

ABSTRACT

This report discusses AI and ML in the field of Computer Vision (CV) through a literature survey titled as “AI-ML On Computer Vision” and with a hands-on project on “Facial Expression And Hand Gesture Recognition” where we display the output in the form of text, which indicate the emotion or expression of the human and hand gesture symbols recognition respectively. This report also focuses on AI, ML, and computer vision algorithms and on some well-known algorithms, such as SVM, CNN, RNN, YoLO, RL and DL, which are frequently used in computer vision tasks, such as object detection, image classification, tracking, image segmentation and so on. We will learn about the mentioned contents by explaining various concepts, such as definitions, terms, and mathematical calculations, and comparing them to better understand the contents in the report.

CONTENTS

S.no	Topic	Page.no
1	Introduction	3-4
2	Support Vector Machine Algorithm	5-9
3	Deep Learning Algorithm	9-17
4	Reinforcement Learning Algorithm	17-22
5	Convolutional Neural Network Algorithm	22-26
6	Recurrent Neural Network Algorithm	27-32
7	YOLO Algorithm	32-38
8	Hand Gesture Recognition	39-50
9	Facial Expression Recognition	51-60
10	Conclusion	60
11	Reference links	61-64

Literature Survey Of AI-ML On Computer Vision

→**What is Artificial Intelligence(AI)?**

Ans:- The systems which can imitate humans which they can do like decision making,solving problems,learning and thinking is know as **Artificial Intelligence**.

→**What is Machine Learning(ML)?**

Ans:-We train the systems by using mathematical models of data such that it enables the system to continue learning and improving itself based on its own experience is known as **Machine Learning** .

→Machine Learning is a part of Artificial Intelligence but partially independent of each other.

→**How are AI & ML interconnected?**

Ans:-In AI,Intelligent Systems think & perform like human.In ML,it develops intelligence of a computer

→**What are the differences between AI & ML?**

ARTIFICIAL INTELLIGENCE	MACHINE LEARNING
AI stands for Artificial intelligence, where intelligence is defined as the ability to acquire and apply knowledge.	ML stands for Machine Learning which is defined as the acquisition of knowledge or skill

AI is the broader family consisting of ML and DL as its components.	Machine Learning is the subset of Artificial Intelligence.
The aim is to increase the chance of success and not accuracy.	The aim is to increase accuracy, but it does not care about the success
AI is aiming to develop an intelligent system capable of performing a variety of complex jobs. decision-making	Machine learning is attempting to construct machines that can only accomplish the jobs for which they have been trained.
It works as a computer program that does smart work.	Here, the tasks systems machine takes data and learns from data.
AI is decision-making.	ML allows systems to learn new things from data.
It is developing a system that mimics humans to solve problems.	It involves creating self-learning algorithms.
AI is a broader family consisting of ML and DL as its components.	ML is a subset of AI.
Three broad categories of AI are : 1. Artificial Narrow Intelligence (ANI) 2. Artificial General Intelligence (AGI) 3. Artificial Super Intelligence (ASI)	Three broad categories of ML are : 1. Supervised Learning 2. Unsupervised Learning 3. Reinforcement Learning
AI can work with structured, semi-structured, and unstructured data.	ML can work with only structured and semi-structured data.
AI's key uses include- <ul style="list-style-type: none"> • Siri, customer service via chatbots • Expert Systems • Machine Translation like Google Translate • Intelligent humanoid robots such as Sophia, 	The most common uses of machine learning- <ul style="list-style-type: none"> • Facebook's automatic friend suggestions • Google's search algorithms • Banking fraud analysis • Stock price forecast

Support Vector Machine(SVM):

→Support Vector Machine or SVM is one of the most popular supervised learning algorithms, which is used for **classification** as well as **regression** problems.

→Primarily, it is used for classification problems in Machine learning.

→The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.

→The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes.

→so that we can easily put the new data point in the correct category in the future.

→This best decision boundary is called a hyperplane.

→If the number of input features is two, then the hyperplane is just a line.

→If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

→SVM chooses the extreme points/vectors that help in creating the hyperplane.

→These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

→The vector points closest to the hyperplane are known as the support vector points because only these two points are contributing to the result of the algorithm, and other points are not.

→If a data point is not a support vector, removing it has no effect on the model. On the other hand, deleting the support vectors will then change the position of the hyperplane.

- The distance of the vectors from the hyperplane is called the margin.
- The line is represented with

$$w \cdot x + b = 0$$

For label 1, $w \cdot x + b \geq 1$

For label -1, $w \cdot x + b \leq -1$

- The distances of the gap in-between two classes is $2/|w|$

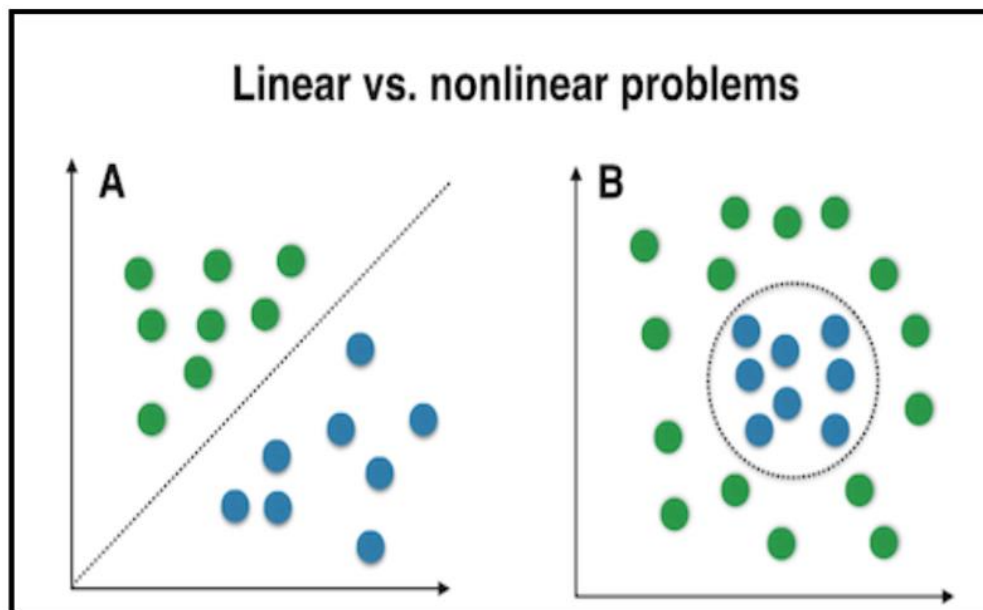
- we want to maximize this distance.

→If we want to find a line that perfectly separates the two classes, we call this type of SVM cost function as Hard-Margin cost Function.

- SVM can be of two types:

1)Linear SVM:- Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM Classifier.

2)Non-linear SVM:- Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.



ADVANTAGES:-

- 1)SVM works very well with higher-dimensional datasets.
- 2)SVM is one of the most memory-efficient classification algorithms.
- 3)The clearer the margin of separation between the categories, the better the SVM works.
- 4)Effective on datasets with multiple features, like financial or medical data.

DISADVANTAGES:-

- 1)SVM does not perform very well when the data set has more noise i.e. target classes are overlapping.

APPLICATIONS:-

1. Hand Written Detection
2. Image Classification
3. Face Detection
4. Bioinformatics
5. Cancer Detection
6. Sentimental Analysis
7. Spam Detection

****Non Linear Support Vector Machines:-**

→If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line.

→Imagine a case - if there is no straight line (or hyper plane) which can separate two classes? In the image shown below, there is a circle in 2D with red and blue data points all over it such that adjacent data points are of different colours.

→So to separate these data points, we need to add one more dimension.

→For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z.

→So ,here we use a kernel function to handle non-linear separable data.

→Kernels are used by classification algorithms to solve non-linear classification problems.

→The kernel function transforms the data into a higher dimensional feature space to make it possible to perform the linear separation.

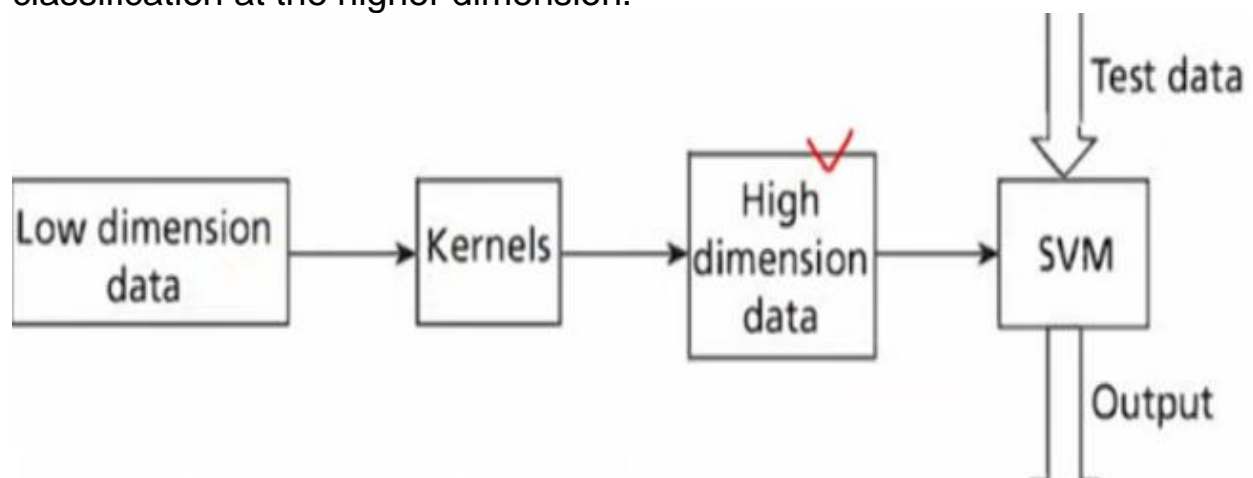
→In machine learning applications, the data can be text, image or video.

→So there is a need to extract features from these prior to classification.

→Hence, in the real world, many classification models are complex and mostly require non-linear hyperplanes.

→Kernels are a set of functions used to transform data from lower dimension to higher dimension and to manipulate data using dot product at higher dimensions.

→The use of kernels is to apply transformation to data and perform classification at the higher dimension.



→The kernel function is a function that may be expressed as the dot product of the mapping function (kernel method) and looks like this, $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$

→There are different types of Kernels are available to convert

non linear to linear.

1. linear: $u \cdot v$

2. polynomial: $(\gamma u \cdot v + \text{coef0})^{\text{degree}}$

3. radial basis (RBF) : $\exp(-\gamma |u-v|^2)$

4. sigmoid : $\tanh(\gamma u \cdot v + \text{coef0})$

→ RBF is generally the most popular one.

→ Deep learning is a subset of Machine Learning that uses multi-layered neural networks, called **deep neural networks**.

The mathematical formula behind RBF is:-

$$K(x, x') = e^{-\gamma \|x - x'\|^2}$$

Gamma is a scalar that defines how much influence a single training example (point) has.

Deep Learning Algorithm(DL):

→ Deep learning is a **type of artificial intelligence** that can **“duplicate” the human brain's** functioning.

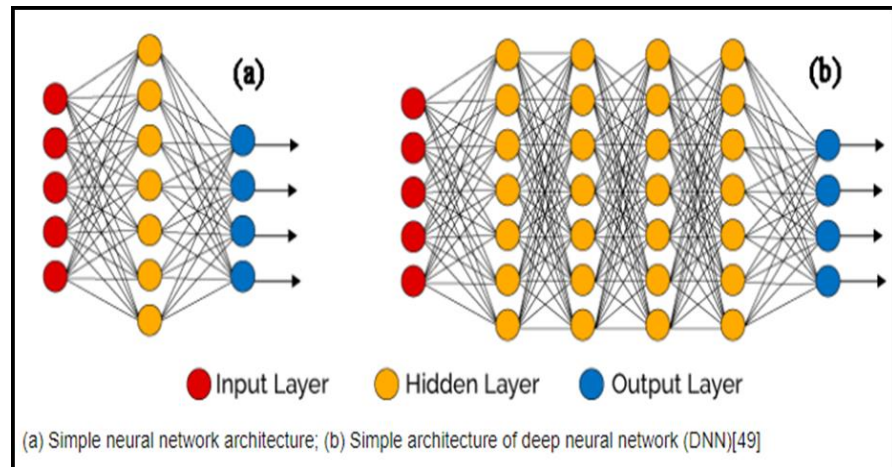
→ The term **"deep"** usually refers to the **number of hidden layers in the neural network**.

→ Models are trained by using a **large set of labelled data** and **neural network architectures** that **contain many layers**.

→ **Deep learning models can recognize complex patterns in pictures, text, sounds, and other data to produce accurate insights and prediction. It improves the ability to classify, recognize, detect and describe using data.**

→ **Neural networks are made up of layers of interconnected nodes, and each node is responsible for learning a specific feature of the**

data.



DIFFERENTIATE B/W AI, ML, DL



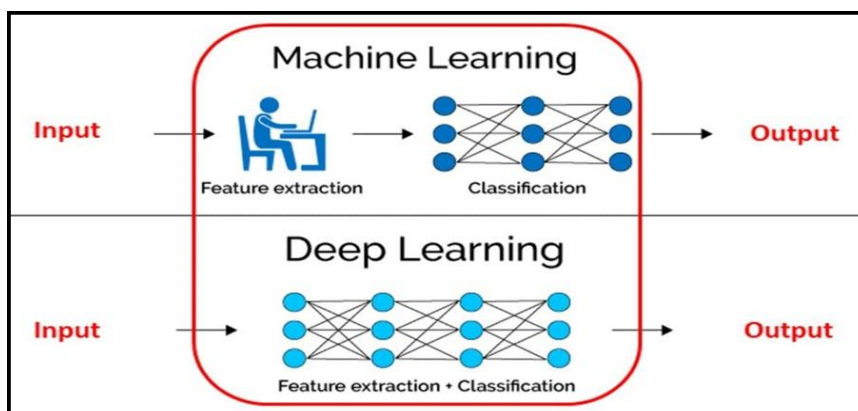
Artificial Intelligence: any technique that allows computer systems to mimic the human intelligence and behaviour



Machine Learning: subset of AI, that uses statistical techniques to enable machines to learn from data and improve with experience



Deep Learning: subset of ML, in which multilayered neural networks learn from vast amounts of data



→Deep learning **eliminates some of the data pre-processing** that is typically involved with machine learning.

Differences	Machine Learning	Deep Learning
Definition	Subset of AI	Subset of ML
Data	Performs well a small and medium dataset	Needs a Big Dataset
Hardware Requirements	Works with low-end machines	Requires machines with GPU
Engineering Particulars	Understands the features and how they represent the data	Needs to understand basic functionality of the data
Training Time	Short	Long
Processing Time	A few Seconds	A Few Hours or Minutes
Number of Algorithms	Many	Few
Data Interpretation	Some ML algorithms are easy to interpret, where some are hardly possible	Difficult
How it Works	Use various types of automated Algorithms that learn to model functions and predict future actions from data	Uses Neural Networks that pass data through many processing layers to interpret data features and relationships
Outputs	Numerical Value, like a classification or score	Anything from numerical values to free-form elements, like free text and sound

PREREQUISITES:-

→To effectively learn Deep Learning, which is a subset of machine learning based on Neural Networks with multiple layers, it is beneficial to have a understanding of the following fundamental concepts and topics:

1. Mathematics:

- 1. Linear Algebra:** Vectors, matrices, matrix operations, Eigen Values, Eigen Vectors, etc.
- 2. Calculus:** Derivatives, chain rule, gradients, partial derivatives, etc.
- 3. Probability and Statistics:** Probability distributions, Bayes' theorem, mean, variance, etc.

