FACIAL LANDMARK IDENTIFICATION VIA MODIFIED VGG NETWORK ARCHITECTURE

Project work submitted to Avinashilingam Institute for Home Science and Higher Education for Women

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

SUBMITTED BY

M. Pavithra (21PIT004)

Under the Guidance of

Dr. Mrs. F. Paulin MCA, M.Phil., Ph.D.

Assistant Professor

Department of Information Technology

AVINASHILINGAM INSTITUTE FOR HOME SCIENCE AND HIGHER EDUCATION FOR WOMEN

SCHOOL OF PHYSICAL SCIENCES AND COMPUTATIONAL SCIENCES

DEPARTMENT OF INFORMATION TECHNOLOGY

COIMBATORE-641043

MAY-2023

DECLARATION

DECLARATION

I hereby declare that the project entitled **"FACIAL LANDMARK IDENTIFICATION VIA MODIFIED VGG NETWORK ARCHITECTURE"** is a record of the original work done by **M. Pavithra(21PIT004)** under the guidance of **Dr. Mrs. F. Paulin MCA, M.Phil., Ph.D.,** Assistant Professor, Department of Information Technology, School of Physical Sciences and Computational Sciences**,** Avinashilingam Institute for Home Science and Higher Educationfor Women in the partial fulfilment for the award of the degree of Master of Science in Information Technology, and this project work has not formed the basis for any Degree/Diploma/Associates.

Place: Coimbatore

Date:

M Pavithra

Signature of the Candidate

19/5/2023

Countersigned By

Dr. Mrs. F. Paulin MCA, M.Phil., Ph.D.,

Assistant Professor,

Department of Information Technology,

School of Physical Sciences and Computational Sciences**.**

CERTIFICATE

CERTIFICATE

MILLION PIXELS
Of Technology

May 02,2023
Coimbatore

TOWHOMSOEVER IT MAY CONCERN

This is to certify that **Ms. Pavithra M (Roll No:21PITO04)**, M.Sc(IT) student from Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore has Successfully completed her internship training entitled "Facial Landmark Identification via Modified VGG Network Architecture" from our esteemed organization. The internship training duration is from 01-02-2023 to 29-04-2023. Her performance and conduct were found to be very good.

During this period, she was sincere and regular in attending all the phases of internship training Program.

Million Pixels of Technology,

Authorized Signatory

2/607 E, Lakshmi Nagar, Malumachampatti (Po), Coimbatore - 641050.
Email : millionpixels24@gmail.com

CERTIFICATE

This is to certify that this project work entitled **"FACIAL LANDMARK IDENTIFICATION VIA MODIFIED VGG NETWORK ARCHITECTURE"** done by **M. Pavithra (21PIT004)** has been submitted to Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore-641043 in partial fulfillment of the requirement for the award of the degreeof **MASTER OF SCIENCE IN INFORMATION TECHNOLOGY.** This Project has not found the basis for the award of any Degree/Associate/fellowship or similar title to any Candidate of any University. Certified as a bonafide record of the work submitted for the Viva voce held on _____.

Signature of the HOD

19/5/2023

Signature of the Guide

Signature of External Examiner

ACKNOWLEGEMENT

ACKNOWLEDGEMENT

I owe my sincere thanks to **Lord Almighty** and **My lovable parents** for showering their generous blessings upon me in all endeavors.

I wish to express my gratitude to **Prof. S. P.Thyagarajan,** Chancellor, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing thefacilities to conduct this study.

I extend my thanks to **Dr. Bharathi Harishankar, Ph.D., FRSA.,** Vice Chancellor, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing flamboyant help towards the completion of the study.

I record my deep sense of gratitude and indebtedness to **Dr. S. Kowsalya,** M.Sc., M.Phil., Ph.D., Registrar, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing adequate help for the study

I grateful record my sincere thanks to **Dr. G. Padmavathi** M.Sc., M.Phil., Ph.D., Dean and Professor, School of Physical Sciences & Computational of Sciences, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for timely help rendered throughout the course of this work.

I heartily Thanks to **Dr. Mrs. D. Shanmugapriya** M.Sc., M.Phil., Ph.D., SET Head of Department of Information Technology for the valuable guidance and encouragement during the course of our project.

I heartily Thank my esteemed project guide **Dr. F. Paulin** MCA. M.Phil., Ph.D., Assistant Professor, Department of Information Technology, for imparting tremendous assistance and well-timed support for triumph of our project.

I express my honorable thanks to our project coordinator Department of Information Technology, for imparting tremendous assistanceand well-timed support for triumph of our project.

I sincerely thank all **the staff members** of Department of Information Technology Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for their help and support.

I would like to express my special thanks to **my parents, my friends** and all **my well-wishers** for their constant encouragement, support and help in carrying out this work successfully.

ABSTRACT

ABSTRACT

This project uses Deep Neural Networks and Data Augmentation methods to recognize important points in face. The project makes use of a Facial Keypoints Dataset with pictures of faces that have 15 critical point positions labelled in them. The study uses a variety of augmentation strategies to improve the size and diversity of the training data because Deep Neural Networks need a lot of labelled data to generalize effectively. The technology known as facial landmark identification is used to identify and monitor face features including the eyes, nose, mouth, and eyebrows in digital photographs. This work presents a technique for identifying face landmarks using the Modified VGG Network (Visual Geometry Group Network), a well-known CNN architecture adding with Stochastic Gradient Descent and Adam optimization algorithms are employed to gain effective performance. Experimental findings show that this method outperforms conventional techniques and reaches state-of-the-art performance with 94% accuracy. This technique can work well for real-time applications because it is computationally efficient. It has a wide range of uses, including augmented reality, virtual reality, facial expression recognition, makeup simulations, and virtual try-ons.

Keywords: Facial landmarks, Face Detection, Deep Learning, VGG Network.

CONTENTS

CONTENTS

INTRODUCTION

1. INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

Facial keypoint identification has evolved significantly over the years, with new techniques and algorithms being developed to improve accuracy and efficiency. Older techniques for facial keypoint identification typically relied on traditional computer vision methods, such as feature detection and template matching. These methods were limited in their ability to handle variations in pose, expression, and lighting, and required a large amount of manual tuning to achieve acceptable results. In contrast, modern facial keypoint identification techniques rely on Deep Learning algorithms, particularly Convolutional Neural Networks (CNNs). CNNs are able to automatically learn features from raw input data, such as images, and can generalize well to new data. This makes them well-suited for facial keypoint identification, where the input data can vary significantly.

One of the most significant advancements in modem facial keypoint identification is the use of pre-trained CNNs. These networks have been trained on large datasets, such as ImageNet, and can identify basic features, such as edges and corners, which can be useful for detecting key points. Pre-trained CNNs can be fine-tuned on smaller datasets specific to the problem at hand, allowing them to learn more specific features related to facial keypoint identification.

Another important advancement in modern facial keypoint identification is the use of data augmentation techniques. Data augmentation is the process of artificially expanding the size of the training dataset by applying various transformations to the original images. These transformations can help the algorithm learn to detect key points despite variations in lighting, pose, and expression.

Overall, modern facial keypoint identification techniques have improved significantly over older techniques, with higher accuracy and greater robustness. This has opened up new opportunities for applications in fields such as healthcare, security, and entertainment.

Facial keypoint identification is a computer vision technique that involves identifying and locating specific points on a person's face, such as the position of the eyes, nose, mouth, and other facial features. These key points are used to describe the geometry and structure of a face and are essential for many computer vision applications, such as facial recognition,

emotion recognition, virtual makeup, augmented reality, and more. This project is achieved through the use of Deep Learning algorithms, which are trained on large datasets of labeled facial images. These algorithms use a combination of Convolutional Neural Networks (CNNs) and regression models to predict the location of key points on new images and we used modified VGG Network architecture.

Facial keypoint identification is a challenging task due to variations in lighting, pose, expression, and occlusion. To address these challenges, various techniques are used to increase the size and diversity of the training dataset, including data augmentation techniques such as rotation, scaling, and flipping, as well as the use of algorithms that can improve accuracy and efficiency.

1.2 ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI), is a field of computer science that aims to create intelligent machines that can perform tasks that typically require human intelligence, such as learning, problem-solving, perception, and decision-making. AI has made significant advances in recent years, and its applications span across various domains, including healthcare, finance, transportation, education, and entertainment.

AI can be broadly classified into two categories: narrow or weak AI and general or strong AI. Narrow AI is designed to perform specific tasks, such as facial recognition or language translation, while general AI aims to achieve human-like intelligence and reasoning abilities.

There are several subfields of AI, including Machine Learning, Natural Language Processing, Computer Vision, Robotics, And Cognitive Computing. Machine Learning, in particular, has seen tremendous growth in recent years and is a key driver of many AI applications. Machine Learning algorithms enable computers to learn from data and improve their performance over time without being explicitly programmed.

AI has several potential benefits, including increased efficiency, accuracy, and productivity in various domains. In healthcare, AI can help diagnose diseases, predict outcomes, and personalize treatments. In finance, AI can help identify fraud, analyze financial data, and provide personalized investment advice. In transportation, AI can assist in autonomous driving, traffic management, and route optimization.

However, there are also potential risks associated with AI, such as the displacement of human workers, bias and discrimination, and the potential misuse of AI for nefarious purposes. Therefore, it is crucial to develop and implement responsible AI policies that address these risks while promoting the benefits of AI. In conclusion, AI is a rapidly evolving field of computer science with significant potential benefits and risks. AI has already demonstrated its ability to transform various domains, and its impact is only expected to grow in the coming years. However, it is crucial to develop and implement responsible AI policies to ensure that the benefits of AI are realized while addressing its potential risks.

1.2.1 MACHINE LEARNING

Machine Learning is a subset of Artificial Intelligence that involves the development of algorithms and statistical models that enable computers to learn from data and improve their performance over time without being explicitly programmed. In other words, Machine Learning enables computers to automatically learn and make predictions or decisions based on patterns and relationships found in data.

Machine Learning can be broadly categorized into three types: supervised learning, unsupervised learning, and reinforcement learning. This project comes under supervised learning.

In supervised learning, the algorithm is trained on labeled data, where each data point is associated with a specific output or target value. The algorithm then uses this labeled data to learn how to predict the output for new, unseen data points.

In unsupervised learning, the algorithm is trained on unlabeled data, where the output or target value is not known. The algorithm then tries to find patterns and structure in the data, such as clusters or groups of similar data points.

Reinforcement learning involves training an algorithm to make decisions based on trial and error. The algorithm receives feedback or rewards for the actions it takes, and it learns to optimize its decision-making process based on maximizing the rewards it receives.

There are many applications of Machine Learning, including image and speech recognition, natural language processing, fraud detection, recommender systems, and autonomous vehicles. Machine Learning is also used in various fields, such as healthcare, finance, retail, and transportation, to improve decision-making and provide insights into complex systems and processes.

However, there are also challenges and limitations to Machine Learning, such as the need for large amounts of high-quality data, the potential for bias in data and algorithms, and the difficulty in interpreting the decisions made by Machine Learning models.

In conclusion, Machine Learning is a powerful technology that enables computers to learn from data and improve their performance over time. Machine Learning has many applications across various domains and has the potential to transform the way we live and work. However, it is crucial to address the challenges and limitations of Machine Learning and develop responsible AI policies to ensure that the benefits of Machine Learning are realized while addressing its potential risks.

1.2.2 DEEP LEARNING

Deep Learning is a subset of Machine Learning that uses Artificial Neural Networks to model and solve complex problems. It involves the use of multiple layers of algorithms that learn es representations of data, with each layer building upon the previous one to extract increasingly complex features from the input data. Deep Learning models are often used for tasks such as image and speech recognition, natural language processing, and other areas where large amounts of data need to be analyzed to identify patterns and make predictions. One of the key advantages of Deep Learning is that it can automatically learn to recognize features and patterns in data without the need for human intervention or explicit programming.

This makes it a powerful tool for solving problems in areas such as computer vision, speech recognition, and natural language processing. Deep Learning is used to solve real-time problems by training Artificial Neural Networks on large amounts of data, and then using the resulting models to make predictions or decisions in real time.

