algorithms **experience** a dataset containing many features, then learn useful properties of the structure of this dataset. In the context of deep learning, entire probability distribution that generated a dataset is to be learnt. Supervised learning algorithms experience a dataset containing features, but each example is also associated with a *label* or *target*. For example, the Iris dataset is annotated with the species of each iris plant.

Roughly speaking, unsupervised learning involves observing several examples of a random vector x, and attempting to implicitly or explicitly learn the probability distribution p(x), or some interesting properties of that distribution, while supervised learning involves observing several examples of a random vector x and an associated value or vector y , and learning to predict y from x, usually by estimating p(y | x).

1.1.3 Common Machine Learning Algorithms

- Neural networks
- Decision trees
- Random forests
- Associations and sequence discovery
- Gradient boosting and bagging
- Support vector machines
- Nearest-neighbour mapping
- k-means clustering
- Self-organizing maps
- Local search optimization techniques (e.g., genetic algorithms)
- Expectation maximization
- Multivariate adaptive regression splines
- Bayesian networks
- Kernel density estimation
- Principal component analysis
- Singular value decomposition
- Gaussian mixture models
- Sequential covering rule building

1.2 Deep Learning: An Overview

In the early days of artificial intelligence, the field rapidly tackled and solved problems that are intellectually difficult for human beings but relatively straightforward for computers—problems that can be described by a list of formal, mathematical rules.

The true challenge to artificial intelligence proved to be solving the tasks that are easy for people to perform but hard for people to describe formally—problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images.

The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. On drawing a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, this approach to AI is called *deep learning*.

Several artificial intelligence projects have sought to hard-code knowledge about the world in formal languages. A computer can reason about statements in these formal languages automatically using logical inference rules. This is known as the *knowledge base* approach to artificial intelligence. The performance of these simple machine learning algorithms depends heavily on the *representation* of the data they are given. Eg. Naïve Bayes algorithm.

Using machine learning to discover not only the mapping from representation to output but also the representation itself, is known as *representation learning*. Learned representations often result in much better performance than can be obtained with hand-designed representations. E.g. Autoencoder.

When designing features or algorithms for learning features, the goal is usually to separate the *factors of variation* that explain the observed data. Such factors are often not quantities that are directly observed. Instead, they may exist either as unobserved objects or unobserved forces in the physical world that affect observable quantities.

A major source of difficulty in many real-world artificial intelligence applications is that many of the factors of variation influence every single piece of data we are able to observe. The individual pixels in an image of a red car might be very close to black at night. The shape of the car's silhouette depends on the viewing angle. Most applications require us to the factors of variation *disentangle* and discard the ones that we do not care about.

Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning allows the computer to build complex concepts out of simpler concepts. Learning or evaluating this mapping seems insurmountable if tackled directly. Deep learning resolves this difficulty by breaking the desired complicated mapping into a series of nested simple mappings, each described by a different layer of the model. The representation also stores state information that helps to execute a program that can make sense of the input.

The quintessential example of a deep learning model is the feedforward deep network or *multilayer perceptron* (MLP). A multilayer perceptron is just a mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions.

The idea of learning the right representation for the data provides one perspective on deep learning. Another perspective on deep learning is that depth allows the computer to learn a multi-step computer program. Each layer of the representation can be thought of as the state of the computer's memory after executing another set of instructions in parallel. Networks with greater depth can execute more instructions in sequence. Sequential instructions offer great power because later instructions can refer back to the results of earlier instructions.

There are two main ways of measuring the depth of a model. The first view is based on the number of sequential instructions that must be executed to evaluate the architecture. Another approach, used by deep probabilistic models, regards the depth of a model as being not the depth of the computational graph but the depth of the graph describing how concepts are related to each other. Because it is not always clear which of these two views is more relevant graphs, there is no single correct value for the depth of an architecture, just as there is no single correct value for the length of a computer program. There is also no consensus about how much depth a model requires to qualify as "deep".

To summarize, deep learning is an approach to AI. Specifically, it is a type of machine learning, a technique that allows computer systems to improve with experience and data. Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

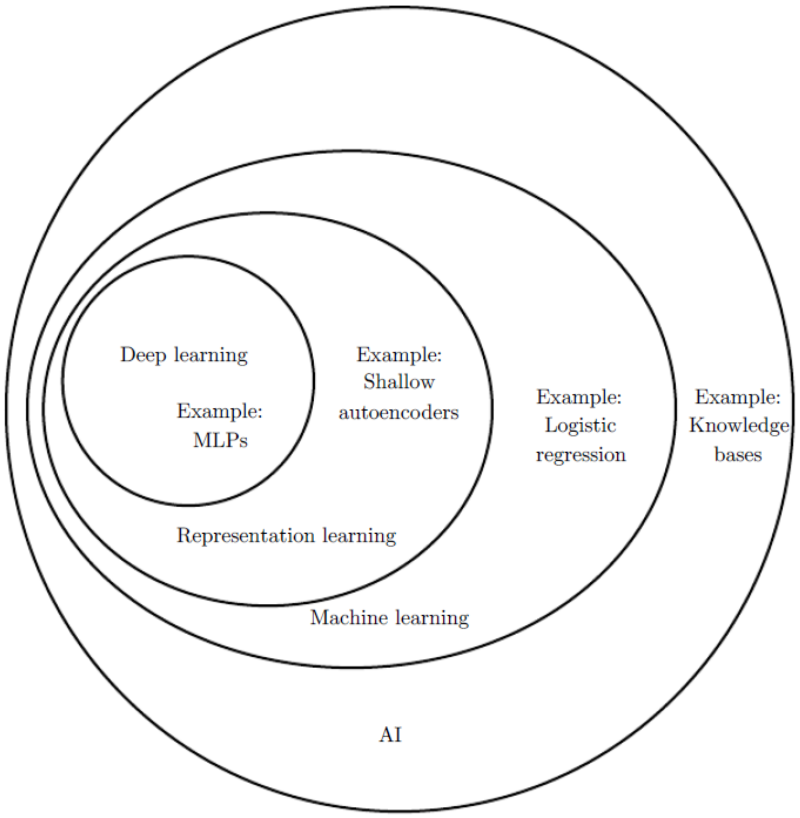


Fig 1.1 A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI.