2. Programming Skills:

1. **Python:** Deep learning frameworks such as TensorFlow and PyTorch are commonly used with Python.
2. **NumPy:** For numerical computing in Python.
3. **Pandas:** For data manipulation and analysis.
4. **Jupyter Notebooks:** For interactive coding and experimentation.
3. **Machine Learning Concepts:**
 1. **Supervised Learning:** Understanding of concepts like regression and classification.
 2. **Unsupervised Learning:** Clustering, Dimensionality reduction, etc.
 3. **Model Evaluation:** Cross-validation, overfitting, underfitting, etc.
4. **Data Preprocessing:**

→Cleaning, normalisation, feature scaling, handling missing data etc.
5. **Basic Understanding of Neural Networks:**
 1. **Perceptron:** The basic building block of neural networks.
 2. **Activation Functions:** Sigmoid, ReLU, tanh, etc.
 3. **Back Propagation:** The algorithm used to train neural networks.
6. **Deep Learning Frameworks:**

→Familiarity with popular deep learning libraries such as TensorFlow or Keras is essential.

REASONS TO USE DEEP LEARNING:

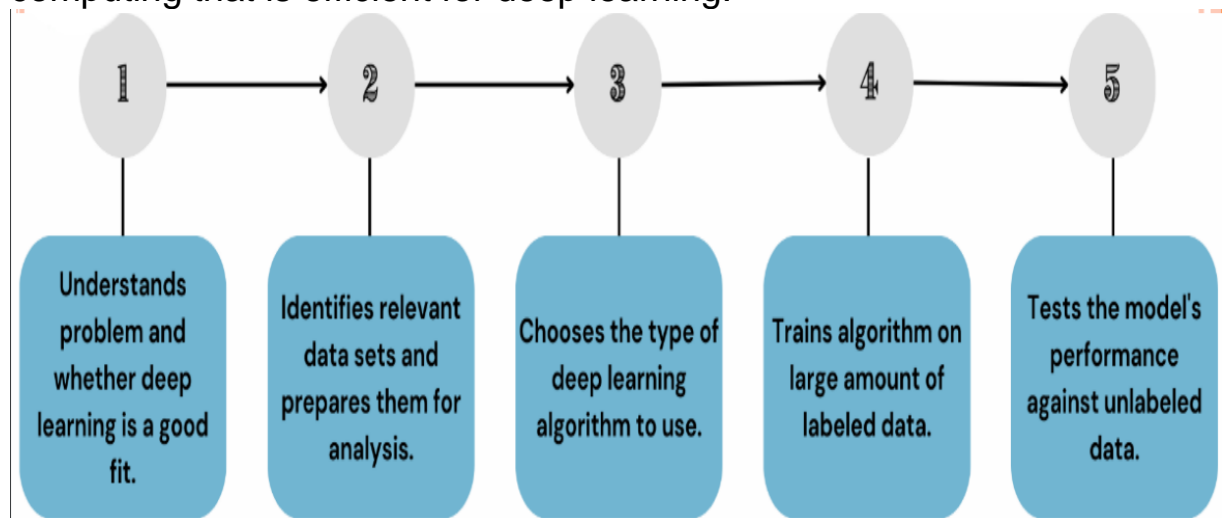
1. **Analysing Unstructured Data:** Deep learning algorithms can be trained to look at text data by analysing social media posts, news, and surveys to provide valuable business and customer insights.
2. **Feature engineering:** A deep learning algorithm can save time because it does not require humans to extract features manually from raw data.
3. **Efficiency:** When a deep learning Algorithm is properly trained, it can perform thousands of tasks over and over again, faster than humans.
4. **Training:** The Neural Networks used in deep learning have the ability to be applied to many different data types and applications. Additionally, a deep learning model can adapt by retraining it with new data. DL is usually a more complex and high-performance GPU to analyze all images.
5. **Handling large and complex data:** Deep Learning algorithms can deal with large and complex datasets that would be challenging for traditional Machine Learning algorithms to handle. This makes it useful for finding insights from big data, such as posts on social media, webpages, videos, audio files, and sensor data.
6. **Handling Non-Linear Relationships:** This allows it to model complex phenomena and capture higher-level abstractions. Some of the examples of handling non-linear relationships by Deep Learning algorithms are: a)

Physical Systems: A Deep Potential Molecular Dynamics (DPMD) achieved an accuracy of 98.9% on predicting the potential energy surface of water molecules, which is a highly non-linear function of atomic positions. b)

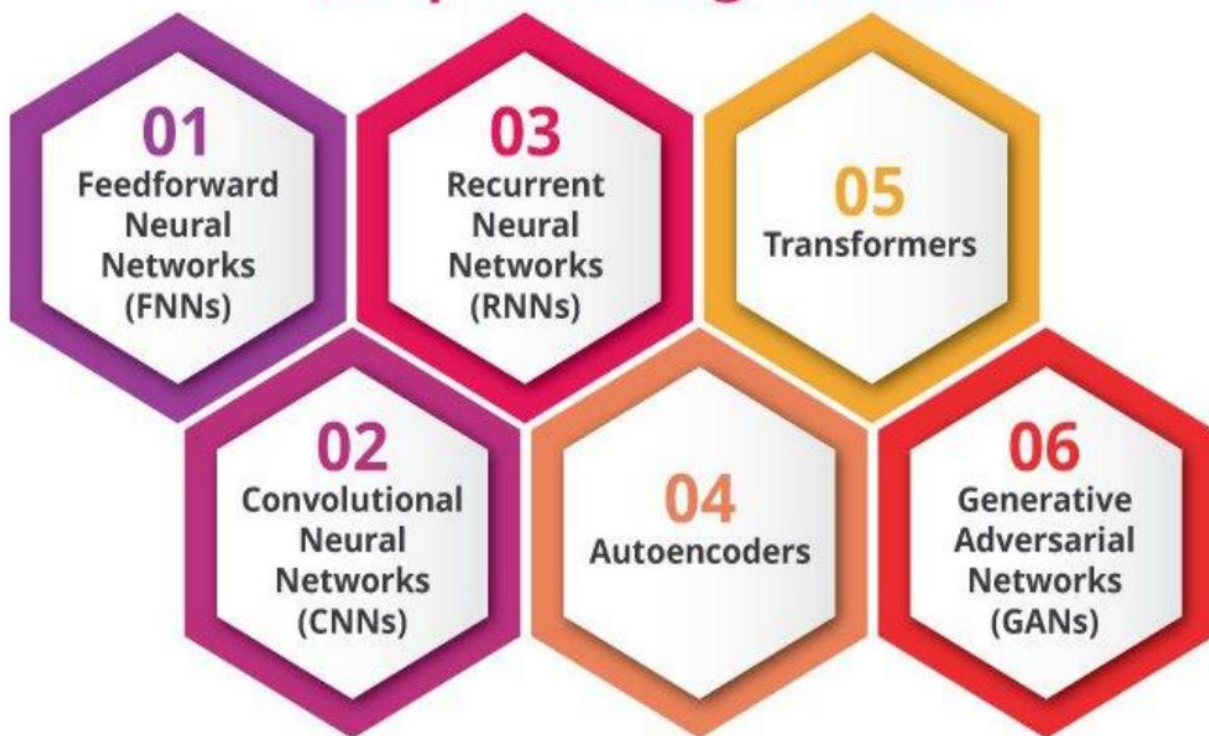
Biological Systems: A Deep Learning model called DeepChrome achieved an accuracy of 89.9% on predicting the chromatin state of genes.

How DEEP LEARNING WORKS

1. Deep learning work on Neural Network Architectures Deep learning is implemented by the help of deep networks, which are nothing but neural networks with multiple hidden layers.
2. It came in the 1980s, but due to a lack of labeled data and computing power, it was not popular at that time. Today, we have large amounts of labeled data.
3. For example, a driverless car requires millions of images and thousands of hours of video to train with high accuracy.
4. The number of hidden layers in the neural network usually refers to “deep”. Hidden layers in deep neural networks can have as many as 150.
5. These models are trained by using large sets of labeled data & and neural networks learn features directly from the data.
6. For computing power, we have high-performance GPUs, cloud computing that is efficient for deep learning.



Types Of Deep Learning models



1. Feed Forward Neural Networks(FNNs):

1. FNNs, also known as Multilayer Perceptrons (MLPs), are the simplest type of artificial neural network.
2. In an FNN, information moves in only one Direction. It can be moved from Input Layer to Output Layer. There are no back-loops in the feed-forward network.
3. To minimize the prediction error, the back propagation algorithm can be used to update the weight values.
4. FNNs are widely used for simple classification and regression problems.

Applications:

1. Data Compression
2. Pattern Recognition
3. Computer Vision
4. Sonar Target Recognition
5. Speech Recognition
6. Handwritten Characters Recognition

2. Convolutional Neural Networks(CNNs):

1. CNNs are a type of deep learning model that are especially effective for processing grid-like data such as images, making them extremely

useful for image recognition, image classification, clustering of images and object recognition etc.

2. An advanced example of CNN is its use in selfdriving cars, specifically in the system responsible for detecting objects around the vehicle.
3. The real-time application where the CNN doesn't just classify static images but continuously processes video frames to identify and classify objects such as other vehicles, pedestrians, traffic signs, and lanes.

Applications:

1. Identify Faces, Street Signs, Tumours.
2. Image Recognition.
3. Video Analysis.
4. NLP.
5. Anomaly Detection.
6. Drug Discovery.
7. Checkers Game.
8. Time Series Forecasting.

3. Recurrent Neural Networks(RNNs):

1. Recurrent neural networks get feedback from output neurons. RNNs are designed to recognize patterns in sequences of data, such as text, genomes, handwriting, or the spoken word, and are therefore very effective for natural language processing and speech recognition tasks.
2. Before Transformers came to age, a special variant of RNN Long Short-Term Memory Networks (LSTM) was prevalent in machine translation and text generation.
3. The Recurrent neural network mainly accesses the preceding info of existing iterations.
4. For example, to guess the succeeding word in any sentence, one must have knowledge about the words that were previously used. It not only processes the inputs but also shares the length as well as weights crossways time.

Applications:

1. Machine Translation
2. Robot Control
3. Time Series Prediction
4. Speech Recognition
5. Speech Synthesis
6. Time Series Anomaly Detection

7. Rhythm Learning
8. Music Composition

4. Auto Encoders:

1. Autoencoders are a specific type of neural network architecture used for learning efficient representations of input data in an unsupervised manner.
2. They are composed of two parts: An encoder, which transforms the input data into a lower-dimensional code, and a Decoder, which attempts to reconstruct the original input data from this code.
3. Auto encoders are often employed for anomaly detection in various types of data, from time-series sensor data to complex images.

Applications:

1. Classification.
2. Clustering.
3. Feature Compression.

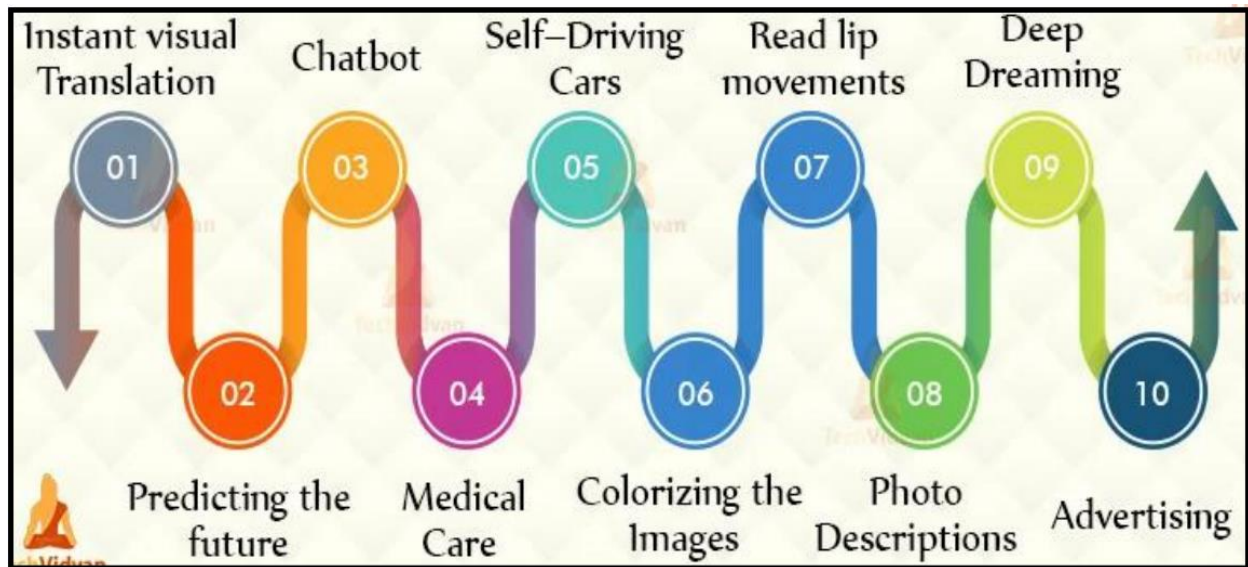
5. Transformers:

1. Transformers, specifically the “self-attention” mechanism within them, have been a game-changer in the field of deep learning.
2. They are especially powerful in handling sequence data for tasks such as language understanding, translation, and text summarization.
3. The best-known implementation of a transformer is OpenAI’s GPT-3 model, which can generate human-like text given some input.
4. It can be used for tasks like writing essays, answering questions, creating poetry, and even writing software code.

6. Generative Adversarial Networks (GANs):

1. GANs are a type of neural network that is used for generating new data samples that are similar to a training dataset.
2. They consist of two networks: a generator and a discriminator. The generator tries to generate new data samples, while the discriminator tries to distinguish between the generated samples and the training data.

Applications: GANs can be used to generate new examples for image datasets in various domains, such as medical imaging, satellite imagery, and natural language processing



DEEP LEARNING FRAMEWORKS:

1. Tensor Flow: It is one of the most popular deep learning frameworks. Developed by the Google Brain team, TensorFlow supports languages such as Python, C++, and R to create deep learning models along with wrapper libraries. It is available on both desktop and mobile. The most well-known use case of TensorFlow has got to be Google Translate coupled with capabilities such as natural language processing, text classification, summarization, speech/image/handwriting recognition, forecasting, and tagging.