For example, in computer vision applications, Deep Learning models can be trained on millions of images to recognize objects, faces, and other visual patterns. Once the model is trained, it can be used in real time to identify objects in new images or video streams. Similarly, in natural language processing applications, Deep Learning models can be trained on vast amounts of text data to understand the meaning of words and phrases, and then used to generate responses to queries or to translate text from one language to another.

In many cases, Deep Learning models are deployed on specialized hardware such as GPUs or TPUs to enable real-time processing of large amounts of data. This allows for the rapid processing of data streams and the generation of predictions or decisions in near real

time. Overall, Deep Learning is a powerful tool for solving real-time problems across a wide range of applications, from computer vision and natural language processing to robotics and autonomous systems.

There are several types of Deep Learning models, each designed to solve different types of problems. Some of the most common types of Deep Learning models are:

- Feedforward Neural Networks: These are the most basic type of Deep Learning model, consisting of layers of interconnected nodes that process input data in a forward direction. Feedforward neural networks are used for tasks such as image classification and speech recognition.
- Convolutional Neural Networks (CNNs): CNNs are designed for image and video processing, and use specialized layers to extract features from visual data.
- Recurrent Neural Networks (RNNs): RNNs are used for sequential data, such as time series or natural language processing, and use feedback loops to process data over time.
- Long Short-Term Memory (LSTM) Networks: LSTMs are a type of RNN that are designed to overcome the vanishing gradient problem, which can occur when training RNNs on long sequences of data.
- Generative Adversarial Networks (GANS): GANs are a type of Deep Learning model that consists of two networks, a generator and a discriminator, which are trained together to generate new data that is similar to the training data.
- Autoencoders: Autoencoders are used for unsupervised learning, and are designed to learn representations of input data by encoding it into a lower-dimensional space and then decoding it back to its original form.

There are many other types of Deep Learning models as well, each with its own strengths and weaknesses. Choosing the right model for a particular problem requires an understanding of the data and the task at hand.

1.2.3 ARTIFICIAL NEURAL NETWORK

Artificial Neural Networks(ANN) are a subset of Machine Learning algorithms that are loosely modeled after the structure and function of the human brain. ANNs consist of

interconnected nodes, or "neurons," which are organized into layers. Each neuron receives input from the neurons in the previous layer and applies a mathematical function to the input to produce an output.

The connections between neurons in ANNs are assigned weights that are learned from data during training. These weights are used to determine the strength of the connection between neurons and ultimately the output of the network. During training, the weights are adjusted to minimize the difference between the predicted output of the network and the actual output.

There are many different types of ANNs, including feedforward networks, convolutional networks, and recurrent networks. Feedforward networks are the simplest type of ANN, and they consist of one or more layers of neurons that process the input and produce an output. Convolutional networks are commonly used in image and video processing, and they apply convolutional filters to the input to extract features. Recurrent networks are used for processing sequential data, such as natural language, and they include connections between neurons that allow information to be passed from one time step to the next.

ANNs have many applications in various domains, including image and speech recognition, natural language processing, and decision-making. ANNs have achieved state-of-the-art performance on many benchmark datasets and are considered a powerful tool for solving complex problems.

However, ANNs also have limitations and challenges, such as the need for large amounts of high-quality data, the potential for overfitting, and the difficulty in interpreting the decisions made by ANNs. It is crucial to address these challenges and develop responsible AI policies to ensure that the benefits of ANNs are realized while addressing their potential risks. In conclusion, ANNs are a powerful subset of Machine Learning algorithms that are inspired by the structure and function of the human brain. ANNs have many applications and have achieved state-of-the-art performance on many benchmarks. However, ANNs also have challenges and limitations, and it is crucial to address these challenges and develop

responsible AI policies to ensure that the benefits of ANNs are realized while addressing their potential risks.

1.2.3.1 CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Networks (CNNs) are a type of Deep Learning model that are primarily used for image and video processing tasks, such as object detection, classification, and segmentation. CNNs are designed to learn features from the input data in a hierarchical manner, allowing them to recognize patterns and structures within the data. CNNs are made up of several layers, including convolutional layers, pooling layers, and fully connected layers. The input to a CNN is typically a 2D array of pixels representing an image, although it can also be applied to other types of data such as audio and video. The first layer of a CNN is typically a convolutional layer, which applies a set of learnable filters to the input image, producing a set of feature maps. Each filter is designed to detect a specific feature, such as edges or corners, and the output of each filter represents the degree to which that feature is present in the input image. After the convolutional layer, a pooling layer is typically applied to reduce the dimensionality of the feature maps, by down-sampling the output of the convolutional layer. This helps to reduce the computational cost of the model and also helps to prevent overfitting. After several layers of convolutional and pooling layers, the output is typically passed through one or more fully connected layers, which perform the final classification or regression task. During training, the parameters of the CNN are learned using backpropagation and gradient descent, by comparing the predicted output of the model to the true output, and adjusting the weights of the filters to minimize the error.

One of the key advantages of CNNs is their ability to learn hierarchical representations of the input data, allowing them to recognize increasingly complex features and patterns. This makes them well-suited for tasks such as image classification, where objects can have multiple layers of features, such as edges, textures, and shapes. Overall, CNNs have become a popular and effective tool for image and video processing tasks, and have been used in a wide range of applications, from self-driving cars to medical imaging.

1.2.3.2 VISUAL GEOMETRY GROUP NETWORK

VGG (Visual Geometry Group) network is a deep Convolutional Neural Network architecture proposed by the Visual Geometry Group at the University of Oxford in 2014. It achieved breakthrough performance on the ImageNet dataset, which led to its popularity and widespread adoption in the computer vision community.

The VGG network is characterized by its deep architecture, which consists of 16 or 19 layers of convolutional and fully connected layers. It is a supervised learning model that is trained on a large dataset of labeled images to perform image recognition tasks such as object classification, object detection, and image segmentation.

The architecture of the VGG network consists of several layers of convolutional layers with small receptive fields of size 3x3 and a stride of 1 pixel. Each convolutional layer is followed by a non-linear activation function (ReLU) and a max-pooling layer with a 2x2 window and a stride of 2 pixels. The max-pooling layer helps reduce the spatial resolution of the output feature maps and makes the network more robust to small shifts in the input image.

After several convolutional layers, the VGG network consists of several fully connected layers, which perform the final classification of the input image. The fully connected layers are followed by a softmax layer, which produces the probability distribution over the classes.

One of the key advantages of the VGG network is its simplicity and ease of implementation. The network architecture is straightforward and easy to understand, making it a popular choice for researchers and practitioners in the computer vision community. The VGG network also achieved state-of-the-art performance on several benchmark datasets, including ImageNet, CIFAR-10, and CIFAR-100.

However, the main disadvantage of the VGG network is its high computational cost

and memory requirements. The large number of layers and parameters make it difficult to train the network on low-end hardware, and it may take a long time to train the model on large datasets. To address this issue, several variants of the VGG network have been proposed, including the VGG-D, VGG-E, and VGG-S networks, which reduce the number of layers or parameters while maintaining high performance.

1.2.3.3 OPTIMIZATION ALGORITHMS

SGD (Stochastic Gradient Descent) and Adam (Adaptive Moment Estimation) are two popular optimization algorithms used in this project to train neural networks with VGG Net architecture.

SGD is a simple and widely used optimization algorithm that updates the parameters of the model by taking small steps in the direction of the negative gradient of the loss function. In other words, it moves the parameters of the model in the direction that reduces the loss function the most. The learning rate determines the step size and controls how much the parameters are updated at each iteration. While SGD is simple and easy to implement, it can be slow to converge and may get stuck in local minima.

Adam is an adaptive learning rate optimization algorithm that combines the advantages of both SGD and RMSProp (Root Mean Square Propagation). It computes the learning rate for each parameter based on the first and second moments of the gradient, which are estimated using exponential moving averages. The learning rate is adaptive and changes based on the variance and mean of the gradient, which can improve the convergence speed and stability of the training process. Adam also uses bias correction to adjust the moving averages of the gradients, which can improve the accuracy of the optimization.

One of the main advantages of Adam is that it is less sensitive to hyperparameter settings than SGD. It does not require manual tuning of the learning rate or other hyperparameters, which can save time and effort in the training process. Adam also has been

shown to work well in practice on a variety of tasks and datasets, and it is a popular choice for Deep Learning applications.

However, Adam can suffer from some disadvantages, such as slower convergence speed than SGD with momentum and more memory usage due to the storage of the additional moments. In some cases, SGD with momentum may perform better than Adam, especially on small datasets or with specific architectures.

In summary, both SGD and Adam are widely used optimization algorithms for training neural networks. While SGD is simple and easy to implement, it can be slow to converge and may get stuck in local minima. Adam, on the other hand, is an adaptive learning rate optimization algorithm that can converge faster and is less sensitive to hyperparameter settings. However, the choice between the two depends on the specific task and dataset, and both can be useful in different scenarios.

1.3 ABOUT PLATFORM

1.3.1 PYTHON

Python is a high-level, interpreted programming language that was first released in 1991. It is widely used for a variety of purposes such as Web Development, Data Analysis, Artificial Intelligence, Scientific Computing, and more. Python is known for its simplicity, readability, and ease of use. It uses an indentation-based syntax that helps developers write clean, readable code. The language is dynamically typed, which means that variable types are determined at runtime. This allows for greater flexibility and faster development times, but also requires careful attention to variable types to avoid errors. Python has a large standard library that provides many useful modules for various purposes such as string manipulation, file LO, network communication, and more. There are also numerous third-party libraries and frameworks available, such as Django and Flask for web development, NumPy and Pandas for scientific computing and data analysis. TensorFlow and PyTorch for Machine Learning and Artificial Intelligence, and many more. Python's popularity has increased significantly in recent years due to its versatility, ease of use, and community support. It is often used as a first programming language for beginners, but is also widely used by experienced developers in a variety of industries. Overall, Python is a powerful, flexible, and easy-to-lear programming language that is well-suited for a wide range of applications. Its popularity is Likely to continue to grow as more and more developers discover its many benefits.

Usage

Python is a popular programming language for developing Artificial Intelligence, Machine Learning, And Artificial Neural Network projects due to its simplicity, ease of use, and availability of many useful libraries and frameworks. In Al projects, Python is used to develop Natural Language Processing (NLP) applications, chatbots, and virtual assistants. Python libraries such as NLTK, SpaCy, and Gensim are commonly used for NLP tasks such as sentiment analysis, entity recognition, and topic modeling. In Machine Learning projects, Python is used to develop algorithms that can learn from data and make predictions or decisions. Popular Machine Learning libraries such as Scikit-learn, TensorFlow, and PyTorch provide a wide range of tools for building, training, and deploying Machine Learning models. These libraries also offer many pre-built models that can be easily integrated into a project.

In artificial neural network projects, Python is used to develop models that can simulate the behavior of the human brain. Python libraries such as Keras, TensorFlow, and PyTorch provide a high-level API for building neural networks and offer many pre-built models such as Convolutional Neural Networks (CNNs), recurrent neural networks (RNNs), and long short- term memory (LSTM) networks. Overall, Python's simplicity, ease of use and availability of many useful libraries and frameworks make it an ideal programming language for Al, Machine Learning, and Artificial Neural Network projects.

What Can Python Do?

Python can do a wide variety of things. Like…

- Web Development: Python can be used for server-side programming to develop web applications. Popular web frameworks like Django and Flask are built in Python.
- Data Analysis: Python provides several libraries such as Pandas and NumPy that help in data analysis, manipulation, and visualization. Python also integrates well with other tools like SQL databases and Excel spreadsheets.
- Machine Learning: Python provides several libraries such as TensorFlow, Scikit-learn, and PyTorch that help in building and deploying Machine Learning models.
- Scientific Computing: Python is used in scientific computing for simulations, data modeling, and data visualization. Libraries like SciPy, SymPy, and Matplotlib provide powerful tools for scientific computing.
- Automation: Python is used for automating repetitive tasks like testing, data extraction, and report generation.
- Game Development: Python is used for game development, and many popular games like Battlefield 2 and Civilization IV were built in Python.

Why Python?

Python is a popular programming language for several reasons:

- Simplicity: Python has a simple syntax that is easy to learn and read. It is a high-level language that abstracts away many low-level details, making it easier for

developers to focus on solving problems rather than worrying about implementation details.