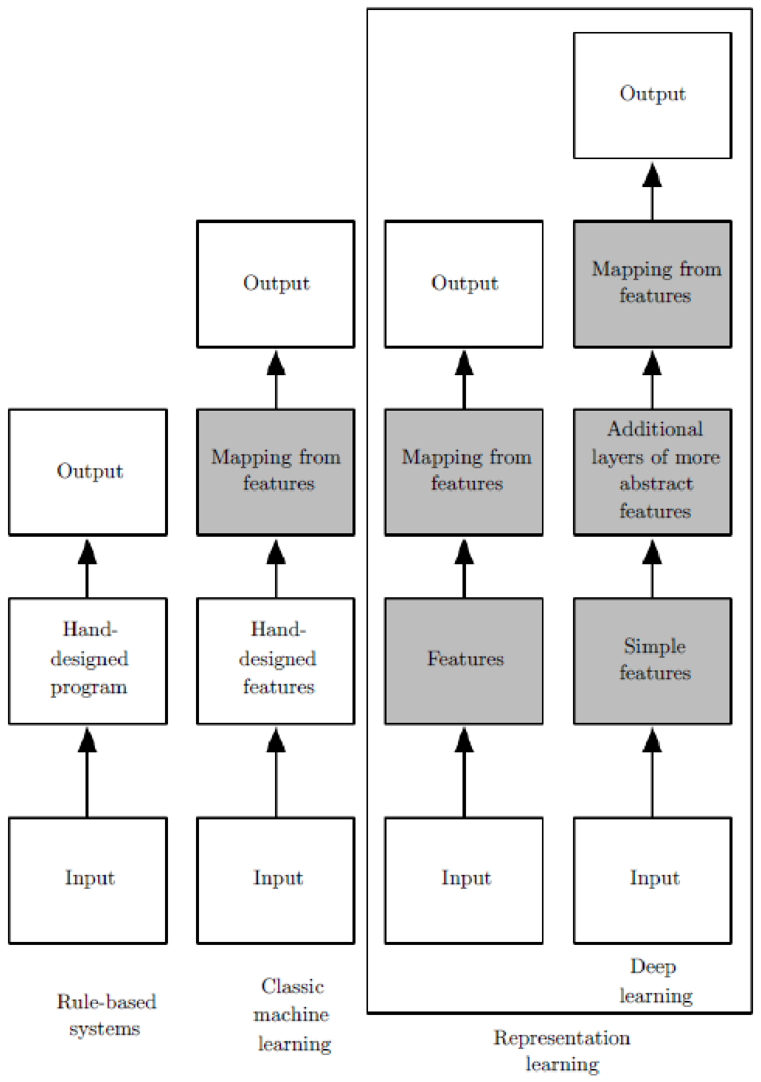


Fig 1.2 Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines. Shaded boxes indicate components that are able to learn from data. Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines. Shaded boxes indicate components that are able to learn from data.

There is also no *Distributed Representation* follows the idea that each input to a system should be represented by many features, and each feature should be involved in the representation of many possible inputs. For example, suppose we have a vision system that can recognize cars, trucks, and birds and these objects can each be red, green, or blue. One way of representing these inputs would be to have a separate neuron or hidden unit that activates for each of the nine possible combinations: red truck, red car, red bird, green truck, and so on. This requires nine different neurons, and each neuron must independently learn the concept of colour and object identity. One way to improve on this situation is to use a distributed representation, with three neurons describing the colour and three neurons describing the object identity. This requires only six neurons total instead of nine, and the neuron describing redness is able to learn about redness from images of cars, trucks and birds, not only from images of one specific category of objects.

Another major accomplishment of the connectionist movement was the successful use of back-propagation to train deep neural networks with internal representations.

The age of “Big Data” has made machine learning much easier because the key burden of statistical estimation—generalizing well to new data after observing only a small amount of data—has been considerably lightened. Another key reason that neural networks are wildly successful today after enjoying comparatively little success since the 1980s is that we have the computational resources to run much larger models today.

2. Facial Expression Recognition using Deep Learning Techniques

2.1 Work-Industry Overview

The Indian Institute of Technology Bombay is a public engineering and research institution located in Powai, Mumbai, India. In the *QS World University Rankings 2014,* IIT Bombay was ranked as India’s top university. It is the second-oldest (after Indian Institute of Technology Kharagpur) institute of the Indian Institutes of Technology system. IIT Bombay was founded in 1958.

IIT Bombay has a comprehensive graduate program offering doctoral degrees in Science, Technology, Engineering and Mathematics. Currently, IIT Bombay has a total of 14 academic departments, six centres, one school, and three interdisciplinary programmes. Educational programmes here extend beyond the physical sciences and engineering into humanities and social sciences such as Economics, English, Philosophy, Psychology and Sociology and into management studies.

IIT Bombay has 17 departments, 13 multi-disciplinary centres, and 3 schools of excellence. IIT Bombay offers a wide variety of courses of study in engineering, pure sciences, design, management and humanities with a primary focus on engineering. The university is a member of "Links to Asia by Organizing Traineeship and Student Exchange" (LAOTSE), an international network of leading universities in Europe and Asia exchanging students and senior scholars.

Faculty members from IIT Bombay undertake industry sponsored research and consultancy projects that are made available through the institute. These are funded by various national agencies like the Department of Science and Technology, Department of Electronics, Department of Space, Aeronautical Development Agency, Department of Atomic Energy, and Oil and Natural Gas Commission (ONGC). Many are also working on projects of national importance. A few projects are also being funded by international agencies. Typically in one year, there are about 400 on-going sponsored projects. The sponsored research has ushered in intense research activity leading to the formation of active research groups and has helped in the creation of modern research facilities in key areas.

The office of the Dean (R&D) provides the necessary liaison with industry and sponsoring agencies. The office helps industry to identify faculty expertise and institutional facilities, and assists faculty in identifying industry problems.

Prof Virendra Singh is an Associate Professor at IIT Bombay and is the head of the Computer Architecture and Dependable Systems Lab (CADSL). His research interests include Computer Architecture, Processor architecture & micro-architecture, Memory system design, Reconfigurable computing, Adaptive computing/architectures, Compiler support for modern architectures, Fault-tolerant computing, Robust design and architectures, Self-healing system design, VLSI testing and design for testability, SoC/NoC design and test, Post Silicon Debug, High level synthesis, Formal verification, Trusted computing, FPGA based acceleration, Trusted hardware design.

2.2 Job Description

The student was a research-intern at the Computer Architecture and Dependable Systems Lab (CADSL) of the Electrical Engineering Department of the Indian Institute of Technology, Bombay. CADSL was established in 2012 to conduct research in the area of: (a) Advance and futuristic architecture and system including compiler and operating system support for architecture, (b) Advanced dependable system including formal verfication and VLSI testing, and (c) Computer Aided design of VLSI and hardware accelerator.

The project was conducted under the guidance of Prof Virendra Singh of the Electrical Engineering Department. The student's participation in the project spanned a little over two months, from 9th May, 2016 to 2nd July, 2016.

2.3 Project Description

The automatic recognition of facial expressions has been an active research topic since the early nineties. There have been several advances in the past few years in terms of face detection and tracking, feature extraction mechanisms and the techniques used for expression classification.

A facial expression is one or more motions or positions of the muscles beneath the skin of the face. According to one set of controversial theories, these movements convey the emotional state of an individual to observers. Facial expressions are a form of nonverbal communication. They are a primary means of conveying social information between humans, but they also occur in most other mammals and some other animal species. Humans can adopt a facial expression voluntarily or involuntarily, and the neural mechanisms responsible for controlling the expression differ in each case. Voluntary facial expressions are often socially conditioned and follow a cortical route in the brain. Conversely, involuntary facial expressions are believed to be innate and follow a subcortical route in the brain.

A facial recognition system is a computer application capable of identifying or verifying a person from a digital image or a video frame from a video source. One of the ways to do this is by comparing selected facial features from the image and a facial database. It is typically used in security systems and can be compared to other biometrics such as fingerprint or eye iris recognition systems. Recently, it has also become popular as a commercial identification and marketing tool.

Face expression analysis and recognition has been one of the fast developing areas due to its wide range of application areas such as emotion analysis, biometrics, image retrieval and is one of the subjects on which lots of research has been done through solving the problems occurring in recognition of the face expressions under different illuminations, orientations and numerous other variations.

Different methods, all aiming to meet different requirements, have been used in solving facial expression analysis problems. These methods consist of pre-processing and processing parts. The detection and extraction of face images from the input data together with the normalization process, which aims to align these extracted images independent of varying environmental conditions such as illumination and orientation, form up the pre-processing part. The processing part on the other hand; aims to extract specific features from the already pre-processed images, and recognize the facial action units/facial expressions depending on these features. Several methods try to find different optimised algorithms mainly for the processing part.