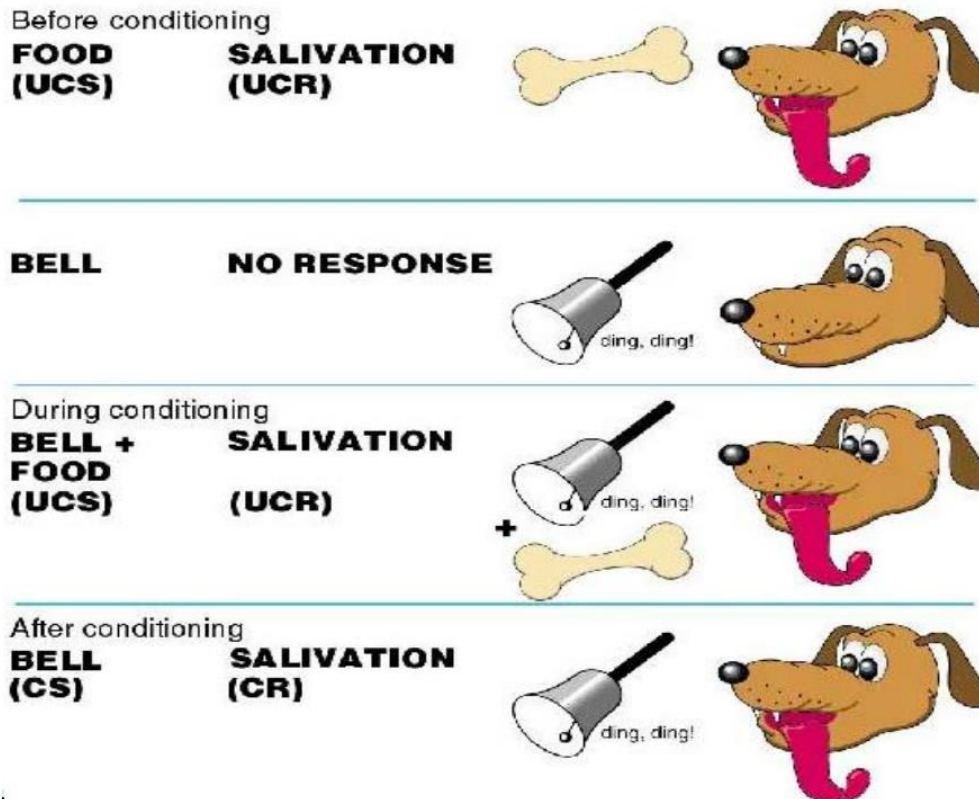
2.Keras: Keras library was developed, and Written in Python, the Keras neural networks library supports both convolutional and recurrent networks that are capable of running on either TensorFlow or Theano. The primary usage of Keras is in classification, text generation, and summarization, tagging, translation along with speech recognition, and others. The main use of Keras is , it supports multiple deep learning backends.

Reinforcement Learning Algorithm(RL):

- Reinforcement Learning is defined as a Machine Learning method that is concerned with how software agents should take actions in an environment.
- Reinforcement learning can be thought of as a hit and trial method of learning.

- The machine gets a Reward or Penalty point for each action it performs. If the option is correct, the machine gains the reward point or gets a penalty point in case of a wrong response.
- Reinforcement Learning(RL) is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences.

Example



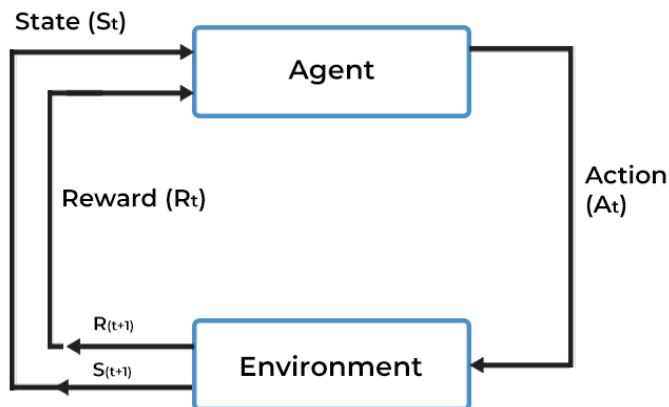
Why use Reinforcement Learning?

- It helps you to find which situation needs action.
- Reinforcement Learning also provides the learning agent with a reward function.
- It also allows it to figure out the best method for obtaining large rewards.
- The learning agent is based on exploration and exploitation.
- Exploration is when the learning agent acts on trial and error .And
- Exploitation is when it performs an action based on the knowledge gained from the environment and rewards the agent

for the environment for every correct action, which is the reinforcement signal.



REINFORCEMENT LEARNING MODEL



Terms used in Reinforcement Learning:

- **Agent:** It is an assumed entity which performs actions in an environment to gain some reward.
- **Environment (e):** A scenario that an agent has to face.
- **Action (a):** All the possible moves that the agent can take.
- **State (s):** Current situation returned by the environment.
- **Policy (π):** It is a strategy which is applied by the agent to decide the next action based on the current state.
- **Reward (R):** An immediate return given to an agent when he or she performs specific action or task.

→ There are 3 different approaches to implementing Reinforcement Learning.

- **1. Value Based**
- **2. Policy Based**
- **3. Model based**

1. Value Based : In a value-based Reinforcement Learning method, you should try to maximise a value function $V(s)$.

2. Policy Based : You try to come up with such a policy that the action performed in every state helps you to gain maximum reward in the future. **3. Model Based :** you need to create a virtual model for each environment.

→ There are two important learning models in reinforcement learning:

- **1. Markov Decision Process**

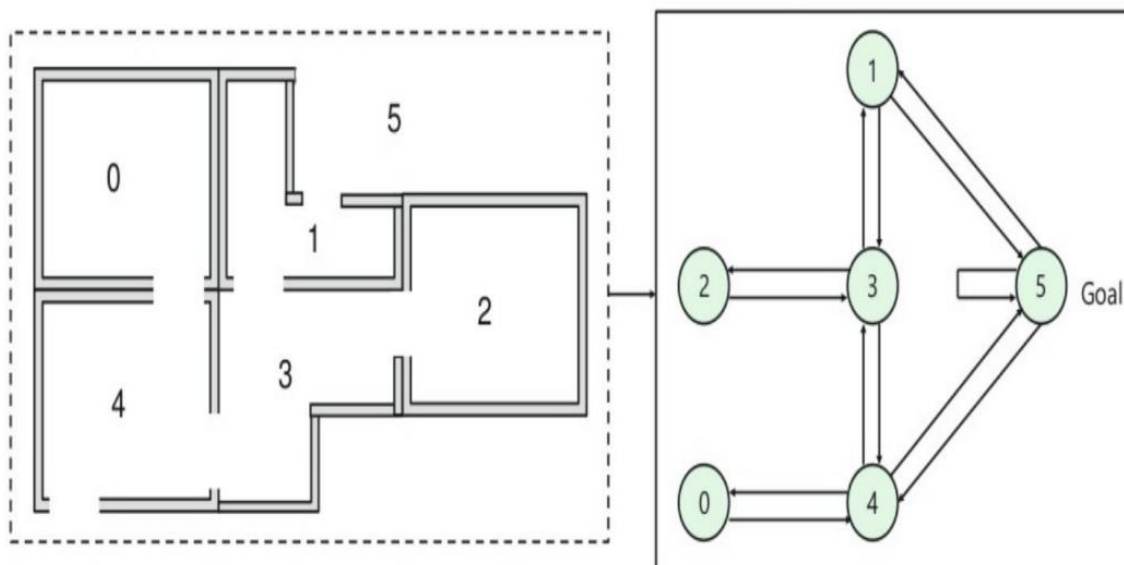
- **2. Q learning**

1. Markov Decision Process : The following parameters are used to get a solution:

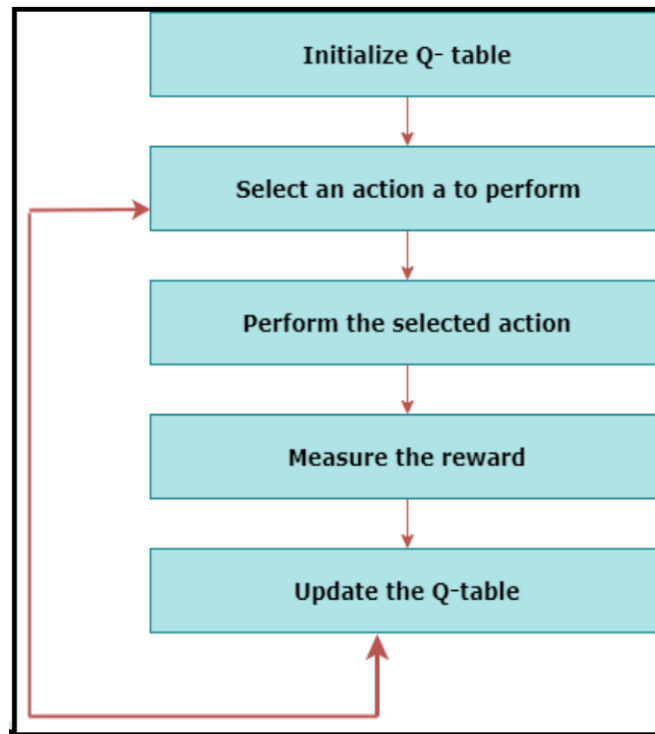
- Set of actions- A
- Set of states -S
- Reward- R
- Policy- π
- Value- V

2. Q – Learning: Q learning is a value-based method of supplying information to inform which action an agent should take. Let's understand this method by the following example:

- There are five rooms in a building which are connected by doors.
- Each room is numbered 0 to 4 .
- The outside of the building can be one big outside area (5).
- Doors number 1 and 4 lead into the building from room 5.



- **Algorithm:** Q-learning is an Off policy RL algorithm, which is used for temporal difference Learning. The temporal difference learning methods are the way of comparing temporally successive predictions.
- It learns the value function $Q(S, a)$, which means how good to take action "a" at a particular state "s."
- The below flowchart explains the working of Q- learning:



- Next, you need to associate a reward value to each door:
- Doors which lead directly to the goal have a reward of 100.
- Doors which are not directly connected to the target room give zero reward.
- As doors are two-way, and two arrows are assigned for each room. Every arrow in the above image contains an instant reward value.

Example:

For example, an agent traverse from room number 2 to 5

Initial state = state 2

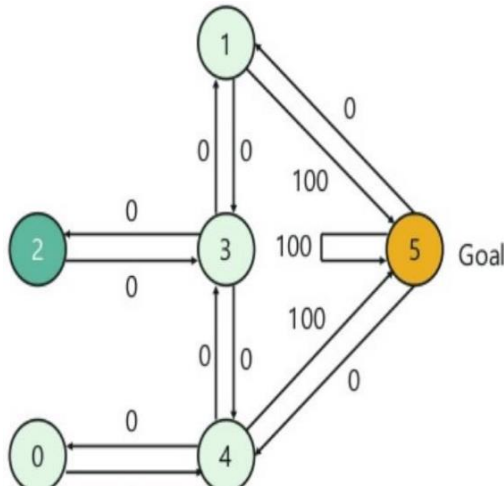
State 2-> state 3

State 3 -> state (2,1,4)

State 4-> state (0,5,3)

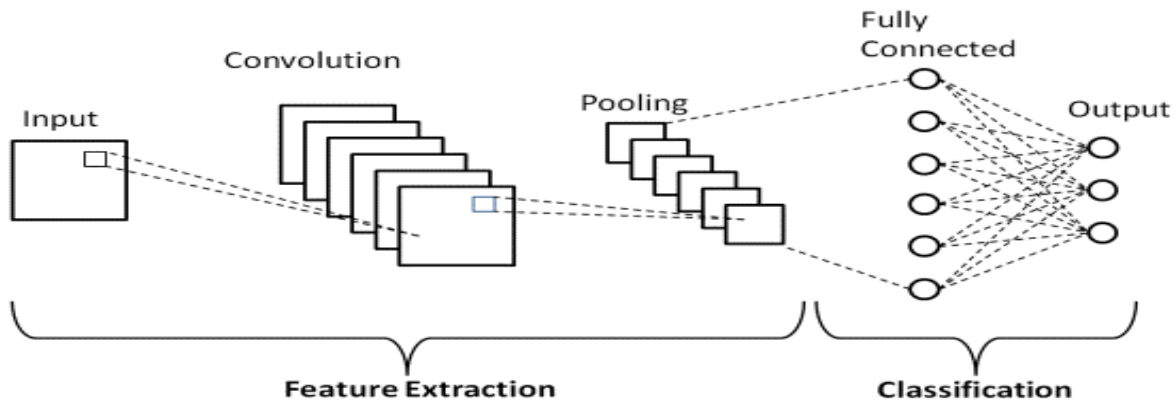
State 1-> state (5,3)

State 0-> state 4

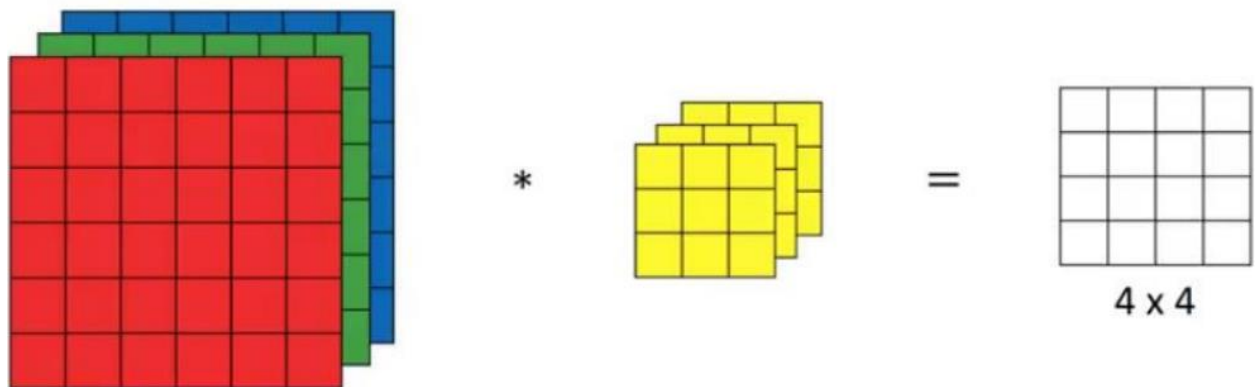


Convolutional Neural Network(CNN):

1. Convolutional Neural Networks (ConvNets or CNNs) are a category of Neural Networks .
2. Convolutional Neural Networks are a form of Feedforward Neural Networks.
3. It is mainly used in Computer Vision Applications. Such as Image recognition and Classification etc.
4. This type of architecture is dominant to recognize objects from a picture or video.
5. ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars.
6. Convolutional neural networks are very good at picking up on patterns in the input image, such as lines, gradients, circles, or even eyes and faces. In the consists of 3 Layers. Those are
 1. Convolution Layer
 2. Pooling Layer
 3. Fully Connected Layer or Dense Layer



- CNN takes an image as input, which is classified and process under a certain category such as dog, cat, lion, tiger, etc.
- The computer sees an image as an array of pixels and depends on the resolution of the image.
- Based on image resolution, it will see as $h * w * d$, where h = height w = width and d = dimension.
- For example, An RGB image is $6 * 6 * 3$ array of the matrix, and the grayscale image is $4 * 4 * 1$ array of the matrix. 7 In CNN, each input image will pass through a sequence of convolution layers along with pooling, fully connected layers, filters (Also known as kernels).
- After that, we will apply the Activation function to classify an object with probabilistic values.