- Versatility: Python can be used for a wide range of applications, from web development and scientific computing to Machine Learning and automation. It is a multi-purpose language that can be used in many different industries and domains.
- Large community and ecosystem: Python has a large and active community of developers who contribute to its development and create many useful libraries and frameworks. The availability of many libraries makes Python a powerful tool for many different tasks.
- Portability: Python code can run on many different platforms and operating systems, including Windows, macOS, Linux, and more.
- Integration: Python integrates well with other technologies and tools, making it easy to use with databases, web servers, and other applications.

1.3.2 ANACONDA JUPYTER NOTEBOOK

Anaconda is a popular distribution of the Python programming language that includes many pre-installed libraries and tools commonly used in data science and Machine Learning. Jupyter Notebook, on the other hand, is a web-based interactive environment that allows users to create and share documents that contain live code, equations, visualizations, and narrative text.

When you install Anaconda on your computer, it includes a version of Jupyter Notebook, along with other data science tools such as NumPy, Pandas, Matplotlib, and scikit-learn. You can access Jupyter Notebook by opening the Anaconda Navigator, which is a graphical interface that provides easy access to all the tools and libraries included in Anaconda. From the Navigator, you can launch Jupyter Notebook, which will open in your web browser. Once you have Jupyter Notebook open, you can create a new notebook and begin writing Python code, which will be executed in real-time as you type.

Jupyter Notebook allows you to split your code into cells, which can be executed individually or all at once, making it easy to test and debug your code. You can also use Jupyter Notebook to create interactive visualizations and to document your code using

Markdown text. Jupyter Notebook is a powerful tool for data science and Machine Learning, as it provides an interactive and flexible environment for developing and testing code. It allows you to easily experiment with different algorithms and models, and to visualize and explore data in real- kime. Additionally, Jupyter Notebook makes it easy to share your work with others, as notebooks can be exported to HTML, PDF, or other formats.

2. LITERATURE REVIEW

This section introduces related works briefly on face detection, face alignment, landmark localization convolution design. According to the author savina jassica colaco and dong seog han [1] they add multi-scale fully connected layers to the standard network known as EfficientNet in order to anticipate the facial landmarks on human faces that are shown on the identified face in real-time. By consistently scaling the width, depth, and resolution dimensions, the suggested model with the compound scaling method produced a scalable model. Both larger and smaller models are evaluated using an adaptive wing loss function. With various head positions and occlusion situations, we also tested the model's robustness. With a model size of 24.6 MB, the suggested model, which trained on a sizable dataset, may reach 90% accuracy for larger models and roughly 88%–89% accuracy for smaller models.

Among different proposed face detection methods, multi-task cascaded convolutional networks (MTCNN) [2] and faster region-based Convolutional Neural Networks (R-CNN) [3] are well-known. Many object detection methods popularly use faster R-CNN where it generates bounding boxes based on predefined anchors with object classification. Subsequently, it crops the feature maps with object detection and refines the bounding box proposals for better results. Faster R-CNN with ResNet as a backbone is used for face detection.

The face alignment methods can be categorized as handcrafted-feature-based and deep-learning-based methods. In the hand-crafted-feature-based methods, the tree structure part model (TSPM) used a deformable part-based model for face shape modeling with a mixture of other tree models for landmark localization, pose estimation and detection in parallel. For capturing the face appearance, cascade regression-based methods with scale-invariant feature transform (SIFT) features were used but the methods were incapable to find an unrestricted face with extreme poses. The statistical methods such as the constrained local model (CLM) and active appearance model (AAM) maximize the confidence of keypoints in an image. A realtime face aligning with facial landmarks using an ensemble of regression trees in [4]. The drawbacks of conventional approaches are expensive computation, high complexity of the model and less robustness.

Deep Learning-based approaches have been adopted since they outperform the conventional approaches. We briefly introduce the works in landmarks localization. A multi-task

learning network [5], called tasks-constrained deep convolutional network (TCDCN), learns pose attributes and landmarks locations jointly.

TCDCN is difficult to train because of its multi-task approach. Trigeorgis et al. [6] proposed a model for facial alignment with an end-to-end recurrent convolution from coarse to fine, where the model is termed as mnemonic descent method (MDM). Lv et al. [7] proposed an architecture with the two-stage re-initialization (TSR) scheme using deep regression, which boosts detection accuracy by dividing the whole face into several parts. Yang et al. [8] proposed a network for landmarks detection which is assisted by head pose angles including yaw, pitch and roll. Jourabloo and Liu [9] proposed a pose-invariant face alignment (PIFA) model which estimates a 3D to 2D projection matrix using deep cascade regressors which later extended with the Convolutional Neural Network [10]. Zhu et al. [11] proposed a model with the face depth in Z-buffer and later fits 2D images in a 3D model.

Kumar and Chellappa [12] designed a convolution neural network with a single dendritic, named pose conditioned dendritic convolution neural network (PCD-CNN), for the improvement of detection accuracy by associating classification network with modular and second classification networks.

Honari et al. [13] designed sequential multitasking (SeqMT) network equivariant landmark transformation (ELT) loss term. Dong et al. [14] created the style-aggregated network (SAN) for robust face landmark detection with the intrinsic variance of image styles. Wu et al. [15] proposed a face alignment algorithm considering the boundary information as a geometric structure attribute for human faces. The landmarks are derived from boundary information to avoid ambiguities. Fan and Zhou [16] use multiple CNNs to increase robustness and improve accuracy for coarse-to-fine prediction.

Weng et al. [17] proposed a feature set matching approach for partial face matching by explicitly constrain the affine matrix. Ranjan et al. [18] proposed a novel architecture called HyperFace by fusing the intermediate layer of the CNN with post-processing methods namely iterative region proposals and landmarks-based non-maximum suppression to improve the overall performance.

Xiao et al. [19] developed a recurrent dual refinement model (RDR) to focus on large-pose

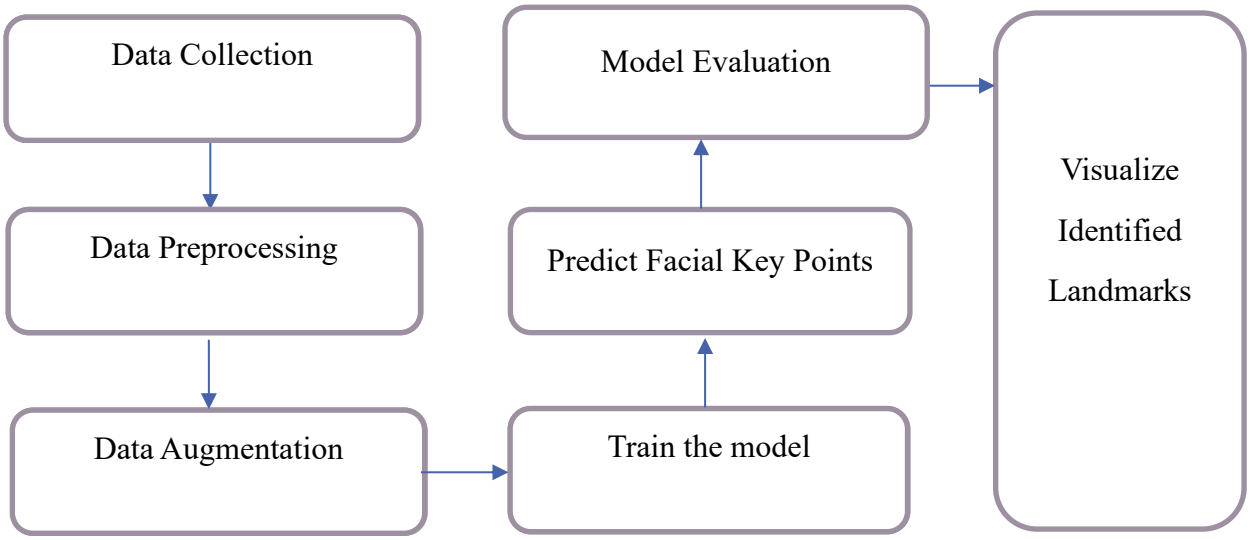
facial landmark detection with an end-to-end framework. Lai et al. [20] proposed a novel recurrent network to refine the estimated coordinates of facial landmarks iteratively. Junfeng and Haifeng [21] presented a global exemplar stacked auto-encoder network (GECSAN) face shape initialization and local information preserve stacked autoencoder networks (LIPSAN) for shape refinement to achieve a robust face refinement.

Xia et al. [22] proposed a CNN based head pose estimation with facial landmarks using heatmaps. Kim et al. [23] proposed an extended multi-task CNN (EMTCNN) model for facial landmarks detection in real-time. Zheng et al. [24] presented a model that aims at providing an efficient coarse-to-fine network by utilizing lightweight coordinate regression and heatmap regression. Zhang et al. [25] developed a structural hourglass network to predict the facial landmarks with corresponding heatmaps. The CNN based facial landmarks localization gets the high-level features from the face and predicts all the keypoints simultaneously.

3. METHODOLOGY

Methodology refers to the systematic approach and set of principles or procedures used to conduct research or solve a problem. It outlines the steps, techniques, and tools employed to gather data, analyze information, and draw conclusions. Methodology is an essential component of any research or project as it ensures that the results are reliable, valid, and replicable.

Proposed Methodology



Algorithm

Here's an algorithm for the work steps involved:

1. Load the facial keypoints dataset from Kaggle.
2. Remove rows with null values from the dataset.
3. Preprocess the image data.
4. Convert the pixel values from strings to arrays.
5. Reshape the image arrays to the desired dimensions (e.g., (96, 96)).
6. Scale the image pixel values to a range between 0 and 1.
7. Extract the keypoints from the dataset.

8. Define a function to visualize images and keypoints together.
9. Apply linear contrast and Gaussian blur augmentation techniques to the images.
10. Apply rotation and scaling augmentation techniques to the images and adjust the corresponding keypoints.
11. Apply horizontal flip augmentation to the images and update the x-coordinates of the keypoints.
12. Shuffle the augmented data.
13. Build a deep neural network model, such as VGG Net with Optimization Algorithms such as Stochastic Gradient Descent and Adam.
14. Define custom callbacks for monitoring training progress.
15. Compile and train the model on the augmented data.
16. Plot the training and validation loss and mean absolute error curves.
17. Use the trained model to predict keypoints for test images.
18. Visualize the predicted keypoints on selected test images.

3.1 DATA COLLECTION

Data collected from Kaggle called Facial Key points Detection. Kaggle is a popular platform for data scientists and Machine Learning enthusiasts to work on various data-related projects, including data collection. The platform offers a vast repository of datasets that can be used for analysis, modeling, and Machine Learning tasks.

Data collection in Kaggle can be done in two ways: by uploading your own dataset or by downloading datasets shared by other users. In either case, it's essential to understand the best practices for collecting data to ensure the quality and accuracy of the data you're working with. When uploading your own dataset to Kaggle, it's important to ensure that it's cleaned, pre-processed, and formatted correctly.

This includes removing any irrelevant or missing data, ensuring consistency in data formats, and making sure that the data is relevant to the problem you're trying to solve. On the other hand, when downloading datasets from Kaggle, it's crucial to read the documentation carefully to understand the data source, collection methodology, and any limitations or biases

associated with the data. It's also recommended to validate the data's quality by performing exploratory data analysis (EDA) and data cleaning to ensure that the data is relevant and reliable.

Dataset for Facial Key Points Detection from Kaggle

In the space of pixel indices, each projected keypoint is described as a (x,y) real-valued pair. The following features of the face are represented by the 15 keypoints.
left_eye_center, right_eye_center, left_eye_inner_corner, left_eye_outer_corner, right_eye_inner_corner, right_eye_outer_corner, left_eyebrow_inner_end, left_eyebrow_outer_end, right_eyebrow_inner_end, right_eyebrow_outer_end, nose_tip, mouth_left_corner, mouth_right_corner, mouth_center_top_lip, mouth_center_bottom_lip.

The data files' final field contains the input image, which is a list of pixels (sorted by row) represented as integers in the range of 0 to 255. Images are 96x96 pixels in size.

Data Files

- **Training.csv**

List of training photos (27049) in training.csv. 15 keypoints' (x,y) coordinates are contained in each row, along with image data in the form of a row-ordered list of pixels.

- **Test.csv**

List of 4783 test photos in test.csv. Each row includes an ImageId and a list of pixels organised by row for the picture data.