Deep Learning and Convolutional Neural Networks have achieved far higher levels of accuracy and performance than traditional and previous methods of image recognition. Thus, the aim of this project has been to create a Facial Expression Recognition system, which utilizes CNNs from video.

The project aims to successfully recognize the expression of a person in an input image, and classify the expression into one of seven categories:

1. Anger
2. Disgust
3. Fear
4. Happiness
5. Sorrow
6. Surprise
7. Neutral

2.4 Need for Project

The universality hypothesis is the assumption that certain facial expressions are signals of specific emotions (happiness, sadness, anger, fear, surprise, and disgust) that are recognized by people everywhere, regardless of culture or language. The evolutionary basis of these kinds of facial expressions can be traced back to Darwin's *The Expression of the Emotions in Man and Animals.*

The uses of automatic facial expression recognition are countless. It can be used to gauge the attentiveness of students in classes, in hospitals to gauge if a patient is undergoing stress or pain, in cameras to automatically take a picture when the subject smiles etc.

2.5 Work Done and Implementation

The project is divided into two sections:

1. Face detection
2. Classification of face expression

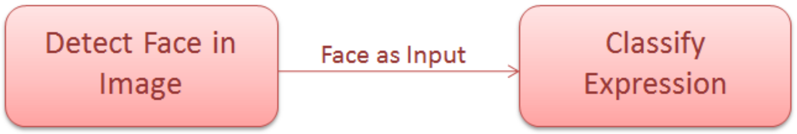


Fig 3.1 Project Tasks

The first task is completed using a combination of image processing and machine learning, with the Viola-Jones object detection framework. The second task is performed using deep learning through Convolutional Neural Networks.

2.5.1 Face Detection

The Viola–Jones object detection framework is the first object detection framework to provide competitive object detection rates in real-time proposed in 2001 by Paul Viola and Michael Jones. Although it can be trained to detect a variety of object classes, it was motivated primarily by the problem of face detection.

The problem to be solved is detection of faces in an image. A human can do this easily, but a computer needs precise instructions and constraints. To make the task more manageable, Viola–Jones requires full view frontal upright faces. Thus in order to be detected, the entire face must point towards the camera and should not be tilted to either side. While it seems these constraints could diminish the algorithm's utility somewhat, because the detection step is most often followed by a recognition step, in practice these limits on pose are quite acceptable.

Their ground-breaking 2001 paper describes a visual object detection framework that is capable of processing images extremely rapidly while achieving high detection rates. There are three key contributions. The first is the introduction of a new image representation called the “Integral Image” which allows the features used by the detector to be computed very quickly. The second is a learning algorithm, based on AdaBoost, which selects a small number of critical visual features and yields extremely efficient classifiers. The third contribution is a method for combining classifiers in a “cascade” which allows background regions of the image to be quickly discarded while spending more computation on promising object-like regions. A set of experiments in the domain of face detection are presented. The system yields face detection performance comparable to the best previous systems.

The advantages of the Viola–Jones algorithm are:

- Robust – very high detection rate (true-positive rate) & very low false-positive rate always.
- Real time – For practical applications at least 2 frames per second must be processed.

- Face detection only (not recognition) - The goal is to distinguish faces from non-faces (detection is the first step in the recognition process).

The algorithm has four stages: Haar Feature Selection, Creating an Integral Image, Adaboost Training and Cascading Classifiers.

The algorithm uses a grey scale image as input. Using Integral Images allows for fast feature evaluation. A set of features like the Haar Basis functions are used. A large amount of features can be used and reduced using a modified AdaBoost procedure: Weak learner is constrained so that each weak classifier returned can depend on only a single feature. Thus, every stage of the boosting process (which selects a new weak classifier) is like a feature selection process. The classifiers are in a cascade structure. This reduces speed drastically, with similar accuracy.

AdaBoost (Adaptive boosting) is used to boost the classification performance of a simple learning algorithm. Does this by combining a collection of weak classification functions to form a stronger classifier. The simple learning algorithm is called a weak learner. The weak learner solves a sequence of learning problems to be boosted. After the first round of learning, examples are reweighted to emphasize those which were incorrectly classified by the previous weak classifier. Training error of the strong classifier approaches zero with increase in the number of rounds.

A Convolution kernel is a matrix used for blurring, sharpening, embossing, detecting edges etc. an image. A Receiver Operating Characteristic (ROC) is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

Integral Image is an intermediate representation of the image. Integral image at (x,y) is the sum of pixels above and to the left of (x,y). It can be computed in one pass over original image. Integral image reduces the amount of initial image processing required.

The image is classified based on features (not pixels), as it works faster. Three kinds of features are used: two-rectangle, three-rectangle and four-rectangle. Rectangle features are sensitive to edges, bars and other simple structures. Steerable filters are good for detailed analysis of boundaries, image compression and texture analysis. However, rectangle features have extreme computational efficiency. Thus, they have been used. Features have been selected such that a single feature can be evaluated at any scale and location.

A Steerable Filter is an orientation-selective convolution kernel used for image enhancement and feature extraction that can be expressed via a linear combination of a small set of rotated versions of itself.

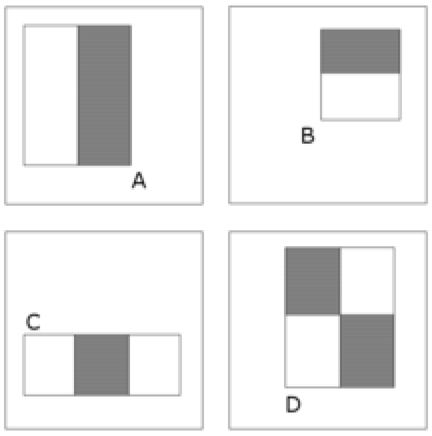


Fig 3.2 Three kinds of features are used: two-rectangle, three-rectangle and four-rectangle. Rectangle features are sensitive to edges, bars and other simple structures. Rectangle features have extreme computational efficiency. Thus, they have been used. Features have been selected such that a single feature can be evaluated at any scale and location.

Adaptive Boosting: There are far too many rectangle features associated with each sub-window (more than the number of pixels). A small number of features can be used to form an effective classifier. Thus, a variant of AdaBoost is used both to select the features and to train the classifier. The training error of the strong classifier approaches zero exponentially in the number of rounds. AdaBoost ensures that a small number of features with good variety are selected (By restricting the weak learner to select a single rectangle feature. For each feature, the weak learner determines a classification function to minimize the number of misclassified functions.)

The Attentional Cascade: Using a cascade of classifiers (instead of a single monolithic one) improves the detection performance and reduces computation time. The cascade structure is homogenous, trading processing time and detection performance. Smaller boosted classifiers reject many negative sub-windows while detecting almost all positive instances. Simple classifiers reject majority of the sub-windows. Then, complex classifiers ensure low false positive rates. Stages in the cascade are constructed by training classifiers using AdaBoost. To reduce number of sub-windows: Evaluate rectangle features, Compute weak classifier for each feature and then Combine weak classifiers. The first classifier evaluates all sub-windows. The positive outcome of a sub-window triggers the second classifier and so on. A negative outcome leads to rejection of a sub-window.

Training a Cascade of Classifiers: Target rates for each stage in the cascade process. Key measure for a classifier is the “positive rate” (proportion of windows labelled as containing the object). Optimization is done by trading off the number of classifier stages, number of features per stage, threshold of each stage to minimize expected number of features. In practice, each layer is trained by AdaBoost, increasing the number of features till they reach the targets (minimum acceptable rates for detection and false positives given by the user) for that level.