LAYERS IN CNN :-

1. Convolution Layer:

- The main purpose of Convolution Layer is Feature Extraction. Convolution layer is the first layer to extract features from an input image.
- It means the network will learn specific patterns within the picture and will be able to recognize it everywhere in the picture.

- It uses the mathematical operation which takes two inputs such as image matrix and a kernel or filter.
- The dimension of the image matrix is $h \times w \times d$. • The dimension of the filter is $f_h \times f_w \times d$.

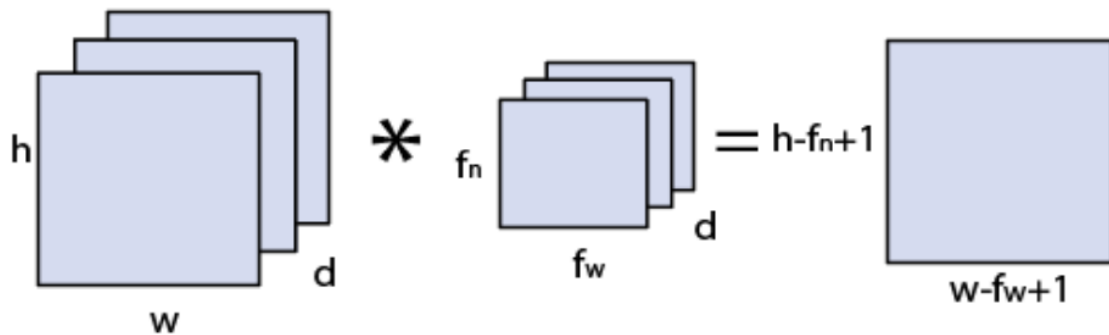


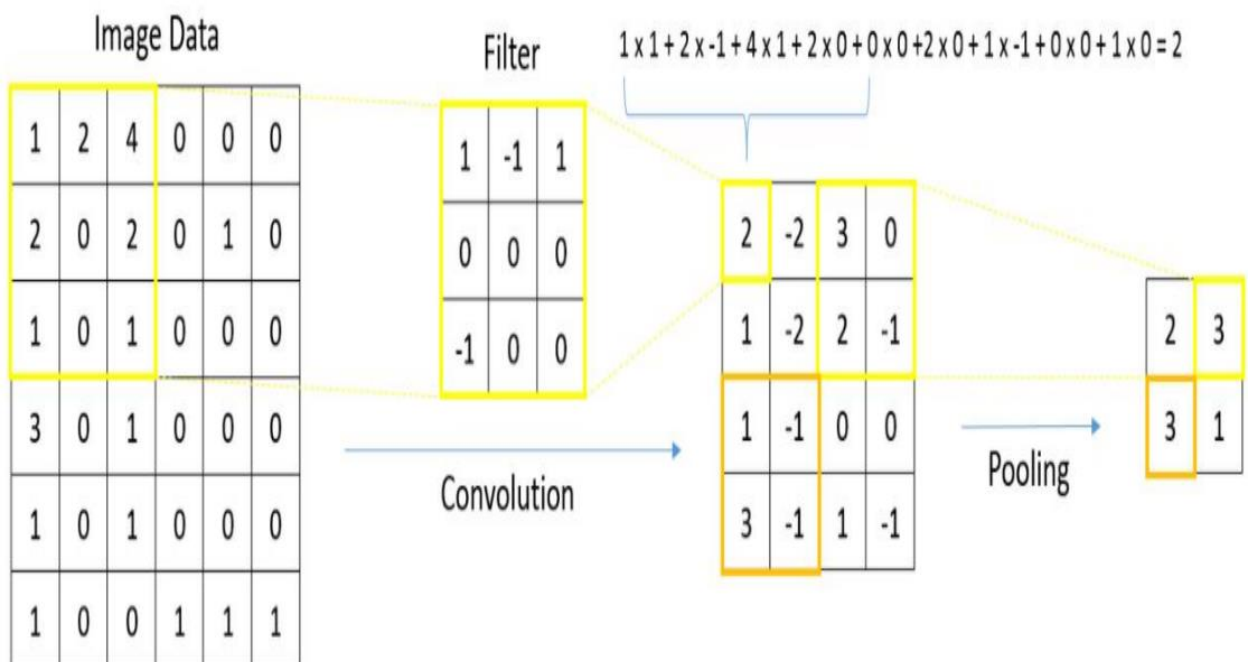
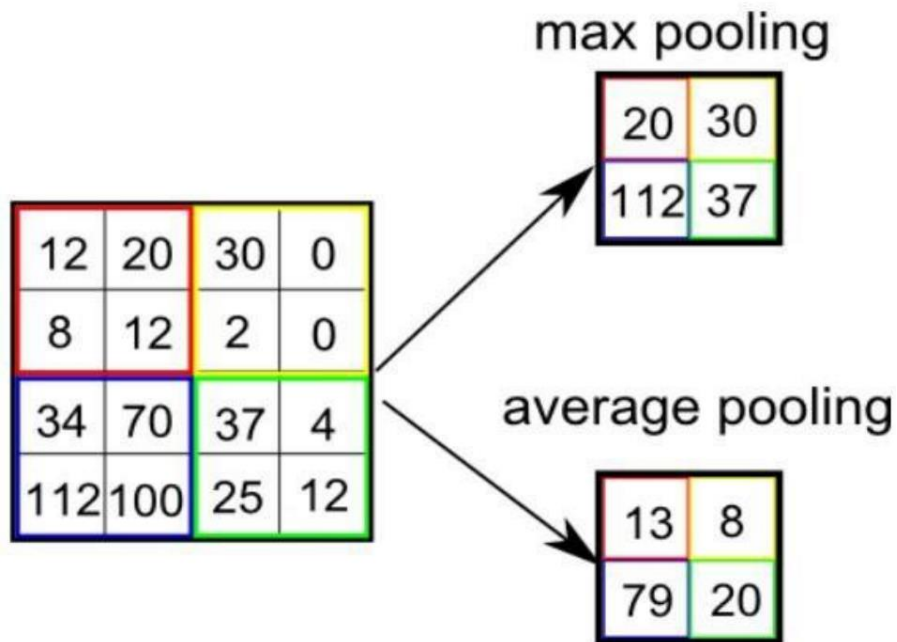
Image matrix multiplies kernel or filter matrix

- Here, Convolution is an element-wise multiplication. First, The computer will scan a part of the image, usually with a dimension of 3×3 and multiplies it to a filter.
 - The output of the element-wise multiplication is called a feature map. This step is repeated until all the image is scanned.
 - Note that, after the convolution, the size of the image is reduced.
- Dimension of the feature map as a function of the input image size(W), feature detector size(F), Stride(S) and Zero Padding on image(P) is**

$$(W - F + 2P) / S + 1$$

2.Pooling Layer:

- Similar to the Convolution Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. Here it is mainly used in Dimensionality Reduction.
- 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.
- On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

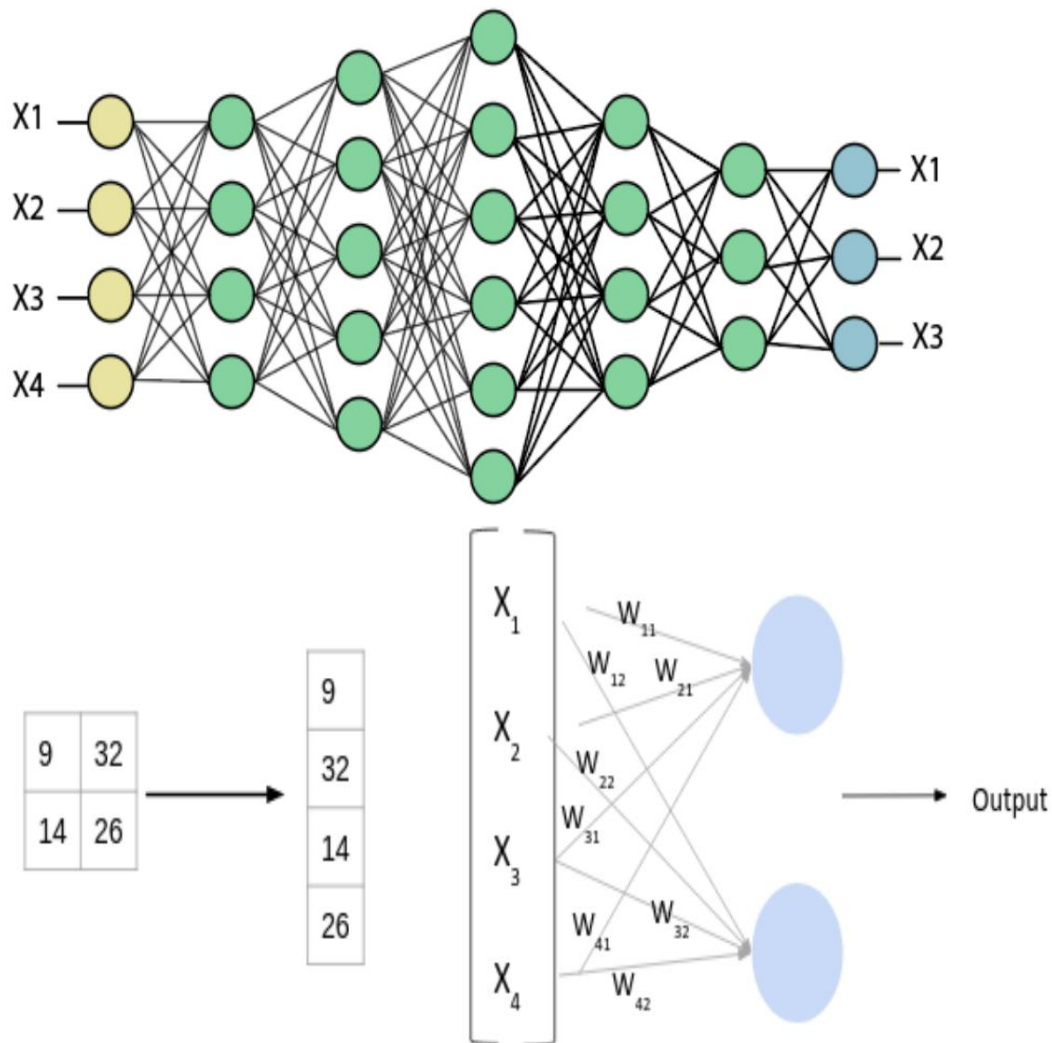


3. Fully Connected Layer:

- The last layer in CNN is the fully connected layer.
- Fully connected means that every output that's produced at the end of the last pooling layer is an input to each node in this fully connected layer.
- The role of the fully connected layer is to produce a list of class scores and perform classification based on image features that

have been extracted by the previous convolutional and pooling layers. So, the last fully connected layer will have as many nodes as there are classes.

Fully Connected Layer



Recurrent Neural Network (RNN):

Main Theme of RNN:

- Main idea: use hidden state to **capture information about the past**

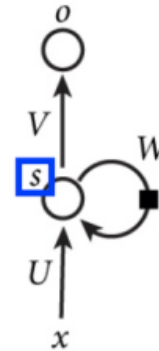
Feedforward Network

Each layer receives input from the previous layer with no loops



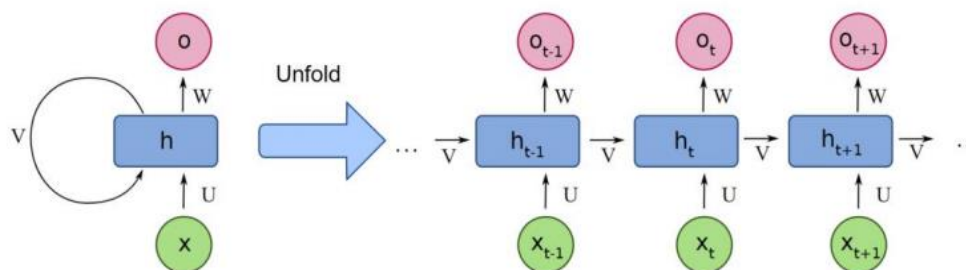
Recurrent Network

Each layer receives input from the previous layer **and the output from the previous time step**



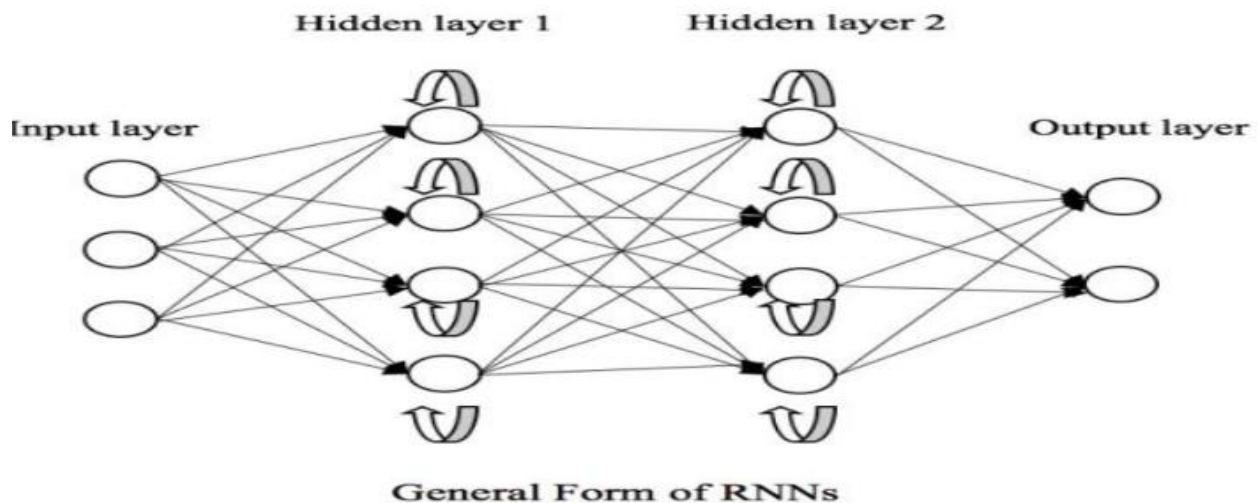
Recurrent Neural Networks (RNNs) are a kind of neural network that is widely utilised in Natural Language Processing. They are a kind of neural network that can utilise past outputs as inputs while having hidden states. RNN features a "memory" notion that remembers all information about what has been computed up to time step t . RNNs are referred to as recurrent because they do the same job for each element in a sequence, with the result dependent on past computations.

Prior to the introduction of attention models, RNNs were the typical recommendation for dealing with sequential data. A deep feedforward model may need specific parameters for each member of the sequence. It may also be limited in its ability to generalise to variable-length sequences. Recurrent Neural Networks apply the same weights for every element of the sequence, reducing the number of parameters and enabling the model to generalise to different length sequences. Because of their architecture, RNNs generalise to structured data other than sequential data, such as geographical or pictorial data.



Architecture of RNN:

Input: $x(t)$ is taken as the input to the network at time step t . For example, x_1 could be a one-hot vector corresponding to a word of a sentence. **Hidden state:** $h(t)$ represents a hidden state at time t and acts as “memory” of the network. $h(t)$ is calculated based on the current input and the previous time step's hidden state: $h(t) = f(Ux(t) + Wh(t-1))$. The function f is taken to be a non-linear transformation such as tanh, ReLU. **Weights:** The RNN has input to hidden connections parameterized by a weight matrix U , hidden-to-hidden recurrent connections parameterized by a weight matrix W , and hidden-to-output connections parameterized by a weight matrix V and all these weights (U, V, W) are shared across time. **Output:** $o(t)$ illustrates the output of the network. In the figure I just put an arrow after $o(t)$ which is also often subjected to non-linearity, especially when the network contains further layers downstream.