3.2 DATA PRE-PROCESSING

Data pre-processing is an essential step in any data-related project, including Machine Learning, Data Analysis, and Data Visualization. It involves cleaning, transforming, and preparing

raw data into a usable format for further analysis.

Here are some common techniques used in data pre-processing

- Data cleaning: This involves removing any irrelevant or missing data, correcting errors, and dealing with outliers that may affect the accuracy of the analysis.

- Data transformation: This includes scaling the data to a specific range, converting categorical variables into numerical variables, and creating new variables from existing ones.

- Data normalization: This is the process of scaling the data to have a mean of zero and a standard deviation of one. This technique helps in reducing the impact of variables with a large range of values.

- Data integration: This involves combining multiple datasets to create a single dataset that can be used for analysis.

- Data reduction: This involves reducing the size of the dataset by eliminating redundant variables, identifying important variables, and performing feature selection.

- Data discretization: This is the process of converting continuous variables into discrete variables. It can be useful for certain types of analysis, such as decision tree algorithms.

Data pre-processing helps in improving the accuracy and reliability of the analysis by ensuring that the data is consistent, relevant, and reliable. By applying these techniques, data scientists can uncover insights and patterns that would otherwise be difficult to detect.

It's important to note that data pre-processing is a time-consuming process that requires careful attention to detail. However, it's a crucial step in any data-related project that can significantly impact the accuracy and reliability of the results.

Data Cleaning

Removed the null value data from the dataset. Identified the null value data using isnull() method and use.sum() and dropped the NaN values by using df_dropped = df.dropna(how='any') method which is supported by the pandas library.

3.3 DATA AUGMENTATION

Data augmentation is a technique used in Machine Learning and computer vision to increase the size and diversity of a dataset by artificially creating new examples from existing data. This technique is particularly useful when working with limited data, as it can help improve model performance by exposing the model to more variations of the same data.

Here are some common techniques used in data augmentation

- Image flipping and rotation: This involves flipping images horizontally or vertically or rotating them to create new variations.
- Image cropping and scaling: This involves cropping images to different sizes or scaling them up or down.
- Image color and brightness adjustment: This involves adjusting the color and brightness of images to create new variations.
- Image distortion: This involves distorting images by adding noise, blur, or other effects to create new variations.
- Text augmentation: This involves adding noise to text data, such as replacing words with synonyms, misspellings, or random variations.

Data augmentation can help improve the accuracy and robustness of Machine Learning models by exposing them to a wider range of data variations. It's a popular technique used in

computer vision applications, such as object recognition, where models need to be trained on large datasets to achieve high accuracy.

It's important to note that data augmentation should be done carefully, as it can also introduce noise and bias into the data. It's essential to ensure that the augmented data is representative of the original data and does not introduce any new biases or inconsistencies. Therefore, data scientists need to choose the appropriate augmentation techniques based on the nature of the data and the problem they are trying to solve.

Add Noises on the image

- Added Gaussian Blur and Linear Contrast on images.

- Added Augmentation techniques using imgaug library such as
  - Horizontal Flip.
  - Rotation.
  - Scaling.

3.4 TRAIN THE MODEL

In this project we used Modified VGG (Visual Geometry Group) NET architecture to train the model. The architecture was developed for image recognition tasks, and has achieved state-of-the-art performance on several benchmark datasets The VGG Net architecture(Figure 3.4.1) is characterized by its use of small 3x3 convolutional filters throughout the network, and its deep structure with up to 19 layers. The use of small filters allows the network to learn more complex features at each layer, while keeping the number of parameters in the model relatively small. The deep structure of the network allows it to learn increasingly abstract features at each layer, resulting in high accuracy on image recognition tasks.

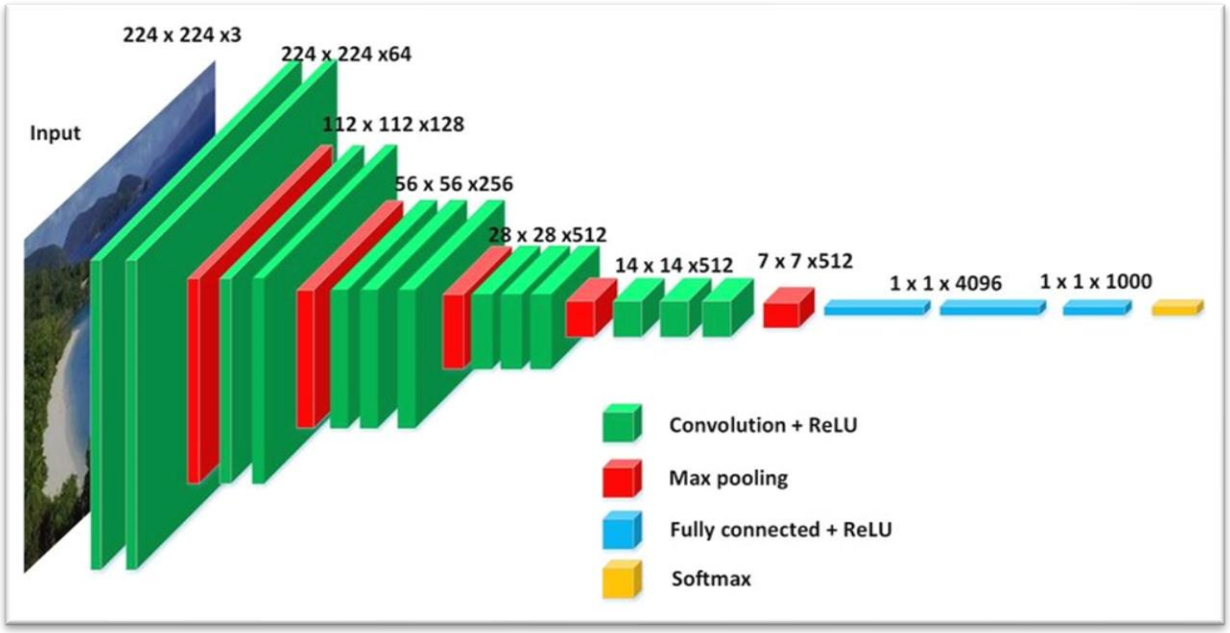


Figure (3.4.1) VGG Network Architecture

The original VGG Net architecture consists of five blocks of convolutional layers, each followed by a max pooling layer, and three fully connected layers. The first two blocks have two convolutional layers each, while the remaining three blocks have three convolutional layers each. The fully connected layers at the end of the network are used for classification.VGG net is a it is a standard deep Convolutional Neural Network (CNN) architecture with multiple layers. The "deep" refers to the number of layers with VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers and modified the model by using

i. **Learning Rate Scheduler:** It is a callback function in the Keras library that adjusts the learning rate of the optimizer during training. The lrdecay function passed to it is used to define how the learning rate should be decayed over time.

learning_rate=0.0010 : Val — 0.1278, Train — 0.1292 at 70th epoch.

learning_rate=0.0015 : Val — 0.1264, Train — 0.1280 at 70th epoch.

learning_rate=0.0020 : Val — 0.1265, Train — 0.1281 at 70th epoch.

learning_rate=0.0025 : Val — 0.1286, Train — 0.1300 at 70th epoch.

ii. Stochastic Gradient Descent Algorithm

Stochastic Gradient Descent, which is a popular optimization algorithm used for training neural networks. The lr argument is used to set the initial learning rate and the optimizer updates the model parameters.

Stochastic gradient descent (SGD) in contrast performs a parameter update for each training.

Example:

x (i) and label y (i): $\theta=\theta-\eta\cdot\nabla\theta J(\theta;x(i);y(i))$ $\theta = \theta - \eta \cdot \nabla \theta J (\theta ; x (i) ; y (i))$.

iii. Adam: It is another optimization algorithm that is widely used for training Deep Neural Networks. The learning rate argument is used to set the initial learning rate of the optimizer.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] \; v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

Parameters Used:

1. ϵ = a small +ve constant to avoid 'division by 0' error when (vt -> 0). (10-8)
2. $\beta1$ & $\beta2$ = decay rates of average of gradients in the above two methods. ($\beta1 = 0.9$ & $\beta2 = 0.999$)
3. α — Step size parameter / learning rate (0.001)

Since mt and vt have both initialized as 0 (based on the above methods), it is observed that they gain a tendency to be ‘biased towards 0’ as both $\beta1$ & $\beta2 \approx 1$. This Optimizer fixes this problem by computing ‘bias-corrected’ mt and vt. This is also done to control the weights while reaching the global minimum to prevent high oscillations when near it.

3.5 PREDICT FACIAL KEY POINTS

Predicting the Facial Key points on the test.csv file images after completion of training the modified VGG net architecture. Once the model is loaded, the next step is to use it to make predictions on the test dataset. This involves passing each image through the model and obtaining the predicted facial keypoints.

After obtaining the predicted facial keypoints, it is important to post-process the results to ensure that they are accurate and in the correct format. This may involve rounding the predicted keypoint positions, clipping them to ensure they are within the image bounds, or applying any other necessary transformations.

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset First need to download the stop words library using the nltk. to use Gensim and essentially is a library that's actually really powerful and can be used for natural language processing. Remove the stopwords and the unnecessary data in the dataset.

3.6 MODEL EVALUATION

Test the project module for facial key point detection with image and key point augmentation using the Imgaug library, one can follow the following steps:

a) Load the saved model: Once the model is trained, save the model weights and load them in the testing module. This can be done using the load_weights() function in Keras.
b) Load the test data in the same way as done for training. For facial key point detection, the test data should also include the input images and the corresponding key points.
c) Apply the same image and key point augmentations to the test data as done during the training phase. This can be done using the Imgaug library in Python.
d) Use the evaluate() function in Keras to evaluate the performance of the model on the test data. This function returns the value of the loss and the accuracy of the model on the test data.
e) Visualize the output of the model on the test data. This can be done by plotting the input images along with the predicted key points. This will help to evaluate the performance of the model qualitatively.

The above steps are used to test the facial key point identification model with image.

3.7 VISUALIZE IDENTIFIED LANDMARKS

Visualizing identified facial landmarks is a crucial step in many computer vision and facial analysis tasks. Facial landmarks are specific points of interest on a human face, such as the corners of the eyes, nose, and mouth. By visualizing these landmarks, we can better understand the spatial relationships between different parts of the face and extract valuable insights for tasks like facial recognition, emotion detection, and virtual makeup.

Here are some common techniques used to visualize identified facial landmarks

- Overlaying landmarks on the image: One common way to visualize facial landmarks is to overlay them on the original image of the face. This can be done by drawing circles or other markers at the locations of the identified landmarks, such as the corners of the eyes and the tip of the nose.

- Creating a mesh grid: Another way to visualize facial landmarks is to create a mesh grid that connects the identified landmarks. This can be done using triangulation algorithms that create a mesh of triangles that cover the entire face.

- 3D visualization: For tasks that involve analyzing the 3D structure of the face, it can be useful to visualize the identified landmarks in 3D space. This can be done using tools like Blender or Unity, which allow for interactive 3D visualization and manipulation.

- Landmark tracking over time: In tasks that involve tracking the movement of facial landmarks over time, it can be useful to visualize the trajectory of each landmark over time using line plots or animations.

Visualizing identified facial landmarks can provide important insights into the structure and relationships between different parts of the face. It can also be useful for quality control, allowing us to verify that the identified landmarks are accurate and consistent across different images or frames.

Visualizing identified facial landmarks is an important step in many computer vision and facial analysis tasks. There are several techniques available for visualizing facial landmarks, including overlaying them on the original image, creating a mesh grid, 3D visualization, and landmark tracking over time. In this project we used Overlaying landmarks on the image method to visualize predicted facial landmarks. With careful attention to detail and appropriate visualization techniques, we can gain a better understanding of the structure and relationships within the face and extract valuable insights for various facial analysis tasks.

4. EXPERIMENTAL RESULTS AND DISCUSSION

Experimental results and discussion are key components of a research paper or scientific report. In this section, researchers present and interpret the findings of their study, highlighting the outcomes of their experiments or investigations and discussing their implications.