Tightly cropped images got slightly worse results. Due to the features used, the size of sub-windows does not affect speed. Speed is linked to the number of features evaluated per sub-window, i.e. the number of features that depends on the image. Sub-windows were normalized to reduce the effect of lighting. Multiple detections occur around each face in a scanned image. Thus, post-processing is required to combine overlapping detections.

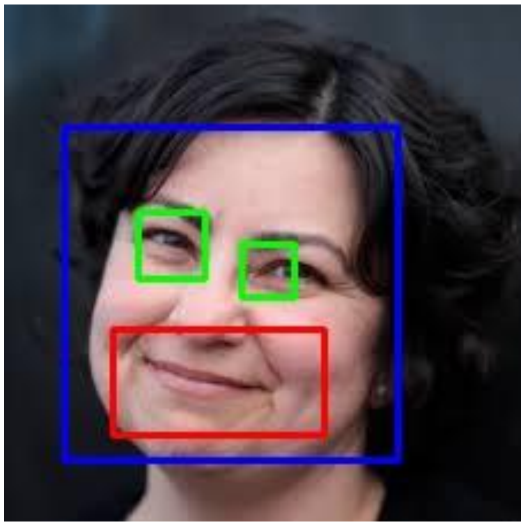


Fig 3.3 The results of the Viola-Jones detector, used to detect the face, eyes and mouth.

2.5.2 Expression Classification

The face detection module selects the face (or faces) from the entire image. This is then sent to the next module where the expression is recognised.

2.5.2.1 Selecting and Enabling a Platform

Deep learning requires high computational resources. Graphics Processing Units (GPUs) are required to perform accelerated matrix calculations. The student enabled the NVIDIA GeForce 8400 GPU with CUDA 7.0 on her system to successfully implement deep learning programs.

Apart from this, a DL library is also required for efficient programming. There exist many DL libraries, some of which were considered below.

Theano is a Python framework developed by the LISA group (now MILA) run by Yoshua Bengio at the University of Montreal for research and development into state of the art deep learning algorithms. It is better described as a mathematical expression compiler where one symbolically defines what one needs and the framework complies one's program to run efficiently on GPUs or CPUs. It is a research platform more than a deep learning library. A lot of work needs to be done to create the models. For example, there are no neural network classes. The Deep Convolutional Networks, Stacked Denoising Auto-Encoders, Deep Belief Networks are some of the architectures which can be created using this library. It is easy to create models from scratch but hard to implement existing models

Theano is actually an ecosystem and in practice it is not used directly. There are many libraries built on top of Theano that provides a handy wrapper API. Some more popular projects include:

- Keras
- Lasagne
- Blocks
- Pylearn2

These are becoming very large projects in and of themselves, providing helpful APIs into the underlying Theano platform, greatly accelerating the speed at which you can put models together.

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow 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. Keras is compatible with**: Python 2.7-3.5.** Some features of Keras are:

- It allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- It supports both convolutional networks and recurrent networks, as well as combinations of the two.
- It supports arbitrary connectivity schemes (including multi-input and multi-output training).
- It runs seamlessly on CPU and GPU.

Caffe is a Python deep learning library developed by Yangqing Jia at the Berkeley Vision and Learning Center for supervised computer vision problems. The primary focus is Convolutional Neural Networks and it may be the world leader. A big benefit of the library is the number of pre-trained networks that can be downloaded from the Caffe Model Zoo and used immediately. This includes state of the art models that can achieve world class results on standard computer vision datasets. For example here are some tutorials for world class models: Alex's CIFAR-10 tutorial with Caffe, Training LeNet on MNIST with Caffe, ImageNet with Caffe.

It is used more for development. It is easy to implement existing models and thus for beginners. However, it is hard to create models from scratch (there is a need to alter underlying code in which caffe is written).

Torch is an open source machine learning library, a scientific computing framework, and a script language based on the Lua programming language. It was developed by Ronan Collobert, Clement Farabet and Koray Kavukcuoglu for research and development into deep learning algorithms. It was used and promoted by the CILVR Lab at NYU (home to Yann LeCun). Torch is used and has been further developed by the Facebook AI lab, Google DeepMind, Twitter and a host of others.

Under the covers Torch makes use of C/C++ libraries as well as CUDA for GPU. It has a goal of speed whist adopting the C-friendly language Lua to provide a less intimidating interface. *The goal of Torch is to have maximum flexibility and speed in building your scientific algorithms while making the process extremely simple.* There is a lot of documentation, but it is a mess. Popular applications of Torch are for supervised image problems with Convolutional Neural Networks and agents in more complex domains with deep reinforcement learning. It provides a wide range of algorithms for deep machine learning, and uses an extremely fast scripting language LuaJIT, and an underlying C implementation. It does not work on Windows. It is only used by Facebook and Google DeepMind. It is used more for research

After evaluating the advantages and disadvantages of the above models, the student decided to use Theano with a Keras wrapper for the project.

2.5.2.2 Convolutional Neural Networks

In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex, whose individual neurons are arranged in such a way that they respond to overlapping regions tiling the visual field. Convolutional networks were inspired by biological processes and are variations of multilayer perceptrons designed to use minimal amounts of preprocessing. They have wide applications in image and video recognition, recommender systems[and natural language processing.

When used for image recognition, convolutional neural networks (CNNs) consist of multiple layers of small neuron collections which process portions of the input image, called receptive fields. The outputs of these collections are then tiled so that their input regions overlap, to obtain a better representation of the original image; this is repeated for every such layer. Tiling allows CNNs to tolerate translation of the input image.

Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters. They also consist of various combinations of convolutional and fully connected layers, with pointwise nonlinearity applied at the end of or after each layer. To reduce the number of free parameters and improve generalization, a convolution operation on small regions of input is introduced. One major advantage of convolutional networks is the use of shared weight in convolutional layers, which means that the same filter (weights bank) is used for each pixel in the layer; this both reduces memory footprint and improves performance.

Some time delay neural networks also use a very similar architecture to convolutional neural networks, especially those for image recognition and/or classification tasks, since the tiling of neuron outputs can be done in timed stages, in a manner useful for analysis of images.

Compared to other image classification algorithms, convolutional neural networks use relatively little pre-processing. This means that the network is responsible for learning the filters that in traditional algorithms were hand-engineered. The lack of dependence on prior knowledge and human effort in designing features is a major advantage for CNNs.

The design of convolutional neural networks follows the discovery of visual mechanisms in living organisms. Early 1968 work showed that the animal visual cortex contains complex arrangements of cells, responsible for detecting light in small, overlapping sub-regions of the visual field, called receptive fields. The paper identified two basic cell types: simple cells, which respond maximally to specific edge-like patterns within their receptive field, and complex cells, which have larger receptive fields and are locally invariant to the exact position of the pattern. These cells act as local filters over the input space.

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a m x m x r image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has r=3. The convolutional layer will have k filters (or kernels) of size n x n x q where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size m−n+1. Each map is then subsampled typically with mean or max pooling over p x p contiguous regions where p ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map. The figure below illustrates a full layer in a CNN consisting of convolutional and subsampling sublayers. Units of the same colour have tied weights. After the convolutional layers there may be any number of fully connected layers. The densely connected layers are identical to the layers in a standard multilayer neural network.

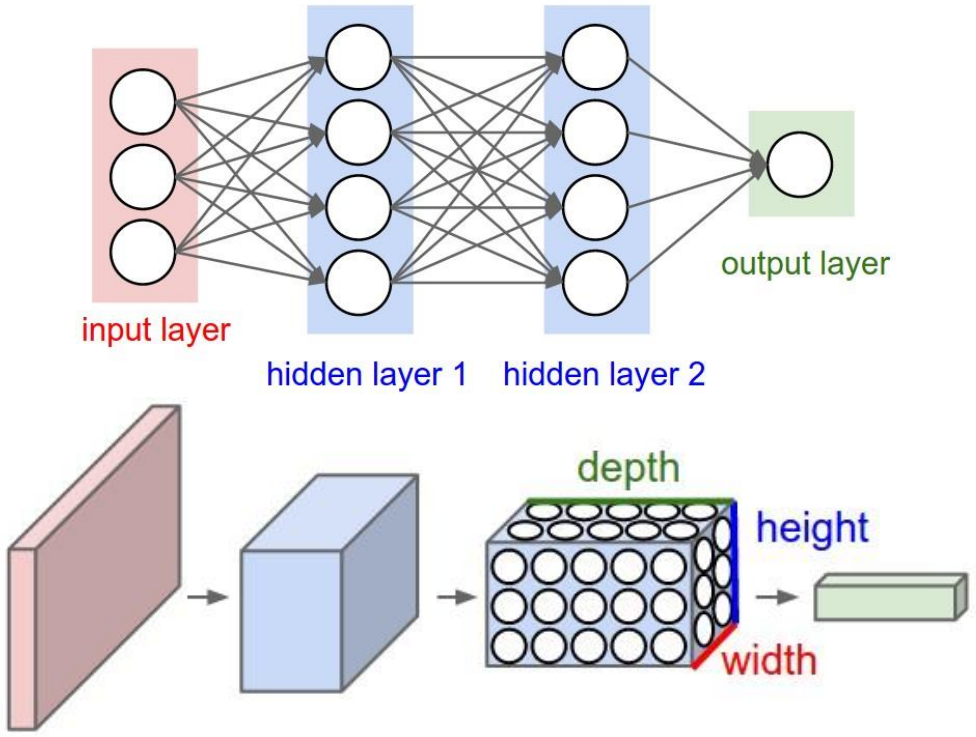


Fig 3.3 Top: A regular 3-layer Neural Network. Bottom: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height

would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

2.5.2.3 Architecture of the Model

The model is trained using the FER2013 dataset. The dataset consists of 28709 training examples and 7178 testing examples. It consists of both frontal and non-frontal faces, with different angles of rotation. All images in the dataset are of 48x48 dimensions.

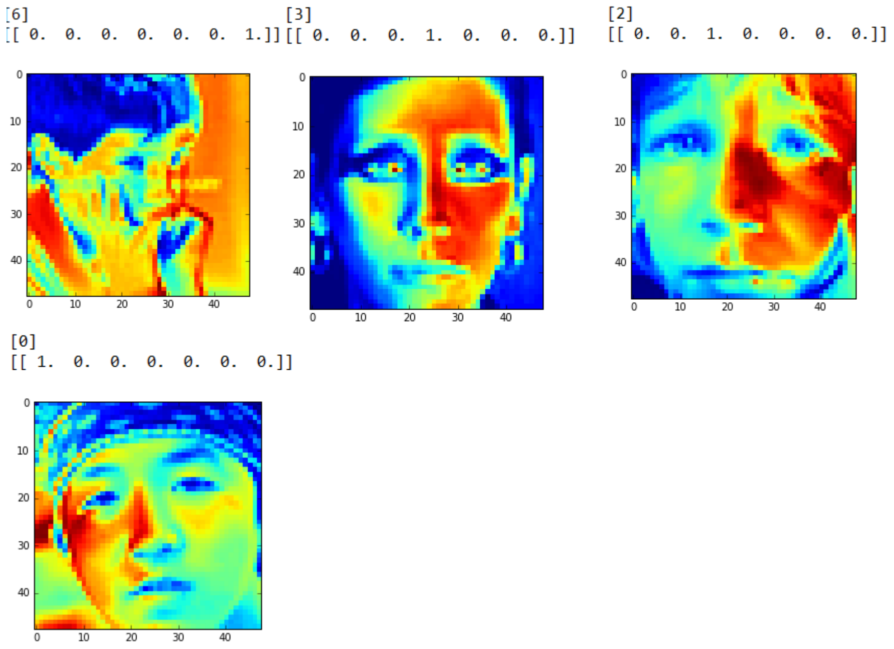
The model consists of 32 convolutional filters, a size of 2 for max pooling, a size of 3 for the convolution kernel, a batch size of 128, 50 epochs and 7 output classes. The data is read as a .csv file.

The model is a sequential model which consists of a three 2D convolution layers and ReLU activation layers. This is followed by a 2D MaxPooling layer and a Dropout layer of 0.25. This entire sequence is repeated thrice. The final activation layer contains a Softmax function which computes the final probabilities of the seven classes.

2.6 Results, Conclusions and Future Scope

The model succeeded in detecting the expressions of faces of the FER2013 dataset. The model can also handle rotated and non-frontal faces.

The model has a loss value of 1.12 and a test accuracy of 84.9%. It correctly identified the emotions of all the examples given below.



```
[6]
[[ 0.  0.  0.  0.  0.  0.  1.]]
[3]
[[ 0.  0.  0.  1.  0.  0.  0.]]
[2]
[[ 0.  0.  1.  0.  0.  0.  0.]]
[0]
[[ 1.  0.  0.  0.  0.  0.  0.]]
```

Fig 3.4 Results from the FER2013 dataset

The model can be adapted to recognising expressions in video. As the model is lightweight it has a low computation time during test time. Another method is to iteratively run the model after short bursts of time, thus creating an illusion of functioning in real-time.

The accuracy of the model can be increased further with a more complicated architecture. However, the model can be said to have succeeded in the task of Facial Expression Recognition.

References

AL-Azzi, M. M., Jun, H. D., & Abbas, T. **A new approach using Camshift Algorithm for multiple Vehicle Tracking.**

Allen, J. G., Xu, R. Y., & Jin, J. S. (2004, June). **Object tracking using camshift algorithm and multiple quantized feature spaces**. In *Proceedings of the Pan-Sydney area workshop on Visual information processing* (pp. 3-7). Australian Computer Society, Inc..

Baker, S., & Matthews, I. (2003). **Lucas-Kanade 20 years on: A unifying framework: Part 2.** *The Robotics Institute, Carnegie Mellon University*.

Baker, S., & Matthews, I. (2004). **Lucas-Kanade 20 years on: A unifying framework.** *International journal of computer vision*, *56*(3), 221-255.

Čehovin, L., Leonardis, A., & Kristan, M. (2015). **Visual object tracking performance measures revisited**. *arXiv preprint arXiv:1502.05803*.

Chibelushi, C. C., & Bourel, F. (2003). **Facial expression recognition: A brief tutorial overview.** *CVonline: On-Line Compendium of Computer Vision, 9*.

Denil, M., Bazzani, L., Larochelle, H., & de Freitas, N. (2012). **Learning where to attend with deep architectures for image tracking**. *Neural computation,24*(8), 2151-2184.

Exner, D., Bruns, E., Kurz, D., Grundhöfer, A., & Bimber, O. (2010, June). **Fast and robust CAMShift tracking.** In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on* (pp. 9-16). IEEE.

Gall, J., Yao, A., Razavi, N., Van Gool, L., & Lempitsky, V. (2011). **Hough forests for object detection, tracking, and action recognition.** *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *33*(11), 2188-2202.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). **Rich feature hierarchies for accurate object detection and semantic segmentation**. In*Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

Guraya, F. F. E., Bayle, P. Y., & Cheikh, F. A. (2009). **People tracking via a modified camshift algorithm**. *1Department of Computer Science and Media Technology, Gjovik University College, 2Department of Computer Science, Universite de Bourgogne*.

Henriques, J. F., Caseiro, R., Martins, P., & Batista, J. (2012). **Exploiting the circulant structure of tracking-by-detection with kernels.** In *Computer Vision–ECCV 2012* (pp. 702-715). Springer Berlin Heidelberg.

Henriques, J. F., Caseiro, R., Martins, P., & Batista, J. (2015). **High-speed tracking with kernelized correlation filters.** *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *37*(3), 583-596.

Hochreiter, S., & Schmidhuber, J. (1997). **Long short-term memory**. *Neural computation*, *9*(8), 1735-1780.

Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach* (pp. 1059-74). GMD-Forschungszentrum In

Kalal, Z., Mikolajczyk, K., & Matas, J. (2012). **Tracking-learning-detection**. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *34*(7), 1409-1422.

Karasulu, B., & Korukoglu, S. (2013). **Moving Object Detection and Tracking in Videos.** In *Performance Evaluation Software* (pp. 7-30). Springer New York.

Kristan, M., Matas, J., Leonardis, A., Felsberg, M., Cehovin, L., Fernandez, G., ... & Pflugfelder, R. (2015). **The Visual Object Tracking VOT2015 Challenge Results**. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (pp. 1-23).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). **Imagenet classification with deep convolutional neural networks**. In *Advances in neural information processing systems* (pp. 1097-1105).

Laaraiedh, M. (2012). **Implementation of Kalman filter with python language**.*arXiv preprint arXiv:1204.0375*.

Li, Q., Niaz, U., & Merialdo, B. (2012, June). **An improved algorithm on Viola-Jones object detector**. In *Content-Based Multimedia Indexing (CBMI), 2012 10th International Workshop on* (pp. 1-6). IEEE.

Liao, S., Jain, A., & Li, S. (2014). **A Fast and Accurate Unconstrained Face Detector**.

LIRIS, F. **The Visual Object Tracking VOT2014 challenge results**.

Mnih, V., Heess, N., & Graves, A. (2014). **Recurrent models of visual attention**. In *Advances in Neural Information Processing Systems* (pp. 2204-2212).

Sivaraman, S., & Trivedi, M. M. (2013). **Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis**. *Intelligent Transportation Systems, IEEE Transactions on*, *14*(4), 1773-1795.

Smola, A., & Vishwanathan, S. V. N. (2008). Introduction to machine learning.*Cambridge University, UK*, *32*, 34.

Sobral, A., & Bouwmans, T. (2014). **BGS Library: A Library Framework for Algorithm's Evaluation in Foreground/Background Segmentation**. *Handbook on" Background Modeling and Foreground Detection for Video Surveillance", Chapter 23*.

Valenti, R., Sebe, N., Gevers, T., & Cohen, I. (2008). **Machine learning techniques for face analysis**. In *Machine Learning Techniques for Multimedia*(pp. 159-187). Springer Berlin Heidelberg.

Various sources from the internet.

Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). **Show and tell: A neural image caption generator**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3156-3164).

Viola, P., & Jones, M. (2001). **Robust real-time object detection.** *International Journal of Computer Vision*, *4*.

Xia, J., Wu, J., Zhai, H., & Cui, Z. (2009). **Moving vehicle tracking based on double difference and camshift.** In *Proceedings of the International Symposium on Information Processing* (Vol. 2).

Xia, J., Rao, W., Huang, W., & Lu, Z. (2013). **Automatic multi-vehicle tracking using video cameras: An improved CAMShift approach.** *KSCE Journal of Civil Engineering*, *17*(6), 1462-1470.

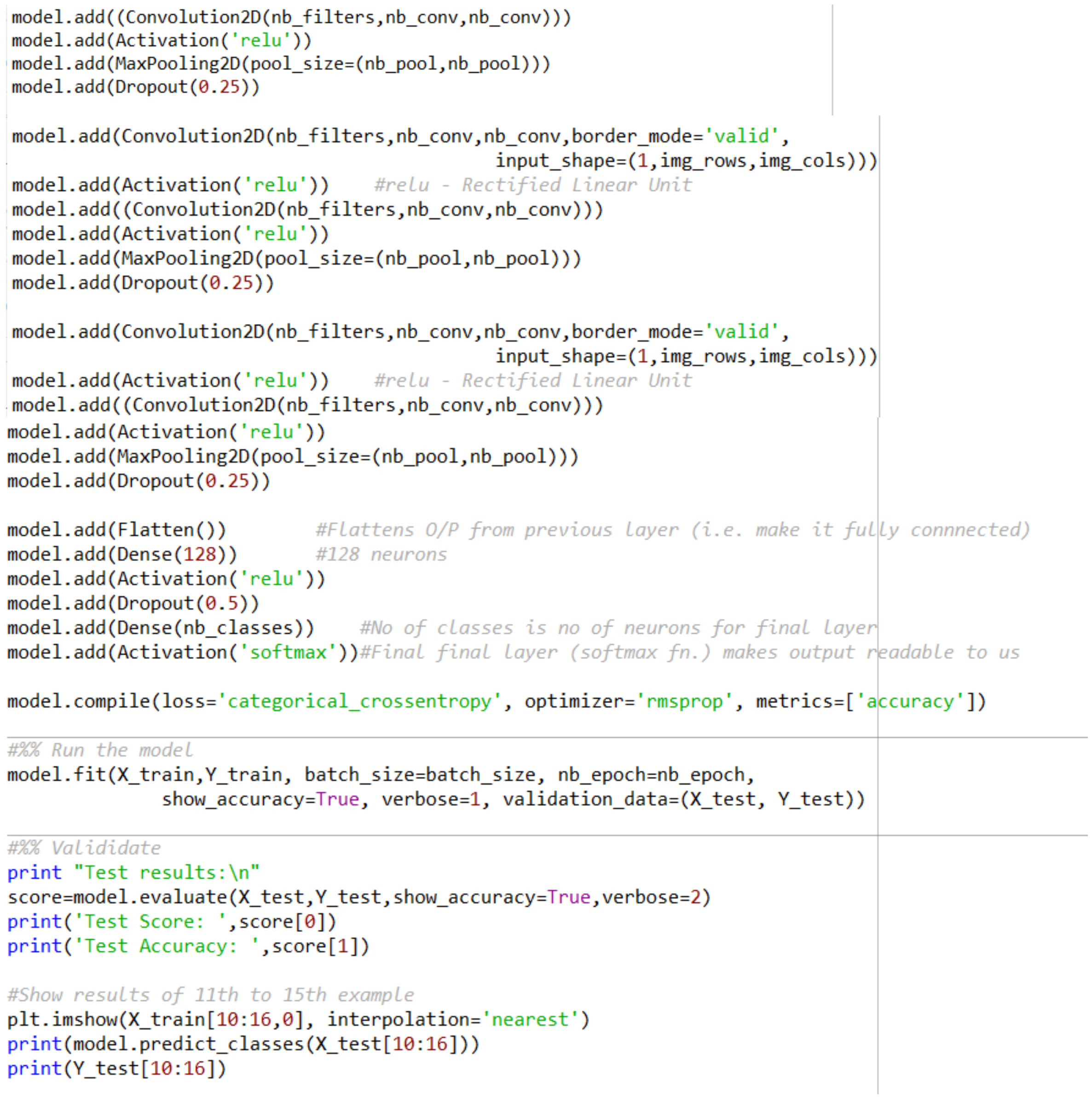
Xu, K., Ba, J., Kiros, R., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015). **Show, attend and tell: Neural image caption generation with visual attention.** *arXiv preprint arXiv:1502.03044*.

Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., & Toderici, G. (2015). **Beyond short snippets: Deep networks for video classification**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4694-4702).

Zhang, L., Chu, R., Xiang, S., Liao, S., & Li, S. Z. (2007). **Face detection based on multi-block lbp representation**. In *Advances in biometrics* (pp. 11-18). Springer Berlin Heidelberg.

Appendix: Code for Facial Expression Recognition using Convolutional Neural Networks

```
#CNN on Facial Expression Recognition, 2013 Dataset (From Kaggle)
#Seven emotions: 0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral

import numpy as np
import csv
from keras.utils import np_utils
import matplotlib.pyplot as plt

from keras.models import Sequential #type of model
from keras.layers.core import Dense, Dropout, Activation, Flatten #layers
from keras.layers.convolutional import Convolution2D, MaxPooling2D #convolution layers

#%% Define values
nb_filters=32            #Number of convolutional filters to be used
nb_pool=2                #Size of pooling area for max pooling
nb_conv=3                #Size of convolution kernel
batch_size=128           #Batch size to train
nb_epoch=50              #Number of epochs to train

img_rows, img_cols=48, 48    #Dimensions of input images
nb_classes=7             #Number of output classes (7 emotions)

#%% Extract training and testing data from a CSV file
'''
X_train - Training data        Y_train - Labels for data
X_test - Testing data          Y_test - Actual output (used to measure accuracy)
X_train.shape     Output: No_of_samples, no_of_channels, imgage_dimensions

To extract the data from .csv files:
    a=str(row[1])          #Extract all pixel positions as one long string        '1 2 3 4'
    b=a.split(' ')         #Split pixel values using ' ' as delimiter           ['1','2','3','4']
    b=map(int,b)           #Convert list of Strings to list of integers         [1,2,3,4]
    b=np.array(b)          #Convert to numpy array
'''

csvr = csv.reader(open('C:/Users/Rheeya/Desktop/fer2013.csv'))
header = csvr.next()
rows = [row for row in csvr]

#Extract as lists and convert to ndarrays
X_train = np.array([map(int,str(row[1]).split(' ')) for row in rows if row[2] == 'Training'])
Y_train = np.array([row[0] for row in rows if row[2] == 'Training'], dtype='uint8')
X_test = np.array([map(int,str(row[1]).split(' ')) for row in rows if row[2] == 'Testing'])
Y_test = np.array([row[0] for row in rows if row[2] == 'Testing'], dtype='uint8')

#Reshape the data
X_train=X_train.reshape(X_train.shape[0],1,img_rows,img_cols)
X_test=X_test.reshape(X_test.shape[0],1,img_rows,img_cols)

#Change datatype from 64 bit float to 32 bit float
X_train=X_train.astype('float32')
X_test=X_test.astype('float32')

#Normailize the data
X_train/=255
X_test/=255

Y_train=np_utils.to_categorical(Y_train, nb_classes)     #Convert label of '1' to [0,1,0,0,0,0,0]
Y_test=np_utils.to_categorical(Y_test, nb_classes)       #Convert label of '1' to [0,1,0,0,0,0,0]

#%% Define Model

model=Sequential()  #Declare a sequential model

model.add(Convolution2D(nb_filters,nb_conv,nb_conv,border_mode='valid',
                                              input_shape=(1,img_rows,img_cols)))
model.add(Activation('relu'))     #relu - Rectified Linear Unit
model.add((Convolution2D(nb_filters,nb_conv,nb_conv)))
model.add(Activation('relu'))
```



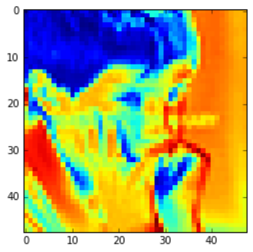
```
model.add((Convolution2D(nb_filters,nb_conv,nb_conv)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool,nb_pool)))
model.add(Dropout(0.25))

model.add(Convolution2D(nb_filters,nb_conv,nb_conv,border_mode='valid',
                                            input_shape=(1,img_rows,img_cols)))
model.add(Activation('relu'))    #relu - Rectified Linear Unit
model.add((Convolution2D(nb_filters,nb_conv,nb_conv)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool,nb_pool)))
model.add(Dropout(0.25))

model.add(Convolution2D(nb_filters,nb_conv,nb_conv,border_mode='valid',
                                            input_shape=(1,img_rows,img_cols)))
model.add(Activation('relu'))    #relu - Rectified Linear Unit
model.add((Convolution2D(nb_filters,nb_conv,nb_conv)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool,nb_pool)))
model.add(Dropout(0.25))

model.add(Flatten())        #Flattens O/P from previous layer (i.e. make it fully connnected)
model.add(Dense(128))       #128 neurons
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))    #No of classes is no of neurons for final layer
model.add(Activation('softmax'))#Final final layer (softmax fn.) makes output readable to us

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

#%% Run the model
model.fit(X_train,Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
              show_accuracy=True, verbose=1, validation_data=(X_test, Y_test))

#%% Valididate
print "Test results:\n"
score=model.evaluate(X_test,Y_test,show_accuracy=True,verbose=2)
print('Test Score: ',score[0])
print('Test Accuracy: ',score[1])

#Show results of 11th to 15th example
plt.imshow(X_train[10:16,0], interpolation='nearest')
print(model.predict_classes(X_test[10:16]))
print(Y_test[10:16])
```

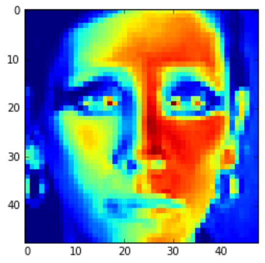
```
Train on 28709 samples, validate on 7178 samples
Epoch 1/50
28709/28709 [==============================] - 60s - loss: 1.2122 - acc: 0.5381 - val_loss: 1.1148 - val_acc: 0.5699
Epoch 2/50
28709/28709 [==============================] - 60s - loss: 1.1985 - acc: 0.5453 - val_loss: 1.1135 - val_acc: 0.5718
Epoch 3/50
28709/28709 [==============================] - 61s - loss: 1.1933 - acc: 0.5462 - val_loss: 1.0703 - val_acc: 0.5896
Epoch 4/50
28709/28709 [==============================] - 61s - loss: 1.1834 - acc: 0.5506 - val_loss: 1.0733 - val_acc: 0.5888
Epoch 5/50
28709/28709 [==============================] - 61s - loss: 1.1774 - acc: 0.5517 - val_loss: 1.0667 - val_acc: 0.5901
Epoch 6/50
28709/28709 [==============================] - 62s - loss: 1.1759 - acc: 0.5544 - val_loss: 1.0734 - val_acc: 0.5922
Epoch 7/50
28709/28709 [==============================] - 61s - loss: 1.1616 - acc: 0.5583 - val_loss: 1.0461 - val_acc: 0.5960
Epoch 8/50
28709/28709 [==============================] - 60s - loss: 1.1553 - acc: 0.5615 - val_loss: 1.0575 - val_acc: 0.5955
Epoch 9/50
28709/28709 [==============================] - 62s - loss: 1.1471 - acc: 0.5652 - val_loss: 1.0549 - val_acc: 0.5986
Epoch 10/50
28709/28709 [==============================] - 61s - loss: 1.1423 - acc: 0.5691 - val_loss: 1.0609 - val_acc: 0.5936
Epoch 11/50
28709/28709 [==============================] - 61s - loss: 1.1369 - acc: 0.5687 - val_loss: 1.0191 - val_acc: 0.6088
Epoch 12/50
28709/28709 [==============================] - 61s - loss: 1.1313 - acc: 0.5714 - val_loss: 1.0163 - val_acc: 0.6116
Epoch 13/50
28709/28709 [==============================] - 67s - loss: 1.1262 - acc: 0.5737 - val_loss: 1.0043 - val_acc: 0.6098
Epoch 14/50
28709/28709 [==============================] - 62s - loss: 1.1192 - acc: 0.5746 - val_loss: 1.0192 - val_acc: 0.6101
Epoch 15/50
28709/28709 [==============================] - 61s - loss: 1.1189 - acc: 0.5745 - val_loss: 0.9656 - val_acc: 0.6285
Epoch 16/50
28709/28709 [==============================] - 61s - loss: 1.1153 - acc: 0.5745 - val_loss: 0.9788 - val_acc: 0.6282
Epoch 17/50
28709/28709 [==============================] - 61s - loss: 1.1053 - acc: 0.5814 - val_loss: 1.0089 - val_acc: 0.6057
Epoch 18/50
28709/28709 [==============================] - 62s - loss: 1.1013 - acc: 0.5842 - val_loss: 0.9815 - val_acc: 0.6240
Epoch 19/50
28709/28709 [==============================] - 62s - loss: 1.0989 - acc: 0.5833 - val_loss: 0.9745 - val_acc: 0.6236
Epoch 20/50
28709/28709 [==============================] - 61s - loss: 1.0949 - acc: 0.5836 - val_loss: 0.9712 - val_acc: 0.6337
Epoch 21/50
28709/28709 [==============================] - 61s - loss: 1.0901 - acc: 0.5871 - val_loss: 0.9529 - val_acc: 0.6344
Epoch 22/50
28709/28709 [==============================] - 61s - loss: 1.0911 - acc: 0.5911 - val_loss: 0.9188 - val_acc: 0.6507
Epoch 23/50
28709/28709 [==============================] - 61s - loss: 1.0805 - acc: 0.5924 - val_loss: 0.9514 - val_acc: 0.6330
Epoch 24/50
28709/28709 [==============================] - 62s - loss: 1.0725 - acc: 0.5950 - val_loss: 0.9300 - val_acc: 0.6384
Epoch 25/50
28709/28709 [==============================] - 62s - loss: 1.0730 - acc: 0.5957 - val_loss: 0.9906 - val_acc: 0.6169
Epoch 26/50
28709/28709 [==============================] - 61s - loss: 1.0792 - acc: 0.5964 - val_loss: 0.9155 - val_acc: 0.6513
Epoch 27/50
28709/28709 [==============================] - 61s - loss: 1.0699 - acc: 0.5949 - val_loss: 0.9084 - val_acc: 0.6518
Epoch 28/50
28709/28709 [==============================] - 62s - loss: 1.0696 - acc: 0.5946 - val_loss: 0.9295 - val_acc: 0.6435
Epoch 29/50
28709/28709 [==============================] - 62s - loss: 1.0681 - acc: 0.5996 - val_loss: 0.9001 - val_acc: 0.6545
Epoch 30/50
28709/28709 [==============================] - 61s - loss: 1.0603 - acc: 0.6028 - val_loss: 0.9119 - val_acc: 0.6483
Epoch 31/50
28709/28709 [==============================] - 62s - loss: 1.0556 - acc: 0.6027 - val_loss: 0.9271 - val_acc: 0.6512
Epoch 32/50
28709/28709 [==============================] - 61s - loss: 1.0604 - acc: 0.6005 - val_loss: 0.9325 - val_acc: 0.6453
Epoch 33/50
28709/28709 [==============================] - 62s - loss: 1.0511 - acc: 0.6014 - val_loss: 0.8907 - val_acc: 0.6634
Epoch 34/50
28709/28709 [==============================] - 61s - loss: 1.0592 - acc: 0.6028 - val_loss: 0.9119 - val_acc: 0.6540
Epoch 35/50
28709/28709 [==============================] - 60s - loss: 1.0510 - acc: 0.6046 - val_loss: 0.9344 - val_acc: 0.6462
Epoch 36/50
28709/28709 [==============================] - 60s - loss: 1.0505 - acc: 0.6035 - val_loss: 0.8739 - val_acc: 0.6652
Epoch 37/50
28709/28709 [==============================] - 60s - loss: 1.0491 - acc: 0.6057 - val_loss: 0.9065 - val_acc: 0.6531
```

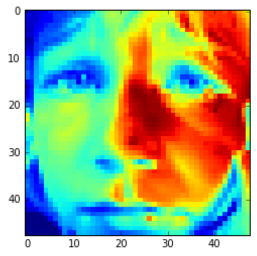
```
Epoch 38/50
28709/28709 [==============================] - 60s - loss: 1.0455 - acc: 0.6075 - val_loss: 0.8779 - val_acc: 0.6721
Epoch 39/50
28709/28709 [==============================] - 60s - loss: 1.0385 - acc: 0.6071 - val_loss: 0.8592 - val_acc: 0.6740
Epoch 40/50
28709/28709 [==============================] - 60s - loss: 1.0421 - acc: 0.6083 - val_loss: 0.8975 - val_acc: 0.6558
Epoch 41/50
28709/28709 [==============================] - 60s - loss: 1.0315 - acc: 0.6103 - val_loss: 0.8622 - val_acc: 0.6720
Epoch 42/50
28709/28709 [==============================] - 60s - loss: 1.0363 - acc: 0.6111 - val_loss: 0.8419 - val_acc: 0.6775
Epoch 43/50
28709/28709 [==============================] - 61s - loss: 1.0312 - acc: 0.6126 - val_loss: 0.8594 - val_acc: 0.6811
Epoch 44/50
28709/28709 [==============================] - 62s - loss: 1.0345 - acc: 0.6106 - val_loss: 0.8479 - val_acc: 0.6832
Epoch 45/50
28709/28709 [==============================] - 60s - loss: 1.0270 - acc: 0.6125 - val_loss: 0.8933 - val_acc: 0.6572
Epoch 46/50
28709/28709 [==============================] - 60s - loss: 1.0281 - acc: 0.6140 - val_loss: 0.8527 - val_acc: 0.6799
Epoch 47/50
28709/28709 [==============================] - 60s - loss: 1.0207 - acc: 0.6125 - val_loss: 0.8486 - val_acc: 0.6818
Epoch 48/50
28709/28709 [==============================] - 61s - loss: 1.0238 - acc: 0.6134 - val_loss: 0.8737 - val_acc: 0.6656
Epoch 49/50
28709/28709 [==============================] - 62s - loss: 1.0228 - acc: 0.6132 - val_loss: 0.8902 - val_acc: 0.6625
Epoch 50/50
28709/28709 [==============================] - 60s - loss: 1.0176 - acc: 0.6161 - val_loss: 0.8253 - val_acc: 0.6854
```

```
Test results:

('Test Accuracy: ', 0.8495945945945946)
```

```
[6]                               [3]                               [2]
[[ 0.  0.  0.  0.  0.  0.  1.]]   [[ 0.  0.  0.  1.  0.  0.  0.]]   [[ 0.  0.  1.  0.  0.  0.  0.]]
```







```
[0]
[[ 1.  0.  0.  0.  0.  0.  0.]]
```