Forward Pass

The figure does not specify the choice of activation function for the hidden units. Before we proceed we make few assumptions: 1) we assume the hyperbolic tangent activation function for hidden layer. 2) We assume that the output is discrete, as if the RNN is used to predict words or characters. A natural way to represent discrete variables is to regard the output \mathbf{o} as giving the un-normalized log probabilities of each possible value of the discrete variable. We can then apply the softmax operation as a post-processing step to obtain a vector $\hat{\mathbf{y}}$ of normalized probabilities over the output.

The RNN forward pass can thus be represented by below set of equations.

$$\begin{aligned}\mathbf{a}^{(t)} &= \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \\ \mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}) \\ \mathbf{o}^{(t)} &= \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \\ \hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)})\end{aligned}$$

This is an example of a recurrent network that maps an input sequence to an output sequence of the same length. The total loss for a given sequence of \mathbf{x} values paired with a sequence of \mathbf{y} values would then be just the sum of the losses over all the time steps. We assume that the outputs $\mathbf{o}(t)$ are used as the argument to the softmax function to obtain the vector $\hat{\mathbf{y}}$ of probabilities over the output. We also assume that the loss \mathbf{L} is the negative log-likelihood of the true target $\mathbf{y}(t)$ given the input so far.

Backward Pass

The gradient computation involves performing a forward propagation pass moving left to right through the graph shown above followed by a backward propagation pass moving right to left through the graph. The runtime is $O(\tau)$ and cannot be reduced by parallelization because the forward propagation graph is inherently sequential; each time step may be computed only after the previous one. States computed in the forward pass must be stored until they are reused during the backward pass, so the memory cost is also $O(\tau)$. The back-propagation algorithm applied to the unrolled graph with $O(\tau)$ cost is called back-propagation through time (BPTT). Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps.

Working of RNN:

Formula for calculating current state:

$$h_t = f(h_{t-1}, x_t)$$

Formula for applying Activation function(tanh):

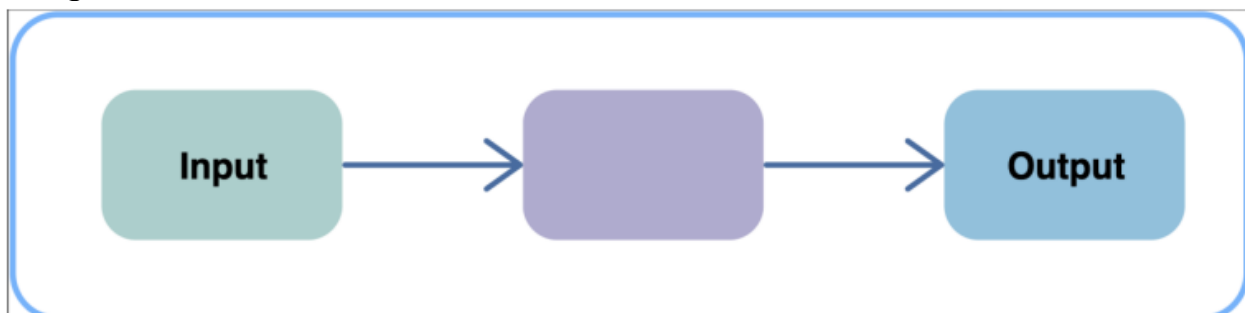
$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

Formula for calculating output:

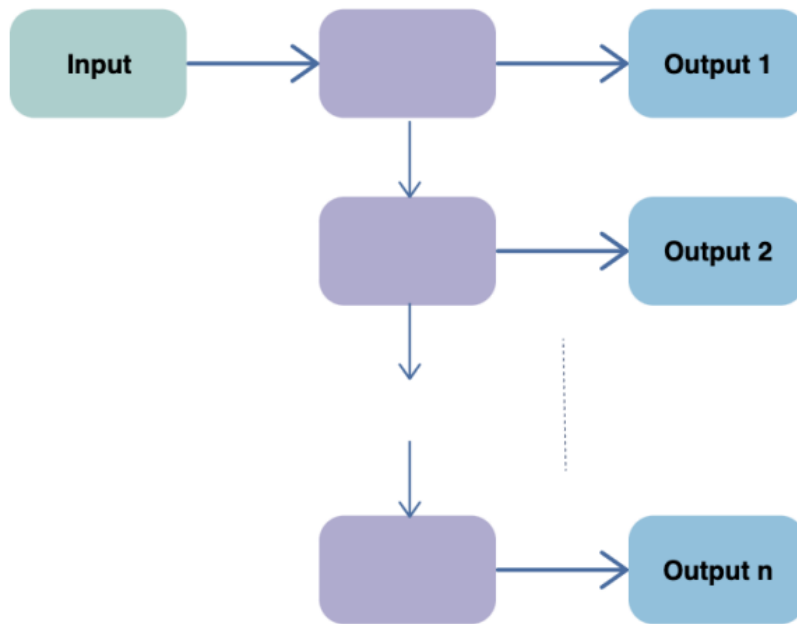
$$y_t = W_{hy}h_t$$

Types of Architectures:

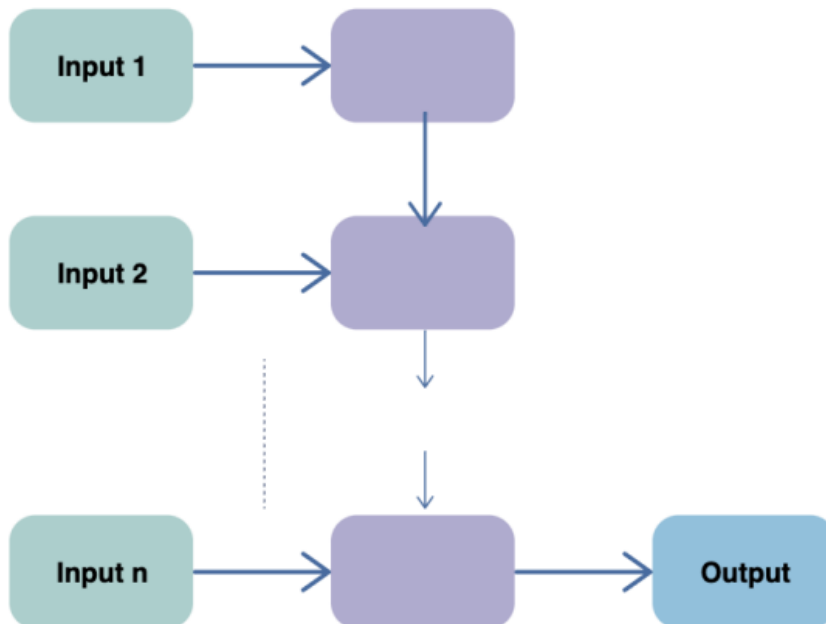
1.One-to-One: The simplest type of RNN is One-to-One, which allows single input and a single output. It has fixed input and output sizes and acts as a traditional neural network. The One-to-One application can be found in ImageClassification.



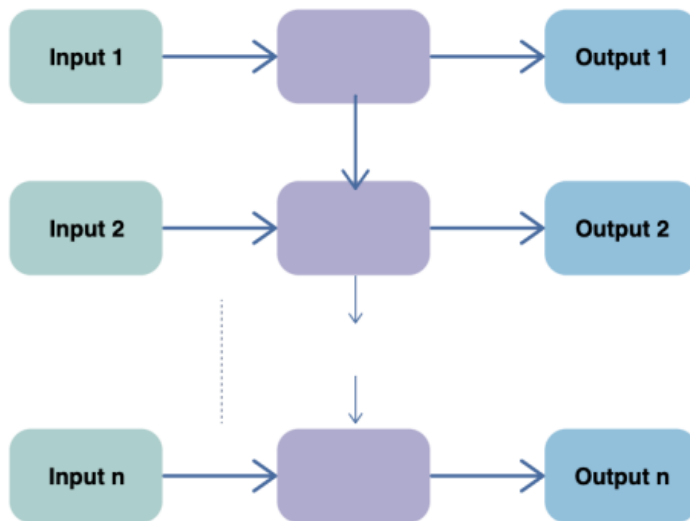
2.One-to-Many: One-to-Many is a type of RNN that gives multiple outputs when given a single input. It takes a fixed input size and gives a sequence of data outputs. Its applications can be found in Music Generation and Image Captioning



3.Many-to-One: Many-to-One is used when a single output is required from multiple input units or a sequence of them. It takes a sequence of inputs to display a fixed output. Sentiment Analysis is a common example of this type of Recurrent Neural Network.



4.Many-to-Many: Many-to-Many is used to generate a sequence of output data from a sequence of input units. Equal Unit Size: In this case, the number of both the input and output units is the same. A common application can be found in Name-Entity Recognition.

**Advantages:**

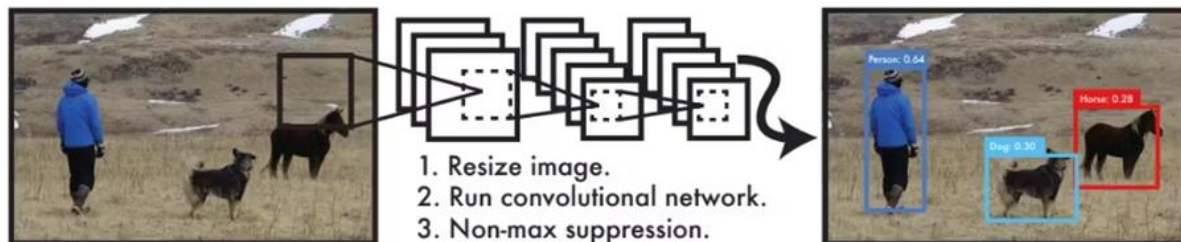
1. An RNN remembers every piece of information throughout time. It is only effective in time series prediction because of the ability to recall past inputs. This is referred to as Long Short Term Memory.
2. Possibility of processing input of any length.
3. Model size not increasing with size of input.
4. Weights are shared across time.
5. Recurrent neural networks are also employed in conjunction with convolutional layers to increase the effective pixel neighbourhood.

Limitations:

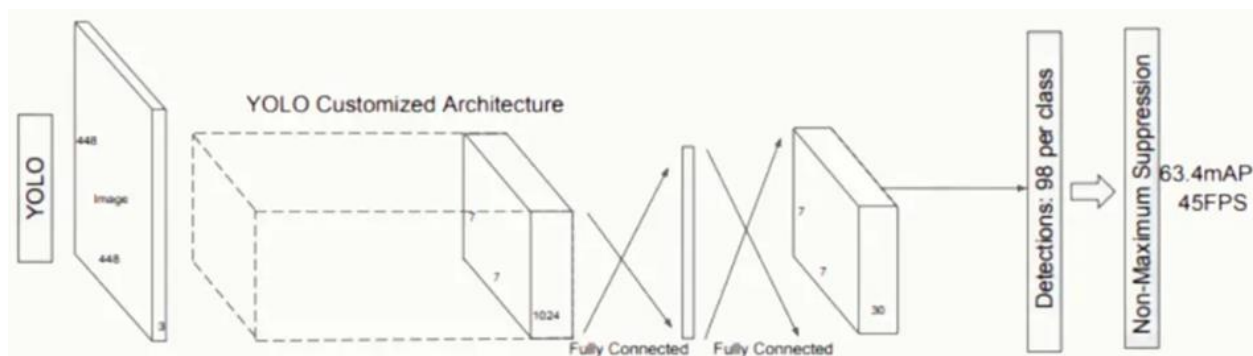
1. The computation time is slow as it is recurrent.
2. Issues with vanishing gradient and exploding.
3. It is quite complicated to train an RNN.
4. When utilising tanh or relu as an activation function, it cannot analyse extremely lengthy sequences.

YOLO Algorithm(You Only Look Once):

It is a family of algorithms in deep learning that deals with object detection by utilising both supervised and unsupervised learning datasets with the help of CNN as the backbone.



You Only Look Once: Unified, Real-Time Object Detection



What are the algorithms that were previously used before YOLO?

A) Sliding window, RCNN, fast RCNN, faster RCNN, etc.,.

The speciality of YOLO is to detect objects in a single pass in real time, whereas other object detection algorithms require multiple passes.

Note: what YOLO does in single pass is:

- Proposing object potential and bounding boxes for objects.
- Classifying the boxes and refining their locations.

YOLO working:

There are several steps in YOLO, in order to get a better output. They are:

1) Image preprocessing:

Here the input image is resized to a size which is suitable and easy for YOLO to process.

2) Feature extraction:

YOLO uses CNN to extract the features from the input image.

3) Bounding box, prediction:

Here YOLO uses CNN to form bounding boxes around an object for location and size and find confidence scores to prove the presence of the object in the bounding.

4) Class prediction:

Here also we use CNN to know the class of objects present in the bounding box.

5) This step is mostly used depending on the image contents such as:

Note: Above mentioned steps possible with one object in the image.

a) What if the input image has multiple objects?

A) Here we use the grid to find multiple objects by getting the vectors of every cell that has a feature. So that we can form multiple bounding boxes.

Note: Representation of a vector (It is a 1D matrix) of a cell.

$$\begin{bmatrix} P_c \\ C_x \\ C_y \\ B_w \\ B_h \\ C_1 \\ C_2 \\ \vdots \\ C_n \end{bmatrix}$$

where

P_c (probability) = 1 or 0 for present or absent of an object respectively.

C_x and C_y these are the coordinates of the centre of the object bounded within a box.

B_w and B_h these are the width and height of the objects bounding box.


C_1, C_2, \dots, C_n are various classes in the classification with '1' for the class to which the input image belongs to and '0' for the remaining class.

Note: The vector only represents one cell of the image which is capable of storing only one feature in it.

b) What if there are multiple bounding boxes around a single object?

A) Then we go with concept called NMS(non maxima suppression) in which we use a

technique called IOU(Intersection over union) i.e.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Note: IA= intersection area and UA= union area

if IA=UA IOU=1(100% overlapping)

if IA=0 IOU=0(0% overlapping)

if the IOU > 0.5 then we remove the anchor boxes with less confidence score.

c) what if there are multiple bounding boxes inside an another bounding box of an object?(which indirectly say multiple small objects close to a single large object which makes their centre close and sometimes same centre)

A) In such cases we try to combine the vectors of multiple anchor objects into

Pc1
Cx1
Cy1
Bw1
Bh1
C1
C2
...
Cn
Pc2
Cx2
Cy2
Bw2
Bh2
C1
C2
...
Cn
...

one single vector. It looks similar to the below mentioned one.

6) final object detection(S):

S=object classification probability x C [object classification probability is given by CNN]

where

C = confidence score(object presence in the bounding)

C = probability of object presence x IOU

Why do we get multiple bounding boxes over a single object?

A) Because YoLO assumes that one object has multiple ones due to training in various ways and this is due to the ground truth box.

What is a ground truth box?

A) It is a user defined object bounding box which is used during training to compare the output of YoLO with human desired output.