Presenting Experimental Results:

4.1 DISTRIBUTION OF KEYPOINTS

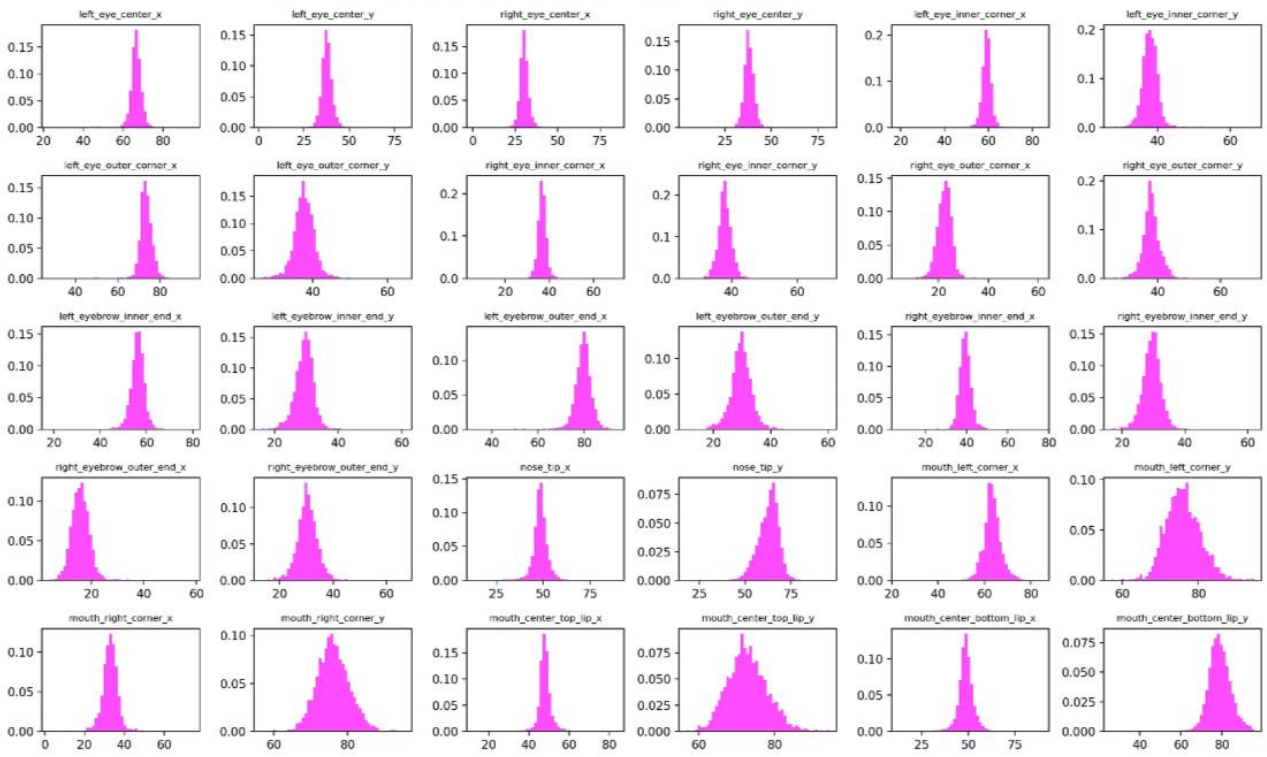


Figure (4.1.1) Distribution of keypoints

Figure (4.1.1) depicts distribution of keypoints. One of the simplest methods for data imputation would be to substitute the distribution mean for the NaN entries because the keypoints actually exhibit normal distribution. From an ML standpoint, it is essential to divide the data into train and test sets before applying the transformation for data imputation; otherwise, we run the risk of causing data leakage.

4.2 TRAINING AND VALIDATION LOSS

In Machine Learning, the training and validation loss are measures of how well a model is performing during the training and validation phases.

Figure (4.2.1) depicts Training Loss refers the error rate of the model on the training dataset during the training phase. The aim of training is to minimize the training loss, which means that the model is learning to fit the training data as accurately as possible.

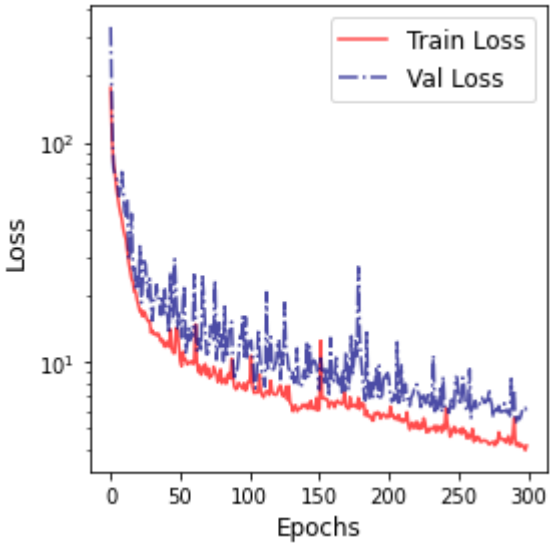


Figure (4.2.1) Training and Validation Loss

Validation loss, on the other hand, refers to the error rate of the model on a separate validation dataset during the training phase. The aim of validation is to check how well the model is generalizing to new data, which is not part of the training set.

The validation loss provides an estimate of how well the model will perform on unseen data, and it is used to evaluate and fine-tune the model during training. If the training loss continues to decrease while the validation loss starts to increase, it is an indication that the model is overfitting to the training data and is not generalizing well to new data.

In summary, the training loss measures how well the model is fitting the training data, while the validation loss measures how well the model is generalizing to new data. The goal is to minimize both the training and validation loss to build a good performing model.

4.3 TRAINING AND VALIDATION FOR MEAN ABSOLUTE ERROR

Figure (4.3.1**)** depicts the validation MAE is calculated by measuring the MAE of the model's predictions on a separate validation dataset. This metric provides an indication of how well the model is generalizing to unseen data. The aim is to minimize the validation MAE as much as possible to ensure that the model can make accurate predictions on new data.

During the training process, the model's parameters are updated based on the gradient of the loss function with respect to those parameters. The loss function used for MAE is the mean of the absolute differences between the predicted and actual values. The optimizer tries to minimize this loss function by updating the parameters in the direction of steepest descent. The model is trained until the training MAE and validation MAE converge to a stable value.

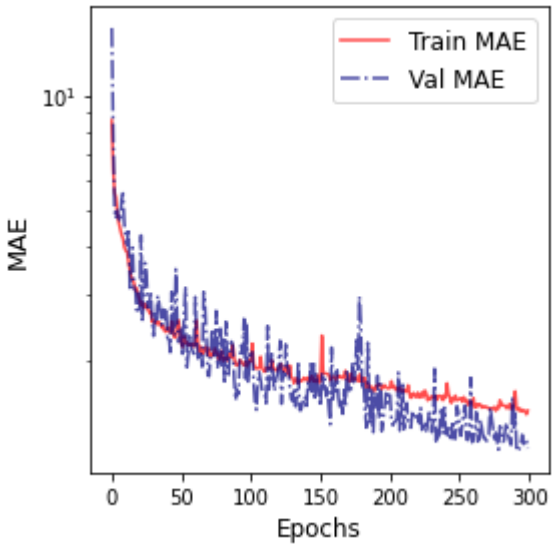


Figure (4.3.1**)** Same as left panel but for Mean Absolute Error

It is important to monitor both training and validation MAE during the training process to avoid overfitting. Overfitting occurs when the model becomes too complex and learns to fit to the noise in the training data. This can cause the model to perform well on the training data but poorly on the validation data. To avoid overfitting, it is common practice to use techniques such as regularization or early stopping during the training process.

In summary, the MAE is a commonly used metric to evaluate the performance of a regression model. The training MAE measures how well the model is fitting to the training data,

while the validation MAE measures how well the model is generalizing to unseen data. Both metrics are used to monitor the performance of the model during the training process and to avoid overfitting.

4.4 PERFORMANCE OF ALGORITHM

When evaluating the performance of an ANN model for facial landmark identification, accuracy, precision, and F1 score are commonly used metrics.

Accuracy: Accuracy is a measure of how well the model correctly predicts the facial landmarks. It is calculated by dividing the number of correctly predicted landmarks by the total number of landmarks.

Accuracy = (True positives + True Negatives)/ (True positives + True negatives + False positives + False negatives)

Accuracy provides an overall assessment of the model's performance, indicating the proportion of correctly identified landmarks. However, accuracy alone may not provide a complete picture, as it does not differentiate between different types of prediction errors.

Precision: Precision is a metric that evaluates the proportion of correctly predicted landmarks out of the total landmarks predicted by the model. Precision focuses on the positive predictions made by the model and indicates the model's ability to avoid false positives. In the context of facial landmark identification, precision measures how well the model identifies the correct facial landmarks and avoids wrongly predicting other points as landmarks.

Precision = TruePositives / (TruePositives + FalsePositives)

F1 Score: The F1 score is a measure that combines precision and recall (which measures the proportion of actual landmarks correctly predicted by the model) into a single value. It provides a balanced evaluation of the model's performance. F1 score is particularly useful when there is an imbalance between the number of positive and negative landmarks. The formula to calculate

the F1 score is

F1 = 2 * (precision * recall) / (precision + recall).

Algorithm	**Accuracy**	**Precision**	**F1 Score**
Modified VGG Network Architecture	0.94	0.90	0.83

Table 4.4.1

Table 4.4.1 shows the Accuracy, Precision and F1 score of this project. When evaluating the performance of an ANN model for facial landmark identification, it is common to calculate accuracy, precision, and F1 score using a validation or test dataset. These metrics provide insights into the model's ability to accurately identify facial landmarks and the trade-off between precision and recall. High accuracy, precision, and F1 score indicate that the model performs well in identifying facial landmarks accurately and consistently.

5. CONCLUSION

This work presented a technique for identifying face landmarks using the Modified VGG Network CNN architecture and pre-trained Kaggle's Facial Key-point Detection dataset. method included data augmentation strategies and employed Stochastic Gradient Descent and Adam Optimization Algorithms for improved performance. The experimental results demonstrated that this technique outperformed conventional methods and achieved state-of-the-art performance with 94% accuracy. This approach was computationally efficient, making it suitable for real-time applications. This work highlighted the various applications of Facial Keypoint Identification Technology in industries such as augmented reality, virtual reality, medical diagnosis and security systems. This work showcased the strength and adaptability of Deep Learning in computer vision applications and demonstrated the potential of this technology in various industries.

6. SCOPE FOR FUTURE ENHANCEMENT

The following section describes the work that will be implemented with future scope of the project:

- Real-time facial expression recognition: Facial landmark detection can be used to recognize different facial expressions, such as happiness, sadness, and anger, in real-time. This can have applications in various fields, including virtual reality, gaming, and mental health diagnosis.
- Augmented reality: Facial landmark detection can be used to overlay virtual objects onto a user's face in real-time. This can be used for various applications such as makeup and beauty filters, virtual try-on, and advertising.
- Medical diagnosis: Facial landmark detection can be used for medical diagnosis, such as detecting facial asymmetry caused by nerve or muscle disorders, detecting changes in facial features caused by aging or disease, and detecting symptoms of genetic disorders.
- Biometric authentication: Facial landmark detection can be used for biometric authentication, such as face recognition for secure access control and identification.
- Human-robot interaction: Facial landmark detection can be used to enable robots to recognize and respond to human emotions and expressions, improving their ability to interact with humans.
- Surveillance and security: Facial landmark detection can be used for surveillance and security applications, such as monitoring for suspicious behavior, tracking suspects, and identifying criminals.
- Virtual assistants and chatbots: Facial landmark detection can be used to enable virtual assistants and chatbots to recognize and respond to human emotions and expressions, improving their ability to interact with humans.