What is an anchor box?

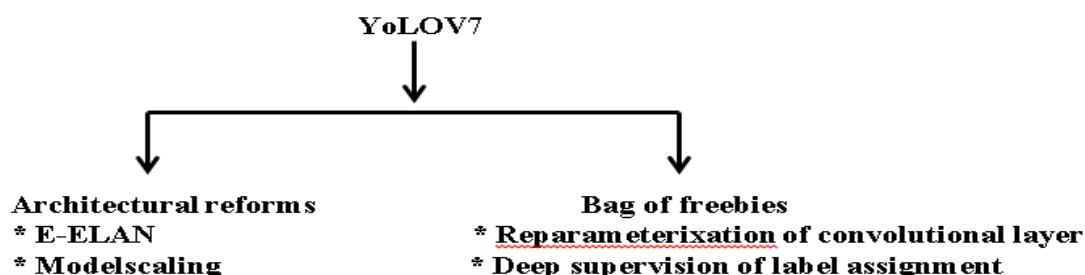
A) It is a predefined bounding box with various aspect ratios which finally will be a final bounding box by removing other anchor boxes because we won't be getting an anchor box in the final output.

What are the various versions of YOLO?

A) There are various series of YOLO and some of the most famous one's are YOLO series from V1 to V9 and YOLOX. Among them YoLOV7 is the most popular and latest one. All the versions of YOLO have their unique capability and are used based on the user requirement.

Why is YoLOV7 better?

A) It is famous for its architectural reforms and optimised training process which we call a trailerable bag of freebies. This version also addresses the memory and gradient propagation challenges.



E-ELAN:

Extended efficient layer aggression network. This develops more robust feature maps

Model

scaling:

This helps to develop the variants of V7 model by works on various scales.

Ex: using the stack and compound scaling YoLOV7tiny produces YoLOV7 and YoLOV7X. Using only compound scaling YoLOV7W6 produce YoLOV7E6 and YoLOV7D6

Applications

in

computer

vision:

1. Autonomous Vehicles:

- **Real-time Object Detection:** YOLO's ability to detect and localise objects in real-time is crucial for autonomous vehicles. It allows them to identify pedestrians, vehicles, traffic signs, and other obstacles on the road, enabling safe navigation and collision avoidance.
- **Traffic Monitoring and Management:** YOLO can be used in traffic management systems to monitor traffic flow, detect congestion, and identify potential accidents.

2. Surveillance Systems:

- **Object Detection and Tracking:** YOLO can be employed in surveillance systems to detect people, objects of interest, and suspicious activities. It can also track the movement of objects over time for security purposes.
- **Anomaly Detection:** YOLO can be used to identify unusual events or objects deviating from the norm in a scene, potentially indicating suspicious activity.

3. Robotics and Automation:

- **Object Recognition and Manipulation:** Robots can leverage YOLO to recognize objects in their environment, enabling them to grasp, manipulate, and interact with objects safely and efficiently.
- **Inventory Management:** YOLO can be used in warehouses and logistics to automate inventory management by detecting and tracking objects within storage areas.

4. Augmented Reality (AR) and Virtual Reality (VR):

- **Real-time Object Interaction:** YOLO can be used in AR applications to enable real-time interaction with virtual objects overlaid on the real world. For instance, it can detect physical objects and allow users to virtually interact with them.

- **Object Recognition and Augmentation:** VR applications can utilize YOLO to recognize objects in the real world and integrate them into the virtual environment, creating a more immersive and interactive experience.

5. Public Safety and Security:

- **Crowd Monitoring:** YOLO can be used in crowd control situations to monitor crowd density and identify potential safety hazards.
- **Facial Recognition:** YOLO can be integrated with facial recognition systems to detect and identify individuals in real-time, aiding in security applications.

6. Video Analysis and Content Understanding:

- **Action Recognition:** YOLO can be used to analyse videos and recognize the actions taking place within them. This has applications in sports analysis, video surveillance, and activity monitoring.
- **Automatic Video Summarization:** YOLO can be used to identify key objects and events in videos, enabling the creation of automatic video summaries.

Conclusion:

We can also combine these algorithms to solve various types of complex challenges. Given these developments and advancements in deep learning can be used to produce accurate results and can provide futuristic solutions.

Hand Gesture Recognition

Models

Palm Detection Model

To detect initial hand locations, we designed a [single-shot detector](#) model optimized for mobile real-time uses in a manner similar to the face detection model in [MediaPipe Face Mesh](#). Detecting hands is a decidedly complex task: our [lite model](#) and [full model](#) have to work across a variety of hand sizes with a large scale span (~20x) relative to the image frame and be able to detect occluded and self-occluded hands. Whereas faces have high contrast patterns, e.g., in the eye and mouth region, the lack of such features in hands makes it comparatively difficult to detect them reliably from their visual features alone. Instead, providing additional context, like arm, body, or person features, aids accurate hand localization.

Our method addresses the above challenges using different strategies. First, we train a palm detector instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modelled using square bounding boxes (anchors in ML terminology) ignoring other aspect ratios, and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects (similar to the RetinaNet approach). Lastly, we minimize the focal loss during training to support a large amount of anchors resulting from the high scale variance.

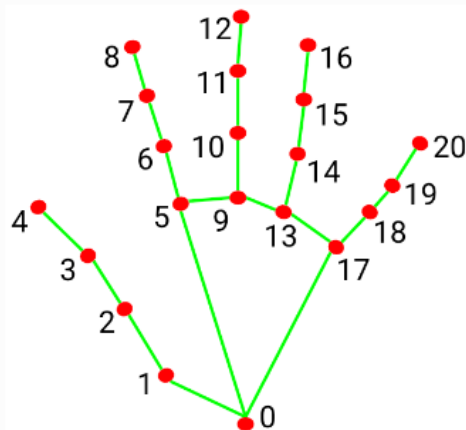
With the above techniques, we achieve an average precision of 95.7% in palm detection. Using a regular cross entropy loss and no decoder gives a baseline of just 86.22%.

Hand Landmark Model

After the palm detection over the whole image our subsequent hand landmark [model](#) performs precise keypoint localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

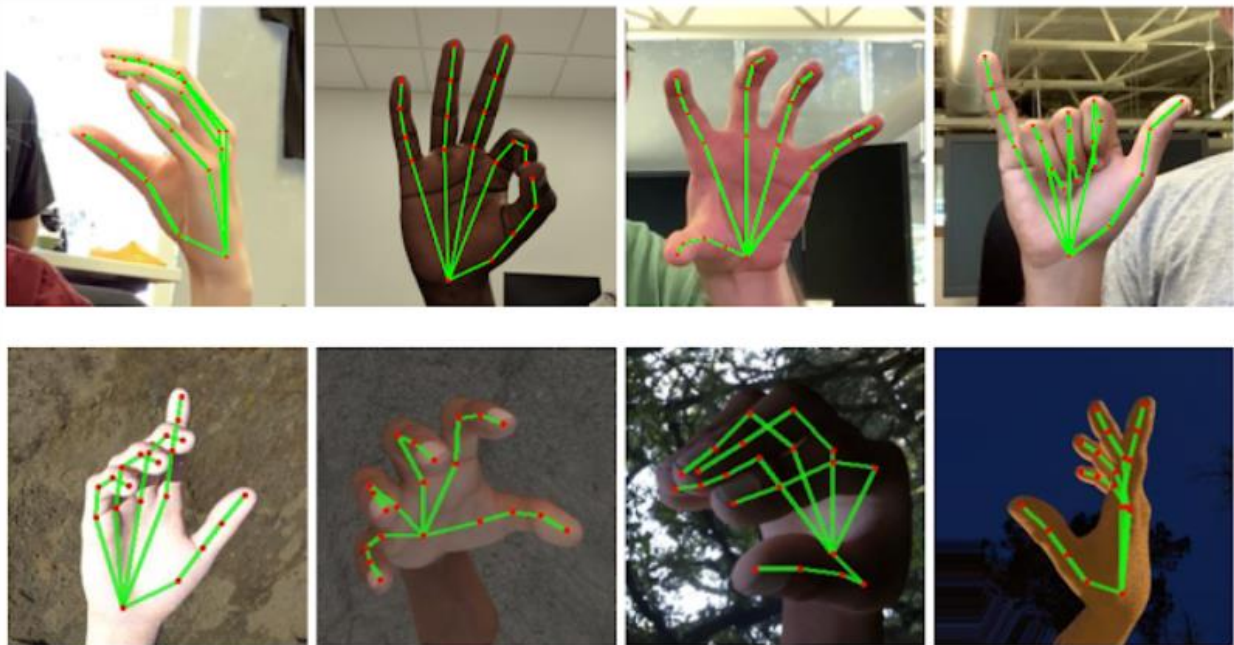
To obtain ground truth data, we have manually annotated ~30K real-world images with 21 3D coordinates, as shown below (we take Z-value from image depth map, if it exists per corresponding coordinate). To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, we also render a high-quality synthetic hand model over various backgrounds and

map it to the corresponding 3D coordinates.



- | | |
|-----------------------|-----------------------|
| 0. WRIST | 11. MIDDLE_FINGER_DIP |
| 1. THUMB_CMC | 12. MIDDLE_FINGER_TIP |
| 2. THUMB_MCP | 13. RING_FINGER_MCP |
| 3. THUMB_IP | 14. RING_FINGER_PIP |
| 4. THUMB_TIP | 15. RING_FINGER_DIP |
| 5. INDEX_FINGER_MCP | 16. RING_FINGER_TIP |
| 6. INDEX_FINGER_PIP | 17. PINKY_MCP |
| 7. INDEX_FINGER_DIP | 18. PINKY_PIP |
| 8. INDEX_FINGER_TIP | 19. PINKY_DIP |
| 9. MIDDLE_FINGER_MCP | 20. PINKY_TIP |
| 10. MIDDLE_FINGER_PIP | |

Hand_Landmarks.png



Solution APIs

Configuration Options:

Naming style and availability may differ slightly across platforms/languages.

Static_image_mode:

If set to **false**, the solution treats the input images as a video stream. It will try to detect hands in the first input images, and upon a successful detection further localises the hand landmarks. In subsequent images, once all [max_num_hands](#) hands are detected and the corresponding hand landmarks are localised, it

simply tracks those landmarks without invoking another detection until it loses track of any of the hands. This reduces latency and is ideal for processing video frames. If set to `true`, hand detection runs on every input image, ideal for processing a batch of static, possibly unrelated, images. Default to `false`.

Max_num_hands:

Maximum number of hands to detect. Default to `2`.

Model_complexity:

Complexity of the hand landmark model: `0` or `1`. Landmark accuracy as well as inference latency generally go up with the model complexity. Default to `1`.

Min_detection_confidence:

Minimum confidence value (`[0.0, 1.0]`) from the hand detection model for the detection to be considered successful. Default to `0.5`.

Min_tracking_confidence:

Minimum confidence value (`[0.0, 1.0]`) from the landmark-tracking model for the hand landmarks to be considered tracked successfully, or otherwise hand detection will be invoked automatically on the next input image. Setting it to a higher value can increase robustness of the solution, at the expense of a higher latency. Ignored if [static_image_mode](#) is `true`, where hand detection simply runs on every image. Default to `0.5`.

Output:

Naming style may differ slightly across platforms/languages.

Multi_hand_landmarks:

Collection of detected/tracked hands, where each hand is represented as a list of 21 hand landmarks and each landmark is composed of `x`, `y` and `z`. `x` and `y` are normalised to `[0.0, 1.0]` by the image width and height respectively. `z` represents the landmark depth with the depth at the wrist being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of `z` uses roughly the same scale as `x`.

Multi_hand_world_landmarks:

Collection of detected/tracked hands, where each hand is represented as a list of 21 hand landmarks in world coordinates. Each landmark is composed of `x`, `y` and `z`: real-world 3D coordinates in metres with the origin at the hand's approximate geometric center.

Multi_handedness:

Collection of handedness of the detected/tracked hands (i.e. is it a left or right hand). Each hand is composed of a `label` and `score`. `label` is a string of values

either "Left" or "Right". `score` is the estimated probability of the predicted handedness and is always greater than or equal to 0.5 (and the opposite handedness has an estimated probability of `1 - score`).

Note that handedness is determined assuming the input image is mirrored, i.e., taken with a front-facing/selfie camera with images flipped horizontally. If it is not the case, please swap the handedness output in the application.

Python Solution API

Please first follow general [instructions](#) to install the MediaPipe Python package, then learn more in the companion [Python Colab](#) and the usage example below. Supported configuration options:

- [static image mode](#)
- [max num hands](#)
- [model complexity](#)
- [min detection confidence](#)
- [min tracking confidence](#)

```
import cv2
import mediapipe as mp
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
# For static images:
IMAGE_FILES = []
with mp_hands.Hands(
    static_image_mode=True,
    max_num_hands=2,
    min_detection_confidence=0.5) as hands:
    for idx, file in enumerate(IMAGE_FILES):
        # Read an image, flip it around y-axis for correct handedness output (see
        # above).
        image = cv2.flip(cv2.imread(file), 1)
        # Convert the BGR image to RGB before processing.
        results = hands.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        # Print handedness and draw hand landmarks on the image.
        print('Handedness:', results.multi_handedness)
        if not results.multi_hand_landmarks:
            continue
        image_height, image_width, _ = image.shape
        annotated_image = image.copy()
        for hand_landmarks in results.multi_hand_landmarks:
            print('hand_landmarks:', hand_landmarks)
            print(
                f'Index finger tip coordinates: (',
                f'{hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].
x * image_width}, ',
                f'{hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].
y * image_height})')
            mp_drawing.draw_landmarks(
                annotated_image,
```

```

        hand_landmarks,
        mp_hands.HAND_CONNECTIONS,
        mp_drawing_styles.get_default_hand_landmarks_style(),
        mp_drawing_styles.get_default_hand_connections_style())
cv2.imwrite(
    '/tmp/annotated_image' + str(idx) + '.png', cv2.flip(annotated_image,
1))
# Draw hand world landmarks.
if not results.multi_hand_world_landmarks:
    continue
for hand_world_landmarks in results.multi_hand_world_landmarks:
    mp_drawing.plot_landmarks(
        hand_world_landmarks, mp_hands.HAND_CONNECTIONS, azimuth=5)
# For webcam input:
cap = cv2.VideoCapture(0)
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:
    while cap.isOpened():
        success, image = cap.read()
        if not success:
            print("Ignoring empty camera frame.")
            # If loading a video, use 'break' instead of 'continue'.
            continue
        # To improve performance, optionally mark the image as not writeable to
        # pass by reference.
        image.flags.writeable = False
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        results = hands.process(image)
        # Draw the hand annotations on the image.
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                mp_drawing.draw_landmarks(
                    image,
                    hand_landmarks,
                    mp_hands.HAND_CONNECTIONS,
                    mp_drawing_styles.get_default_hand_landmarks_style(),
                    mp_drawing_styles.get_default_hand_connections_style())
        # Flip the image horizontally for a selfie-view display.
        cv2.imshow('MediaPipe Hands', cv2.flip(image, 1))
        if cv2.waitKey(5) & 0xFF == 27:
            break
cap.release()

```

Keypoint_Classification.ipynb:

```
import csv

import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

RANDOM_SEED = 42
```

#Specify each path

```
dataset = 'model/keypoint_classifier/keypoint.csv'
model_save_path = 'model/keypoint_classifier/keypoint_classifier.hdf5'
```

#Setting the number of categories

```
NUM_CLASSES = 5
```

#loading the Training data

```
x_dataset = np.loadtxt(dataset, delimiter=',', dtype='float32', usecols=list(range(1, (21 * 2) + 1)))

y_dataset = np.loadtxt(dataset, delimiter=',', dtype='int32', usecols=(0))

x_train, x_test, y_train, y_test = train_test_split(x_dataset, y_dataset, train_size=0.75, random_state=RANDOM_SEED)
```

#Model building

```
Click to add a breakpoint models.Sequential([
    tf.keras.layers.Input((21 * 2, )),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])

model.summary() # tf.keras.utils.plot_model(model, show_shapes=True)

# Model checkpoint callback
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    model_save_path, verbose=1, save_weights_only=False)

# Early stopping callback
es_callback = tf.keras.callbacks.EarlyStopping(patience=20, verbose=1)
```

#compile the model

```
model.compile(  
    optimizer='adam',  
    loss='sparse_categorical_crossentropy',  
    metrics=['accuracy']  
)
```

#Model Training

```
model.fit(  
    X_train,  
    y_train,  
    epochs=1000,  
    batch_size=128,  
    validation_data=(X_test, y_test),  
    callbacks=[cp_callback, es_callback]  
)
```

#Model evaluation

```
val_loss, val_acc = model.evaluate(X_test, y_test, batch_size=128)
```

```
# Loading a saved model
```

```
model = tf.keras.models.load_model(model_save_path)
```

```
# Inference test
```

```
predict_result = model.predict(np.array([X_test[0]]))
```

```
print(np.squeeze(predict_result))
```

```
print(np.argmax(np.squeeze(predict_result)))
```

```

# Mix rows and columns

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

def print_confusion_matrix(y_true, y_pred, report=True):
    labels = sorted(list(set(y_true)))
    cmx_data = confusion_matrix(y_true, y_pred, labels=labels)

    df_cmx = pd.DataFrame(cmx_data, index=labels, columns=labels)

    fig, ax = plt.subplots(figsize=(7, 6))
    sns.heatmap(df_cmx, annot=True, fmt='g', square=False)
    ax.set_ylim(len(set(y_true)), 0)
    plt.show()

    if report:
        print('Classification Report')
        print(classification_report(y_test, y_pred))

Y_pred = model.predict(X_test)
y_pred = np.argmax(Y_pred, axis=1)

print_confusion_matrix(y_test, y_pred)

# Convert to a model for Tensorflow-Lite
# Save as inference-only model
model.save(model_save_path, include_optimizer=False)

# Transform the model (quantize it)
tflite_save_path = 'model/keypoint_classifier/keypoint_classifier.tflite'

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quantized_model = converter.convert()

open(tflite_save_path, 'wb').write(tflite_quantized_model)

# Inference test

interpreter = tf.lite.Interpreter(model_path=tflite_save_path)
interpreter.allocate_tensors()

# Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

```

```
interpreter.set_tensor(input_details[0]['index'], np.array([X_test[0]]))

%%time
# Inference execution
interpreter.invoke()
tflite_results = interpreter.get_tensor(output_details[0]['index'])

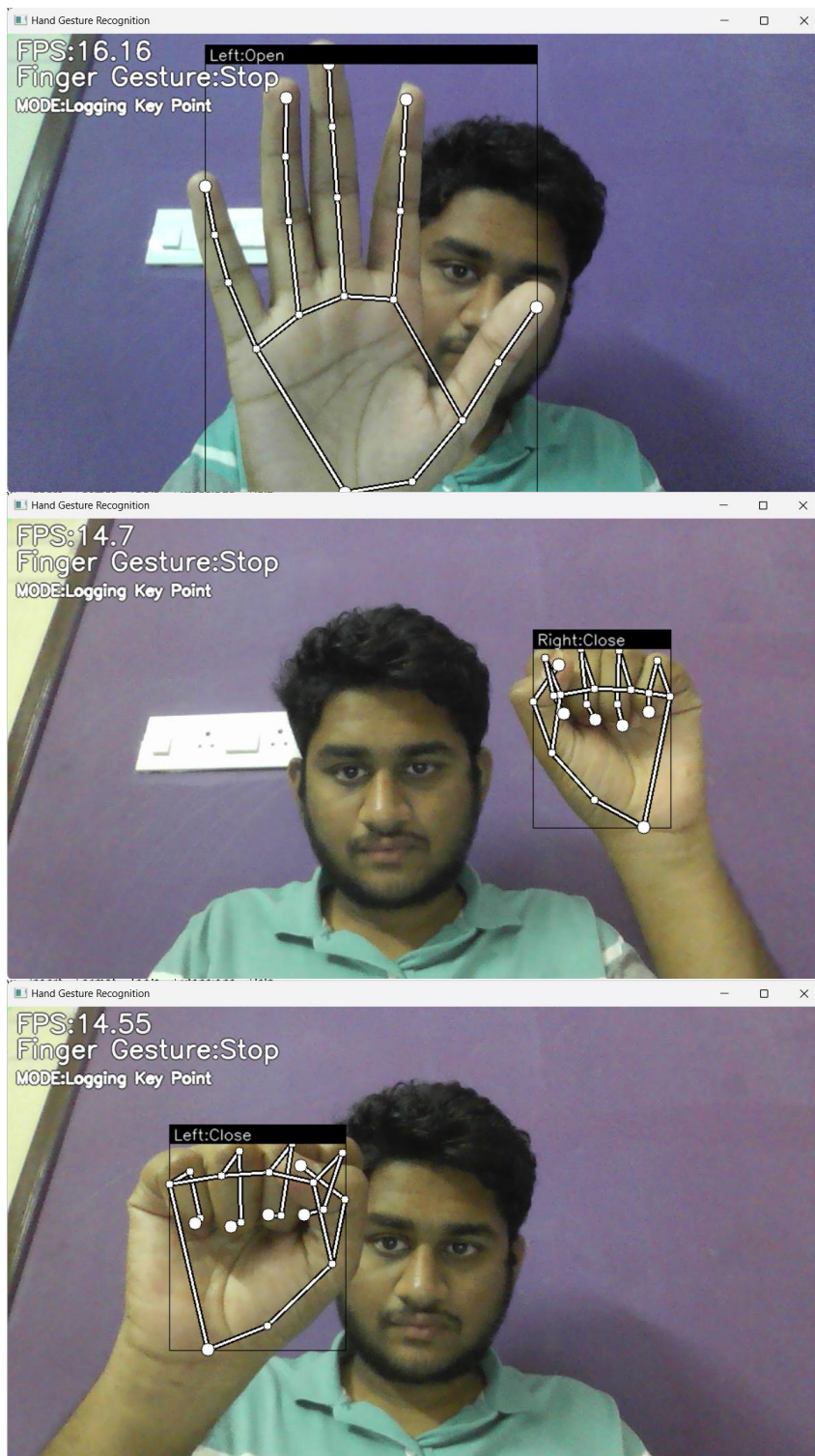
print(np.squeeze(tflite_results))
print(np.argmax(np.squeeze(tflite_results)))
```

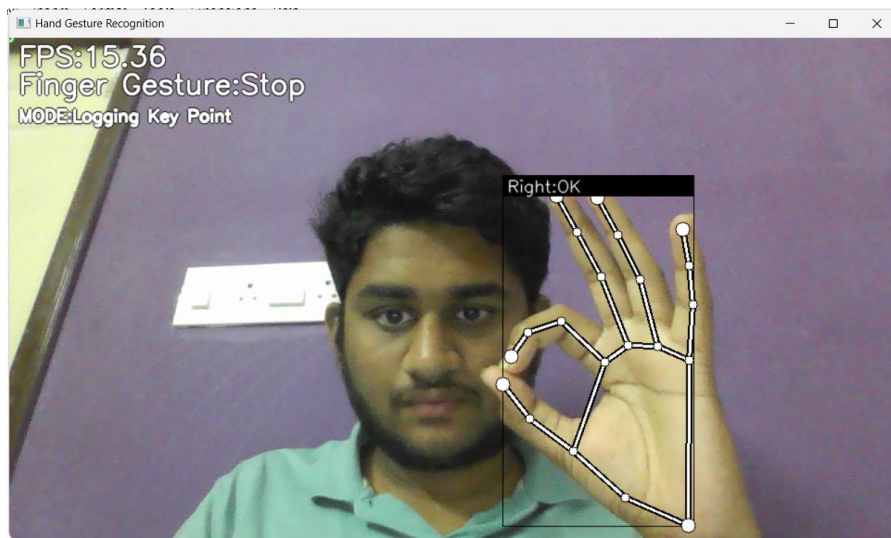
Keypoint_Classifier_label.csv:

- 1.Right open
- 2.Right close
- 3.Left open
- 4.Left close
- 5.Right pointer
- 6.Left pointer
- 7.Right ok

Output:-







FACIAL EXPRESSION RECOGNITION

REAL TIME FACIAL EMOTION RECOGNIZER

Real-time facial emotion recognition is a technology that uses computer vision and deep learning to analyze a person's facial expressions in real-time and determine their emotional state.

Overview

The Real-Time Facial Emotion Recognition System is a Python-based project that utilizes computer vision and deep learning techniques to perform real-time emotion detection from live video streams captured by a camera source. A CNN model is trained to recognize a range of 7 emotions, including 'Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', and 'Surprise'.

Workflow

1. It all starts with training a CNN model.
2. The dataset used to train and test the model is [DATASET] (<https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset> "DATASET").
3. You can download the dataset or directly use the dataset in [KAGGLE] (<https://kaggle.com>) if you want to perform any changes in `real-time-facial-emotion-classification-cnn-using-keras.ipynb` file.
4. The trained model generated will be in .h5 format. Here the emotion detection model is `model.h5` file.
5. This model attained 72% training accuracy and 60% validation accuracy.
! [3eb93191-ea63-4715-b6d2-dcefdb42d0b7] (<https://github.com/SHAIK-AFSANA/facialemotionrecognizerinrealtime/assets/146961917/5b9e98a5-3d37-42a8-8d47-5fc4c9060f63>)
6. The system employs the Haar Cascade Classifier, loaded from the `haarcascade_frontalface_default.xml` file, to detect faces in the video frames.
7. The code continuously captures video frames from the camera source, making real-time processing possible.
8. The detected faces are passed through the emotion detection model to classify the emotion, and the corresponding emotion label is overlaid on the video frame.
9. Here is the sample output. For all emotion outputs you can check out `OutputScreenshots` folder.
! [Screenshot (273)] (<https://github.com/SHAIK-AFSANA/facialemotionrecognizerinrealtime/assets/146961917/6b473e7a-a0b1-4875-afed-8f3dc24b54b9>)

Prerequisites

- * Python
- * TensorFlow
- * Keras
- * OpenCV
- * Pandas
- * Numpy
- * Seaborn
- * Matplotlib

Usage

1. You can start by cloning the project repository to your local system or by downloading the zip file and extracting it in your working folder.

2. Ensure you have the pre-trained emotion detection model (model.h5) and the Haar Cascade Classifier XML file (haarcascade_frontalface_default.xml) placed in the project directory. These models are essential for the system to function.
3. Execute the Python script `main.py` to start the real-time facial emotion recognition system.

Importing Libraries

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os

# Importing Deep Learning Libraries

from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Input, Dropout, GlobalAveragePooling2D, Flatten, Conv2D, BatchNormalization, Activation, MaxPooling2D
from keras.models import Model, Sequential
from keras.optimizers import Adam, SGD, RMSprop
```

Displaying Images

```
picture_size = 48
folder_path = "../input/face-expression-recognition-dataset/images/"
```

```

expression = 'disgust'

plt.figure(figsize= (12,12))
for i in range(1, 10, 1):
    plt.subplot(3,3,i)
    img = load_img(folder_path+"train/"+expression+"/"+
                    os.listdir(folder_path + "train/" + expression)[i], target_size=(picture_size, picture_size))
    plt.imshow(img)
plt.show()

```



Making Training and Validation Data

```
batch_size = 128

datagen_train = ImageDataGenerator()
datagen_val = ImageDataGenerator()

train_set = datagen_train.flow_from_directory(folder_path+"train",
                                              target_size = (picture_size,picture_size),
                                              color_mode = "grayscale",
                                              batch_size=batch_size,
                                              class_mode='categorical',
                                              shuffle=True)

test_set = datagen_val.flow_from_directory(folder_path+"validation",
                                           target_size = (picture_size,picture_size),
                                           color_mode = "grayscale",
                                           batch_size=batch_size,
                                           class_mode='categorical',
                                           shuffle=False)
```

Model Building

```
from keras.optimizers import Adam,SGD,RMSprop

no_of_classes = 7

model = Sequential()

#1st CNN layer
model.add(Conv2D(64,(3,3),padding = 'same',input_shape = (48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

#2nd CNN layer
model.add(Conv2D(128,(5,5),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#3rd CNN layer
model.add(Conv2D(512,(3,3),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#4th CNN layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
```



```

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

#Fully connected 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(no_of_classes, activation='softmax'))

opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

```

Fitting the Model with Training and Validation Data

```

from keras.optimizers import RMSprop,SGD,Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLR0nPlateau

checkpoint = ModelCheckpoint("./model.h5", monitor='val_acc', verbose=1, save_best_only=True, mode='max')

early_stopping = EarlyStopping(monitor='val_loss',
                                min_delta=0,
                                patience=3,
                                verbose=1,
                                restore_best_weights=True
                                )

reduce_learningrate = ReduceLR0nPlateau(monitor='val_loss',
                                          factor=0.2,
                                          patience=3,
                                          verbose=1,
                                          min_delta=0.0001)

callbacks_list = [early_stopping,checkpoint,reduce_learningrate]

epochs = 48

model.compile(loss='categorical_crossentropy',
              optimizer = Adam(lr=0.001),
              metrics=['accuracy'])

```