7. REFERENCES

1. Colaco, S. J., & Han, D. S. (2022). Deep Learning-Based Facial Landmarks Localization Using Compound Scaling. *IEEE Access*, *10*, 7653-7663.
2. Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE signal processing letters*, *23*(10), 1499-1503.
3. H. Jiang and E. Learned-Miller, ‘‘Face detection with the faster R-CNN,’’ in Proc. 12th IEEE Int. Conf. Autom. Face Gesture Recognit. (FG), May 2017, pp. 650–657.
4. Kazemi, V., & Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1867-1874).
5. Z. Zhang, P. Luo, C. C. Loy, and X. Tang, ‘‘Facial landmark detection by deep multi-task learning,’’ in Proc. ECCV, 2014, pp. 94–108.
6. G. Trigeorgis, P. Snape, M. A. Nicolaou, E. Antonakos, and S. Zafeiriou, ‘‘Mnemonic descent method: A recurrent process applied for end-to-end face alignment,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 4177–4187.
7. J. Lv, X. Shao, J. Xing, C. Cheng, and X. Zhou, ‘‘A deep regression architecture with two-stage re-initialization for high performance facial landmark detection,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 3691–3700.
8. H. Yang, W. Mou, Y. Zhang, I. Patras, H. Gunes, and P. Robinson, ‘‘Face alignment assisted by head pose estimation,’’ 2015, arXiv:1507.03148.
9. A. Jourabloo and X. Liu, ‘‘Pose-invariant 3D face alignment,’’ in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 3694–3702.
10. A. Jourabloo, M. Ye, X. Liu, and L. Ren, ‘‘Pose-invariant face alignment with a single CNN,’’ in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 3219–3228.
11. X. Zhu, Z. Lei, X. Liu, H. Shi, and S. Z. Li, ‘‘Face alignment across large poses: A 3D solution,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 146–155.
12. A. Kumar and R. Chellappa, ‘‘Disentangling 3D pose in a dendritic CNN for unconstrained 2D face alignment,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 430–439.

13. S. Honari, P. Molchanov, S. Tyree, P. Vincent, C. Pal, and J. Kautz, ‘‘Improving landmark localization with semi-supervised learning,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 1546–1555.
14. X. Dong, Y. Yan, W. Ouyang, and Y. Yang, ‘‘Style aggregated network for facial landmark detection,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 379–388.
15. W. Wu, C. Qian, S. Yang, Q. Wang, Y. Cai, and Q. Zhou, ‘‘Look at boundary: A boundary-aware face alignment algorithm,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 2129–2138.
16. H. Fan and E. Zhou, ‘‘Approaching human level facial landmark localization by Deep Learning,’’ Image Vis. Comput., vol. 47, pp. 27–35, Mar. 2016.
17. R. Weng, J. Lu, and Y.-P. Tan, ‘‘Robust point set matching for partial face recognition,’’ IEEE Trans. Image Process., vol. 25, no. 3, pp. 1163–1176, Mar. 2016.
18. R. Ranjan, V. M. Patel, and R. Chellappa, ‘‘HyperFace: A deep multitask learning framework for face detection, landmark localization, pose estimation, and gender recognition,’’ IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 1, pp. 121–135, Jan. 2019.
19. S. Xiao, J. Feng, L. Liu, X. Nie, W. Wang, S. Yan, and A. Kassim, ‘‘Recurrent 3D-2D dual learning for large-pose facial landmark detection,’’ in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 1642–1651.
20. H. Lai, S. Xiao, Y. Pan, Z. Cui, J. Feng, and C. Xu, ‘‘Deep recurrent regression for facial landmark detection,’’ IEEE Trans. Circuits Syst. Video Technol., vol. 28, no. 5, pp. 1144–1157, May 2016.
21. J. Zhang and H. Hu, ‘‘Exemplar-based cascaded stacked auto-encoder networks for robust face alignment,’’ Comput. Vis. Image Understand., vol. 171, pp. 95–103, Jun. 2018.
22. J. Xia, L. Cao, G. Zhang, and J. Liao, ‘‘Head pose estimation in the wild assisted by facial landmarks based on Convolutional Neural Networks,’’ IEEE Access, vol. 7, pp. 48470–48483, 2019.
23. H.-W. Kim, H.-J. Kim, S. Rho, and E. Hwang, ‘‘Augmented EMTCNN: A fast and accurate facial landmark detection network,’’ Appl. Sci., vol. 10, no. 7, p. 2253, Mar. 2020.
24. S. Zheng, X. Bai, L. Ye, and Z. Fang, ‘‘HafaNet: An efficient coarse-to-fine facial landmark detection network,’’ IEEE Access, vol. 8, pp. 123037–123043, 2020.
25. J. Zhang, H. Hu, and S. Feng, ‘‘Robust facial landmark detection via heatmap-offset regression,’’ IEEE Trans. Image Process., vol. 29, pp. 5050–5064, 2020.

8. ANNEXURE

8.1 SAMPLE CODING

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split
from google.colab import drive
drive.mount('/content/drive/')
data_path = '/content/drive/My Drive/Colab
Notebooks/AutoEncoders/Advanced_ML_HSE/Face_Keypoint_Kaggle'
# check the training file:
train_read = pd.read_csv(data_path + '/training.csv', sep=',')
print ('training data shape; ', train_read.shape)
train_read.head(3).T
fig, axes = plt.subplots(5, 6, figsize=(15, 9))
ax = axes.ravel()
for i in range(30):
  ax[i].hist(train_read[train_read.columns[i]], bins=50, density=True, alpha=0.7, color='magenta')
  ax[i].set_title(train_read.columns[i],fontsize=8)
  # ax[i].axes.get_xaxis().set_visible(False)
plt.tight_layout()
plt.savefig(data_path + 'points_dist.png', dpi=200)
train_clean = train_read.dropna(axis=0, how='any', inplace=False)
train_clean = train_clean.reset_index(drop=True)
# Most of the columns have null values
# we will impute them by 'pad/ffill' method
# as we expect the missing value of facial kaypoints to be somewhere close to the others
# train_read.fillna(method='pad', inplace=True)
```

```
# print ('nan in every cols after imputing: ', train_read.isna().sum())
print ('data-frame shape with no null values: ', train_clean.shape)
print ('data-frame columnn names; ', train_clean.columns)
# train_read[['Image']].head(2)
all_imgs = []
print (train_read[['Image']].shape)
for i in range(0, 7049):
  x = train_read['Image'][i].split(' ') # split the pixel values based on the space
  x = [y for y in x] # create the listed pixels
  all_imgs.append(x)
all_imgs_arr = np.array(all_imgs, dtype='float') # arrays are always better than lists :)
# train_clean[['Image']].head(2)
clean_imgs = []
# print (train_clean[['Image']].shape)
for i in range(0, len(train_clean)):
x_c = train_clean['Image'][i].split(' ') # split the pixel values based on the space
x_c = [y for y in x_c] # create the listed pixels
clean_imgs.append(x_c)
clean_imgs_arr = np.array(clean_imgs, dtype='float') # arrays are always better than lists
clean_imgs_arr = np.reshape(clean_imgs_arr, (train_clean.shape[0], 96, 96, 1))
train_ims_clean = clean_imgs_arr/255.
# plt.imshow(all_imgs_arr[10].reshape(96, 96), cmap='gray')
clean_keypoints_df = train_clean.drop('Image', axis=1)
print ('check shape after dropping Image col in clean df: ', clean_keypoints_df.shape)
clean_keypoints_arr = clean_keypoints_df.to_numpy()
print ('check shape of clean key points arr: ', clean_keypoints_arr.shape)
# normalize the points so that training is easier
# we didn't use it later
# print ('check few values: ', keypoints_arr[0], keypoints_arr[1])
def standardize_keypoint(key points):
y_points = (key points - 48.)/48.
  print ('check key points max and min: ', np.max(y_points), np.min(y_points))
  return y_points
# train_points_arr = standardize_keypoint(keypoints_arr)
```

```
# train_points_arr = standardize_keypoint(clean_keypoints_arr)
# print ('\n')
# print ('check few vals after standardize: ', train_points_arr[0], train_points_arr[1])
# fig, ax = plt.subplots(figsize=(5, 4))
def vis_im_keypoint(img, points, axs):
  # fig = plt.figure(figsize=(6, 4))
  axs.imshow(img.reshape(96, 96))
  # points should be in the standardized
  xcoords = 48* (points[0::2] + 1.)
  ycoords = 48* (points[1::2] + 1.)
  axs.scatter(xcoords, ycoords, color='red', marker='o')
# vis_im_keypoint(train_ims[5], train_points_arr[5], ax)
def vis_im_keypoint_notstandard(img, points, axs): # same function as before but deals with key points when they are not standardized
  # fig = plt.figure(figsize=(6, 4))
  axs.imshow(img.reshape(96, 96))
  xcoords = (points[0::2] + 0.)
  ycoords = (points[1::2] + 0.)
  axs.scatter(xcoords, ycoords, color='red', marker='o')
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dropout, \
    Flatten, BatchNormalization, Dense, Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.activations import elu, relu
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
# from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2
input_im = Input(shape=(96, 96, 1))
def model():
layer 1:
  conv1 = Conv2D(32, (3, 3), activation='relu', )(input_im) #96 x 96 x 32
  conv2 = Conv2D(32, (3, 3), activation='relu', )(conv1) #96 x 96 x 32
  pool1 = MaxPooling2D((2, 2))(conv2)
  conv3 = Conv2D(64, (3, 3), activation='relu', )(pool1) #48 x 16 x 64
```

```
    conv4 = Conv2D(64, (3, 3), activation='relu', )(conv3)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv4)
    conv5 = Conv2D(128, (3, 3), padding='same', activation='relu',)(pool2)
    conv6 = Conv2D(128, (3, 3), padding='same', activation='relu',)(conv5)
    conv7 = Conv2D(128, (3, 3), padding='same', activation='relu',)(conv6)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv7)
    conv8 = Conv2D(256, (3, 3), padding='same', activation='relu',)(pool3)
    conv9 = Conv2D(256, (3, 3), padding='same', activation='relu',)(conv8)
    conv10 = Conv2D(256, (3, 3), padding='same', activation='relu',)(conv9)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv10)
    flat = Flatten()(pool4)
    den1 = Dense(128, activation='relu')(flat)
    den1 = Dropout(0.20)(den1)
    den2 = Dense(64, activation='relu')(den1)
    den2 = Dropout(0.20)(den2)
    pred = Dense(clean_keypoints_arr.shape[1])(den2)
    model = Model(inputs=input_im, outputs=pred, name='VGG_Like')
    return model
 face_key_model = model()
 face_key_model.summary()
class customCallbacks(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    self.epoch = epoch + 1
    if self.epoch % 50 == 0:
      print ('epoch num {}, train acc: {}, validation acc: {}'.format(epoch, logs['mae'],
logs['val_mae']))
learning_rate = 1e-3
def lrdecay(epoch):
  lr = learning_rate
  if epoch > 1600:
    lr *= 1e-1
  elif epoch > 800:
    lr *= 3e-1
  elif epoch > 400:
```

```
    lr *= 5e-1
  elif epoch > 200:
lr *= 7e-1
  elif epoch > 100:
    lr *= 9e-1
  if epoch % 50 == 0:
    print('Learning rate: ', lr)
  return lr
def lrexpdecay(epoch):
  decay = 0.1
  lr = learning_rate*(np.exp(-decay*epoch))
  return lr
def earlystop(mode):
  if mode=='acc':
    estop = tf.keras.callbacks.EarlyStopping(monitor='val_acc', patience=20, mode='max')
  elif mode=='loss':
    estop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=20, mode='min')
  return estop
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_mae', factor=0.8,
                          patience=25, min_lr=1e-5, verbose=1)
  # if epoch < 40:
  #   return 0.01
  # else:
  #   return 0.01 * np.math.exp(0.03 * (40 - epoch))
lrdecay = tf.keras.callbacks.LearningRateScheduler(lrdecay) # learning rate decay
sgd = SGD(lr=learning_rate, momentum = 0.9,nesterov=True)
adam = Adam(learning_rate=learning_rate)
```

Include Augmentation

1. Horizontal Flip.
2. Linear Contrast.
3. Gaussian Blur.
4. Rotation.
5. Scaling.