```

history = model.fit_generator(generator=train_set,
                              steps_per_epoch=train_set.n//train_set.batch_size,
                              epochs=epochs,
                              validation_data = test_set,
                              validation_steps = test_set.n//test_set.batch_size,
                              callbacks=callbacks_list
                              )

```

Plotting Accuracy & Loss

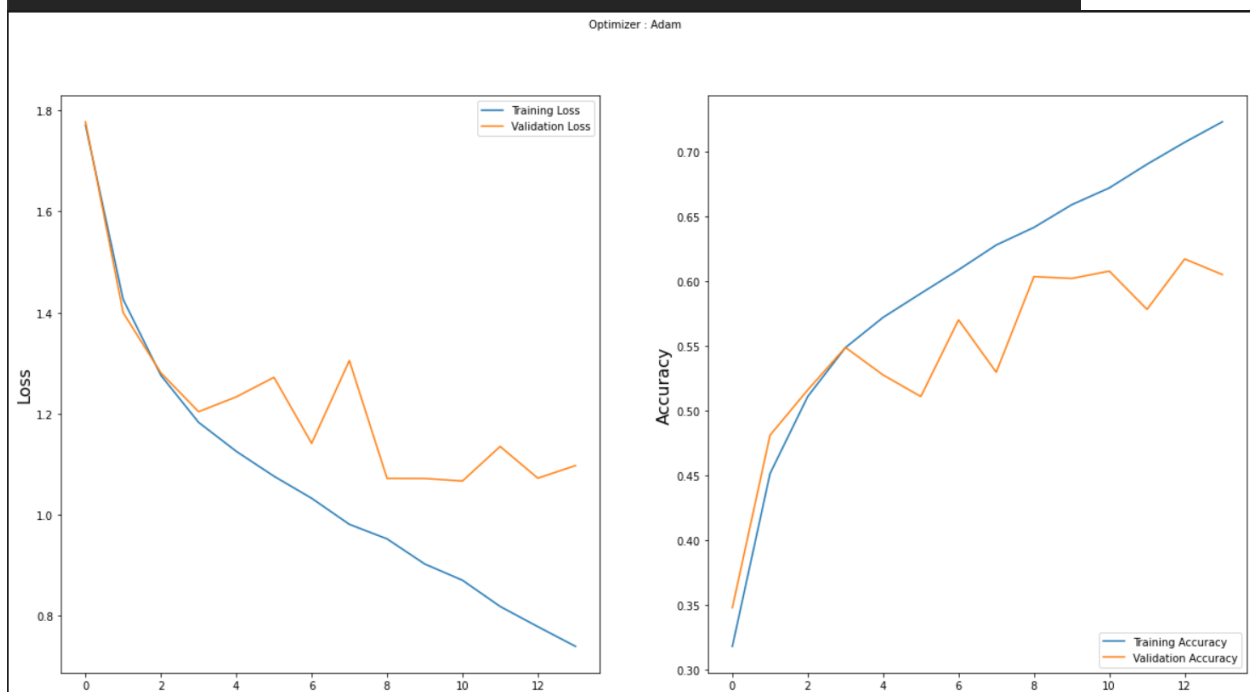
```

plt.style.use('dark_background')

plt.figure(figsize=(20,10))
plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.show()

```



main.py:

```
import tensorflow
from tensorflow import keras
from keras.models import load_model
from time import sleep
from keras.preprocessing.image import img_to_array
from keras.preprocessing import image
import os
import cv2
import numpy as np

face_classifier =
cv2.CascadeClassifier(r'C:\Users\Admin\OneDrive\Desktop\facialemotionrecog
nizerinrealtime-main\haarcascade_frontalface_default.xml')
classifier
=load_model(r'C:\Users\Admin\OneDrive\Desktop\facialemotionrecognizerinrea
ltime-main\model.h5')

emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad',
'Surprise']

cap = cv2.VideoCapture(0)

while True:
    _, frame = cap.read()
    labels = []
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,255), 2)
        roi_gray = gray[y:y+h,x:x+w]
        roi_gray =
cv2.resize(roi_gray, (48,48), interpolation=cv2.INTER_AREA)

        if np.sum([roi_gray])!=0:
            roi = roi_gray.astype('float')/255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi,axis=0)

            prediction = classifier.predict(roi)[0]
            label=emotion_labels[prediction.argmax()]
            label_position = (x,y)
            cv2.putText(frame,label,label_position,cv2.FONT_HERSHEY_SIMPLE
x,1, (0,255,0), 2)
        else:
            cv2.putText(frame, 'No
Faces', (30,80), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)
```

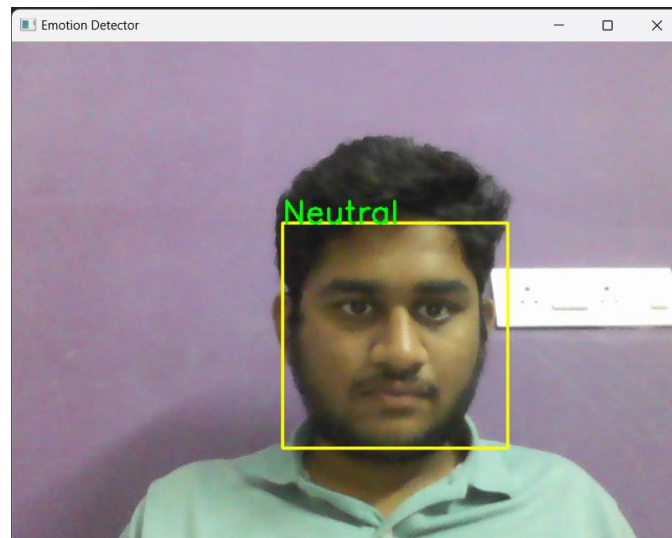
```
cv2.imshow('Emotion Detector',frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

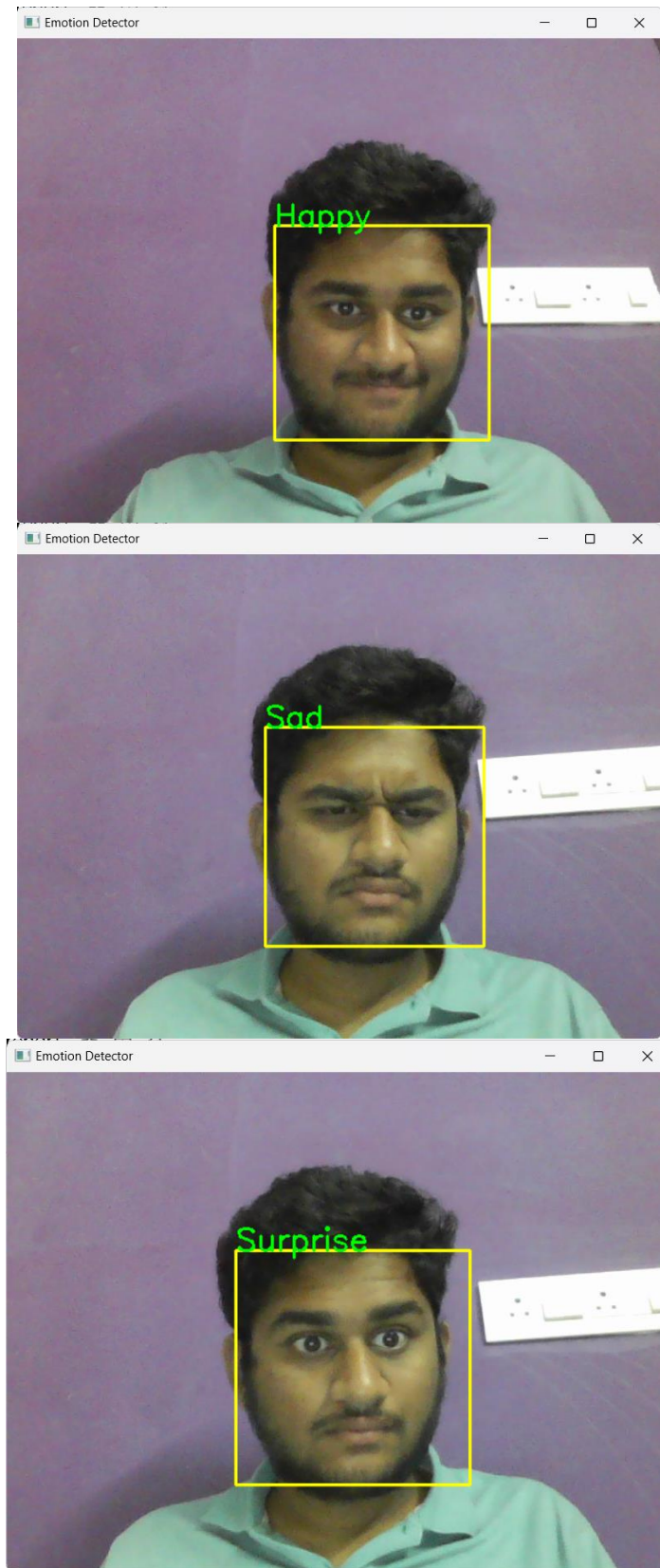
cap.release()
cv2.destroyAllWindows()
```

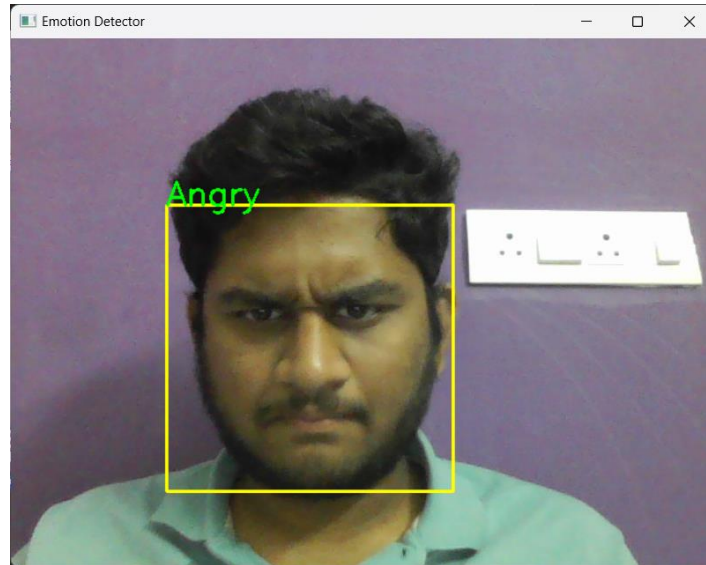
Requirements.txt:-

```
Python
TensorFlow
Keras
OpenCV
Pandas
Numpy
Seaborn
Matplotlib
```

Output:-







Conclusion

The hand gesture and facial expression recognition systems both utilize advanced machine learning and computer vision techniques for real-time recognition tasks. The hand gesture recognition system, optimized for mobile applications, follows a two-step approach with a palm detection model achieving an average precision of 95.7%, followed by a precise hand landmark model robust to occlusions. On the other hand, the facial expression recognition system employs a CNN model to classify emotions in real-time from video streams, achieving 72% training accuracy and 60% validation accuracy. Both systems effectively demonstrate the integration of machine learning models for interactive and responsive applications, providing valuable insights and real-time tracking capabilities in their respective domains.

Summary

Both assignments leverage advanced machine learning and computer vision techniques to achieve real-time recognition and tracking. The hand gesture recognition system is optimized for mobile use and achieves high precision in palm detection and robust hand landmark tracking. The facial expression recognition system, on the other hand, focuses on real-time emotion detection from facial expressions using a pre-trained CNN model, achieving moderate accuracy. Both systems demonstrate the potential of integrating machine learning models for interactive and responsive applications.

Reference links:

1. https://en.m.wikipedia.org/wiki/Computer_vision
2. <http://hdl.handle.net/1721.1/11589>
3. <https://www.zebra.com/ap/en/resource-library/faq/what-is-the-difference-between-machine-vision-computer-vision.html>
4. Rosenfeld, A., 1988. Computer Vision. Science, 253 5025, pp. 1249-54 . [https://doi.org/10.1016/S0065-2458\(08\)60261-2](https://doi.org/10.1016/S0065-2458(08)60261-2)
5. Charan, A., Chowdary, C., & Komal, P., 2022. The Future of Machine Vision in Industries- A systematic review. IOP Conference Series: Materials Science and Engineering, 1224. <https://doi.org/10.1088/1757-899X/1224/1/012027>
6. Zhilenkov, A., 2023. Prospects for the Application and Development of Computer vision Technologies. Artificial societies. <https://doi.org/10.18254/s207751800025011-1>
7. Batchelor, B., & Charlier, J., 1998. Machine vision is not computer vision. , 3521. <https://doi.org/10.1117/12.326946>
8. Zhao, H., 2017. Research Progress of Machine Vision Technology in Artificial Intelligence. DEStech Transactions on Computer Science and Engineering. <https://doi.org/10.12783/dtcse/aiea2017/14920>
9. Kita, Y., Ishikawa, H., & Masuda, T., 2017. Guest Editorial: Machine Vision Applications. International Journal of Computer Vision, 122, pp. 191-192. <https://doi.org/10.1007/s11263-017-0990-1>
10. López, M., Sergiyenko, O., & Tyrsa, V., 2008. Machine Vision: Approaches and Limitations. <https://doi.org/10.5772/6156>
11. Tian, H., Wang, T., Liu, Y., Qiao, X., & Li, Y., 2020. Computer vision technology in agricultural automation —A review. Information Processing in Agriculture, 7, pp. 1-19. <https://doi.org/10.1016/j.inpa.2019.09.006>
12. <http://www.pvpsit.ac.in/autonomous20/31/CSE/20CS4501D.pdf>

13. <http://www.pvpsit.ac.in/autonomous20/32/cse/20CS3602.pdf>
14. <https://doi.org/10.1016/B978-0-12-815739-8.00006-7>
15. <https://doi.org/10.1109/ICMCCE.2017.49>
16. <https://shorturl.at/8E6Op>
17. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
18. <https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm>
19. <https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/>
20. <https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb>
21. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/>
22. <https://doi.org/10.38094/jastt20165>
23. <https://doi.org/10.17849/in-sm-47-01-31-39.1>
24. <https://link.springer.com/article/10.1007/s11749-016-0481-7>
25. <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
26. <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
27. <https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial>
28. <https://shorturl.at/gWGAz>
29. <https://www.analyticsvidhya.com/blog/2021/06/image-processing-using-cnn-a-beginners-guide/>
30. <https://www.analyticsvidhya.com/blog/2022/06/yolo-algorithm-for-custom-object-detection/>
31. <https://encord.com/blog/yolo-object-detection-guide/>

32. <https://viso.ai/computer-vision/what-is-computer-vision/>
33. <https://www.geeksforgeeks.org/computer-vision/>
34. <https://www.mygreatlearning.com/blog/what-is-computer-vision-the-basics/>
35. <https://opencv.org/blog/what-is-computer-vision/>
36. <https://shorturl.at/3qGJi>
37. <https://shorturl.at/UUWpO>
38. <https://shorturl.at/jbtim>
39. <https://www.faqs.com.pk/the-importance-of-machine-learning/>
40. <https://365datascience.com/question/can-we-use-svm-for-multi-class-classification/>
41. https://www.researchgate.net/figure/A-linear-support-vector-machine_fig1_315316855
42. <https://www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm/>
43. <https://medium.com/@balajicena1995/support-vector-machine-with-numerical-example-8dfe81eae4f0>
44. <https://shorturl.at/0LjAN>
45. <https://youtu.be/kPw1IGUAoY8?si=kttUTAfXyUBoZCa>
46. <https://youtu.be/EESZtSOdhEQ?si=sRY5FfumXglPaTpo>
47. <https://youtu.be/bfmFfD2RIcg?si=lccSCMgCyJNJYHS>
48. <https://youtu.be/ZBnHoVMeeJE?si=nTvT-dV6u3DZVTQL>
49. <https://medium.com/@sachinsoni600517/concept-of-yolov1-the-evolution-of-real-time-object-detection-d773770ef773#:~:text=Limitations%20of%20YOU%20v1%20%3A,images%20with%20>

50. <https://shorturl.at/Dcpg7>
51. <https://youtu.be/zEXiGAPGKFk?si=iEYKyKS3iXduvRcA>
52. <https://youtu.be/F-irLP2k3Dk?si=24jLw265wVVaMd97>
53. <https://youtu.be/F-irLP2k3Dk?si=aBl0CTnSWICS2-o0>