```
def flip_im_points1(img, points):
  flip_im = np.fliplr(img)
  xcoords = points[0::2]
  ycoords = points[1::2]
  new_points = []
  for i in range(len(xcoords)):
    xp = xcoords[i]
    yp = ycoords[i]
    new_points.append(xp*(-1))
    new_points.append(yp)
  return flip_im, np.asarray(new_points)
def flip_im_points0(img, points): # flip key points when they are not standardized
  flip_im = np.fliplr(img)
  xcoords = points[0::2]
  ycoords = points[1::2]
  new_points = []
  for i in range(len(xcoords)):
    xp = xcoords[i]
    yp = ycoords[i]
    new_points.append(96-xp)
    new_points.append(yp)
  return flip_im, np.asarray(new_points)
import imgaug as ia
import imgaug.augmenters as iaa
def gnoise_lincontrast(im_tr, pt_tr):
  seq = iaa.Sequential([iaa.LinearContrast((0.6, 1.5)),
                        iaa.Sometimes(
        0.80, iaa.GaussianBlur(sigma=(0., 2.0)))])
  aug_ims = []
  aug_pts = []
  for im, pt in zip(im_tr, pt_tr):
    #f_im, f_pts = flip_im_points1(im, pt)
    f_im = seq(image=im)
    aug_ims.append(im)
```

```
    aug_ims.append(f_im)
    aug_pts.append(pt)
    aug_pts.append(pt)
  return np.asarray(aug_ims), np.asarray(aug_pts)
aug_ims_train_clean_g, aug_points_train_clean_g = gnoise_lincontrast(train_ims_clean,
clean_keypoints_arr)
print (type(aug_ims_train_clean_g), aug_ims_train_clean_g.shape,
aug_points_train_clean_g.shape)
fig = plt.figure(figsize=(12, 10))
npics= 16
count = 1
for i in range(npics):
  ipic = i # use this to see original and augmented image side by side
#   ipic = np.random.choice(aug_ims_train_clean.shape[0])
  ax = fig.add_subplot(npics/4 , 4, count, xticks=[],yticks=[])
  vis_im_keypoint_notstandard(aug_ims_train_clean_g[ipic], aug_points_train_clean_g[ipic], ax)
  count = count + 1
# plt.title('Gaussian Blur and Linear Contrast')
plt.tight_layout()
plt.savefig(data_path+'/aug_ims_kps_gauss_cont.png', dpi=200)
plt.show()
# include rotation augmentation

from imgaug.augmentables import Keypoint, KeypointsOnImage
def rotate_aug(im_tr, pt_tr):
  seq = iaa.Sequential([iaa.Affine(rotate=15, scale=(0.8, 1.2))])
  #image_aug, kps_aug = seq(image=image, key points=kps)
  aug_ims = []
  aug_pts = []
  coordlist = []
  for im, pt in zip(im_tr, pt_tr):
    #f_im, f_pts = flip_im_points1(im, pt)
    xcoord = pt[0::2]
    ycoord = pt[1::2]
```

```
    for i in range(len(xcoord)):
      coordlist.append(Keypoint(xcoord[i], ycoord[i]))
    kps = KeypointsOnImage(coordlist, shape=im.shape)
    f_im, f_kp = seq(image=im, key points=kps)
    #new_xcoords = []
    #new_ycoords = []
    all_coords = []
    for k in range(len(kps.keypoints)):
      before = kps.keypoints[k]
      after = f_kp.keypoints[k]
      # print("Keypoint %d: (%.8f, %.8f) -> (%.8f, %.8f)" % (
      #     i, before.x, before.y, after.x, after.y)  # )
      all_coords.append(after.x)
      all_coords.append(after.y)
      all_coords_arr = np.asarray(all_coords)
    aug_ims.append(im)
    aug_ims.append(f_im)
    aug_pts.append(pt)
    aug_pts.append(all_coords)
    coordlist.clear()
  return np.asarray(aug_ims), np.asarray(aug_pts)
aug_ims_train_clean_g2, aug_points_train_clean_g2 = rotate_aug(aug_ims_train_clean_g,
aug_points_train_clean_g)
print (type(aug_ims_train_clean_g2), aug_ims_train_clean_g2.shape,
aug_points_train_clean_g2.shape)
fig = plt.figure(figsize=(12, 9))
npics= 20
count = 1
for i in range(npics):
  ipic = i # use this to see original and augmented image side by side
#   ipic = np.random.choice(aug_ims_train_clean.shape[0])
  ax = fig.add_subplot(npics/4 , 5, count, xticks=[],yticks=[])
  vis_im_keypoint_notstandard(aug_ims_train_clean_g2[ipic], aug_points_train_clean_g2[ipic],
ax)
```

```
  count = count + 1
plt.margins(0,0)
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.tight_layout()
plt.tight_layout()
plt.savefig(data_path+'/aug_ims_kps_rot_scale.png', dpi=200, bbox_inches = 'tight', pad_inches =
0)
plt.show()
### add the flipped images in the training data-set
def aug_flip(im_tr, pt_tr):
  aug_ims = []
  aug_pts = []
  for im, pt in zip(im_tr, pt_tr):
    f_im, f_pts = flip_im_points1(im, pt)
    aug_ims.append(im)
    aug_ims.append(f_im)
    aug_pts.append(pt)
    aug_pts.append(f_pts)
  return np.asarray(aug_ims), np.asarray(aug_pts)
def aug_flip0(im_tr, pt_tr):
  aug_ims = []
  aug_pts = []
  for im, pt in zip(im_tr, pt_tr):
    f_im, f_pts = flip_im_points0(im, pt)
    aug_ims.append(im)
    aug_ims.append(f_im)
    aug_pts.append(pt)
    aug_pts.append(f_pts)
  return np.asarray(aug_ims), np.asarray(aug_pts)
aug_ims_train_clean_g3, aug_points_train_clean_g3 = aug_flip0(aug_ims_train_clean_g2,
                                                aug_points_train_clean_g2)
print ('size of training data now: ', aug_ims_train_clean_g3.shape,
aug_points_train_clean_g3.shape)
```

```
fig = plt.figure(figsize=(13, 10))
npics= 24
count = 1
for i in range(npics):
  ipic = i # use this to see original and augmented image side by side
#   ipic = np.random.choice(aug_ims_train_clean.shape[0])
  ax = fig.add_subplot(npics/4 , 6, count, xticks=[],yticks=[])
  vis_im_keypoint_notstandard(aug_ims_train_clean_g3[ipic], aug_points_train_clean_g3[ipic],
ax)
  count = count + 1
plt.margins(0,0)
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.tight_layout()
plt.savefig(data_path+'/aug_ims_kps_flip_h.png', bbox_inches = 'tight', pad_inches = 0, dpi=200)
plt.show()
fig = plt.figure(figsize=(10, 9))
npics= 24
count = 1
for i in range(npics):
  #ipic = i # use this to see original and augmented image side by side
  ipic = np.random.choice(aug_ims_train_clean_g3.shape[0])
  ax = fig.add_subplot(npics/4 , 6, count, xticks=[],yticks=[])
  vis_im_keypoint_notstandard(aug_ims_train_clean_g3[ipic], aug_points_train_clean_g3[ipic],
ax)
  count = count + 1
plt.tight_layout()
plt.show()
fig = plt.figure(figsize=(13, 10))
npics= 24
count = 1
for i in range(npics):
  ipic = i # use this to see original and augmented image side by side
#   ipic = np.random.choice(aug_ims_train_clean.shape[0])
```

```
  ax = fig.add_subplot(npics/4 , 6, count, xticks=[],yticks=[])
  vis_im_keypoint_notstandard(aug_ims_train_clean_g3[ipic], aug_points_train_clean_g3[ipic],
ax)
  count = count + 1
plt.margins(0,0)
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.tight_layout()
plt.savefig(data_path+'/aug_ims_kps_flip_h.png', bbox_inches = 'tight', pad_inches = 0, dpi=200)
plt.show()
fig = plt.figure(figsize=(10, 9))
npics= 24
count = 1
for i in range(npics):
  #ipic = i # use this to see original and augmented image side by side
  ipic = np.random.choice(aug_ims_train_clean_g3.shape[0])
  ax = fig.add_subplot(npics/4 , 6, count, xticks=[],yticks=[])
  vis_im_keypoint_notstandard(aug_ims_train_clean_g3[ipic], aug_points_train_clean_g3[ipic],
ax)
  count = count + 1
plt.tight_layout()
plt.show()
ipic = np.random.choice(train_ims.shape[0])
 check_flip_im1, check_flip_points1 = flip_im_points1(train_ims[ipic], train_points_arr[ipic])
 fig, ax = plt.subplots(figsize=(4, 4))
 vis_im_keypoint(train_ims[ipic], train_points_arr[ipic], ax)
 plt.title('Before Horizontal Flip')
 fig, ax1 = plt.subplots(figsize=(4, 4))
 vis_im_keypoint(check_flip_im1, check_flip_points1, ax1)
 plt.title('After Horizontal Flip')
 plt.show()
from sklearn.model_selection import train_test_split
 imgs_train, imgs_val, points_train, points_val = train_test_split(train_ims, train_points_arr,
                                                  test_size=0.15, random_state=21)
```

```
imgs_train, imgs_val, points_train, points_val = train_test_split(train_ims, keypoints_arr,
                                                 test_size=0.15, random_state=21)
 print ('train image data size: ', imgs_train.shape)
 print ('train key points data size: ', points_train.shape)
 print ('validation image data size: ', imgs_val.shape)
 imgs_train_clean, imgs_val_clean, points_train_clean, points_val_clean =
train_test_split(train_ims_clean, clean_keypoints_arr, test_size=0.15, random_state=21)
 print ('train clean image data size: ', imgs_train_clean.shape)
 print ('train clean key points data size: ', points_train_clean.shape)
  conv1 = Conv2D(filter1, (1,1), padding='same', activation='relu')(input_layer)
  bn1 = BatchNormalization()(conv1)
  # 3x3 conv
  conv3 = Conv2D(filter2, (3,3), padding='same', activation='relu')(input_layer)
  bn3 = BatchNormalization()(conv3)
  # 5x5 conv
  conv5 = Conv2D(filter3, (5,5), padding='same', activation='relu')(input_layer)
  bn5 = BatchNormalization()(conv5)
  # 3x3 max pooling
#   pool = MaxPooling2D((3,3), strides=(1,1), padding='same')(input_layer)
  pool = MaxPooling2D((2,2), strides=(1,1), padding='same')(input_layer)
  # concatenate filters, assumes filters/channels last
  layer_out = Concatenate(axis=-1)([bn3, bn5, pool])
  return layer_out
input_im = Input(shape=(96, 96, 1))
def model2():
#   x = Conv2D(64, (3, 3), padding='same', strides=(2, 2), activation='relu', )(input_im)
#   x = MaxPooling2D((3, 3), padding='same', strides=(2, 2), )(x)
#   x = Conv2D(64, (1, 1), padding='same', strides=(1, 1), activation='relu', )(x)
# #   x =  Conv2D(64, (3, 3), padding='same', strides=(1, 1), activation='relu', )(x)
#   x = Conv2D(96, (3, 3), padding='same', strides=(1, 1), activation='relu', )(x)
#   x = MaxPooling2D((3, 3), padding='same', strides=(2, 2) )(x)
#   x = Conv2D(16, (3, 3), padding='same', activation='relu', )(input_im)
#   x = Conv2D(32, (3, 3), padding='same', activation='relu', )(input_im)
#   x = Conv2D(64, (3, 3), padding='same', activation='relu', )(x)
```

```
  x1 = inception_like(input_im, 64, 64, 32)
  x1 = MaxPooling2D((3, 3), padding='same', strides=(2, 2) )(x1)
  x2 = inception_like(x1, 64, 64, 32)
  x2 = MaxPooling2D((3, 3), padding='same', strides=(2, 2) )(x2)
  x2_1 = inception_like(x2, 96, 96, 64)
  x2_1 = MaxPooling2D((3, 3), padding='same', strides=(2, 2) )(x2_1)
  x3 = inception_like(x2_1, 96, 128, 64)
  #x3 = MaxPooling2D((3, 3), padding='same', strides=(2, 2) )(x3)
  x3 = MaxPooling2D()(x3)
  x3_1 = inception_like(x3, 128, 256, 128)
  #x3_1 = MaxPooling2D((3, 3), padding='same', strides=(2, 2) )(x3_1)
  x3_1 = GlobalAveragePooling2D()(x3_1)  x4 = Flatten()(x3_1)
  x4 = Dense(1024, kernel_regularizer=l2(l2=0.03))(x4)
  x4 = Dropout(0.2)(x4)
  #x5 = Dense(128, kernel_regularizer=l2(l2=0.02))(x4)
  #x5 = Dropout(0.1)(x5)
  pred = Dense(30)(x4)
  model = Model(inputs=input_im, outputs=pred, name='Inception_Like')
  return model
face_key_model2_aug = model2()
face_key_model2_aug.summary()
face_key_model2_aug.compile(loss='mse',
                optimizer=Adam(learning_rate=3e-3),
                metrics=['mae'])
face_key_model2_aug_train_clean = face_key_model2_aug.fit(aug_ims_train_final,
aug_points_train_final,
                                validation_split= 0.05,
                                batch_size=64, epochs=300,
                                callbacks=[customCallbacks(), reduce_lr],
                            verbose=0)
mae = face_key_model2_aug_train_clean.history['mae']
# mae = [i for i in mae if i<60]
print (type(mae))
val_mae = face_key_model2_aug_train_clean.history['val_mae']
```

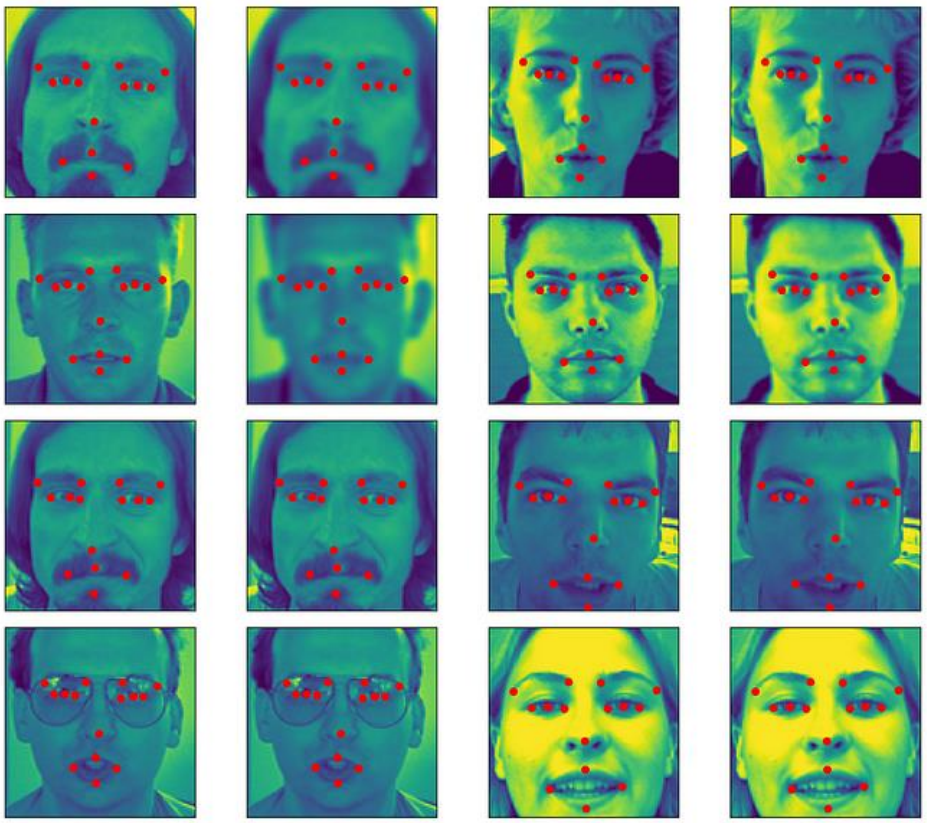
```
# val_mae = [i for i in val_mae if i<60]
loss = face_key_model2_aug_train_clean.history['loss']
# loss = [i for i in loss if i<1200]
val_loss = face_key_model2_aug_train_clean.history['val_loss']
# val_loss = [i for i in val_loss if i<1200]
fig = plt.figure(figsize=(8, 4))
fig.add_subplot(121)
plt.plot(range(len(loss)), loss, linestyle='-', color='red', alpha=0.7, label='Train Loss')
plt.plot(range(len(val_loss)), val_loss, linestyle='-.', color='navy', alpha=0.7, label='Val Loss')
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Loss', fontsize=13)
plt.yscale('log')
plt.legend(fontsize=12)
fig.add_subplot(122)
plt.plot(range(len(mae)), mae, linestyle='-', color='red', alpha=0.7, label='Train MAE')
plt.plot(range(len(val_mae)), val_mae, linestyle='-.', color='navy', alpha=0.7, label='Val MAE')
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('MAE', fontsize=13)
plt.yscale('log')
plt.legend(fontsize=12)
plt.tight_layout()
plt.savefig(data_path+'train_loss_mae.png', dpi=150)
plt.show()
predict_points_aug2_clean = face_key_model2_aug.predict(test_ims)
print ('check shape of predicted points: ', predict_points_aug2_clean.shape)
fig = plt.figure(figsize=(10, 8))
npics= 12
count = 1
for i in range(npics):
  # ipic = i
  ipic = np.random.choice(test_ims.shape[0])
  ax = fig.add_subplot(npics/3 , 4, count, xticks=[],yticks=[])
  vis_im_keypoint_notstandard(test_ims[ipic], predict_points_aug2_clean[ipic], ax)
  count = count + 1
```

```
plt.tight_layout()
plt.savefig(data_path+'/prediction_keypoints.png', dpi=200, bbox_inches = 'tight', pad_inches = 0)
plt.show()
```

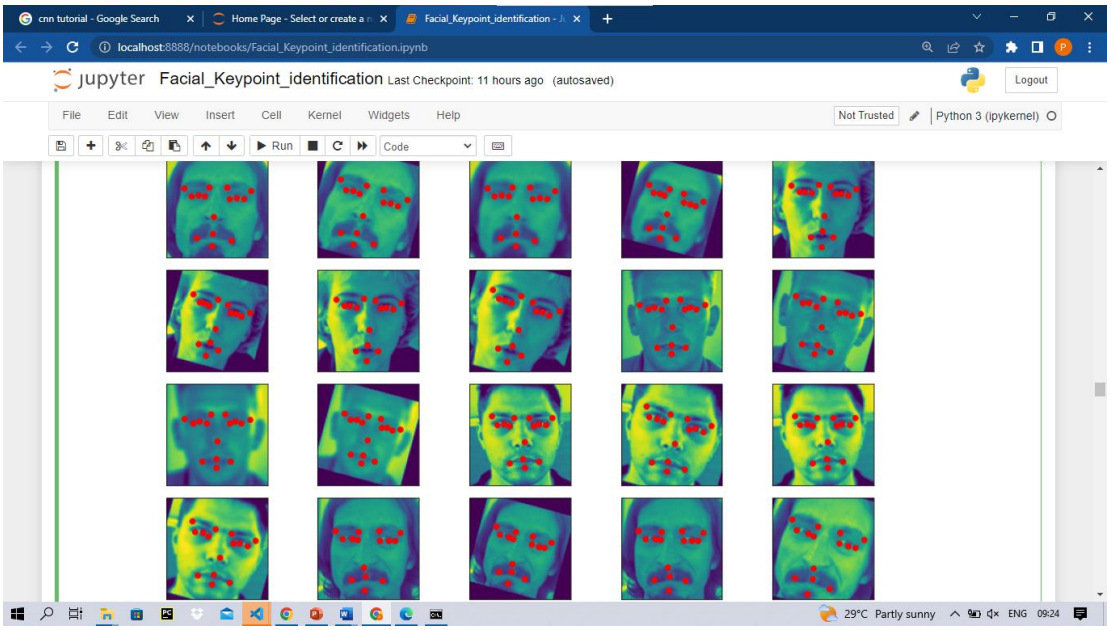
Run Command

```
C:\Users\elcot>D:
D:\>dir
Directory of D:\
D:\>cd facial-keypoints-identification
D:\facial-keypoints-identification>python -m venv facial-keypoints-identification
D:\facial-keypoints-identification>.\facial-keypoints-identification\Scripts\activate
(facial-keypoints-identification) D:\facial-keypoints-identification>jupyter notebook
```

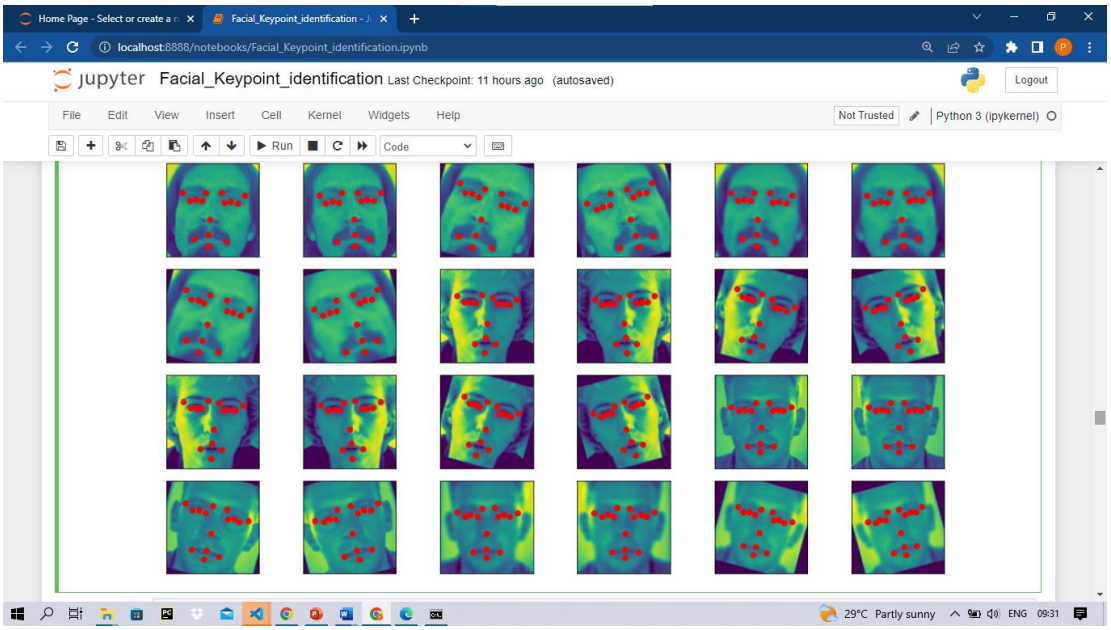
8.2 OUTPUT



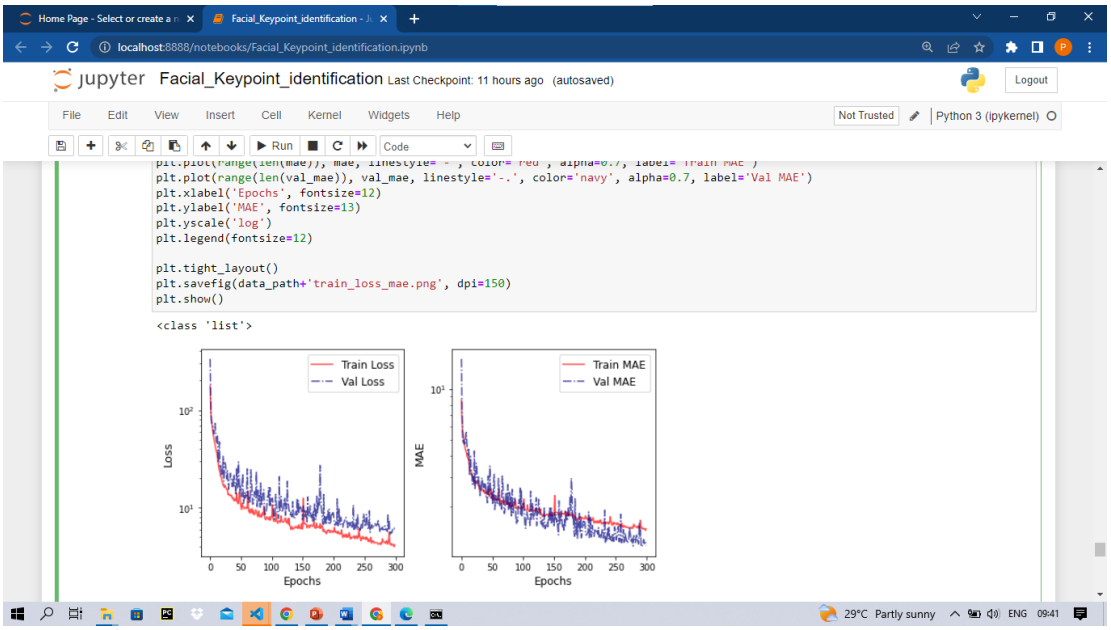
After rotation augmentation:



After flipping augmentation:



Loss function and mean absolute error graph representation for training and validation:



Facial Landmark Identification in all types of images: