FPT UNIVERSITY

Capstone Project Document

Deep Learning in Age-Invariant Face Recognition

Deep Learning in Age-Invariant Face Recognition	
Group Members	Le Quang Minh – SE151488 Ton Nu Quyen Mi – SE150215 Nguyen Lam Nguyen – SE151345
Supervisor	Nguyen Quoc Trung
Capstone Project code	SP23AI10

– Ho Chi Minh, April, 2023 –

ACKNOWLEDGEMENT

During my time studying and researching to complete the thesis, we received guidance and support from teachers, friends, and colleagues.

We wish to express my special thanks to Ph.D. Nguyen Quoc Trung, lecturer in the Department of Artificial Intelligence for sharing expertise, and giving valuable guidance and encouragement extended to me.

We are also grateful to FPT University, for providing me with all the necessary facilities for the research.

We take this opportunity to express gratitude to all of the Department faculty members for their help and support.

We also place on record my sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture.

AUTHOR CONTRIBUTIONS

Conceptualization, Quang Minh, Quyen Mi, and Lam Nguyen; methodology, Quang Minh; software, Quyen Mi, Quang Minh and Lam Nguyen; validation, Lam Nguyen; formal analysis, Quang Minh, Quyen Mi, and Lam Nguyen; investigation, Quyen Mi; resources, Quang Minh and Lam Nguyen; data curation, Lam Nguyen; writing - original draft preparation, Quyen Mi; writing - review and editing, Quang Minh and Lam Nguyen; visualization, Quyen Mi; supervision, Quang Minh; project administration, Quang Minh. All authors have read and agreed to the Final Capstone Project document.

ABTRACT

Face recognition across aging has grown into an extremely prominent and challenging job in the field of facial recognition in recent times. The ability to find missing children and identify them even after they have grown up is one of the uses for age-invariant facial recognition. Additionally, it can be used to check a watch list and look up suspects who have been missing for a long time. Human faces can change dramatically over time in a variety of ways, including facial texture (such as wrinkles), shape (such as weight increase), facial hair, the use of glasses, etc. This makes the work at hand complex. Additionally, the surroundings and image acquisition settings may alter, which may result in scale and uniform lighting changes. Therefore, this study develops an age-invariant facial recognition system using deep learning. Age-invariant face recognition is basically divided into two categories: Age-Invariant Face Recognition (AIFR) and Age-Invariant Face Verification (AIFV). We will only consider the first type in this work. The model uses convolutional neural networks (CNNs) and Transfer Learning techniques to achieve more optimal face recognition synthesis based on facial feature extraction. We then apply Custom Structure Preservation (CUSP) and self-supervised learning techniques to improve the model and explore the effectiveness of self-supervised learning (SSL), specifically the Bootstrap Your Own Latent (BYOL) technique, to improve the age-invariant face recognition model. The results obtained for the training model on the FGNET dataset are 59.3% on the test set and 75.3% on the validation set. And the CACD dataset obtained 77.9% on the test set and 76.7% on the validation set.

Keywords: Aging model; Self-Supervised Learning; Age-Invariant; Deep learning; Face recognition; ...

CONTENTS

List of figures

List of Tables

List of Abbreviation

Abbreviation	Full word
AIFR	Age-Invariant Face Recognition
AIM	Age-Invariant Model
BYOL	Bootstrap Your Own Latent
CACD	Cross-Age Celebrity Dataset
CNNs	Convolutional neural networks
CUSP	Custom Structure Preservation
GAN	Generative adversarial networks
LPS	Local Pattern Selection
SSL	Self-Supervised Learning

1. INTRODUCTION

The following section aims to present a comprehensive overview of age-invariant face recognition achieved through computer vision methodologies. The initial segment of this exposition seeks to introduce the underlying rationale behind the selection of this topic, while simultaneously highlighting the critical importance of age-invariant face recognition. Subsequently, the section endeavors to shed light on the challenges posed by this field of study, followed by a succinct summary of the current state of research. Finally, the section culminates with a clear delineation of the objectives and contributions of this project.

1.1 Overview

1.1.1 Context

Face recognition involves the application of machine devices that are designed to automatically detect a face within an image or video and then ascertain the identity of the individual in question. This process is accomplished through a combination of complex algorithms and pattern recognition techniques that allow the system to compare the detected facial features with a database of known faces in order to establish a positive identification.

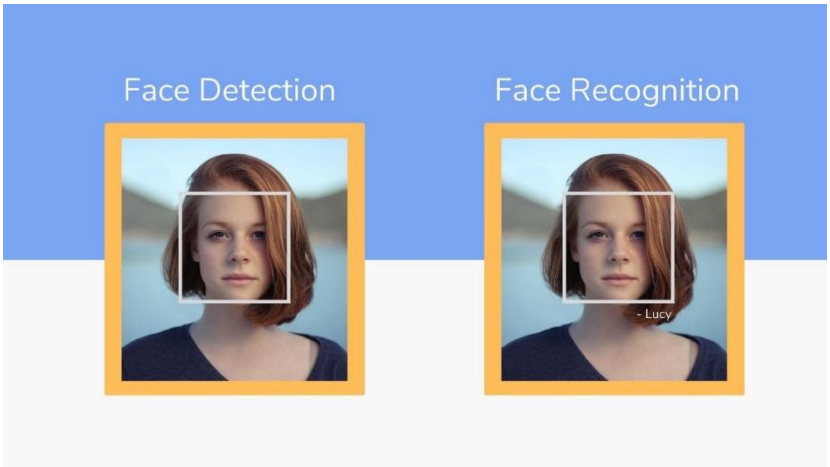


Figure 1. Difference between face detection and face recognition

Facial recognition technology has become increasingly prevalent and is now widely utilized in numerous applications beyond simple identity verification. Facial recognition technology has proven to be a valuable tool in various important fields, including law enforcement, criminal justice, and national security. For example, it is

commonly employed in security systems to monitor and track individuals in public spaces, such as airports and shopping centers. Facial recognition is also being utilized in marketing and advertising to analyze customer demographics and behavior, which can be used to optimize marketing strategies and improve customer engagement. Additionally, the technology is being explored for medical purposes, such as diagnosing certain health conditions through facial analysis. The potential applications of facial recognition technology are continually expanding, making it an area of significant interest and investment for researchers and businesses alike. Nevertheless, the process of accurately identifying an individual from their facial features is fraught with difficulties due to a variety of factors, including changes in photographic angle, expression, brightness, and aging. Of these factors, aging is particularly problematic, as it is a complex and multi-faceted process that involves a wide range of physiological changes that can make it extremely challenging to accurately identify individuals over time.

Aging is a highly variable and complex process that affects individuals differently. It is influenced by a multitude of factors, including geographic location, lifestyle choices, dietary habits, use of cosmetics, physical and emotional health, and many other variables. This complexity makes the development of effective age-invariant face recognition systems a challenging task. In order to develop accurate and reliable systems, researchers must take into account the diverse range of factors that contribute to the aging process and design algorithms that can effectively account for these factors. Only by considering this complexity can we hope to develop age-invariant face recognition systems that can accurately identify individuals across different ages and stages of life. These variations pose numerous challenges, but they are extremely efficient in real applications such as finding missing children, issuing passports, tracing criminals, etc. As a result, biometrics research has focused on facial identification, particularly age-invariant face recognition, in recent years.

Over time, the aging process causes significant changes to an individual's facial features, which can greatly affect the accuracy of facial recognition systems. These changes can include the loss of skin elasticity, changes in skin texture and tone, and alterations in the shape and position of facial features. As a result, the process of

identifying a person's identity through facial recognition becomes more challenging as they age. This is because the algorithms used in these systems rely heavily on identifying specific facial features that may no longer be present or have changed significantly. As a result, researchers face a significant challenge in developing age-invariant face recognition systems that can account for these changes and accurately identify individuals across different ages and stages of life.

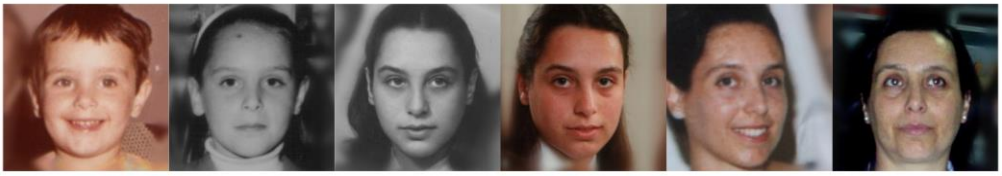


Figure 2. Images of a woman's faces through the ages

Figure 3. Images of people's faces from different ages

Therefore, it is necessary to deeply understand and improve the functions and techniques for automatic facial recognition according to age changes, in order to conquer the challenge that is the age - invariant face recognition.

1.1.2 Research overview

Human face detection has been a fundamental problem in computer vision for many years, and numerous approaches have been proposed to address it. These approaches range from template matching and neural networks to more sophisticated algorithms based on haar-like features. Among these methods, the use of haar-like features has proven to be particularly effective, and it is now widely considered the most reliable approach for detecting human faces in images and videos.

This method relies on analyzing the distribution of intensities in different regions of the image to identify facial features, such as eyes, nose, and mouth. By detecting these features and using them to define a set of facial characteristics, haar-like features can accurately identify and localize human faces even in complex and challenging environments. As a result, this method has become a cornerstone of many modern facial

recognition systems and is widely regarded as one of the most effective and reliable approaches for detecting human faces in a variety of different contexts.

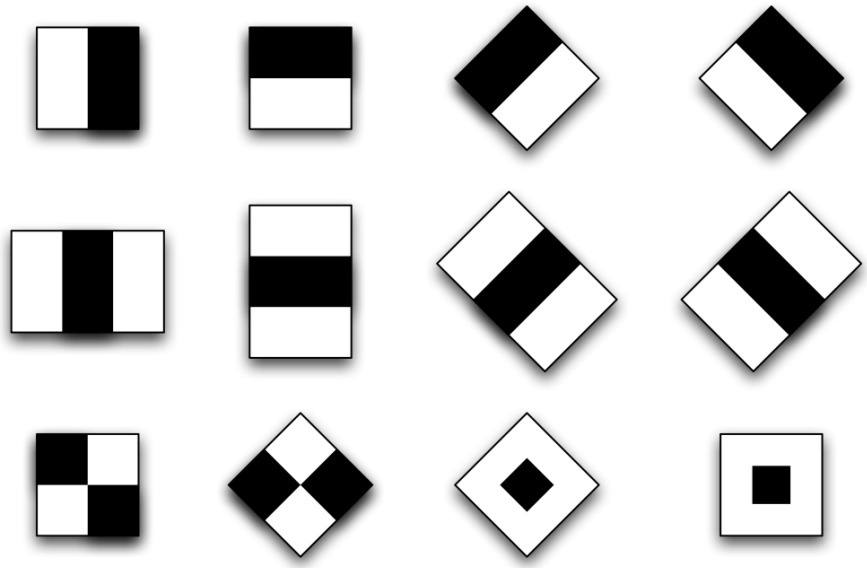


Figure 4. Haar-Like features

Figure 4 shows the Haar-Like feature. A Haar-like feature is represented by taking a rectangular part of an image and dividing that rectangle into multiple parts. They are often visualized as black-and-white adjacent rectangles.

In recent years, there has been significant progress in developing age-invariant face recognition systems, and researchers have explored a range of different approaches to this problem. Broadly speaking, these approaches can be divided into three main categories: generative methods, discriminative methods, and methods based on convolutional neural networks (CNNs).

Generative methods involve modeling the aging process explicitly and using this model to generate synthetic facial images that correspond to different ages. By comparing these synthetic images with real images, researchers can develop algorithms that can accurately recognize individuals across different ages. Discriminative methods, on the other hand, focus on identifying features that are discriminative across different ages and using these features to train age-invariant recognition models. Finally, CNNs have emerged as a particularly effective approach to age-invariant facial recognition, leveraging deep learning techniques to identify complex patterns in facial features that are invariant to age-related changes.

Overall, each of these approaches has its own strengths and weaknesses, and researchers continue to explore and refine each of them in pursuit of more accurate and reliable age-invariant facial recognition systems.

1.2 Capstone project goals and contributions

The objective of the project is to develop an age-invariant face recognition model that combines a convolutional neural network (CNNs) model with transfer learning techniques. To enhance the model's accuracy, a combination of Custom Structure Preservation (CUSP) image augmentation methods will be applied in the training set. Additionally, self-supervised learning techniques will be employed to further improve the model's performance.

The project aims to explore the possibility of enhancing the model's accuracy using self-supervised learning techniques and to propose new directions for developing age-invariant face recognition systems based on these techniques. By leveraging self-supervised learning, the model can learn to identify relevant facial features that are invariant to age-related changes, allowing for more accurate and reliable facial recognition across different ages and stages of life.

Through this project, we hope to contribute to the development of more robust and accurate age-invariant face recognition systems that can benefit a range of different applications, from law enforcement and national security to social media and e-commerce. By leveraging the latest advances in deep learning and computer vision, we believe that it is possible to overcome the challenges posed by age-related changes and develop truly age-invariant facial recognition models that can reliably identify individuals across different ages and stages of life.

Test results on two different data sets, FGNET and CACD, to see the recognition accuracy, especially on the most challenging aging dataset FGNET with only 1002 images divided into 82 classes. and unequal age distribution.

2. RELATED WORKS

This section elucidates the literature review conducted in the concerned field. In order to undertake the research, a thorough examination of numerous relevant works was carried out, with a particular emphasis on two principal areas: Age-invariant face recognition and self-supervised learning techniques applied in face recognition.

2.1 Age-Invariant Face recognition

Initially, in order to obtain a comprehensive understanding of the domain of age-invariant face recognition, we conducted an analysis of the research article entitled "Age Invariant Face Recognition Methods: A Review" Paper [1] provided us with extensive insight into the approaches utilized within this area. The article mentions many methods such as generative methods, discriminative, and using convolutional neural networks (CNN).

Generative methods (aging simulation) include age-invariant face detection methods based on facial geometry recognition [2] using Local Pattern Selection (LPS) [3] that a new feature descriptor used on two-level learning model. The works [4] propose a 2D/3D face aging pattern space that enables the generation of a facial image that matches a target face image prior to recognition, where [4] employs a 2D face aging model to simulate facial aging and achieve face across aging.

Discriminative approaches for facial recognition utilize facial components that are classified into two categories: age-invariant factors and aging factors. To identify these factors, a hidden factor analysis (HFA) model, described in [5] develops a linear model that separates the two factors into distinct subspaces, followed by the computation of cosine distance between the identity components of gallery and probe samples for facial recognition. However, assuming independence between identity factors and age in [5] is an oversimplification, as the extent of facial changes due to aging can vary considerably among individuals. To overcome this issue, [6] introduces an updated HFA model that accounts for correlated changing factors during facial recognition.

Convolutional neural networks (CNNs) have emerged as a highly popular technique for Computer Vision applications, with a notable emphasis on face recognition. In particular, CNNs have been found to be the most optimal technique for

face recognition models, including age-invariant face recognition. Researchers have employed various types of CNNs in their studies, resulting in promising outcomes.

To address the challenges posed by the nonlinear and smooth transformation of aging in face images, several neural network models have been proposed. For instance, the coupled auto-encoder network (CAN), which bridges two auto-encoders with two shallow neural networks, was introduced in [7] The LF-CNN (Latent Factor guided Convolutional Neural Network) model [8] employs a well-designed CNN model to acquire age-invariant deep features. The AE-CNN (Age Estimation-guided CNN) model [9] is specifically trained to separate age-related variations from identity-specific traits, while the OE-CNN (Orthogonal Embedding CNN) model [10] decomposes deep facial representations into two orthogonal components, one for age-specific features and the other for identity-specific features.

In addition to these models, recent studies have proposed novel approaches, such as the use of Stacked Autoencoder Deep Neural Networks (a type of unsupervised artificial neural network) [11]. In [12] proposes a new deep learning model called the Age-Invariant Model (AIM) for identifying faces in uncontrolled settings. AIM has three main innovations: it combines cross-age face synthesis and recognition to improve performance, it can modify the age of a face while maintaining its identity without requiring paired data or true age information, and it employs novel training strategies to explicitly separate age variation from face representation.

Our research also involves the application of convolutional neural networks, however, we have further enhanced the model with the most recent technique in the field - self-supervised learning.

2.2 Self-Supervised Learning in Face Recognition

Self-supervised learning [13] is a recently popularized keyword in the field of machine learning, and it has not been extensively studied in the context of age-invariant face recognition. Nevertheless, we widely recognized that self-supervised learning has significant potential to be applied in this field. Initially, we investigated the potential of self-supervised learning through some related works in the field of face recognition.

In research on pose invariant face recognition [14] the authors introduced the FRGAN, a self-supervised approach that generates frontal face images from non-frontal images by rotating them to their original pose through the use of reconstruction and adversarial losses. Additionally, the FRGAN utilized Random Swap, a data augmentation technique that enhances performance by swapping important facial regions between the input image and its reconstructed counterpart to generate more realistic synthesized images. Besides, in [15] the author used self-supervised learning to improve the embedded space of the face recognition model and maximize the similarity between the embeddings of each image and its copy in both the source and target domains. This was done to reduce the model's degradation from the source domain to the target domain.

3. PROJECT MANAGEMENT PLAN

In order for the project to run smoothly and complete the work on time, we have implemented project management as follows

Table 1. Project plan

Topic	Deep Learning in Age-Invariant Face Recognition
Schedule with student	8:00 PM on Friday
Project plan on week	
Week 1	Literature review, including approaches to the topic, relevant articles, and data for the project
Week 2	Choose Scope for Proposal and do preliminary test
Week 3	Data processing: Find relevant data sources and collect data
Week 4	Data processing: Standardize the data for training the models
Week 5	Code Transfer Learning and Hyperparameter Tuning for ML algorithm
Week 6	Code Transfer Learning and Hyperparameter Tuning for Resnet,Evaluate Resnet models on FGNet with generated images from SAM face aging model
Week 7	Complete code and hyperparameter tuning for BYOL, Run and collect the results of Transfer Learning Resnet model
Week 8	Train resnet models on datasets with generated images, Run and collect the results of Transfer Learning Resnet model
Week 9	Train and fine-tune model BYOL on FGNet dataset, Fine-tuning CUSP model to generate high preservation face aging images, Create a Web Demo
Week 10	Train and fine-tune model BYOL on CACD dataset, Train ResNet models on datasets with generated images
Week 11	Research and analyze existing results
Week 12	Write the project report, summarize the best results to write the report for the scientific conference

Week 13	Refine the report and create slides for the final project defense.
Week 14	Prepare the thesis defend

Table 2. Detailed project assignment plan

ID		**SE151488**	**SE150215**	**SE151345**
Name		**Le Quang Minh**	**Ton Nu Quyen Mi**	**Nguyen Lam Nguyen**
Division	Task 1	Machine Learning	Data Preprocessing	Face alignment on datasets
	Task 2	Train BYOL on FGNet Dataset	Train ResNet model on both datasets	Using generative model to generate images
	Task 3	Train BYOL on FGNet Dataset	Fine-tune model and build web demo	Test models' performance on generated images
Timeline	Week 1	Research on topic	Research on topic	Research on topic
	Week 2	Choose Scope for Proposal and do preliminary test	Choose Scope for Proposal and do preliminary test	Choose Scope for Proposal and do preliminary test
	Week 3	Research on CACD dataset	Research on FGNet dataset	Research on CASIA dataset
	Week 4	Research for coding	Data pre-processing	Research for coding
	Week 5	Code Transfer Learning and Hyperparameter Tuning for ML algorithm	Data processing	Face alignment

	Week 6	Run and collect the results of ML model	Code Transfer Learning and Hyperparameter Tuning for Resnet	Evaluate Resnet models on FGNet with generated images from SAM face aging model
	Week 7	Complete code and hyperparameter tuning for BYOL	Run and collect the results of Transfer Learning Resnet model on FGNet dataset	Using CUSP to generate face images on FGNet and CACD dataset
	Week 8	Train and fine-tune model BYOL on FGNet dataset	Run and collect the results of Transfer Learning Resnet model on CACD dataset	Train resnet models on datasets with generated images
	Week 9	Train and fine-tune model BYOL on FGNet dataset	Create a Web Demo	Tuning CUSP model to generate high preservation face aging images on FGNet and CACD
	Week 10	Train and fine-tune model BYOL on CACD dataset	Complete web demo	Train ResNet models on datasets with generated images
	Week 11	Research and analyze existing results	Research and analyze existing results	Research and analyze existing results

	Week 12	Write the project report, summarize the best results to write the report for the scientific conference	Write the project report, summarize the best results to write the report for the scientific conference	Write the project report, summarize the best results to write the report for the scientific conference
	Week 13	Refine the report and create slides for the final project defense.	Refine the report and create slides for the final project defense.	Refine the report and create slides for the final project defense.
	Week 14	Prepare the thesis defend	Prepare the thesis defend	Prepare the thesis defend

Table 3. Source code and data

Items	**Link**	**Description**
Data	Click	The dataset has been processed and divided into two training and testing sets with the ratio 7:3 used in the project.
Source code	Click	All source code is used to build the model, CUSP, and SSL

4. MATERIALS AND METHODS

4.1 Materials

4.1.1 Datasets overview

Several databases are commonly used in age-invariant face recognition (AIFR) research. Among them, FGNET and UTKFACE are the most frequently used databases. The FGNET database contains images of faces of different ages and has been widely used to evaluate the performance of AIFR systems. The UTKFACE database is another commonly used dataset that contains a large number of facial images of people from different age groups.

In addition to these databases, several others have been introduced in recent years to facilitate research in age-invariant face recognition. These include the MORPH, CACD, CACD-VS, and CASIA WEBFACE databases. The CACD database is particularly useful for AIFR research, as it contains images of celebrities at different ages, allowing for a more realistic evaluation of AIFR systems.

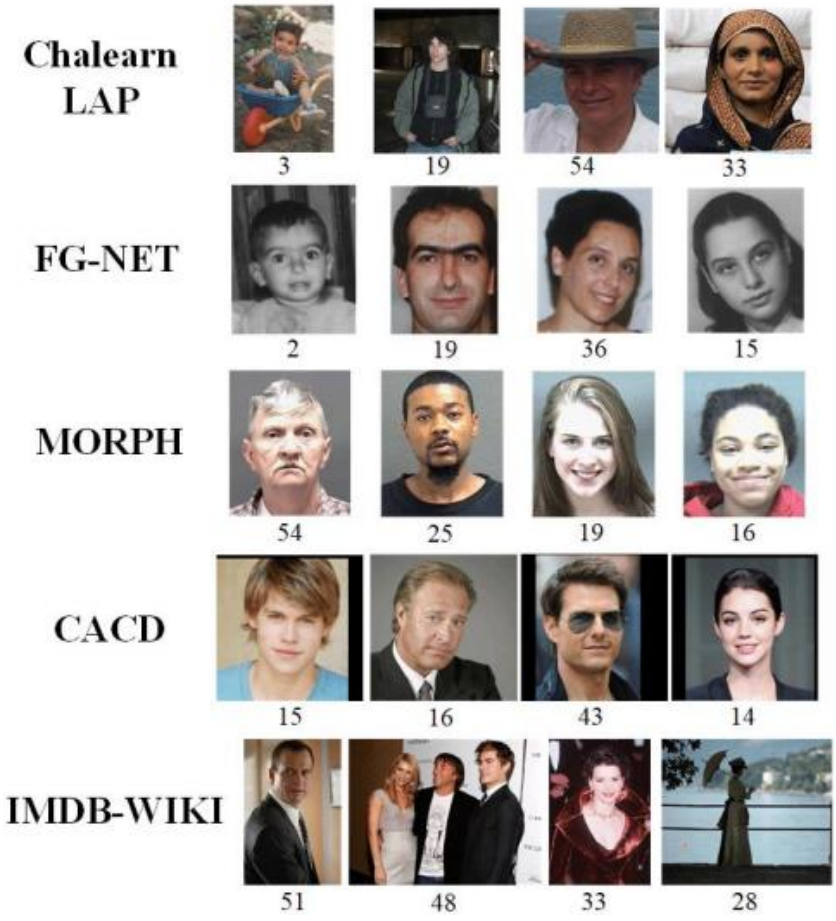


Figure 5. Datasets overview

Overall, the use of these databases has greatly facilitated the development and evaluation of AIFR systems, allowing researchers to test the robustness of their models to age-related changes and identify areas for further improvement. As AIFR continues to gain importance in a range of different applications, the use of these databases will likely become even more widespread, enabling the development of more accurate and reliable age-invariant facial recognition systems.

We have done some preliminary research on the above data sets to select the data set for experiment.

FGNET dataset

FGNET is considered to be one of the largest datasets for facial aging studies and has been used to investigate age-related changes in facial expressions. While FGNET has a relatively small number of images, it covers a wide range of ages and is therefore considered to be a valuable resource for age-invariant face recognition research.

The FGNET dataset consists of 1,002 images of 82 individuals. All of the images are user-submitted snapshots taken throughout the subject's lifetime. The average number of images per individual is 12, with a range of images from 6 to 18. One significant limitation of the FGNET dataset is that it contains fewer images than subjects (82). Although the ages of the subjects range from 0 to 69 years old, half of the subjects in FGNET were under the age of 13. Figure 6 below shows a few sample images in the FGNET dataset, showing data for many people of different ages. The age of the pictures is also included below each picture.

Figure 6. Sample data in dataset FG-NET

CACD dataset

Cross-Age Celebrity Dataset (CACD). The latest mature dataset is CACD, which includes 163,446 images of 2000 web-based esteemed individuals. Every face image is reviewed and checked.

The Celebrity Cross-Age Dataset (CACD) contains 163,446 images from 2,000 celebrities collected from the Internet. The images are collected from search engines using the celebrity's name and year (2004-2013) as keywords. The age of the subjects was 12-62 years old.

The images in the CACD dataset have various lighting, expressions, and poses, making it more realistic and challenging. The dataset also provides a list of the celebrities' birth and photo-taking years, which can be used to calculate the subjects' actual age. The CACD dataset has become one of the most commonly used datasets for age-invariant face recognition research, as it provides a large and diverse dataset for training and testing age-invariant face recognition models.

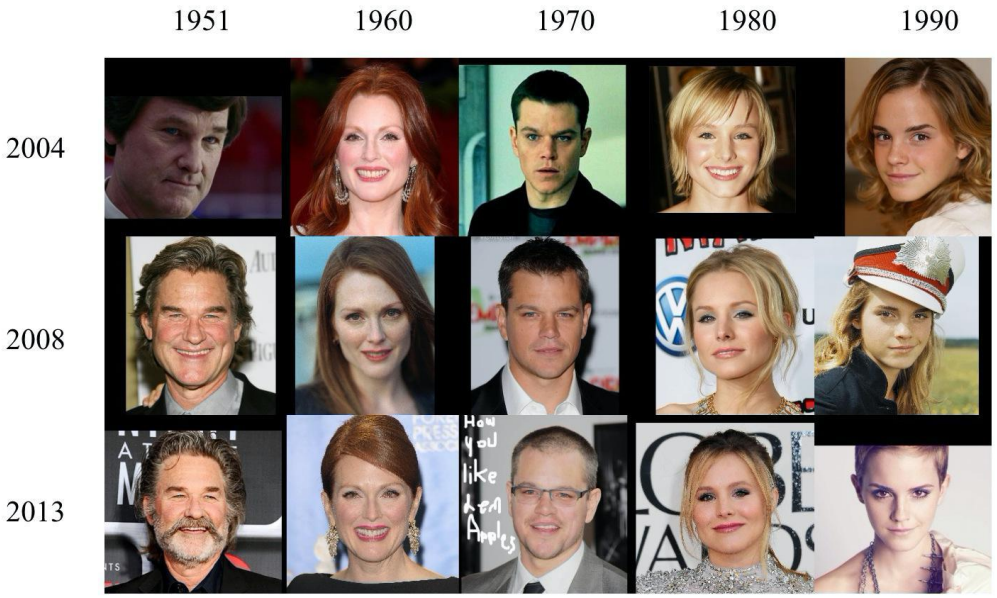


Figure 7. Sample data in dataset CACD

CASIA dataset

The CASIA-WebFace dataset is a widely used dataset for face recognition and verification tasks.

This dataset contains a total of 494,414 face images from 10,575 distinct individuals, all of which were collected from the internet. The dataset is special in that

it contains a large number of images for each individual, with some individuals having over 100 images available for training and testing (Figure 8).

Figure 8. Example of number of pictures of one person (one class) in the set CASIA

This large dataset allows for the development of more accurate and robust facial recognition models.

UTKFace dataset

The UTKFace dataset is a large-scale face dataset with long-term ages (from 0 to 116 years old). The dataset includes more than 20,000 face images with captions for age, gender, and ethnicity. The images include great variation in posture, facial expressions, lighting, occlusion, resolution, and more. This dataset can be used for a variety of tasks, e.g. face detection, age estimation, age progression/regression, landmark location, etc.

Figure 9. Sample data in dataset UTKFace

The IMDB-Wiki dataset

The IMDB-Wiki dataset is a large-scale face dataset that contains over 500,000 face images of individuals. The dataset was created by combining two existing datasets, IMDB and Wiki, which contain images of celebrities and public figures.

The dataset includes annotations for age, gender, and other attributes for each face image. The age range of the subjects in the dataset is from 0 to 100 years old, making it useful for age-invariant face recognition tasks.

The dataset also includes a large number of images for each subject, which can be used for training deep learning models.

Figure 10. Sample data in dataset IMDB-Wiki

4.1.2 Data pre-processing

After having an overview of the data sets, based on our analysis of the available datasets, we have determined that the FGNET and CACD datasets are most appropriate for our project in terms of time, resources, and objectives. Therefore, we have decided to use these two datasets for our research and experimentation.

With the aim of preparing the data for further analysis, it is often necessary to conduct various preprocessing steps. Before engaging in these procedures, however, it is important to have a thorough understanding of the data being used. Thus, in order to gain an overview of two specific datasets, we utilized the basic information that was acquired earlier and proceeded to evaluate both datasets. To achieve a more comprehensive understanding, an age survey was conducted on both datasets, enabling us to gather relevant information and prepare the data for subsequent preprocessing. By conducting these initial steps, we can ensure that the data is prepared in a way that maximizes its utility for analysis and facilitates the generation of valuable insights.

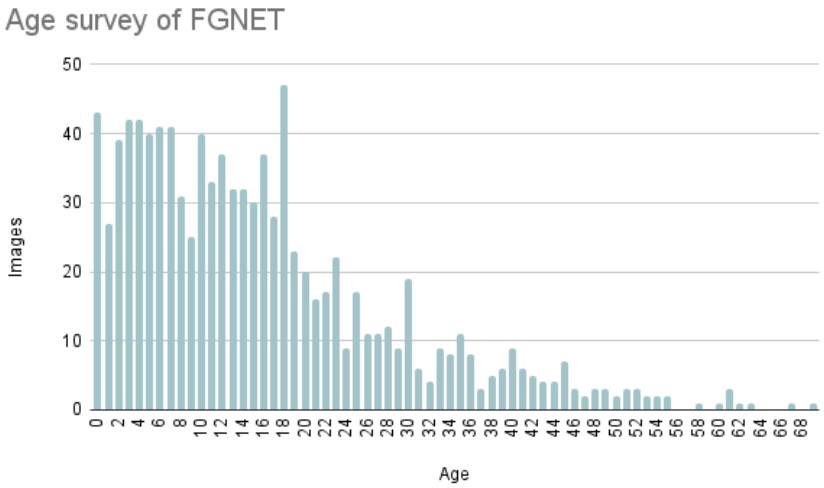


Figure 11. Age survey of FGNET dataset

Upon analyzing the age survey results for the FGNET dataset, it becomes evident that the age distribution of the data is uneven. The majority of the images in this dataset are of individuals aged between 0 to under 22 years old, with the number of images gradually decreasing as the age groups get older. This skewed distribution of ages leads to a shortage of data for the training set, which can pose a challenge for data analysis and modeling.

On the other hand, the CACD dataset (Figure 12) contains a significantly larger number of images compared to FGNET. The age distribution of the images in this dataset is relatively balanced, with a similar number of images available for each age group. This even distribution of ages provides a more diverse range of data for the training set, making it easier to perform analysis and modeling on this dataset.

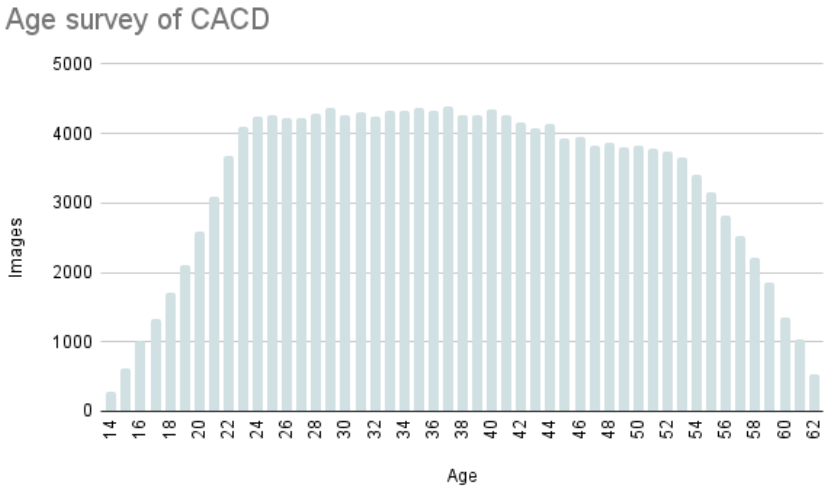


Figure 12. Age survey of FGNET dataset

Data pre-processing is a critical step in building any machine learning model. In the context of age-invariant face recognition. Initially, the original images within the dataset are cropped to extract only the face portion. Next, the background is eliminated and the image is adjusted for optimal quality. The image is then resized to 256 x 256 to facilitate the face detection procedure and reduce the computational load.

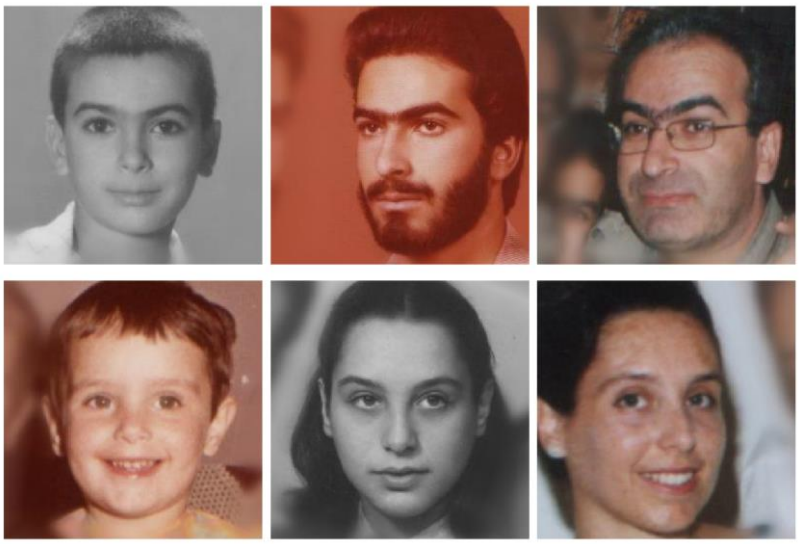


Figure 13. Some examples of images that have been aligned in the dataset

The face detection step is important as it allows the model to focus only on the facial features and ignore other irrelevant details in the image. Once the faces are detected, they are normalized to remove any variations in pose and lighting, ensuring that the model can learn the underlying features of the face regardless of the age of the individual. This normalization process can include steps such as alignment, equalization, and normalization of the image intensity values.

Finally, the dataset is split into training, validation, and test sets to allow for the evaluation of the model's performance on unseen data.

We will describe the detailed implementation and metrics on an FGNET dataset, while the CACD dataset is applied in a similar way.

We divided the FGNET dataset into two training and testing sets with 3 different scales to see the difference in accuracy between the split ratios. The ratio is 9:1, 7:3 and 6:4.

Table 4 gives us an overview of the split ratios in the dataset.

Table 4. Split data of FGNET dataset

Dataset	Ratio	Training set	Testing set
FGNET	9:1	854 images	148 images
	7:3	670 images	332 images
	6:4	564 images	438 images

4.2 Methods

4.2.1 Experimental process

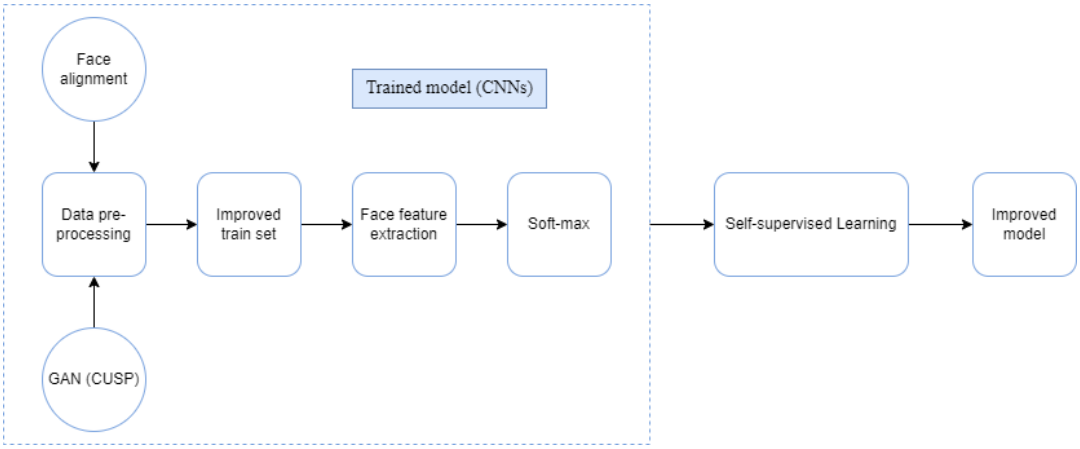


Figure 14. Experimental process

About the main workflows are divided into 2 large sections: Training age-invariant face recognition model and improving trained models using self-supervised learning.

Training age-invariant face recognition model

1. Data preprocessing: After selecting the dataset, we proceed to image processing steps of the dataset such as determining face position, aligning, resizing, removing error data, …
2. Improve the training dataset: Realizing the missing data of the dataset after dividing the training and testing sets, using GAN applying Custom Structure Preservation in Face Aging (CUSP) to add images into the training set.
3. Feature extraction and classification by pre-trained model with transfer learning technique.

Improving trained models using self-supervised learning

4.2.2 Data Augmentation

Data augmentation is a process of artificially increasing the amount of data by generating new data points from existing data. These are the data augmentation techniques used in this project.

4.2.2.1 Standard augmentations techniques

Color Jitter

Color Jitter transform randomly changes the brightness, saturation, and other properties of an image.

Figure 15. Color Jitter

Grayscale

Grayscale converts an image to grayscale.

Figure 16. Grayscale

Random Horizontal Flip

Random Horizontal Flip performs horizontal flip of an image.

Figure 17. Random Horizontal Flip

Gaussian Blur

Gaussian Blur performs gaussian blur transform on an image.

Figure 18. Gaussian Blur

Random Resized Crop

Random Resized Crop crops and image at a random location, and then resizes the crop to a given size.

Figure 19. Random Resized Crop

4.2.2.2 Generative Adversarial Networks (Custom Structure Preservation in Face Aging) [16]

Custom Structure Preservation in Face Aging (CUSP) is a face age transformation model involves reflecting age factors on a given face to create its future or past face images. A transformed face image must retain the identity of the original photo subject and all other details related to the face image.

Model architecture

The model consists of 5 neural nets:

- Style Encoder Network E_s: Produces a style embedding s of an input image **X.** Its final layer discards all image spatial information through global-average-pooling.
- Content Encoder E_c: Creates an image content representation c without taking into account the image's style details.
- Age Encoder E_a: Builds an age embedding vector.
- CUSP module: predicts a blurring mask M, which is leveraged to distinguish between image areas that should remain intact and those that are modified while performing image age transformation.
- Generator Network G: synthesizes a target image from the image style encoding s, its style encoding c, age representation at invoking a blurring mask M. The generator architecture is based on StyleGAN2 with several modifications made to adapt it to the age transformation task.

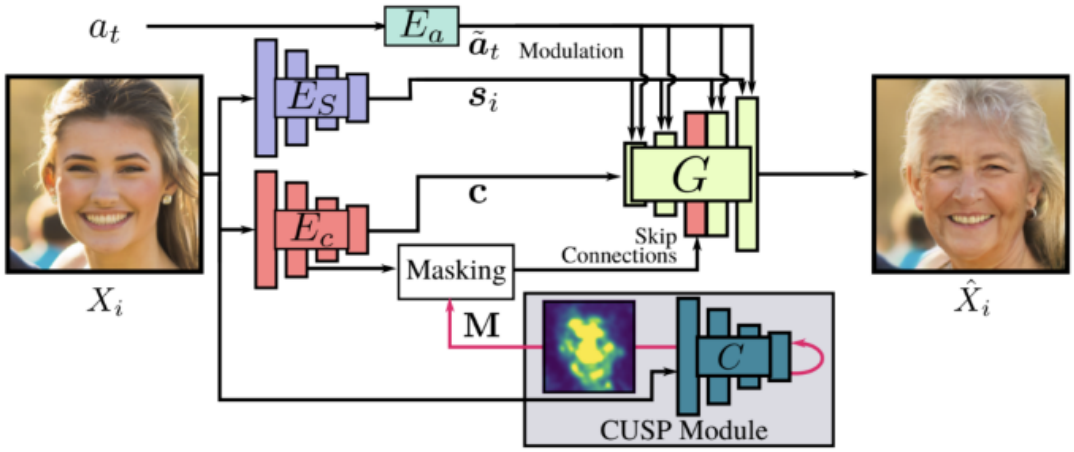


Figure 20. CUSP model architecture

Loss Function Structure

The loss function is composed of three components:

- Reconstruction Loss: measures the target image quality by employing L1 loss between the model output and ground-truth target image (used when a target images is available).
- Age Fidelity Losses: penalizes the constructed image for not matching the target age.
- Cycle-Consistency Loss: encourages a model to preserve not age-related details.

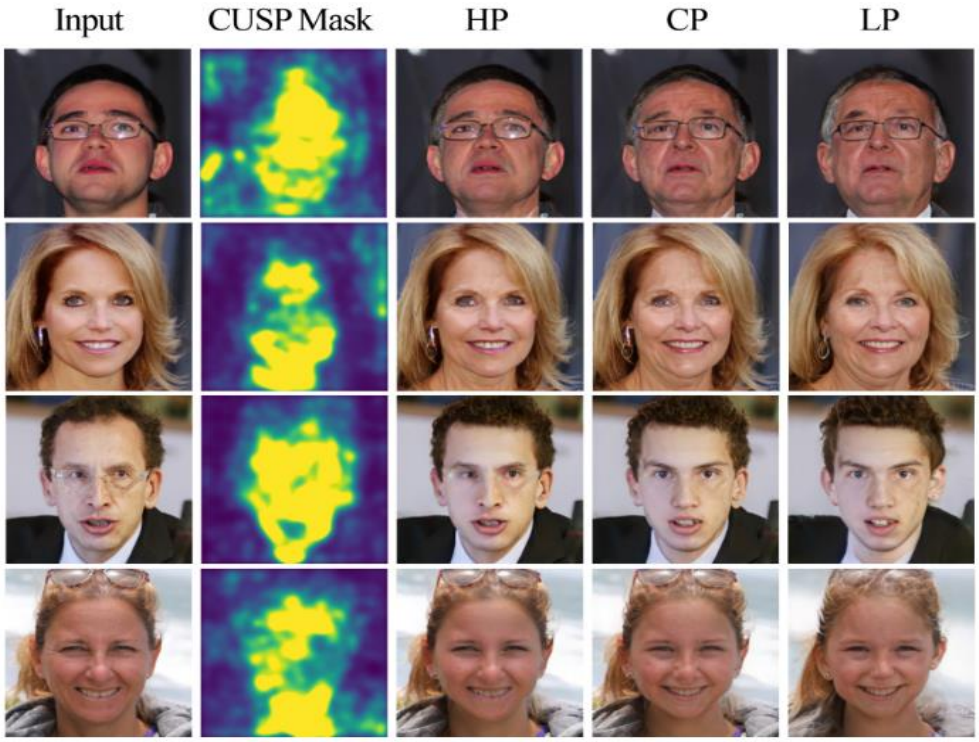


Figure 21. Examples of generated face aging images with High, Custom, Low preservation from the author's paper.

In this project, we will be using a CUSP model pretrained on FFHQ-RR dataset to generate face aging images on FGNet dataset with High preservation to retain the person's identity.

4.2.3 Age-Invariant Face recognition models

As described in the introduction, there are many approaches to building age-invariant face recognition models. In this project, we choose the method of using convolutional neural networks (CNNs) in deep learning.

4.2.3.1 What is deep learning?

Deep Learning is a subset of Machine Learning, capable of being different in some important respects from traditional shallow Machine Learning, allowing computers to solve a wide range of unsolvable complex problems.

An example of a simple, shallow Machine Learning task could predict how ice cream sales will change based on outdoor temperature. Making predictions using only a few data features in this way is relatively simple and can be done using a Machine Learning technique called linear regression with decreasing gradients.

The problem is that a host of real-world problems do not fit into such simple models. An example of one of these complex real-world problems is recognizing handwritten numbers.

To solve this problem, computers need to be able to deal with the great variety of ways in which data is presented. Each digit from 0 to 9 can be written in an infinite number of ways: the exact size and shape of each handwritten digit can vary greatly depending on the writer and under what circumstances.

Dealing with the variability of these features, and the greater interactive clutter between them, is where deep learning and deep neural networks come in handy.

Neural networks are structured mathematical models loosely inspired by the brain.

Each neuron in a neural network is a mathematical function that takes data through the input, transforms that data into a more tunable form, and then spits it out through the output. You can think of neurons in a neural network as arranged in layers, see picture below

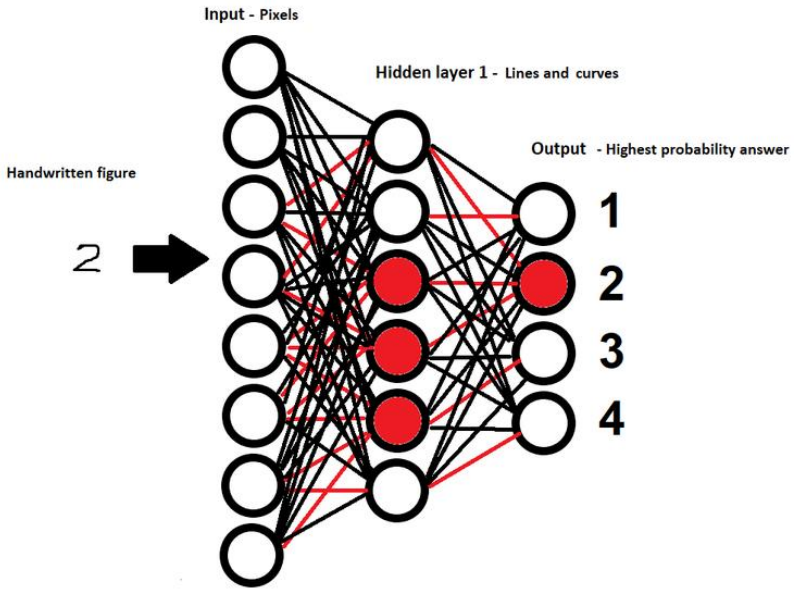


Figure 22. Layered Neural Networks

All neural networks have an input layer, into which the initial data is input, and an output layer, which produces the final prediction. But in a deep neural network, there are many hidden “cell layers” between the input and output layers, each feeding data into the other. Hence, the terms "Deep" in "Deep Learning" and "deep neural networks", it refers to the large number of hidden layers – often larger than three – at the heart of these neural networks.

The simplified diagram above will hopefully help provide an idea of how to structure a simple neural network. In this example, the network has been trained to recognize handwritten figures, such as the number 2 shown here, with the input layer given values representing the pixels that make up the image of the image. a handwritten digit and the output layer predicts which handwritten number is already displayed in the image.

In the diagram above, each circle represents a neuron in the network, with the neurons organized into vertical layers.

As you can see, each neuron is associated with every neuron in the following layer, representing the fact that each neuron generates a value into each neuron in the next layer. The colors of the links in the diagram are also different. The different colors, black and red, represent the importance of connections between neurons. The links in red are the ones of greater significance, which means they amplify the value as it passes

between the layers. In turn, amplifying this value can help activate the neuron to which the value is being input.

A neuron can be said to have been activated when the sum of the values fed into this neuron exceeds a set threshold. In the diagram, activated neurons are shown in red. This activation means varies by class. In the “Hidden Layer 1” shown in the diagram, an activated neuron can mean that the image of the handwritten figure contains a certain combination of pixels that resembles the horizontal line at the beginning of the handwritten number 7. In such a way This one, “Hidden Layer 1” can detect multiple lines and story curves that will eventually come together into a full handwritten figure.

An actual neural network will probably have both hidden layers and more neurons in each layer. For example, “Hidden Layer 2” can be given small lines and curves defined by “Hidden Layer 1” and detect how they combine to form recognizable shapes, forming letters. number, like the entire bottom loop of six. By providing transition data between layers in this way, each subsequent hidden layer handles increasingly higher features.

As mentioned, the activated neuron in the output layer of the diagram has a different meaning. In this case, the neuron is activated corresponding to the estimated number of neural networks it is shown in the image of a handwritten digit that it is fed as input.

As you can see, the output of one layer is the input of the next layer in the network, with data flowing through the network from input to output.

But how do these many hidden layers allow a computer to determine the nature of a handwritten digit? These multiple layers of neurons essentially provide a way for the neural network to build a rough hierarchy of the different features that make up the handwritten digit in question. For example, if the input is an array of values representing individual pixels in an image of a handwritten shape, the next layer can combine these pixels into lines and shapes, the next layer combines the that shape into distinct features like loops in 8 or upper triangle in 4, and so on. By building a picture of these features, modern neural networks can determine – with great accuracy – the number that corresponds to a handwritten digit. Similarly, different types of deep neural networks can be trained to recognize faces in images or to transcribe written speech.

The process of building this increasingly complex hierarchy of handwritten number features is nothing but pixels learned by the network. The learning process is accomplished by the network being able to change the importance of the connections between neurons in each layer. Each link has an attached value called a weight, which modifies the value generated by a neuron as it propagates from one layer to the next. By varying the value of these weights and an associated value called bias, it is possible to emphasize or reduce the importance of the connections between neurons in the network.

For example, in the case of handwritten digit recognition models, these weights can be modified to emphasize the importance of a particular group of pixels forming a line or a pair of intersecting lines forming 7.

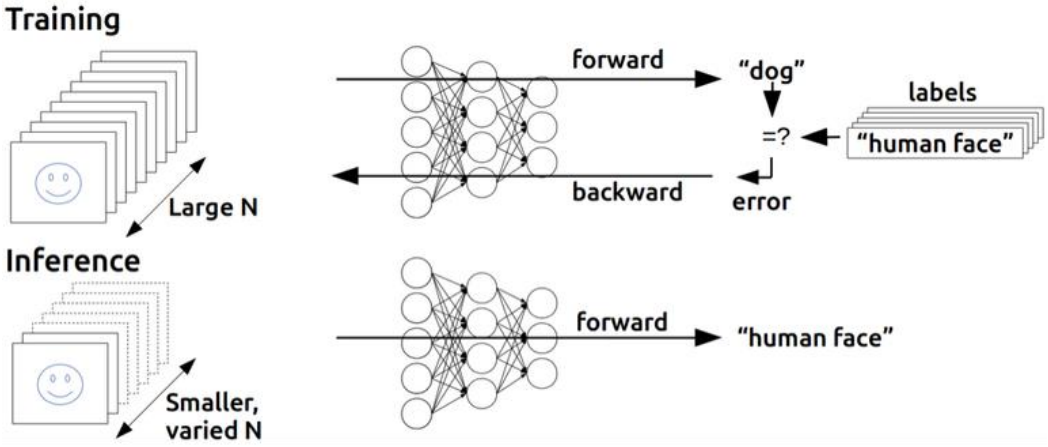


Figure 23. An illustration of the structure of a neural network and how training works

The model that learns the connections between neurons is important in making successful predictions during training. At each step in the training process, the network uses a mathematical function to determine how accurate its latest prediction is compared to the expected one. This function generates a series of error values, which the system can use to calculate how the model should update the value of the weights attached to each link, with the ultimate aim of improving accuracy. accuracy of the network's predictions. The degree to which these values will be changed is calculated by an optimization function such as gradient descent, and those changes are pushed back across the network at the end of each training cycle in a step called backpropagation.

Over many, many training cycles, and with the help of occasional manual parameter tuning, the network will keep on nue to generate better and better predictions

until it gets close to the accuracy. highest body. At this point, for example, when handwritten digits can be recognized with more than 95% accuracy, the Deep Learning model can be said to have been trained.

Essentially, Deep Learning enables Machine Learning to solve a new set of complex problems – such as image, language, and speech recognition – by allowing machines to learn how features in data combined into increasingly higher abstractions. For example, in face recognition, how pixels in an image create lines and shapes, how those lines and shapes produce facial features, and how these facial features are arranged into a face.

4.2.3.2 How to used Deep Learning?

For many tasks, to recognize and generate images, speech, and language, and combine with reinforcement learning to match human performance in games from ancient, like Go, to modern, like Dota 2 and Quake III.

Deep learning systems are a cornerstone of modern online services. Such systems are used by Amazon to understand what you say — both your words and the language you use — with Alexa or Google to translate text when you visit a foreign-language website.

Each Google search uses multiple Machine Learning systems, to understand the language of your query through personalizing your results, so fishing enthusiasts searching for “bass” are not inundated with guitar results.

But beyond the very obvious manifestations of machines and deep learning, such systems are beginning to find an application in every industry. These applications include: computer vision for driverless vehicles, drones and delivery robots; language and language recognition and synthesis for chatbots and service robots; facial recognition for surveillance in countries like China; help radiologists pick out tumors in X-rays, help researchers uncover genetic sequences involved in diseases, and identify molecules that could lead to more effective drugs in care. health care; enable predictive maintenance of infrastructure by analyzing IoT sensor data; reinforcing computer vision makes it possible for the cashless Amazon Go supermarket to provide reasonably accurate transcription and translation for business meetings – the list goes on.

4.2.3.3 Deep Learning techniques

There are many types of deep neural networks, with structures suitable for different types of tasks. For example, Convolutional Neural Networks (CNNs) are commonly used for computer vision tasks, while Recurrent Neural Networks (RNNs) are commonly used for language processing. Each layer has its own specializations, in CNN the layers are initially specialized to extract distinct features from the image, and then fed into a more conventional neural network to allow the image to be classified. Meanwhile, RNNs differ from traditional feed-forward neural networks in that they not only feed data from one neural layer to the next, but also have built-in feedback loops, where the output data from one layer is fed back to the layer before it – giving the network a form of memory. There is a more specialized form of RNN that includes what is called a memory cell and is tuned to handle data with delays between inputs.

The most basic type of neural network is a multilayer perceptron network, the type discussed above in the handwritten metrics example, where data is fed forward between layers of neurons. Each neuron will typically transform the values they are given using an activation function, changing those values into a form that, at the end of the training cycle, will allow the network to compute the coverage. far to make accurate predictions.

There are a large number of different types of deep neural networks. No one network is better than the other, they are just better suited for learning specific types of tasks.

Recently, general adversarial networks (Gans) are expanding what is possible to use neural networks. In this architecture, two neural networks fight, the generator network tries to generate convincing “fake” data and the discriminator tries to tell the difference between fake and real data. . With each training cycle, the generator gets better at generating fake data, and the discriminator gets a sharper eye to spot those fakes. By combining two networks together during training, both can achieve better performance. GANs have been used to perform a number of important tasks.

4.2.3.4 What is Convolutional Neural Networks (CNNs)?

Convolutional Neural Networks (CNNs) are one of the advanced Deep Learning models. It helps us build intelligent systems with high accuracy today. CNN is widely used in the problem of recognizing objects in images.

CNN is a sliding window on a matrix as shown in the figure below.

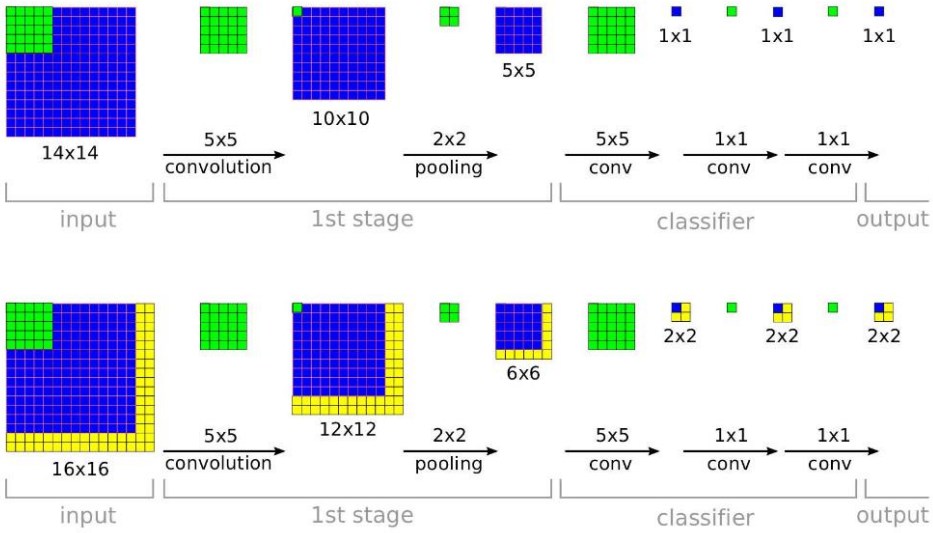


Figure 24. Sliding window (CNNs)

The convolutional layers have parameters (kernel) that have been learned to self-adjust to get the most accurate information without selecting features.

In the example image above, the matrix on the left is a digitized black and white image. The matrix is 5×5 and each pixel has a value of 1 or 0 that is the intersection of the row and column.

Convolution or convolution is multiplying each element in the matrix 3. Sliding Window, also known as kernel, filter or feature detect is a matrix with a small size as in the above example 3×3.

Convolution or convolution is multiplying each element inside a 3×3 matrix by the matrix on the left. The result is a matrix called Convoled feature that is generated from receiving the Filter matrix with the left 5×5 image matrix.

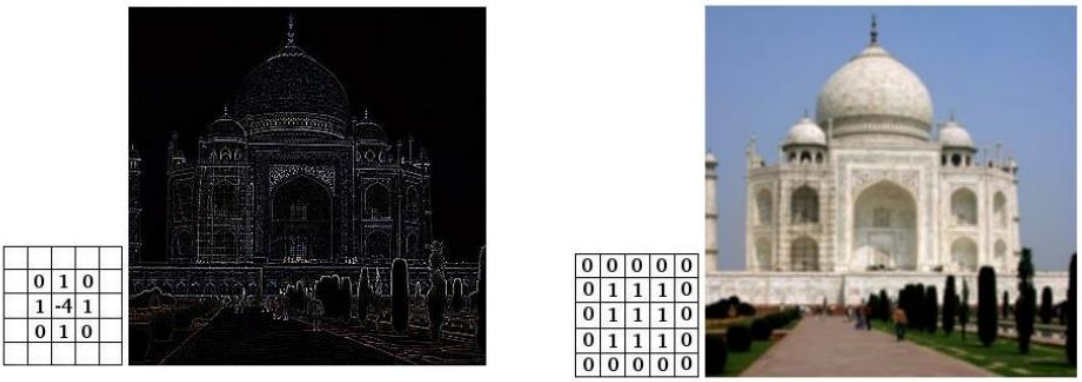


Figure 25. The Convoled feature is generated from receiving a Filter matrix with a 5×5 image matrix.

4.2.3.5 CNN network structure

A CNN network is a collection of Convolution layers that overlap and use nonlinear activation functions like ReLU and tanh to activate the weights in the nodes. Each class after passing activation functions will generate more abstract information for the next classes.

Each class after passing activation functions will generate more abstract information for the next classes. In the feedforward neural network model, each input neuron gives each output neuron in subsequent layers.

This model is called fully connected layer or fully connected network. In the CNNs model, the opposite is true. The layers are linked together through the convolution mechanism.

The next layer is the convolution result from the previous layer, so we have local connections. Thus, each neuron in the next layer is generated from the result of a filter applied to a local image region of the previous neuron.

Each class that uses different filters usually takes hundreds of thousands of such filters and combines their results. In addition, there are some other layers such as pooling/subsampling layer used to filter out more useful information (remove noise information).

During the training of the network, the CNN automatically learns the values through the filter classes based on how you do it. For example, in the image classification task, CNNs will try to find the optimal parameters for the corresponding

filters in the order raw pixels > edges > shapes > facial > high-level features. The last layer is used to layer the image.

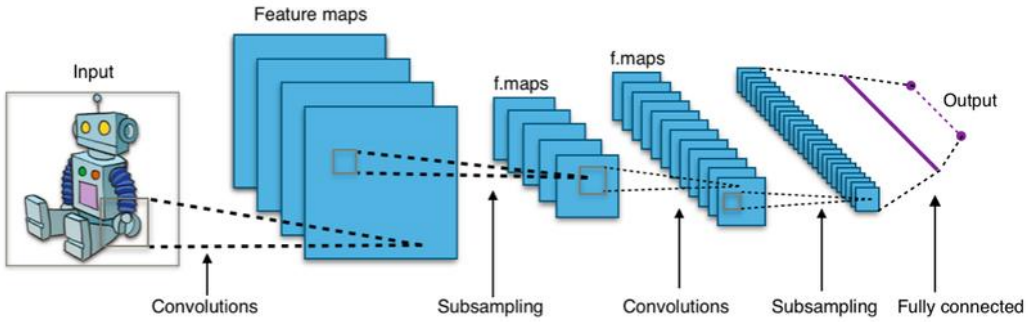


Figure 26. An example of an identity model of CNN

In the CNN model, there are two aspects that need attention: invariance (Location Invariance) and coherence (Compositionality). With the same object, if this object is projected in different degrees (translation, rotation, scaling), the accuracy of the algorithm will be significantly affected.

Pooling layer will give you invariant to translation, rotation, and scaling. Local associativity gives us lower-to-higher and more abstract levels of information representation through convolution from filters.

That is why CNNs produce models with very high accuracy. Just like how humans perceive objects in nature.

The CNN network uses three basic ideas:

- Local receptive fields
- Shared weights
- Pooling

4.2.3.6 Transfer Learning

In transfer learning, the knowledge of an already trained machine learning model is applied to a different but related problem. For example, if you trained a simple classifier to predict whether an image contains a backpack, you could use the knowledge that the model gained during its training to recognize other objects like sunglasses.

With transfer learning, we basically try to exploit what has been learned in one task to improve generalization in another. We transfer the weights that a network has learned at “task A” to a new “task B.”

The general idea is to use the knowledge a model has learned from a task with a lot of available labeled training data in a new task that doesn't have much data. Instead of starting the learning process from scratch, we start with patterns learned from solving a related task.

Transfer learning is mostly used in computer vision and natural language processing tasks like sentiment analysis due to the huge amount of computational power required.

Transfer learning isn’t really a machine learning technique, but can be seen as a “design methodology” within the field, for example, active learning. It is also not an exclusive part or study-area of machine learning. Nevertheless, it has become quite popular in combination with neural networks that require huge amounts of data and computational power.

4.2.3.7 How transfer learning works

In computer vision, for example, neural networks usually try to detect edges in the earlier layers, shapes in the middle layer and some task-specific features in the later layers. In transfer learning, the early and middle layers are used and we only retrain the latter layers. It helps leverage the labeled data of the task it was initially trained on.

Let’s go back to the example of a model trained for recognizing a backpack on an image, which will be used to identify sunglasses. In the earlier layers, the model has learned to recognize objects, because of that we will only retrain the latter layers so it will learn what separates sunglasses from other objects.

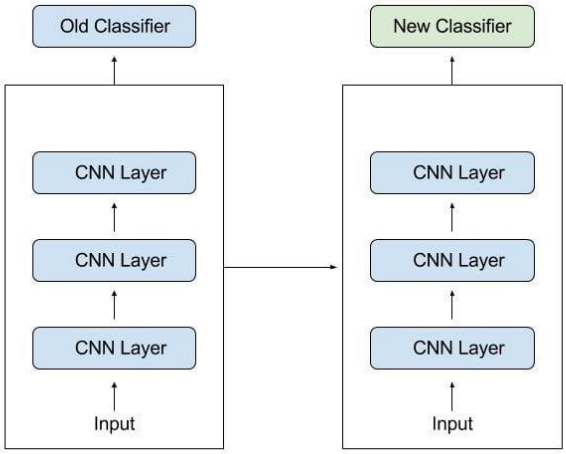


Figure 27. Transfer learning

In transfer learning, we try to transfer as much knowledge as possible from the previous task the model was trained on to the new task at hand. This knowledge can be in various forms depending on the problem and the data. For example, it could be how models are composed, which allows us to more easily identify novel objects.

Specifically, Transfer Learning in Deep Learning is a technique in which:

A pretrained model has been trained on a specific source task, then part or all of the pretrained model can be reused depending on the task of each layer in that model.

A new model uses part or all of the pretrained model to learn a target task, and depending on the task of each layer, the new model can add other layers based on the existing pretrained model.

The use of pretrained models is a big step for those following to follow in the footsteps of their predecessors, taking advantage of existing pretrained models to create new models for more specific target tasks. more practical applicability.

It's not copying an idea, the creators of the pretrained models themselves make their success public in the hope that followers can benefit from those models, or at least use it to solve their problems.

4.2.3.8 Fine-tunning

To use pretrained models effectively, we need two things:

- Add layers that match our target tasks, remove pretrained model layers that we don't use (which you probably already know), but have to make the model more efficient, here's A difficult problem (very very difficult) requires in-depth study of each layer and their purpose.
- Having a good training strategy, this is also not easy, because if you train poorly, it will lose the effectiveness of the pretrained model and thus reduce the ability of the model we are training, even worse. worse than training all over again.

Therefore, fine-tuning was born to help you have an effective training strategy on your transfered model (the first thing as far as I know, it cannot be generalized to create a technique).

Fine-tuning doesn't just help you adjust the weights of the transferred model to match your target tasks. It is not just a fine-tuning like translation of fine-tuning, but

beyond that, it offers an optimal way to train both the pretrained model and the new part in the transferred model to achieve high accuracy on target tasks, making two parts. fit together to form a new model.

In short, fine-tuning is training a transferred model for the purpose of optimizing the accuracy of this model on target tasks. Here are the most commonly used strategies:

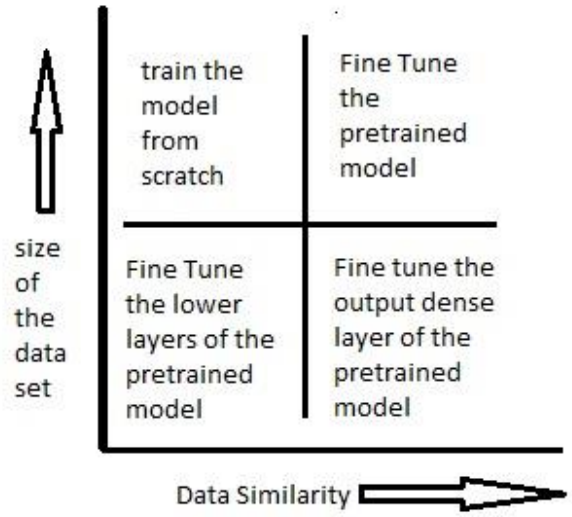


Figure 28. Fine-tunning strategies

- When the dataset for target tasks is large and similar to the dataset for source tasks: this is the ideal case, when you can use the weights of the pretrained model to initialize the pretrained part, then train either the transferred model or just the added part. in, it's up to you.
- When the dataset for target tasks is small and similar to the dataset for source tasks: because the dataset is small, retraining the pretrained part will lead to overfitting, so we only train the added layers with the weights initialized for the pretrained as above.
- When the dataset for target tasks is large and different from the dataset for source tasks: because our dataset is different, using weights from a pretrained model will reduce accuracy because of the difference in tasks and dataset, but also because of the dataset. large, so training the entire transfered model from scratch is the most effective, helping the model better adapt to this dataset.

- When the dataset for target tasks is small and different from the dataset for source tasks: this is the most difficult case, what you should do might be:
    - Interfere with pretrained models, replace pretrained layers far from the input to adapt to the new dataset (high-level features will change into low-level features that have been taken from previous layers) but do not train close layers input of pretrained because the dataset is small will not be able to train these layers effectively and these layers only extract general features from the dataset, which will not affect the target task.
    - Consult an expert or senior for more methods.

4.2.3.9 What is Residual Network (ResNet)?

A Residual Network [17], or ResNet for short, is a type of deep neural network architecture that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.

ResNet was designed to address the problem of vanishing gradients that can occur when training very deep neural networks. This problem arises because as the gradient is propagated through many layers, it can become extremely small, making it difficult for the network to learn.

Vanishing Gradient

First of all, Backpropagation Algorithm is a technique commonly used in tranining. The general idea of the leaf algorithm is to go from the output layer to the input layer and compute the corresponding cost function gradient for each parameter (weight) of the network. Gradient Descent is then used to update those parameters.

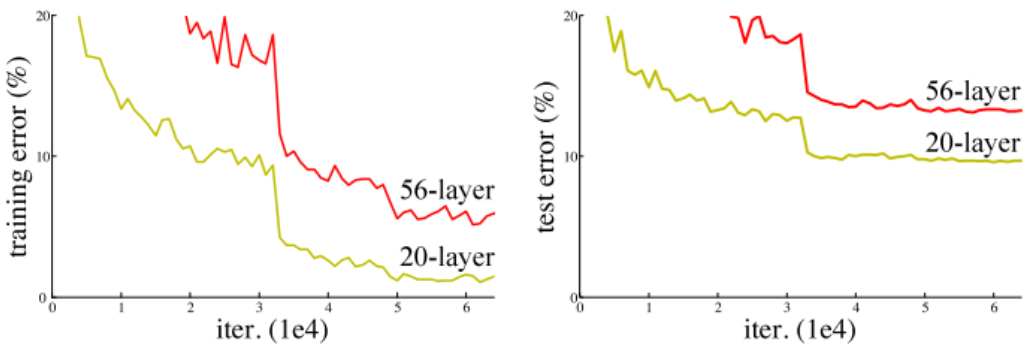


Figure 29. Gradient Descent

The whole process will be repeated until the parameters of the network are converged. Normally we would have a hyperparameter (the number of epochs - the number of times the tranig set is traversed once and the weights are updated) that defines the number of iterations to perform this process.

Vanishing gradient is a common problem that can occur when training deep neural networks. It happens when the gradient of the loss function with respect to the weights of the network becomes very small as it is propagated back through many layers during the training process. When the gradient becomes too small, it can make it difficult or even impossible for the network to learn, because the updates to the weights become negligible.

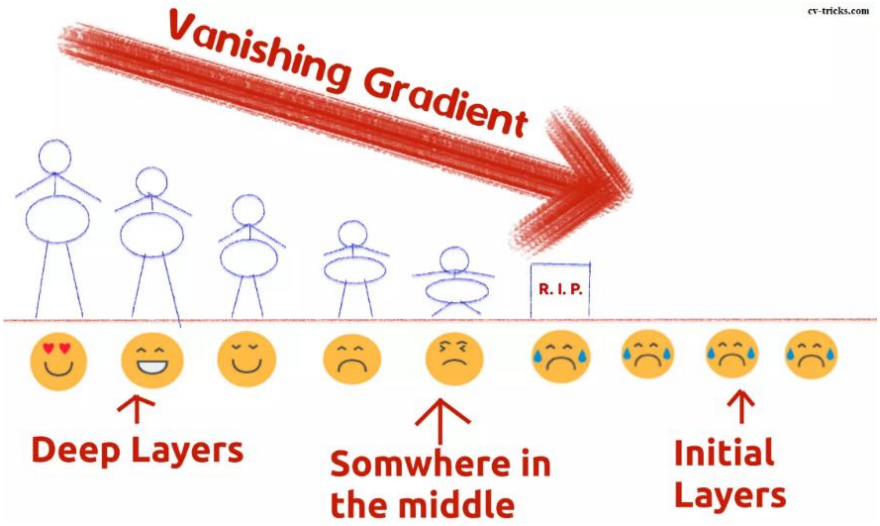


Figure 30. Vanishing Gradient Intuition

This problem can occur because of the way gradients are calculated using the chain rule of calculus. As the gradient is propagated through each layer, it is multiplied by the weights of that layer, which can cause it to become smaller and smaller with each layer. This effect is compounded in deep networks, where there can be many layers. When the gradient becomes too small, the weights of the network are not updated as much as they should be, which can lead to slow or ineffective training. The vanishing gradient problem is especially common in networks that use activation functions that have derivatives that are close to zero over much of their range, such as the sigmoid function.

Residual networks, as mentioned in the previous, are one solution to the vanishing gradient problem. By allowing for shortcut connections that bypass some layers, residual networks can make it easier for gradients to flow through the network, even in very deep architectures. Other solutions include careful weight initialization and the use of activation functions with more desirable properties, such as the rectified linear unit (ReLU) function.

4.2.3.10 ResNet architecture [17]

Deep Residual Network is almost similar to the networks which have convolution, pooling, activation and fully-connected layers stacked one over the other. The only construction to the simple network to make it a residual network is the identity connection between the layers. The figure below shows the residual block used in the network. You can see the identity connection as the curved arrow originating from the input and sinking to the end of the residual block.

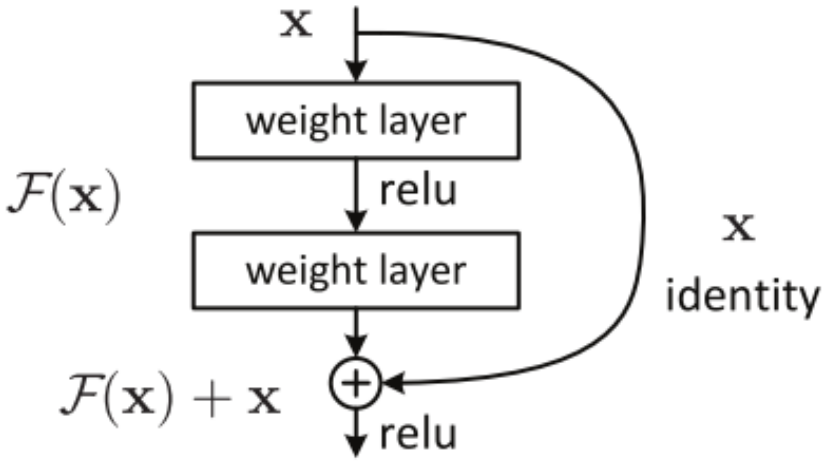


Figure 31. A residual block of Deep Residual Network

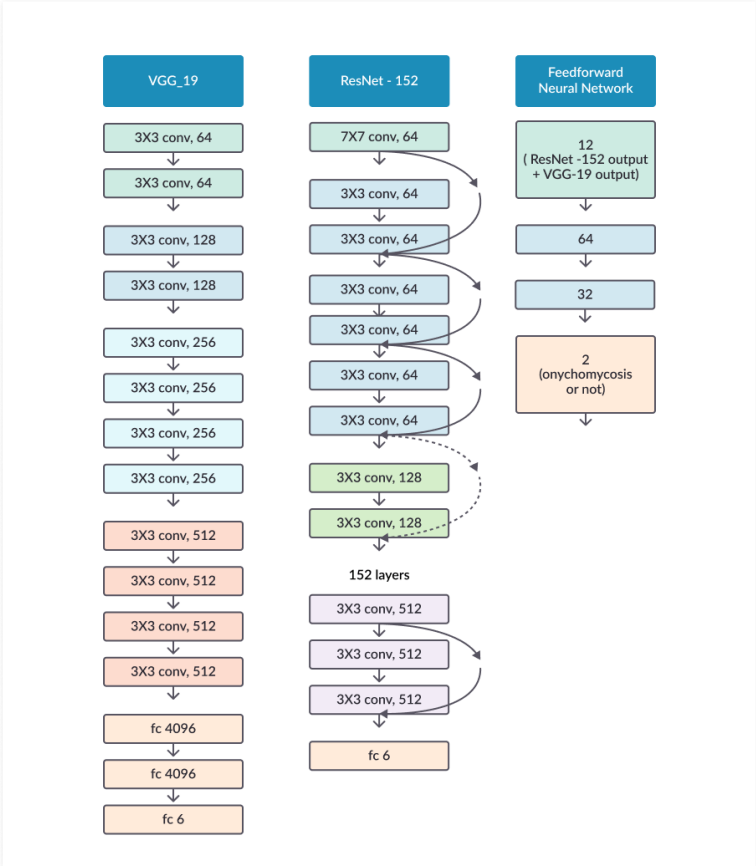
A curved arrow starts at the beginning and ends at the end of the residual block. In other words, it will add Input X to the output of the layer, or the addition we see in the illustration, this will counteract the zero derivative, since X is still added. With $H(x)$ is the predicted value, $F(x)$ is the true value (label), we want $H(x)$ to be equal to or approximately $F(x)$.

The creators of ResNet again thought as the statement goes:

"RATHER THAN EXPECTING STACKED LAYERS TO LEARN TO APPROXIMATE $H(x)$, THE AUTHORS ARE LETTING THE LAYERS TO APPROXIMATE A RESIDUAL FUNCTION i.e. $F(x) = H(x) - x$."

The above statement is explaining that during training the deep residual network, the main focus is to learn the residual function i.e. $F(x)$. So, if the network will somehow learn the difference ($F(x)$) between the input and output, then the overall accuracy can be increased. In other words, the residual value should be learned in a way such that it approaches zero, therefore making the identity mapping optimal. In this way, all the layers in the network will always produce the optimal feature maps i.e. the best case feature map after the convolution, pooling and activation operations. The optimal feature map contains all the pertinent features which can perfectly classify the image to its ground-truth class.

Increasing the number of layers in the network reduces accuracy, but wanting a deeper network architecture can work well.



a) b) c)

Figure 32

In Figure 32:

a) VGG-19 is a CNN model that uses a 3×3 kernel across the entire network, VGG-19 also won the 2014 ILSVRC.

b) ResNet using short connections (directly connecting the input of the layer (n) to $n + x$ is shown as a curved arrow. Through the model it has been demonstrated that it can improve performance in model training process when the model has more than 20 classes.

c) A total of 12 outputs from ResNet-152 and VGG-19 were used as inputs for the network with 2 hidden layers. The final output is calculated through two hidden layers. Stacking layers will not reduce network performance. With this architecture, the upper layers get information more directly from the lower layers, so it will adjust the weights more efficiently.

4.2.3.11 Architecture of ResNet-34

The first ResNet architecture was the Resnet-34 [17] which involved the insertion of shortcut connections in turning a plain network into its residual network counterpart. In this case, the plain network was inspired by VGG neural networks (VGG-16, VGG-19), with the convolutional networks having 3×3 filters. However, compared to VGGNets, ResNets have fewer filters and lower complexity. The 34-layer ResNet achieves a performance of 3.6 bn FLOPs, compared to 1.8bn FLOPs of smaller 18-layer ResNets.

It also followed two simple design rules – the layers had the same number of filters for the same output feature map size, and the number of filters doubled in case the feature map size was halved in order to preserve the time complexity per layer. It consisted of 34 weighted layers.

The shortcut connections were added to this plain network. While the input and output dimensions were the same, the identity shortcuts were directly used. With an increase in the dimensions, there were two options to be considered. The first was that the shortcut would still perform identity mapping while extra zero entries would be padded for increasing dimensions. The other option was to use the projection shortcut to match dimensions.

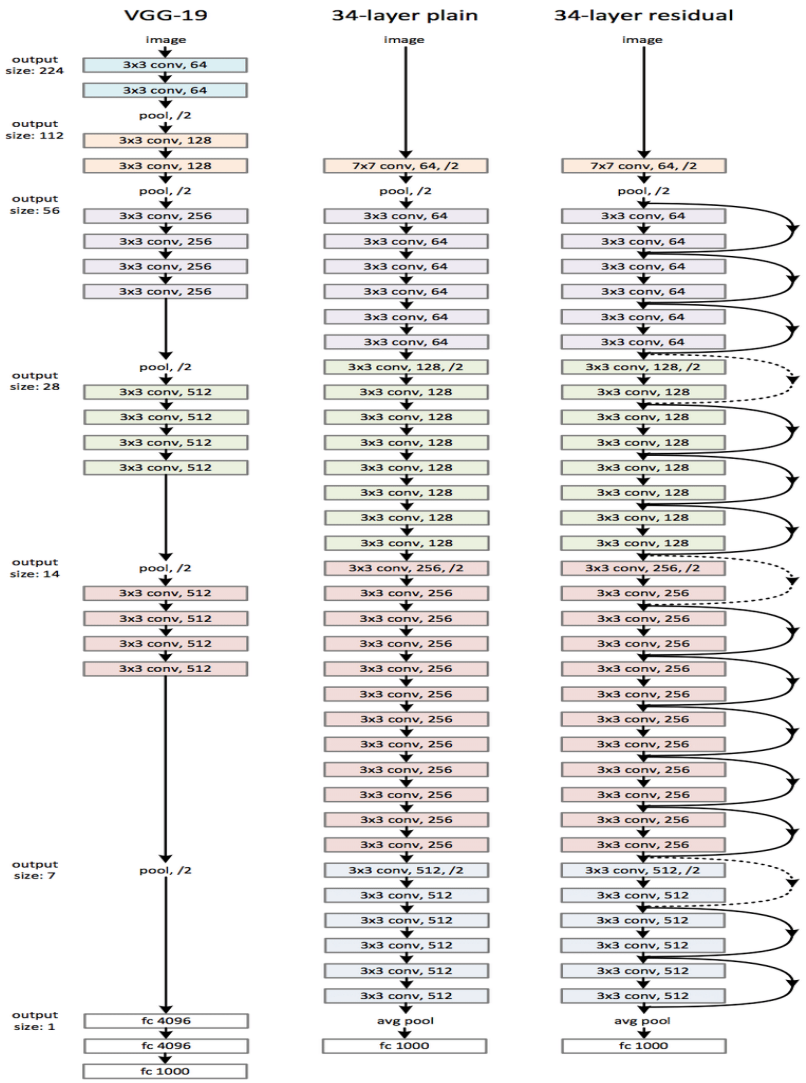


Figure 33. ResNet 34 original architecture

Since ResNets can have variable sizes, depending on how big each of the layers of the model are, and how many layers it has, we will follow the described by the authors in the paper — ResNet 34 — in order to explain the structure after these networks.

If you have taken a look at the paper, you will have probably seen some figures and tables like the following ones, that you are struggling to follow. Lets depict those figures by going into the detail of every step.

In here we can see that the ResNet (the one on the right) consists on one convolution and pooling step (on orange) followed by 4 layers of similar behavior.

Each of the layers follow the same pattern. They perform 3x3 convolution with a fixed feature map dimension (F) [64, 128, 256, 512] respectively, bypassing the input

every 2 convolutions. Furthermore, the width (W) and height (H) dimensions remain constant during the entire layer.

The dotted line is there, precisely because there has been a change in the dimension of the input volume (of course a reduction because of the convolution). Note that this reduction between layers is achieved by an increase on the stride, from 1 to 2, at the first convolution of each layer; instead of by a pooling operation, which we are used to see as down samplers.

In the table, there is a summary of the output size at every layer and the dimension of the convolutional kernels at every point in the structure.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 34. Sizes of outputs and convolutional kernels for ResNet 34

The next figure is the way I prefer to see convolutional models, and from which I will explain every layer. I prefer to observe how actually the volumes that are going through the model are changing their sizes. This way is easier to understand the mechanism of a particular model, to be able to adjust it to our particular needs — we will see how just changing the dataset forces to change the architecture of the entire model. Also, I will try to follow the notation close to the PyTorch official implementation to make it easier to later implement it on PyTorch.

For instance, ResNet on the paper is mainly explained for ImageNet dataset. But the first time I wanted to make an experiment with ensembles of ResNets, I had to do it on CIFAR10. Obviously, since CIFAR10 input images are (32×32) instead of (224×224), the structure of the ResNets need to be modify. If you want to have a control on the modifications to apply to your ResNet, you need to understand the details. This other

tutorial is a simplified of the current one applied to CIFAR10. So, let's go layer by layer!

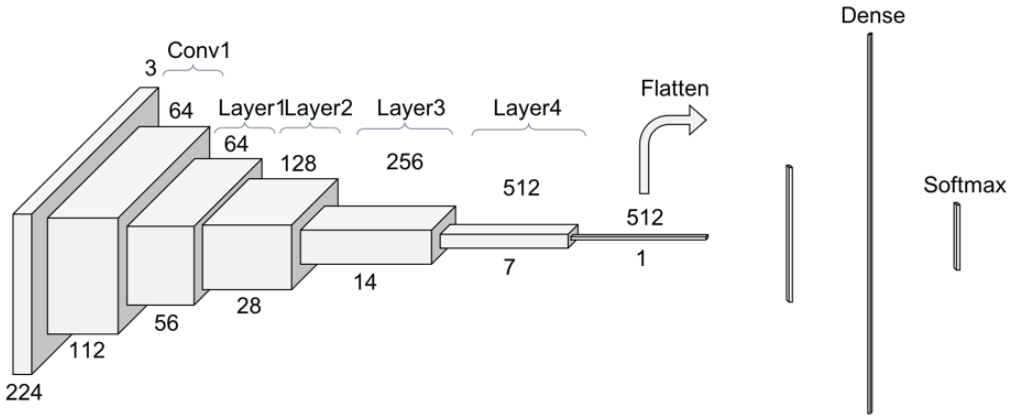


Figure 35. Another look at ResNet 34

Convolution 1

The first step on the ResNet before entering the common layer behavior is a block — called here Conv1 — consisting on a convolution + batch normalization + max pooling operation.

If you don't remember how convolutions and pooling operations where performed, take a quick look at this draws I made to explain them, since I reused part of them here.

So, first there is a convolution operation. In the Figure 33 we can see that they use a kernel size of 7, and a feature map size of 64. You need to infer that they have padded with zeros 3 times on each dimension — and check it on the PyTorch documentation. Taken this into account, it can be seen in Figure 36 that the output size of that operation will be a (112×112) volume. Since each convolution filter (of the 64) is providing one channel in the output volume, we end up with a $(112 \times 112 \times 64)$ output volume — note this is free of the batch dimension to simplify the explanation.

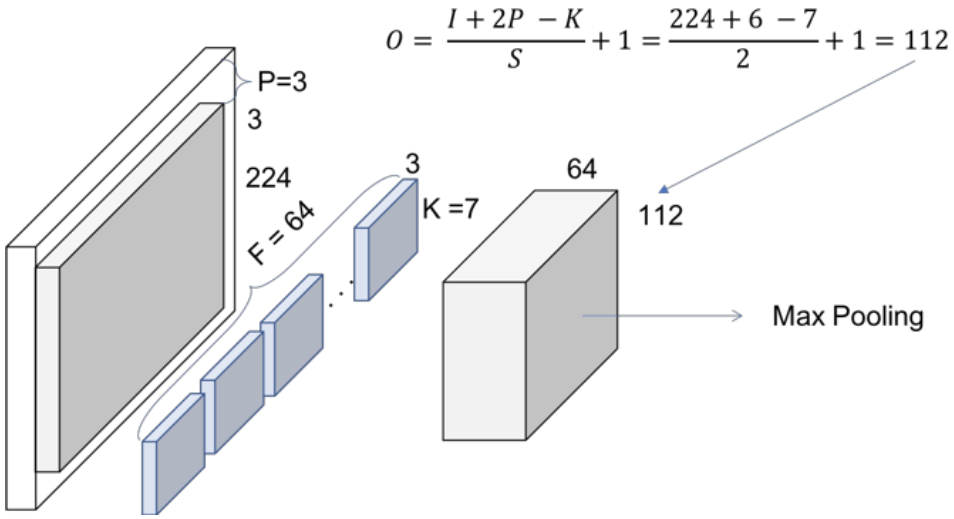


Figure 36. Conv1 — Convolution

The next step is the batch normalization, which is an element-wise operation and therefore, it does not change the size of our volume. Finally, we have the (3×3) Max Pooling operation with a stride of 2. We can also infer that they first pad the input volume, so the final volume has the desired dimensions.

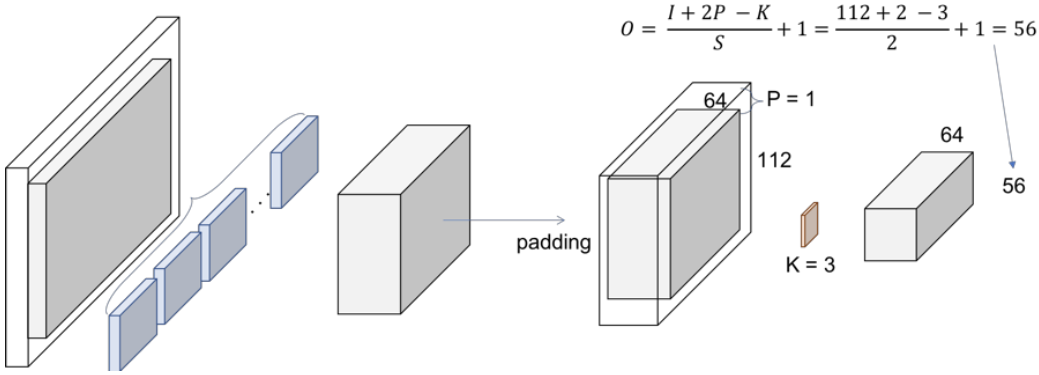


Figure 37. Conv1 — Max Pooling

ResNet Layers

So, let's explain this repeating name, block. Every layer of a ResNet is composed of several blocks. This is because when ResNets go deeper, they normally do it by increasing the number of operations within a block, but the number of total layers remains the same — 4. An operation here refers to a convolution a batch normalization and a ReLU activation to an input, except the last operation of a block, that does not have the ReLU.

Therefore, in the PyTorch implementation they distinguish between the blocks that includes 2 operations — Basic Block — and the blocks that include 3 operations — Bottleneck Block. Note that normally each of these operations is called layer, but we are using layer already for a group of blocks.

We are facing a Basic one now. The input volume is the last output volume from Conv1.

Block 1

1 convolution

We are replicating the simplified operation for every layer on the paper.

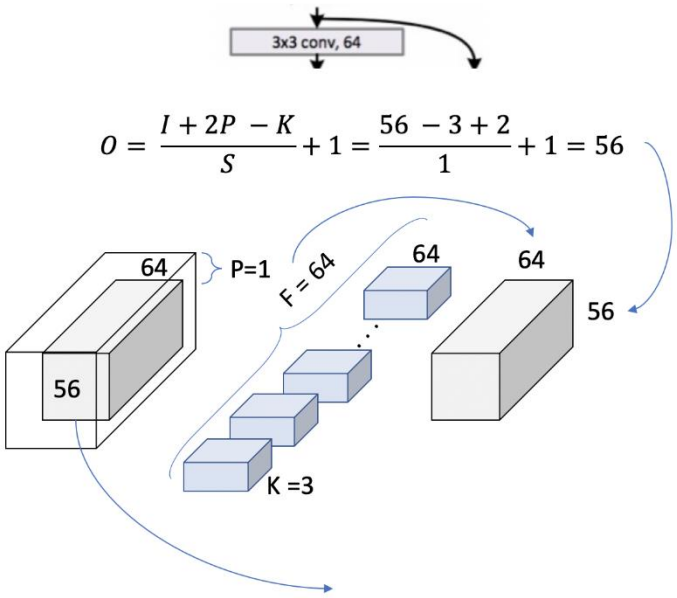


Figure 38. Layer 1, block 1, operation 1

We can double check now in the table from the paper we are using $[3 \times 3, 64]$ kernel and the output size is $[56 \times 56]$. We can see how, as we mentioned previously, the size of the volume does not change within a block. This is because a padding = 1 is used and a stride of also 1. Let's see how this extends to an entire block, to cover the 2 $[3 \times 3, 64]$ that appears in the table.

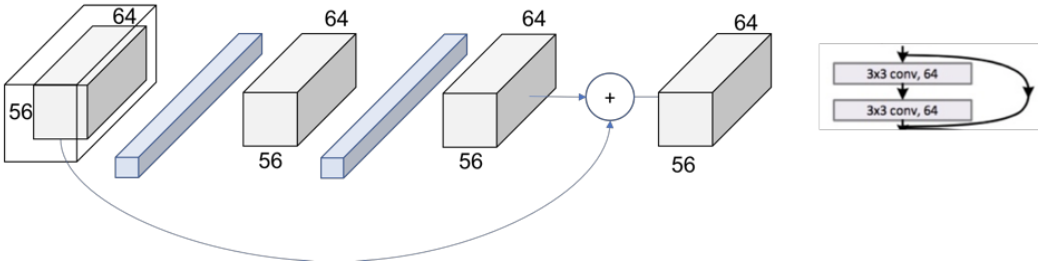


Figure 39. Layer 1, block 1

The same procedure can be expanded to the entire layer then as in Figure 40. Now, we can completely read the whole cell of the table (just recap we are in the 34 layers ResNet at Conv2_x layer.

We can see how we have the $[3 \times 3,64]$ times within the layer.

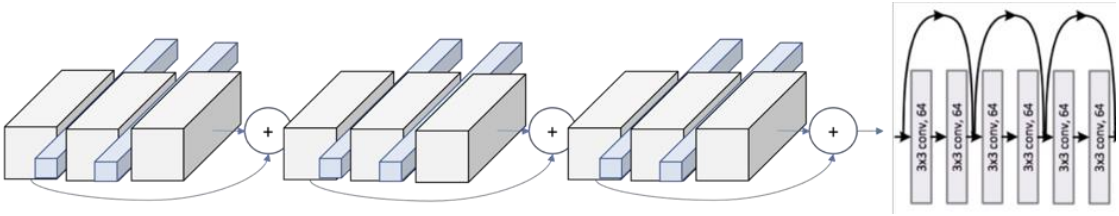


Figure 40. Layer 1

Patterns

The next step is to escalate from the entire block to the entire layer. In the Figure 33 we can see how the layers are differentiable by colors. However, if we look at the first operation of each layer, we see that the stride used at that first one is 2, instead of 1 like for the rest of them.

This means that the down sampling of the volume though the network is achieved by increasing the stride instead of a pooling operation like normally CNNs do. In fact, only one max pooling operation is performed in our Conv1 layer, and one average pooling layer at the end of the ResNet, right before the fully connected dense layer in Figure 33.

We can also see another repeating pattern over the layers of the ResNet, the dot layer representing the change of the dimensionality. This agrees with what we just said. The first operation of each layer is reducing the dimension, so we also need to resize

the volume that goes through the skip connection, so we could add them like we did in Figure 39.

This difference on the skip connections are the so called in the paper as Identity Shortcut and Projection Shortcut. The identity shortcut is the one we have already discussed, simply bypassing the input volume to the addition operator. The projection shortcut performs a convolution operation to ensure the volumes at this addition operation are the same size. From the paper we can see that there are 2 options for matching the output size. Either padding the input volume or perform 1×1 convolutions. Here, this second option is shown.

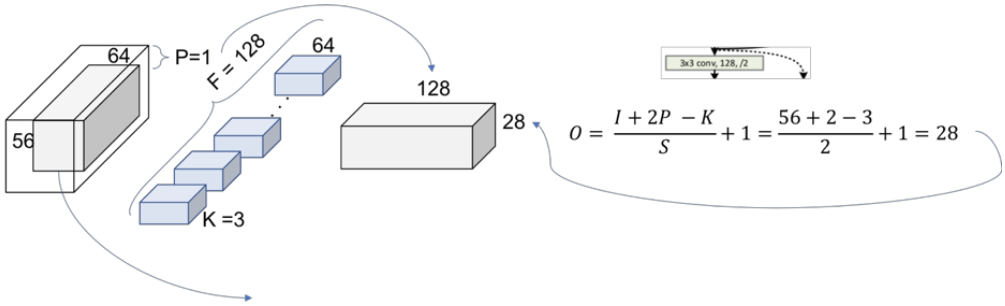


Figure 41. Layer2, Block 1, operation 1

Figure 41 represents this down sampling performed by increasing the stride to 2. The number of filters is duplicated in an attempt to preserve the time complexity for every operation $(56 \times 64) = (28 \times 128)$. Also, note that now the addition operation cannot be performed since the volume got modified. In the shortcut we need to apply one of our down sampling strategies. The 1×1 convolution approach is shown below.

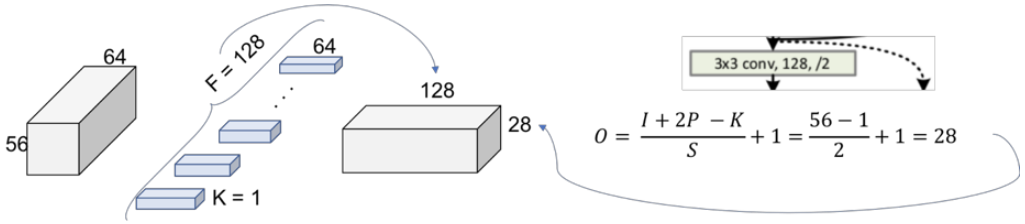


Figure 42. Projection Shortcut

The final picture looks then like in the next figure where now the 2 output volumes of each thread has the same size and can be added.

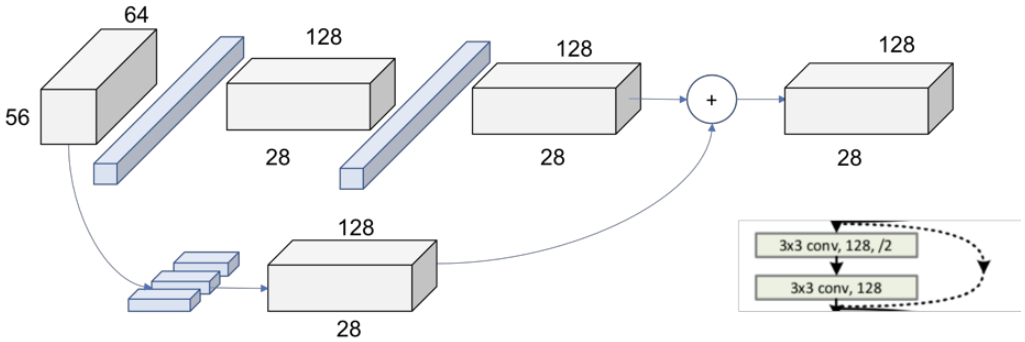


Figure 43. Layer 2, Block 1

We can see the global picture of the entire second layer. The behavior is exactly the same for the following layers 3 and 4, changing only the dimensions of the incoming volumes.

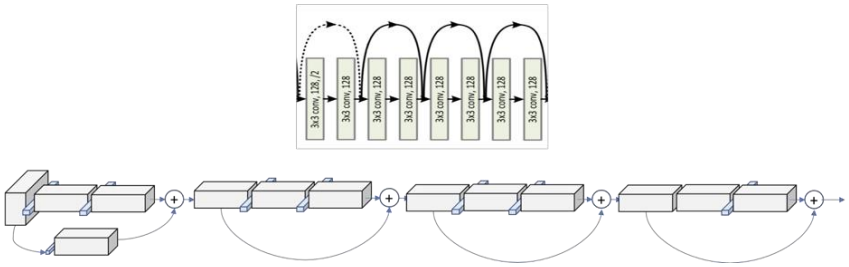


Figure 44. Layer 2

4.2.3.12 Architecture of ResNet-50

Now we'll talk about the architecture of ResNet50. The architecture of ResNet50 has 4 stages as shown in the diagram below. The network can take the input image having height, width as multiples of 32 and 3 as channel width. For the sake of explanation, we will consider the input size as $224 \times 224 \times 3$. Every ResNet architecture performs the initial convolution and max-pooling using 7×7 and 3×3 kernel sizes respectively. Afterward, Stage 1 of the network starts and it has 3 Residual blocks containing 3 layers each. The size of kernels used to perform the convolution operation in all 3 layers of the block of stage 1 are 64, 64 and 128 respectively. The curved arrows refer to the identity connection. The dashed connected arrow represents that the convolution operation in the Residual Block is performed with stride 2, hence, the size of input will be reduced to half in terms of height and width but the channel width will be doubled. As we progress from one stage to another, the channel width is doubled and the size of the input is reduced to half.

For deeper networks like ResNet50, ResNet152, etc, bottleneck design is used. For each residual function F, 3 layers are stacked one over the other. The three layers are $1 \times 1, 3 \times 3, 1 \times 1$ convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions.

Finally, the network has an Average Pooling layer followed by a fully connected layer having 1000 neurons (ImageNet class output).

ResNet – V2

Now we have discussed the ResNet50 version 1 and the ResNet50 version 2 which is all about using the pre-activation of weight layers instead of post-activation. The figure below shows the basic architecture of the post-activation (original version 1) and the pre-activation (version 2) of versions of ResNet.

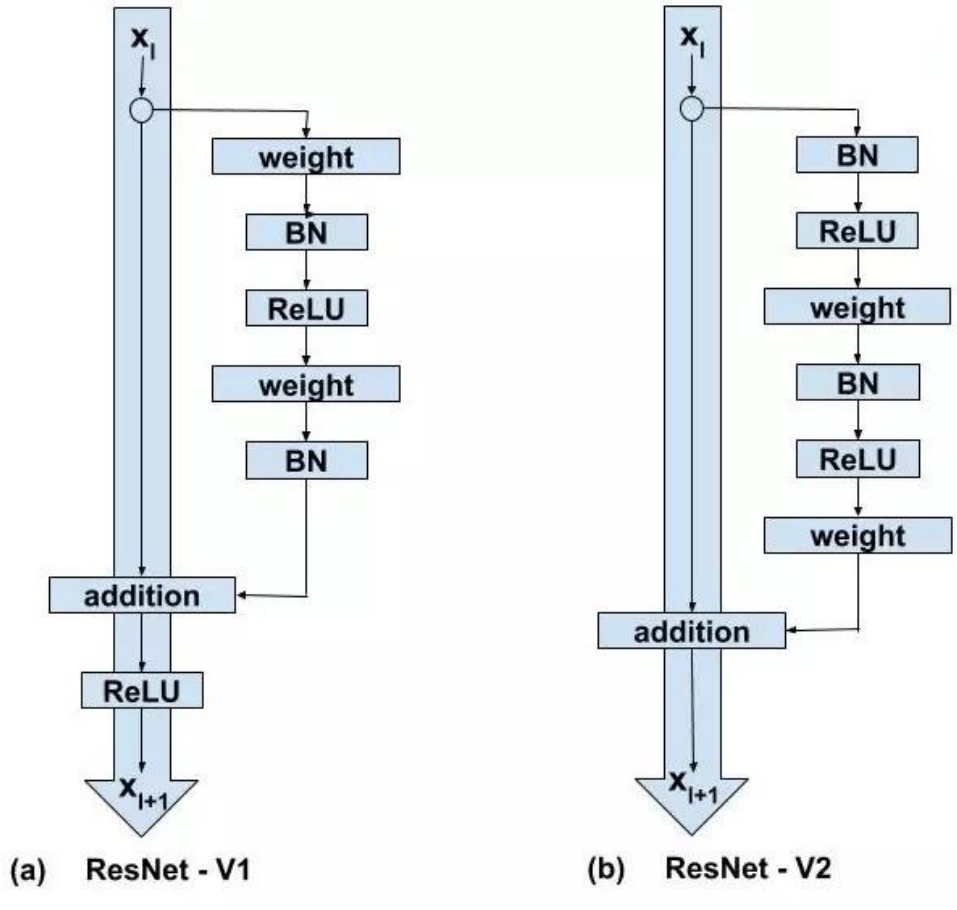


Figure 45. ResNet V1 and ResNet V2

The major differences between ResNet – V1 and ResNet – V2 are as follows:

- ResNet V1 adds the second non-linearity after the addition operation is performed in between the x and $F(x)$. ResNet V2 has removed the last non-linearity, therefore, clearing the path of the input to output in the form of identity connection.
- ResNet V2 applies Batch Normalization and ReLU activation to the input before the multiplication with the weight matrix (convolution operation). ResNet V1 performs the convolution followed by Batch Normalization and ReLU activation.

ResNet - V1	ResNet - V2
$y = x_l + F(x_l, \{W_i\})$ $x_{l+1} = H(x) = ReLU(y)$	$y = h(x_l) + F(x_l, \{W_i\})$ $x_{l+1} = H(x) = f(y)$
y = Addition Output x_{l+1} = Input to Next Block	y = Addition output $h(x_l)$ = Generalized form of input. For ResNet V1, $h(x_l) = x_l$. f = Function applied to 'y'. For ResNet V1, f = ReLU. For ResNet V2, f is an identity mapping.

Figure 46. Difference between ResNet V1 and ResNet V2

The ResNet V2 mainly focuses on making the second non-linearity as an identity mapping i.e. the output of addition operation between the identity mapping and the residual mapping should be passed as it is to the next block for further processing. However, the output of the addition operation in ResNet V1 passes from ReLU activation and then transferred to the next block as the input.

When the function 'f' is an identity function the signal can be directly propagated between any two units. Also, the gradient value calculated at the output layer can easily reach the initial layer without any change in signal.

4.2.3.13 Architecture of ResNet-18

ResNet18 [18] is a 72-layer architecture with 18 deep layers. The architecture of this network is aimed at enabling large amounts of convolutional layers to function efficiently. However, the addition of multiple deep layers to a network often results in a degradation of the output. This is known as the problem of vanishing gradient where neural networks, while getting trained through backpropagation, rely on gradient descent, descending the loss function to find the minimizing weights. Due to the presence of multiple layers, the repeated multiplication results in the gradient becoming smaller and smaller thereby "vanishing" and leading to a saturation in the network performance or even degrading the performance.

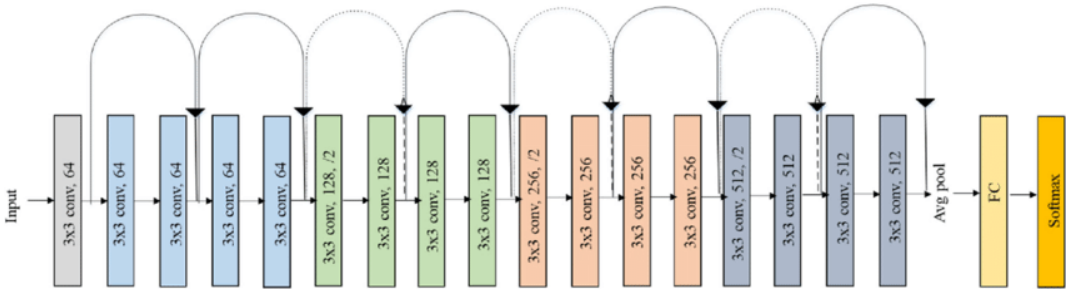


Figure 47. Original ResNet18 architecture

The primary idea of ResNet is the use of jumping connections which are mostly referred to as shortcut connections or identity connections. These connections primarily function by hopping over one or multiple layers forming shortcuts between these layers. The aim of introducing these shortcut connections was to resolve the predominant issue of vanishing gradient faced by deep networks. These shortcut connections remove the vanishing gradient issue by again using the activations of the previous layer. These identity mappings initially do not do anything much except skip the connections, resulting in the use of previous layer activations. This process of skipping the connection compresses the network; hence, the network learns faster. This compression of the connections is followed by the expansion of the layers so that the residual part of the network could also train and explore more feature space. The input size to the

network is $224 \times 224 \times 3$, which is predefined. The network is considered to be a DAG network due to its complex layered architecture and because the layers have input from multiple layers and give output to multiple layers. [19]

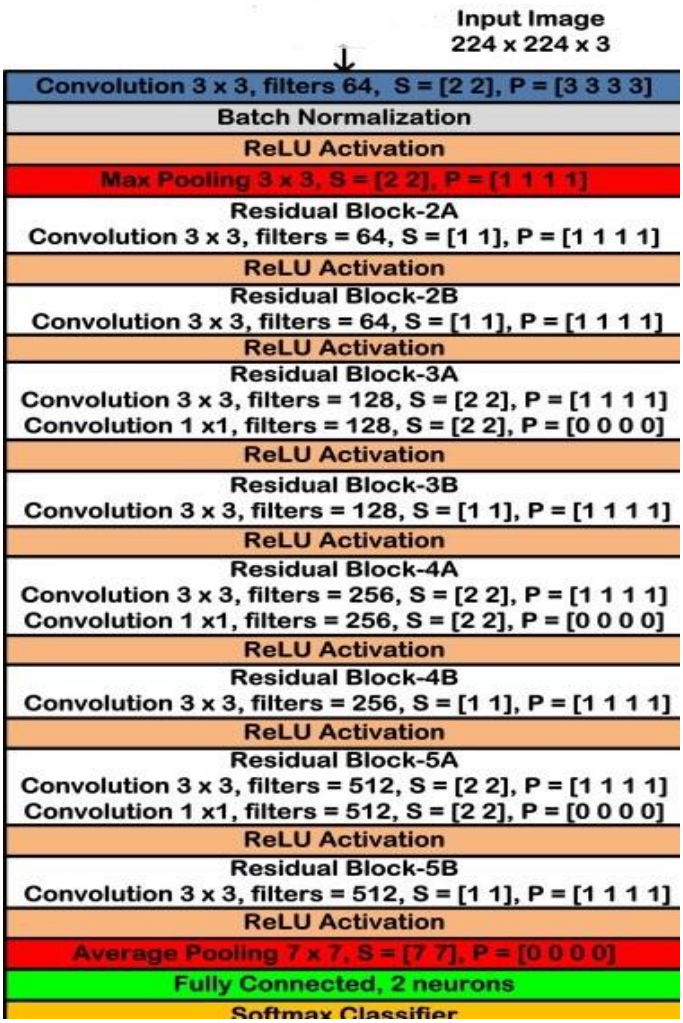


Figure 48. The layer architecture of ResNet18 CNN model

The introduction of residual blocks overcomes the problem of vanishing gradient by implementation of skip connections and identity mapping. Identity mapping has no parameters and maps the input to the output, thereby allowing the compression of the network, at first, and then exploring multiple features of the input. Fig. 47 shows a typical residual block used in ResNet18 CNN model.

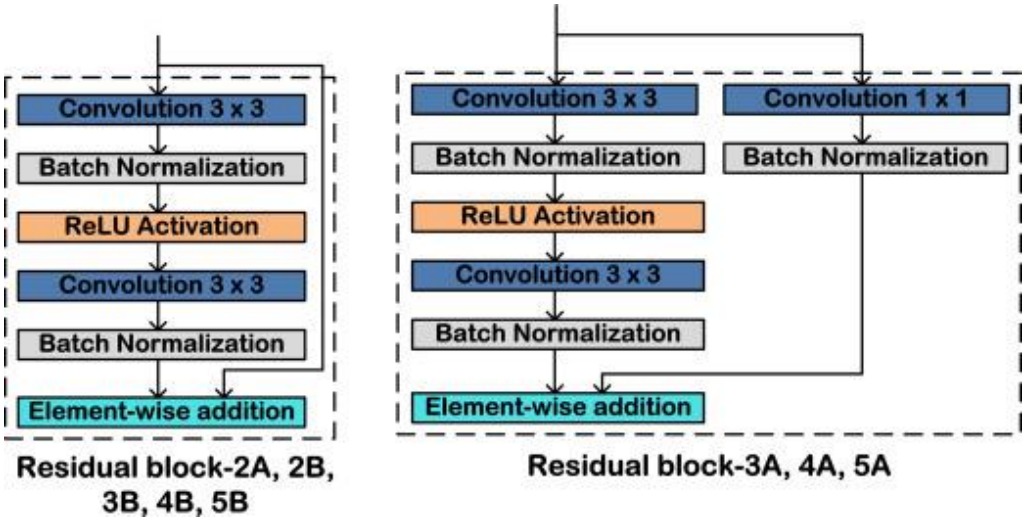


Figure 49. A typical residual block used in ResNet18 CNN model

4.2.4 Self-supervised Learning

4.2.4.1 Self-supervised Learning architecture

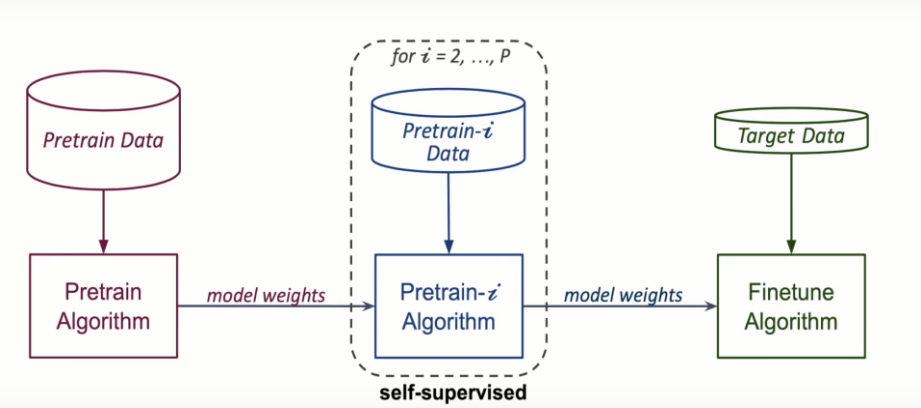


Figure 50. Self-supervised Learning architecture

4.2.4.2 What is Self-Supervised Learning?

Self-supervised learning is a subcategory under unsupervised learning because it leverages the unlabeled data. The key idea is to allow the model to learn the data representation without manual labels. Once the model learned how to represent data, and then it can be used for downstream tasks with a smaller amount of labeled data to achieve similar or better performance than the models without self-supervised learning.

It has three steps:

1. Generate the input data from the unlabeled data programmatically based on the understanding of the data.

2. Pre-training: train the model with data/labels from the previous step.

3. Fine-tune: use the pre-trained model as the initial weights to train for tasks of interests

If we use the data with manual labels instead of automatically generated labels in the second step, it'd be supervised pre-training, known as a step-in transfer learning.

4.2.4.3 Why is Self-Supervised Learning important?

Self-Supervised Learning has been successful in multiple fields i.e., text, image/video, speech, and graph. Essentially, self-supervised learning mines unlabeled data and boosts performance. This self-supervised learning can take millions of bites per sample while supervised learning (the icing) can only take 10 to 10,000 bites. That is to say, **self-supervised learning can gain more useful information from each sample than supervised learning**.

Human-generated labels usually focus on a specific view of the data. For example, we can describe an image of a horse on the grass (like the image shown below) with only one term "horse" for image recognition and provide the pixel coordinates for semantic segmentation. However, there is way more information in the data, e.g., the horse's head and tail are on the opposite side of the body (figure 22), or the horse is usually on top of grass (not underneath). The models can potentially learn better and more complex representations from the data directly instead of manual labels. Not to mention the manual labels can be wrong sometimes, which is harmful to the models.

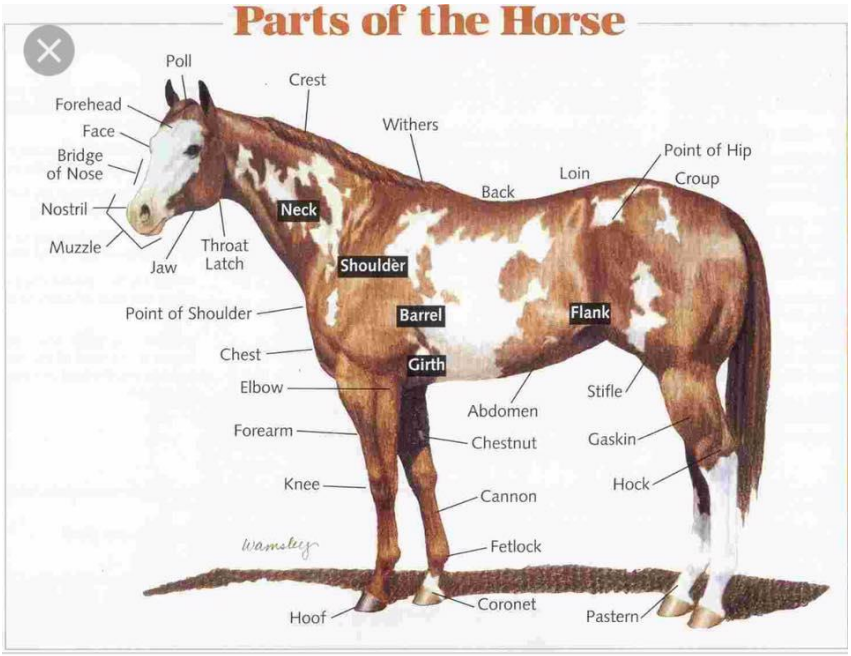


Figure 51. Part of the Horse (example)

Data labeling is costly, time-consuming, and labor-intensive. In addition, the supervised learning approaches would need different labels for new data/labels and new tasks. More importantly, it has been shown that self-supervised pre-training even outperformed supervised pre-training for image-based tasks (i.e., image recognition, object detection, semantic segmentation). In other words, **extracting information directly from data is more helpful than manual labels**. Then maybe we don't need many costly labels with more advanced self-supervised learning now or in the near future depending on the tasks.

The superiority of self-supervised learning has been validated in image-based tasks thanks to the large-scale labeled datasets in the image domain which has a longer history than other domains in the recent deep learning trend. I believe that similar superiority will be proven in other domains in the future as well. Hence, self-supervised learning is essential for advancing the machine learning field.

4.2.4.4 How can it be used?

Usually, when a self-supervised model was released, we can download the pre-trained model. Then, we can fine-tune the pre-trained model and use the fine-tuned model for a specific downstream task. For example, the most well-known example of self-supervised learning is probably BERT . BERT was pre-trained on 3.3 billion words in the self-supervised learning fashion. We can fine-tune BERT for a text-related task, such as sentence classification, with much less effort and data than training a model from scratch.

4.2.4.5 What are the categories of self-supervised learning?

Let me describe each category with a few words and dive into each of them later.

1. **Generative approaches**: recover the original information
   a. Non-autoregressive: masking a token/pixel and predicting the masked token/pixel (e.g., masked language modeling (MLM))
   b. Autoregressive: predicting the next token/pixel
2. **Predictive tasks**: design labels based on the understanding, the clustering, or the augmentation of data
   a: Predict the context (e.g., predict the relative location of the image patches, predicting whether the next fragment is the next sentence)

b: Predict the cluster id of each sample

c: Predict the image rotation angle

3. **Contrastive learning** (aka, contrastive instance discrimination): set up a binary classification problem based on positive and negative sample pairs created by augmentation
4. **Bootstrapping approaches**: use two similar but different networks to learn the same representation from the augmented pairs of the same sample
5. **Regularization**: add loss and regularization terms based on the assumptions/intuitions:

   a: the positive pairs should be similar

   b: the outputs from different samples in the same batch should be different

4.2.4.6 Bootstrap Your Own Latent (BYOL) [20]

Bootstrap Your Own Latent (BYOL):

Description of BYOL:

BYOL's goal is to learn a representation y_θ which can then be used for downstream tasks. As described previously, BYOL uses two neural networks to learn: the online and target networks. The online network is defined by a set of weights θ and is comprised of three stages: an encoder f_θ , a projector g_θ and a predictor q_θ , as shown in Figure 52 and Figure 53. The target network has the same architecture as the online network, but uses a different set of exponential moving average of the online parameters θ. More precisely, given a target decay rate $\tau[0,1]$, weights ξ. The target network provides the regression targets to train the online network, and its parameters ξ are an after each training step we perform the following update

$$\xi \leftarrow \tau\xi + (1-\tau)\theta \quad (1)$$

Given a set of images D, an image $x \sim D$ sampled uniformly from D, and two distributions of image augmentations T and T_j, BYOL produces two augmented views $v = \Delta\, t(x)$ and $v_j = \Delta\, t_j(x)$ from x by applying respectively image, augmentations $t \sim T$ and $t' \sim T'$. From the first augmented view v, the online network outputs a representation $y_\theta = \Delta f_\theta(v)$ and a projection $z_\theta = \Delta\, g_\theta(y)$. The target network outputs $y' = \Delta f_\xi(v')$ and the target projection $z' = \Delta g_\xi(y')$ from the second augmented view v'. We then output a prediction $q_\theta(z_\theta)$ of z' and l_2-normalize both

$q_\theta(z_\theta)$ and z' to $q_\theta(z_\theta) = \Delta \frac{q_\theta(z_\theta)}{||q_\theta(z_\theta)||}$ and $z' = \Delta \frac{z'}{||z'||_2}$. Note that this predictor is only applied to they online branch, making the architecture asymmetric between the online and target pipeline.

Finally, we define the following mean squared error between the normalized predictions and target projections

$$\mathcal{L}_{\theta,\xi} \triangleq \left\|\overline{q_\theta}(z_\theta) - \overline{z}'_\xi\right\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\left\|q_\theta(z_\theta)\right\|_2 \cdot \left\|z'_\xi\right\|_2}. \tag{2}$$

We symmetrize the loss $\mathcal{L}_\theta, \xi$ in Eq. 2 by separately feeding v' to the online network and v to the target network to compute $\mathcal{L}_\theta, \xi$. At each training step, we perform a stochastic optimization step to minimize ˜BYOL $\mathcal{L}_\theta, \xi = \mathcal{L}_\theta, \xi + \tilde{}\mathcal{L}_\theta, \xi$ with respect to θ only, but not ξ, as depicted by the stop-gradient in Figure 53. BYOL's dynamics are summarized as where optimizer is an optimizer and η is a learning rate

$$\begin{aligned} \theta &\leftarrow \text{optimizer}\left(\theta, \nabla_\theta \mathcal{L}_{\theta,\xi}^{\text{BYOL}}, \eta\right), \\ \xi &\leftarrow \tau\xi + (1-\tau)\theta, \end{aligned} \tag{3}$$

Main step:

1. Take two networks with same architecture: a fixed '*target' network (which is randomly initialized)* and a *trainable* '*online' network.*
2. Take an *input image t* and create an *augmented view* t_1
3. Pass *image t* through online network, *image* t_1 through target network and extract *predicted* and *target embeddings* respectively.
4. Minimize the *distance between both embeddings.*
5. Update the target network — which is the moving exponential average of the previous online networks.

Repeat steps 2–5.

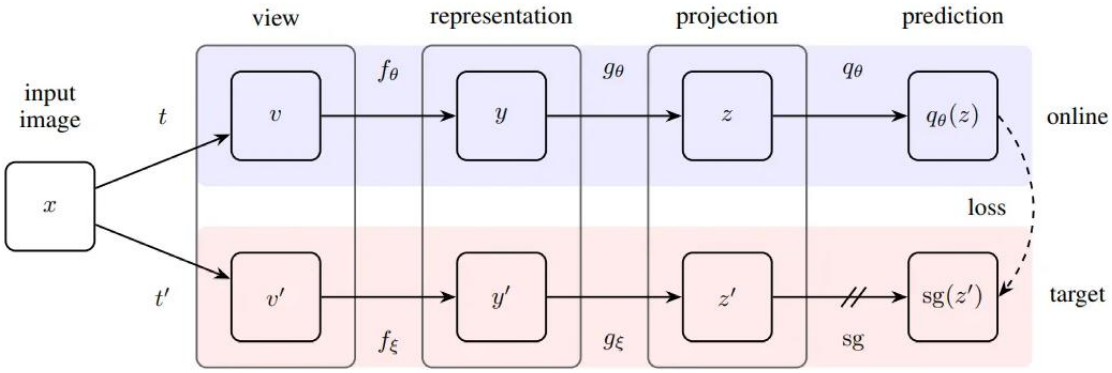


Figure 52. BYOL's architecture. BYOL minimizes a similarity loss between $q_\theta(z_\theta)$ and sg(z'), where θ are the trained weights, ξ are an exponential moving average of θ and sg means stop-gradient. At the end of training, everything but f_θ is discarded, and y_θ is used as the image representation.

The online network consists of an encoder *f*, a projector *g* and a predictor *q* — *this can be seen as a backbone network (encoder f) with a fully connected layer (projection + prediction) on top.* After training, only the *encoder f* is used for generating representations.

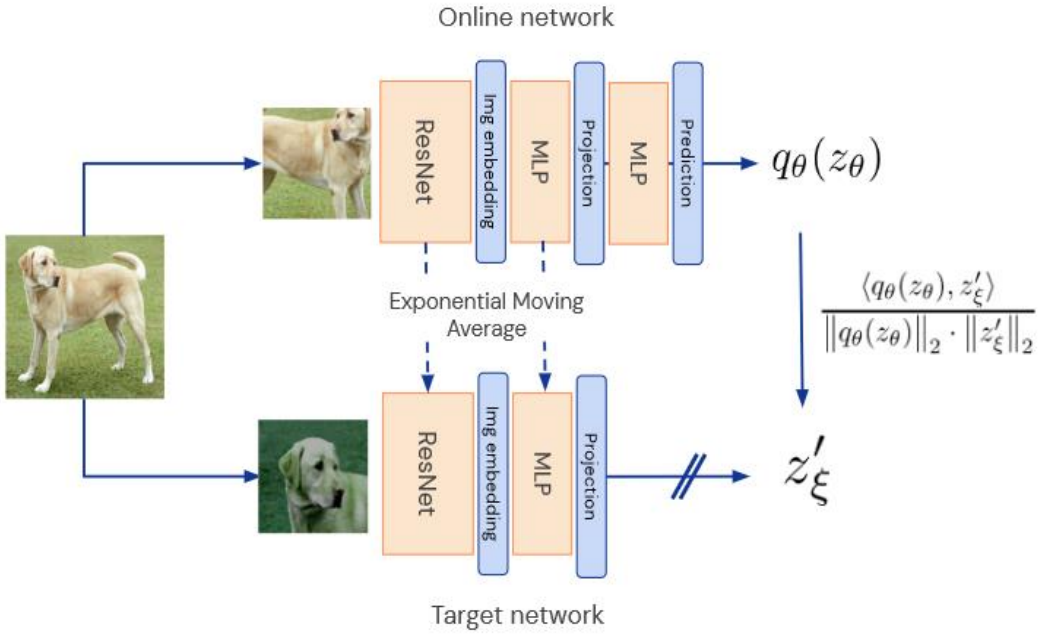


Figure 53. BYOL sketch summarizing the method by emphasizing the neural architecture. [20]

Loss function

The loss function used is mean squared error — *difference between l2 normalized online and target networks' representations.*

$$\mathcal{L}_{\theta,\xi} \triangleq \left\| \overline{q_\theta}(z_\theta) - \overline{z}'_\xi \right\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\left\| q_\theta(z_\theta) \right\|_2 \cdot \left\| z'_\xi \right\|_2}. \tag{4}$$

Where does BYOL fit into Self- Supervised Learning?

In a self-supervised learning setting where there are no labels, every image and its augmented view is passed through online and target network respectively and useful representations are learnt which can be used for downstream tasks having only few labelled data. One major advantage of BYOL is that it does not require labelled negatives which is the case in contrastive learning in general (although not in SimCLR).

How does this approach of using subsequent online networks as target network result in useful embeddings as well as not cause collapsed solution?

The fact that a network trained with a randomly initialized network as a target network produces better results than the random network itself serves as the motivation for using subsequent online networks as target networks and hence learns good representations.

How and where can we apply BYOL in a real-world setting?

For any image classification dataset with less amount of labelled data, one can learn useful representations using BYOL from unlabelled dataset, use it as initialization and then fine-tune a classifier (either the entire network or the FC layer) on the limited labelled data.

Experiment insights:

- BYOL almost matches the best supervised baseline on top-1 accuracy on ImageNet and beasts out the self-supervised baselines.
- BYOL can be successfully used for other vision tasks such as detection
- BYOL is not affected by batch size dynamics as much as SimCLR
- BYOL does not rely on the color jitter augmentation unlike SimCLR. The intuition here is that in SimCLR with just random crops the color histograms of

the augmented views are enough to differentiate the input images. In BYOL however the network is encouraged to keep as much of the information as possible to improve the online predictions.

5. RESULTS

5.1 Experiment

5.1.1 Frameworks and Libraries

- OpenCV and PIL for image processing
- Dlib 68 points landmark detection for face alignment
- Pytorch and Pytorch lightning for building deeplearning models
- Streamlit for web demo

5.1.2 Face alignment

In this project, we used Dlib 68 points landmark detection to detect landmarks and align faces from FGNet and CACD dataset. This algorithm will detect 68 facial landmarks such as the eyes, nose and mouth. With the obtained coordinates from Dlib 68 points landmark detection, we then apply alignment.

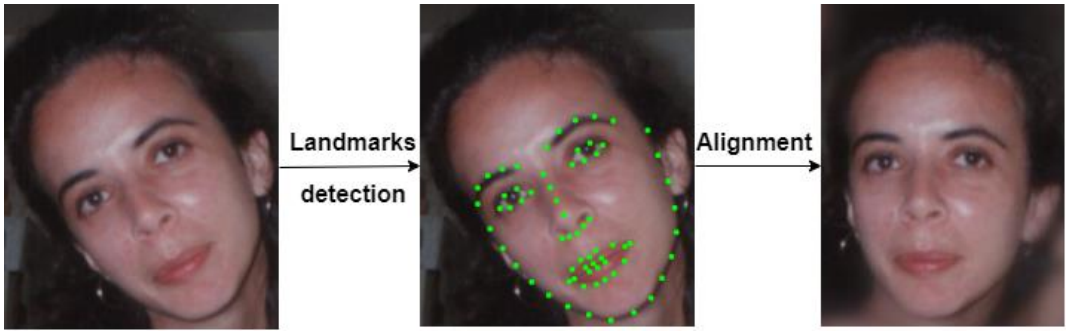


Figure 54. Example of face alignment

The below images show codes to extract and align faces.

```python
LEFT_EYE_INDICES = [36, 37, 38, 39, 40, 41]
RIGHT_EYE_INDICES = [42, 43, 44, 45, 46, 47]
def rect_to_tuple(rect):
    left2right = rect.right()-rect.left()
    left = int(rect.left() - 0.25*left2right)
    right = int(rect.right() + 0.25*left2right)
    top = int(rect.top() -0.4*left2right)
    bottom = int(rect.bottom() +0.1*left2right)
    return left, top, right, bottom

def extract_eye(shape, eye_indices):
    points = map(lambda i: shape.part(i), eye_indices)
    return list(points)

def extract_eye_center(shape, eye_indices):
    points = extract_eye(shape, eye_indices)
    xs = map(lambda p: p.x, points)
    ys = map(lambda p: p.y, points)
    return sum(xs) // 6, sum(ys) // 6

def extract_left_eye_center(shape):
    return extract_eye_center(shape, LEFT_EYE_INDICES)

def extract_right_eye_center(shape):
    return extract_eye_center(shape, RIGHT_EYE_INDICES)

def angle_between_2_points(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    tan = (y2 - y1) / (x2 - x1)
    return np.degrees(np.arctan(tan))

def get_rotation_matrix(p1, p2):
    angle = angle_between_2_points(p1, p2)
    x1, y1 = p1
    x2, y2 = p2
    xc = (x1 + x2) // 2
    yc = (y1 + y2) // 2
    M = cv2.getRotationMatrix2D((xc, yc), angle, 1)
    return M

def crop_image(image, det):
    left, top, right, bottom = rect_to_tuple(det)
    print(left)
    print(right)
    print(top)
    print(bottom)
    return image[top:bottom, left:right]
```

```python
def detect_face(image):
    image = np.array(image)
    det = detector(image, 1)[0]
    height, width = image.shape[:2]
    shape = predictor(image, det)
    left_eye = extract_left_eye_center(shape)
    right_eye = extract_right_eye_center(shape)

    M = get_rotation_matrix(left_eye, right_eye)
    rotated = cv2.warpAffine(image, M, (width, height), flags=cv2.INTER_CUBIC)

    cropped = crop_image(rotated, det)
    return cropped
```

5.1.3 Experiments on FGNet dataset

5.1.3.1 Data augmentations with GAN on FGNet dataset

To overcome data imbalance and high age variance in FGNet dataset, we used a pretrained CUSP model to generate face aging images from 20 to 60 years old, adding them to train set of FGNet dataset. The processed train set has around 30 images per class.

Table 5. Dataset after using GAN

Dataset	Ratio	Original train set	Train set with GAN
FGNet	7:3	670 images	2857 images

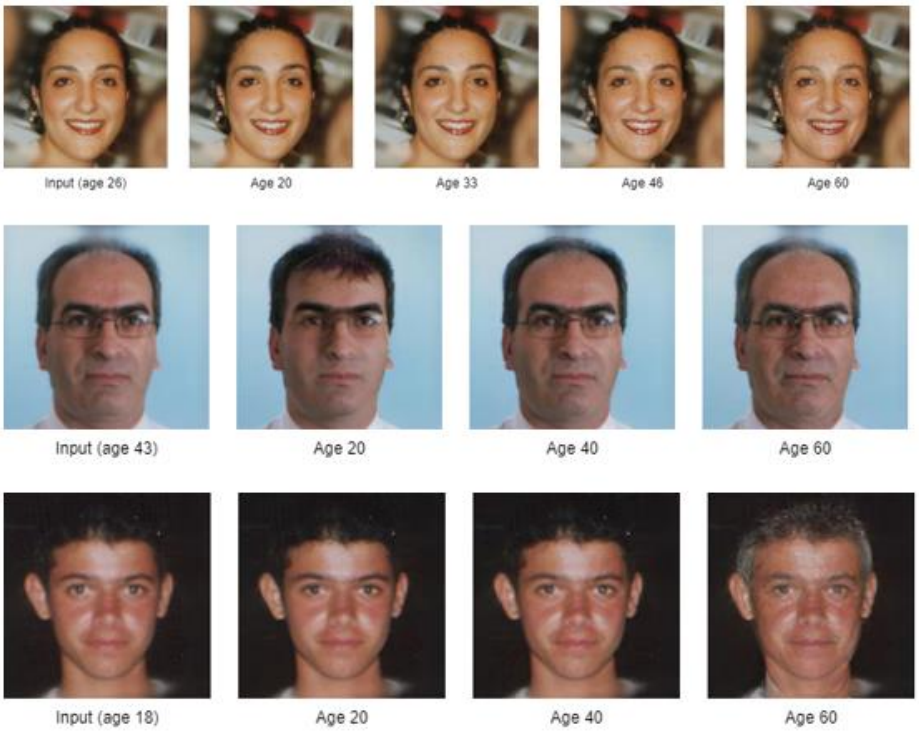


Figure 55. Some examples of generated face aging images using CUSP

5.1.3.2 Building deep learning model to train on FGNet dataset

For FGNet dataset, we will be transfer learning from a pretrained ResNet18 that is available via Pytorch.

```
from torchvision.models import resnet18

#load pretrain
model = resnet18(pretrained='True')

num_ftrs = model.fc.in_features
model.fc = nn.Sequential(
              nn.Linear(num_ftrs, 512),
              nn.ReLU(inplace=True),
              nn.Dropout(0.4),
              nn.Linear(512, len(class_names)))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ResNet                                   [8, 82]                   --
├─Conv2d: 1-1                            [8, 64, 112, 112]         9,408
├─BatchNorm2d: 1-2                       [8, 64, 112, 112]         128
├─ReLU: 1-3                              [8, 64, 112, 112]         --
├─MaxPool2d: 1-4                         [8, 64, 56, 56]           --
├─Sequential: 1-5                        [8, 64, 56, 56]           --
│    └─BasicBlock: 2-1                   [8, 64, 56, 56]           --
│    │    └─Conv2d: 3-1                  [8, 64, 56, 56]           36,864
│    │    └─BatchNorm2d: 3-2             [8, 64, 56, 56]           128
│    │    └─ReLU: 3-3                    [8, 64, 56, 56]           --
│    │    └─Conv2d: 3-4                  [8, 64, 56, 56]           36,864
│    │    └─BatchNorm2d: 3-5             [8, 64, 56, 56]           128
│    │    └─ReLU: 3-6                    [8, 64, 56, 56]           --
│    └─BasicBlock: 2-2                   [8, 64, 56, 56]           --
│    │    └─Conv2d: 3-7                  [8, 64, 56, 56]           36,864
│    │    └─BatchNorm2d: 3-8             [8, 64, 56, 56]           128
│    │    └─ReLU: 3-9                    [8, 64, 56, 56]           --
│    │    └─Conv2d: 3-10                 [8, 64, 56, 56]           36,864
│    │    └─BatchNorm2d: 3-11            [8, 64, 56, 56]           128
│    │    └─ReLU: 3-12                   [8, 64, 56, 56]           --
├─Sequential: 1-6                        [8, 128, 28, 28]          --
│    └─BasicBlock: 2-3                   [8, 128, 28, 28]          --
│    │    └─Conv2d: 3-13                 [8, 128, 28, 28]          73,728
│    │    └─BatchNorm2d: 3-14            [8, 128, 28, 28]          256
│    │    └─ReLU: 3-15                   [8, 128, 28, 28]          --
│    │    └─Conv2d: 3-16                 [8, 128, 28, 28]          147,456
│    │    └─BatchNorm2d: 3-17            [8, 128, 28, 28]          256
│    │    └─Sequential: 3-18             [8, 128, 28, 28]          8,448
│    │    └─ReLU: 3-19                   [8, 128, 28, 28]          --
│    └─BasicBlock: 2-4                   [8, 128, 28, 28]          --
│    │    └─Conv2d: 3-20                 [8, 128, 28, 28]          147,456
│    │    └─BatchNorm2d: 3-21            [8, 128, 28, 28]          256
│    │    └─ReLU: 3-22                   [8, 128, 28, 28]          --
│    │    └─Conv2d: 3-23                 [8, 128, 28, 28]          147,456
│    │    └─BatchNorm2d: 3-24            [8, 128, 28, 28]          256
│    │    └─ReLU: 3-25                   [8, 128, 28, 28]          --
├─Sequential: 1-7                        [8, 256, 14, 14]          --
│    └─BasicBlock: 2-5                   [8, 256, 14, 14]          --
│    │    └─Conv2d: 3-26                 [8, 256, 14, 14]          294,912
│    │    └─BatchNorm2d: 3-27            [8, 256, 14, 14]          512
│    │    └─ReLU: 3-28                   [8, 256, 14, 14]          --
│    │    └─Conv2d: 3-29                 [8, 256, 14, 14]          589,824
│    │    └─BatchNorm2d: 3-30            [8, 256, 14, 14]          512
│    │    └─Sequential: 3-31             [8, 256, 14, 14]          33,280
│    │    └─ReLU: 3-32                   [8, 256, 14, 14]          --
│    └─BasicBlock: 2-6                   [8, 256, 14, 14]          --
│    │    └─Conv2d: 3-33                 [8, 256, 14, 14]          589,824
│    │    └─BatchNorm2d: 3-34            [8, 256, 14, 14]          512
│    │    └─ReLU: 3-35                   [8, 256, 14, 14]          --
│    │    └─Conv2d: 3-36                 [8, 256, 14, 14]          589,824
│    │    └─BatchNorm2d: 3-37            [8, 256, 14, 14]          512
│    │    └─ReLU: 3-38                   [8, 256, 14, 14]          --
├─Sequential: 1-8                        [8, 512, 7, 7]            --
│    └─BasicBlock: 2-7                   [8, 512, 7, 7]            --
│    │    └─Conv2d: 3-39                 [8, 512, 7, 7]            1,179,648
│    │    └─BatchNorm2d: 3-40            [8, 512, 7, 7]            1,024
│    │    └─ReLU: 3-41                   [8, 512, 7, 7]            --
│    │    └─Conv2d: 3-42                 [8, 512, 7, 7]            2,359,296
│    │    └─BatchNorm2d: 3-43            [8, 512, 7, 7]            1,024
│    │    └─Sequential: 3-44             [8, 512, 7, 7]            132,096
│    │    └─ReLU: 3-45                   [8, 512, 7, 7]            --
│    └─BasicBlock: 2-8                   [8, 512, 7, 7]            --
│    │    └─Conv2d: 3-46                 [8, 512, 7, 7]            2,359,296
│    │    └─BatchNorm2d: 3-47            [8, 512, 7, 7]            1,024
│    │    └─ReLU: 3-48                   [8, 512, 7, 7]            --
│    │    └─Conv2d: 3-49                 [8, 512, 7, 7]            2,359,296
│    │    └─BatchNorm2d: 3-50            [8, 512, 7, 7]            1,024
│    │    └─ReLU: 3-51                   [8, 512, 7, 7]            --
├─AdaptiveAvgPool2d: 1-9                 [8, 512, 1, 1]            --
├─Sequential: 1-10                       [8, 82]                   --
│    └─Linear: 2-9                       [8, 512]                  262,656
│    └─ReLU: 2-10                        [8, 512]                  --
│    └─Dropout: 2-11                     [8, 512]                  --
│    └─Linear: 2-12                      [8, 82]                   42,066
==========================================================================================
Total params: 11,481,234
Trainable params: 11,481,234
Non-trainable params: 0
Total mult-adds (G): 14.51
==========================================================================================
Input size (MB): 4.82
Forward/backward pass size (MB): 317.95
Params size (MB): 45.92
Estimated Total Size (MB): 368.70
==========================================================================================
```

Figure 56. Model's architecture

5.1.3.3 Pretraining with self-supervised learning

After initializing the model, we use BYOL to pretrain ResNet18 on FGNet dataset.

```python
from copy import deepcopy
from itertools import chain
from typing import Dict, List

import pytorch_lightning as pl
from torch import optim
import torch.nn.functional as f


def normalized_mse(x: Tensor, y: Tensor) -> Tensor:
    x = f.normalize(x, dim=-1)
    y = f.normalize(y, dim=-1)
    return 2 - 2 * (x * y).sum(dim=-1)


class BYOL(pl.LightningModule):
    def __init__(
        self,
        model: nn.Module,
        image_size: Tuple[int, int] = (128, 128),
        hidden_layer: Union[str, int] = -2,
        projection_size: int = 256,
        hidden_size: int = 4096,
        augment_fn: Callable = None,
        beta: float = 0.999,
        **hparams,
    ):
        super().__init__()
        self.augment = default_augmentation(image_size) if augment_fn is None else augment_fn
        self.beta = beta
        self.encoder = EncoderWrapper(
            model, projection_size, hidden_size, layer=hidden_layer
        )
        self.predictor = nn.Linear(projection_size, projection_size, hidden_size)
        # self.hparams = hparams
        self._target = None

        self.encoder(torch.zeros(2, 3, *image_size))

    def forward(self, x: Tensor) -> Tensor:
        return self.predictor(self.encoder(x))

    @property
    def target(self):
        if self._target is None:
            self._target = deepcopy(self.encoder)
        return self._target

    def update_target(self):
        for p, pt in zip(self.encoder.parameters(), self.target.parameters()):
            pt.data = self.beta * pt.data + (1 - self.beta) * p.data

    # --- Methods required for PyTorch Lightning only! ---

    def configure_optimizers(self):
        optimizer = getattr(optim, self.hparams.get("optimizer", "Adam"))
        lr = self.hparams.get("lr", 1e-4)
        weight_decay = self.hparams.get("weight_decay", 1e-6)
        return optimizer(self.parameters(), lr=lr, weight_decay=weight_decay)

    def training_step(self, batch, *_) -> Dict[str, Union[Tensor, Dict]]:
        x = batch[0]
        with torch.no_grad():
            x1, x2 = self.augment(x), self.augment(x)

        pred1, pred2 = self.forward(x1), self.forward(x2)
        with torch.no_grad():
            targ1, targ2 = self.target(x1), self.target(x2)
        loss = torch.mean(normalized_mse(pred1, targ2) + normalized_mse(pred2, targ1))

        self.log("train_loss", loss.item())
        self.update_target()

        return {"loss": loss}

    @torch.no_grad()
    def validation_step(self, batch, *_) -> Dict[str, Union[Tensor, Dict]]:
        x = batch[0]
        x1, x2 = self.augment(x), self.augment(x)
        pred1, pred2 = self.forward(x1), self.forward(x2)
        targ1, targ2 = self.target(x1), self.target(x2)
        loss = torch.mean(normalized_mse(pred1, targ2) + normalized_mse(pred2, targ1))

        return {"loss": loss}

    @torch.no_grad()
    def validation_epoch_end(self, outputs: List[Dict]) -> Dict:
        val_loss = sum(x["loss"] for x in outputs) / len(outputs)
        self.log("val_loss", val_loss.item())
```

We pretrain the model for 50 epochs using Adam optimizer with learning rate of 1e-4, weight decay of 1e-6 and batch size of 64.

```
model_SSL = resnet18(pretrained=True)
model_SSL.fc = nn.Sequential(
               nn.Linear(num_ftrs, 512),
               nn.ReLU(inplace=True),
               nn.Dropout(0.4),
               nn.Linear(512, len(class_names)))
byol = BYOL(model_SSL, image_size=(224, 224))
trainer = pl.Trainer(
    max_epochs=50,
    gpus=-1,
    logger=logger,
    accumulate_grad_batches=2048 // 128,
    callbacks = [checkpoint_callback],
    enable_model_summary=None,
)

trainer.fit(byol, train_dataloader, val_dataloader)
```

5.1.3.4 Supervised training on FGNet with weights obtained from pretraining

After pretraining the model, we perform supervised training on FGNet using the weights from self-supervised learning to further improve the model's performance.

```
# Load pretrained weights from self-supervised learning
model_improved = resnet18()
model_improved.fc = nn.Sequential(
               nn.Linear(num_ftrs, 512),
               nn.ReLU(inplace=True),
               nn.Dropout(0.4),
               nn.Linear(512, len(class_names)))
model_improved.load_state_dict(torch.load('FGNet_BYOL_baseline_GAN_r18_50e.pth'))
```

For supervised learning task, we use Cross Entropy Loss, Adam optimizer with learning rate of 1e-4, weight decay of 1e-6 and batch size of 64 and maximum amount of epoch is 100.

We also use ModelCheckpoint and EarlyStopping callbacks to save the best model and stop the training process if the model overfits.

```
#model checkpoint
checkpoint_callback = ModelCheckpoint(
                                      dirpath='./checkpoint_improved/',
                                      filename='sample-CACD-{epoch:02d}-{val_loss:.2f}-val{val_acc:.3f}',
                                      save_top_k = 5,
                                      monitor='val_loss',
                                      mode='min'
                                      )
#early stopping
early_stop_callback = EarlyStopping(
                                    monitor="val_loss",
                                    mode="min",
                                    patience=10)
```

This is performance of supervised learning:

```
supervised = SupervisedLightningModule(model_improved)
trainer = pl.Trainer(
    max_epochs=100,
    logger=logger,
    gpus=-1,
    callbacks = [checkpoint_callback, early_stop_callback],
    enable_model_summary=None,
)

trainer.fit(supervised, train_dataloader, val_dataloader)
```

5.1.4 Experiments on CACD dataset

5.1.4.1 Building deeplearning model to train on CACD dataset

For CACD dataset, we will be transfer learning from a pretrained ResNet50.

```
from torchvision.models import resnet50

#load pretrain
model = resnet50(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Sequential(
                nn.Linear(num_ftrs, 512),
                nn.ReLU(inplace=True),
                nn.Dropout(0.4),
                nn.Linear(512, len(class_names)))
```

5.1.4.2 Pretraining with self-supervised learning

After initializing the model, we use the same BYOL training procedure as in FGNet section to pretrain ResNet50 on CACD dataset.

We pretrain the model for 50 epochs using Adam optimizer with learning rate of 1e-4, weight decay of 1e-6 and batch size of 128.

```
model_SSL = resnet50(weights = 'ResNet50_Weights.IMAGENET1K_V1')
model_SSL.fc = nn.Sequential(
                nn.Linear(num_ftrs, 512),
                nn.ReLU(inplace=True),
                nn.Dropout(0.4),
                nn.Linear(512, len(class_names)))
byol = BYOL(model_SSL, image_size=(224, 224))
trainer = pl.Trainer(
    max_epochs=50,
    gpus=-1,
    accumulate_grad_batches=2048 // 128,
    callbacks = [checkpoint_callback],
    enable_model_summary=None,
)

trainer.fit(byol, train_dataloader, val_dataloader)
```

5.1.4.3 Supervised training on CACD with weights obtained from pretraining

With the pretrained weights obtained form self-supervised learning, we then load the weights to a new ResNet50 and perform supervised learning on CACD dataset to further improve model's performance.

```
# Extract the state dictionary, initialize a new ResNet50 model,
# and load the state dictionary into the new model.
#
# This ensures that we remove all hooks from the previous model,
# which are automatically implemented by BYOL.
state_dict = model_SSL.state_dict()
model_improved = resnet50()
model_improved.fc = nn.Sequential(
                nn.Linear(num_ftrs, 512),
                nn.ReLU(inplace=True),
                nn.Dropout(0.4),
                nn.Linear(512, len(class_names)))
model_improved.load_state_dict(state_dict)
```

For supervised learning task, we use Cross Entropy Loss, Adam optimizer with learning rate of 1e-4, weight decay of 1e-6 and batch size of 128 and maximum amount of epoch is 70. CheckpointModel is used to save the best model.

```
checkpoint_callback = ModelCheckpoint(
                                    dirpath='./checkpoint_improved/',
                                    filename='sample-CACD-{epoch:02d}-{val_loss:.2f}',
                                    save_top_k = 1,
                                    monitor='val_loss',
                                    mode='min'
                                    )
```

```
supervised = SupervisedLightningModule(model_improved)
trainer = pl.Trainer(
    max_epochs=70,
    gpus=-1,
    callbacks = checkpoint_callback,
    enable_model_summary=None,
)

trainer.fit(supervised, train_dataloader, val_dataloader)
```

5.2 Results

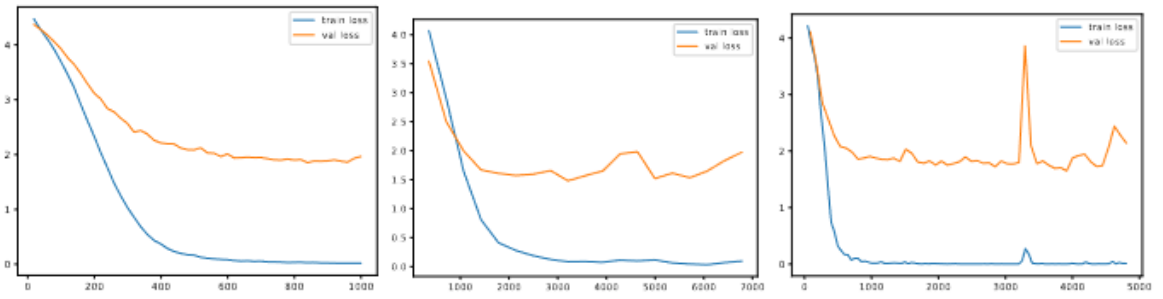
5.2.1 Overview

```
img = Image.open('/content/008A31.JPG')
img.show()
for i in indices[0][:5]:
  print(class_names[i], prob[i].item())
```

```
008 98.1402359008789
011 0.6586174964904785
009 0.2019803524017334
062 0.105869360268116
014 0.10301120579242706
```

Figure 57. Test on image have 98% confidence

5.2.2 On FGNet dataset



(a) Baseline loss on FGNET (b) Loss after using GAN on FGNET (c) Loss after using self-supervised learning on FGNET

Figure 58. Result on FGNet dataset

Baseline accuracy test set on FGNet dataset:

```
[ ] test2 = SupervisedLightningModule.load_from_checkpoint('/content/drive/MyDrive/MINH/7-3/sample-fgnet-epoch=39-val_loss=1.93.ckpt')
    test2.eval()


    test2.cuda()
    acc = sum([accuracy(test2(x.cuda()), y.cuda()) for x, y in test_dataloader]) / len(test_dataloader)
    print(f"Accuracy: {acc:.3f}")
    #clear cache
    torch.cuda.empty_cache()

    Accuracy: 0.513
```

ResNet18 accuracy when adding generated images to train set:

```
: test = SupervisedLightningModule.load_from_checkpoint('./checkpoint_model_gan_r18/FGNet-epoch=15-val_loss=1.53-val_acc=0.660.ckpt
  test.eval()
  test.cuda()
  acc = sum([accuracy(test(x.cuda()), y.cuda()) for x, y in test_dataloader]) / len(test_dataloader)
  print(f"Accuracy: {acc:.3f}")

  Accuracy: 0.579
```

Self-Supervised Learning test set validation on FGNet dataset:

```
  test2 = SupervisedLightningModule.load_from_checkpoint('/content/drive/MyDrive/MINH/7-3/improved-fgnet-epoch=22-val_loss=1.47.ckpt')
  test2.eval()


  test2.cuda()
  acc = sum([accuracy(test2(x.cuda()), y.cuda()) for x, y in test_dataloader]) / len(test_dataloader)
  print(f"Accuracy: {acc:.3f}")
  #clear cache
  torch.cuda.empty_cache()

  Accuracy: 0.519
```

SSL accuracy when adding generated images to FGNet train set:

```
test = SupervisedLightningModule.load_from_checkpoint('./checkpoint_improved_GAN/FGNet-epoch=31-val_loss=1.72-valval_acc=0.000.ck
test.eval()
test.cuda()
acc = sum([accuracy(test(x.cuda()), y.cuda()) for x, y in test_dataloader]) / len(test_dataloader)
print(f"Accuracy: {acc:.3f}")

Accuracy: 0.593
```

5.2.3 On CACD dataset

Baseline accuracy validation on CACD dataset:

```
[ ] test = SupervisedLightningModule.load_from_checkpoint(t[0])
    test.eval()


    test.cuda()
    acc = sum([accuracy(test(x.cuda()), y.cuda()) for x, y in val_dataloader]) / len(val_dataloader)
    print(f"Accuracy: {acc:.3f}")

    INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.9.4 to v2.0.0. To
    Accuracy: 0.757
```

Self-Supervised Learning accuracy validation on CACD dataset:

```
  t = glob.glob('./checkpoint_improved/*')
  name = t[0].split("/")[-1]

  test2 = SupervisedLightningModule.load_from_checkpoint(t[0])
  test2.eval()


  test2.cuda()
  acc = sum([accuracy(test2(x.cuda()), y.cuda()) for x, y in val_dataloader]) / len(val_dataloader)
  print(f"Accuracy: {acc:.3f}")

  #clear cache
  torch.cuda.empty_cache()

  INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.9.4 to v2.0.0. To apply the upgrade to your files
  Accuracy: 0.767
```

Baseline accuracy on CACD test set:

```
test = SupervisedLightningModule.load_from_checkpoint(t[0])
test.eval()


test.cuda()
acc = sum([accuracy(test(x.cuda()), y.cuda()) for x, y in test_dataloader]) / len(test_dataloader)
print(f"Accuracy: {acc:.3f}")
```

```
Accuracy: 0.766
```

Self-Supervised Learning accuracy on CACD test set:

```
test = SupervisedLightningModule.load_from_checkpoint(t[0])
test.eval()


test.cuda()
acc = sum([accuracy(test(x.cuda()), y.cuda()) for x, y in test_dataloader]) / len(test_dataloader)
print(f"Accuracy: {acc:.3f}")
```

```
Accuracy: 0.777
```

5.2.4 Prototype demo

Demo on CACD dataset

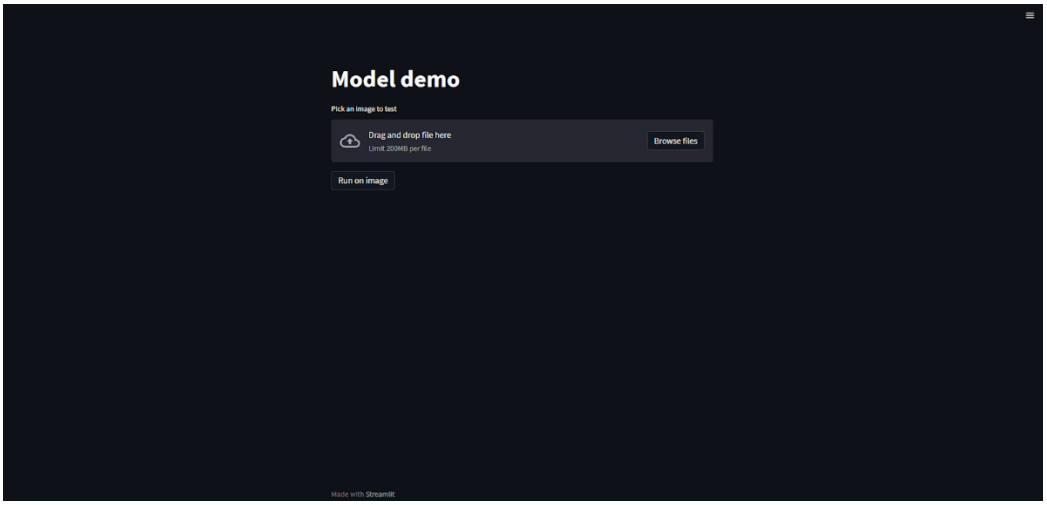


Figure 59. Demo' page main

Step 1: Click **Browse files** to select and upload an image

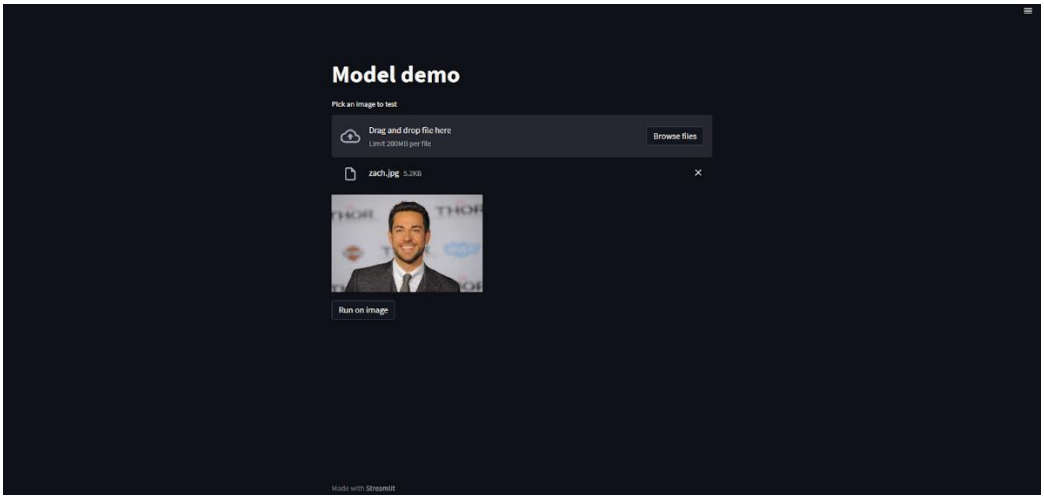


Figure 60. Upload an image

Step 2: Click **Run on image** to run model, model will detect and run prediction on detected face.

Detected face and predictions are shown below:

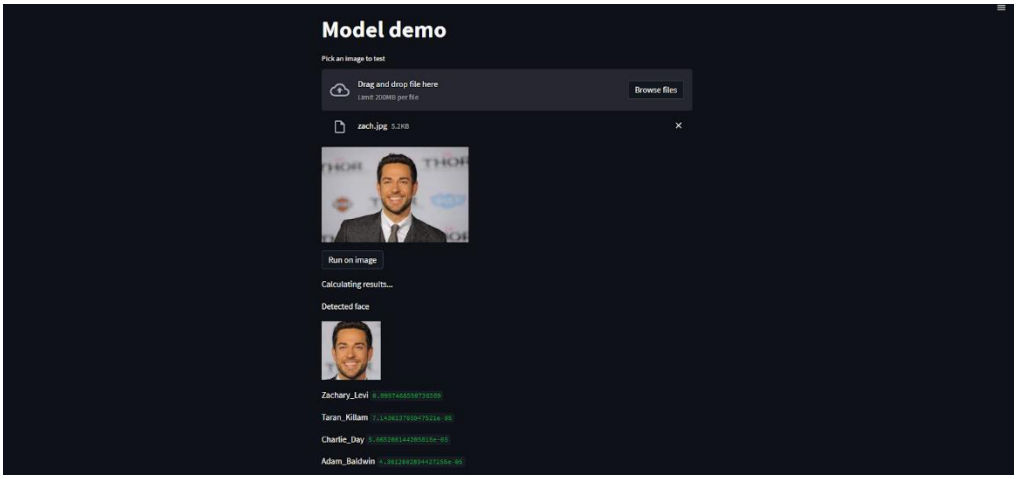


Figure 61. Result on demo

5.2.5 Review of the experimental process

CACD dataset has the number of images for 1 adult, so when generating data, the accuracy of the model has reached 78%. Without hardware and time limitations, there is a possibility for higher accuracy because of the strong self-learning ability of the self-supervised learning method on large data sets.

The FGNET dataset has a low number of images per person, averaging 12 images per person, which is too small for model training. Accuracy is low when training the same model and number of data generation.

5.2.6 Compare with other studies on the same topic

Due to many objective factors, hardware configuration, unstable experimental environment (google colab), the results of the method after many training implementations change, and here will take the best index. experimental time for comparison.

5.2.6.1 On FGNet dataset

The table compares the experimental results of the previous methods [10], and the experimental results of the method in the study.

Table 6. Compare experimental results on the dataset FGNet

Methods	Accuracy	
3D aging model (2010)	37.4%	
Discriminative aging model (2011)	47.5%	
Hidden factor analysis model (2013)	69.0%	
Feature-aging model (2015)	71.3%	
Maximum entropy model (2015)	76.2%	
Pretrained Model + SLL (our method)	**Test set**	**Validation set**
	59.3%	75.3%

5.2.6.2 On CACD dataset

The table compares the experimental results of the previous methods, and the experimental results of the method in the study.

Table 7. Compare experimental results on the dataset CACD

Methods	Accuracy	
High-Dimensional LBP [21]	81.6%	
HFA [22]	84.4%	
CARC [23]	87.6%	
LF-CNNs	98.5%	
Human, Average [24]	85.7%	
Human, Voting [24]	94.2%	
Pretrained Model + SLL (our method)	**Test set**	**Validation set**
	77.9%	76.7%

5.2.6.3 Compare the accuracy after applying SSL

The table shows the results obtained after applying the self-supervised learning method to the age-invariant face recognition model.

Table 8. The results after applying SSL

Sets / Dataset	Test set		Validation set	
	Baseline	**BYOL**	**Baseline**	**BYOL**
FGNet	51.3%	59.3%	69.1%	75.3%
CACD	77.2%	77.9%	75.7%	76.7%

Both data sets have an increase in accuracy after applying Self-Supervised Learning technique. Especially, there is a strong increase in the FGNet dataset - a challenging dataset in this topic

5.2.7 Research paper

The experimental results demonstrate the potential of self-supervised learning for age-invariant face recognition. This advanced technique is effective even on low-precision models and has improved the model's performance significantly. Notably, it has yielded promising results on the challenging FGNET dataset, with an accuracy increase of 8% on the test set and 6.2% on the validation set. We anticipate even more remarkable results when applied to more powerful models.

We used this result to write a research paper called "Using Self-Supervised Learning to improve Age Invariant Face Recognition". The paper has now been submitted to the conference "The 2nd International Conference on Intelligence of Things 2023". We are waiting for the paper to be approved!

Figure 62. The 2nd International Conference on Intelligence of Things 2023

6. DICUSSION

This thesis has proposed an experimental method for age-independent face recognition by computer vision. The goal is to gradually improve the accuracy of recognition and explore the applicability of self-supervised learning techniques in age-invariant face recognition.

The findings of this study contribute to the development of age-invariant face recognition models through the integration of self-supervised learning. Moreover, the methods outlined in this research can be adapted and implemented in other related studies to enhance their models.

However, this result is still not optimal and needs to be studied and improved further. Specifically, it is necessary to learn and research more for the MobileNet network, ... because this is the support network for applications on the smartphone platform.

Another problem is due to the limitation of the computer infrastructure, as well as the GPU is not powerful enough to experiment on many other data sets. So just stop at the experimental level. In the future, if conditions permit, they will experiment on a larger scale to get more objective results.

7. CONCLUSIONS AND PERSPECTIVES

7.1 Contributions and limitations

7.1.1 Contributions

First, the thesis has contributed to the research community a new and more modern direction with the combination of new and more applicable technologies in the explosive revolution 4.0, helping to identify more efficient, simpler in the field of computer vision.

Second, experiments using hand-crafted algorithms and CNN, have shown that with complex image data types, the use of models combining technologies such as Landmark 68 and CNN is a good choice.

Third, the research results have shown that the self-supervised learning technique has high applicability in the field of face recognition in general and age-invariant face recognition in particular.

Besides, through the results achieved, it will inspire research in the field of identification in the real environment, thereby effectively applying it in life.

7.1.2 Limitations

First, the methods indicated the need for a powerful enough computer and enough time to run all the feature extraction techniques. It is also necessary to use the latest CNN networks with better tuning methods.

Second, the accuracy is not really optimal, if there is enough hardware, time and stable environment can be further improved.

Third, more research is needed on CNN networks such as adjusting layers and parameters, reducing computation time to improve accuracy in recognition.

Fourth, it is necessary to learn and experiment with other self-supervised learning models to test and compare techniques.

In order to overcome the above limitations, it takes time and funds to implement, and further research is needed in the future.

7.2 Future works and conclusion

7.2.1 Future works

With the above contributions and limitations, we give some development orientations of the topic in the future:

- Continue to increase the number of datasets to experiment with, along with increasing the number of new human strains (classes).
- Continue to research unsupervised learning methods, especially self-supervised learning because of the large and rich data source.
- Continuing to research and improve feature extraction techniques on hand-crafted algorithms.
- Continue to study Deep learning models to increase time-invariant facial recognition results.
- Develop applications based on the results of experiments to identify in practice.
- Newly generated data will be collected and analyzed from the application deployed and operated in reality. On that basis, reinforcement learning models will be applied to self-learn untrained samples
- Developing a prototype of human identification to apply in the field of finding lost children

7.2.2 Conclusion

Through the process of experiment, comparison and evaluation, it shows that CNN-based methods have given good results (approximately 78% in our study). Because CNNs are designed with multiple layers for self-learning of features, it is possible to learn optimal features directly from the raw pixels of an image. The learned synthetic features are not limited only to shape, texture or color, but also extend to specific types of face features such as structural divisions, eyes, nose, ears, etc.

Experimental results also show that self-supervised learning can create a breakthrough for models in the field of age-invariant face recognition. In particular, this technique clearly demonstrates the ability of this technique to improve the model with low initial accuracy (approximately 6%).

In general, age-invariant face recognition for real-life applications is still challenging and a difficult task for the computer vision industry.

In the future, we will continue to add more new technology combinations to be able to apply this technology to serve the necessary fields of Vietnam.

8. APPENDIX

8.1 Dataset and Source code

Dataset	LINK
Source code	LINK

8.2 Research paper

After conducting the experimental steps, we have calibrated and extracted the best parts of the research to make a scientific paper called "Using Self-Supervised Learning to improve Age Invariant Face Recognition". The paper has now been submitted to the conference "The 2nd International Conference on Intelligence of Things 2023" - an International Conference on the current state of technology and the outcome of ongoing research in the area of the Internet of things, the intelligence of things (IoT2 /AIoT), and related fields of Information Technology, technically sponsored by Springer.

The 2nd ICIT 2023 be organized by Ho Chi Minh City University of Technology (HCMUT), Hanoi University of Mining and Geology (HUMG), Vietnam National University of Agriculture (VNUA), Ho Chi Minh City Open University, and Quy Nhon University.

ICIT 2023 will be hosted by Ho Chi Minh City University of Technology (HCMUT) in Ho Chi Minh City, Vietnam from October 25 to October 27, 2023.

We are waiting for the paper to be approved!

9. REFERENCES

[1] Baruni, Kedimotse and Mokoena, Nthabiseng and Veeraragoo, Mahalingam and Holder, Ross, “Age Invariant Face Recognition Methods: A Review,” trong *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2021, pp. 1657-1662.

[2] Bijarnia, Saroj and Singh, Preety, "Age Invariant Face Recognition Using Minimal Geometrical Facial Features," in *Advanced Computing and Communication Technologies*, Singapore, Springer Singapore, 2016, pp. 71-77.

[3] Li, Zhifeng and Gong, Dihong and Li, Xuelong and Tao, Dacheng, "Aging Face Recognition: A Hierarchical Learning Model Based on Local Patterns Selection," *IEEE Transactions on Image Processing*, vol. 25, pp. 2146-2154, 2016.

[4] Du, Ji-Xiang and Zhai, Chuan-Min and Ye, Yong-Qing, "Face Aging Simulation Based on NMF Algorithm with Sparseness Constraints," in *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2012, pp. 516--522.

[5] Gong, Dihong and Li, Zhifeng and Lin, Dahua and Liu, Jianzhuang and Tang, Xiaoou, “Hidden Factor Analysis for Age Invariant Face Recognition,” trong *2013 IEEE International Conference on Computer Vision*, 2013, pp. 2872-2879.

[6] Li, Haoxi and Zou, Haoshan and Hu, Haifeng, "Modified Hidden Factor Analysis for Cross-Age Face Recognition," *IEEE Signal Processing Letters*, vol. 24, pp. 465-469, 2017.

[7] Chenfei Xu and Qihe Liu and Mao Ye, "Age invariant face recognition and retrieval by coupled auto-encoder networks," *Neurocomputing*, vol. 222, pp. 62-71, 2017.

[8] Wen, Yandong and Li, Zhifeng and Qiao, Yu, "Latent Factor Guided Convolutional Neural Networks for Age-Invariant Face Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4893-4901.

[9] Zheng, Tianyue and Deng, Weihong and Hu, Jiani, "Age Estimation Guided Convolutional Neural Network for Age-Invariant Face Recognition," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 503-511.

[10] Yitong Wang and Dihong Gong and Zheng Zhou and Xing Ji and Hao Wang and Zhifeng Li and W. Liu and T. Zhang, "Orthogonal Deep Features Decomposition for Age-Invariant Face Recognition," *ArXiv*, vol. abs/1810.07599, 2018.

[11] Arora, Raman and Kaur, Parvinder and Kaur, Er. Deepinder, "Age Invariant Face Recogntion Using Stacked Autoencoder Deep Neural Network," in *2020 International Conference on Intelligent Engineering and Management (ICIEM)*, 2020, pp. 358-363.

[12] Zhao, Jian and Yan, Shuicheng and Feng, Jiashi, "Towards Age-Invariant Face Recognition," *{IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 474-487, 2022.

[13] Longlong Jing, Yingli Tian, “Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey”.

[14] Liao, Jiashu and Sanchez, Victor and Guha, Tanaya, "Self-Supervised Frontalization and Rotation Gan with Random Swap for Pose-Invariant Face Recognition," in *2022 IEEE International Conference on Image Processing (ICIP)*, 2022, pp. 911-915.

[15] Lin, Chun-Hsien and Wu, Bing-Fei, "Domain Adapting Ability Of Self-Supervised Learning For Face Recognition," in *2021 IEEE International Conference on Image Processing (ICIP)*, 2021, pp. 479-483.

[16] Guillermo Gomez-Trenado, Stéphane Lathuilière, Pablo Mesejo & Óscar Cordón , "Custom Structure Preservation in Face Aging," in *Lecture Notes in Computer Science*, 2022.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, “Deep Residual Learning for Image Recognition,” 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren & Jian Sun, "Identity Mappings in Deep Residual Networks," in *Lecture Notes in Computer Science*, 2016.

[19] Yashvi Chandola and Jitendra Virmani and H.S. Bhadauria and Papendra Kumar, "Chapter 4 - End-to-end pre-trained CNN-based computer-aided classification system design for chest radiographs," in *Deep Learning for Chest Radiographs*, Academic Press, 2021, pp. 117-140.

[20] Grill, Jean-Bastien and Strub, Florian and Altch, Florent and Tallec, Corentin and Richemond, Pierre and Buchatskaya, Elena and Doersch, Carl and Avila Pires, Bernardo and Guo, Zhaohan and Gheshlaghi Azar, Mohammad and Piot, Bilal and kavukcuoglu, ko, "Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning," in *Advances in Neural Information Processing Systems*, 2020, pp. 21271-21284.

[21] Ramanathan, N. and Chellappa, R, "Modeling age progression in young face," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2006, pp. 387-394.

[22] Danfeng Xie, Lei Zhang, and Li Bai, “Deep Learning in Visual Computing and Signal Processing”.

[23] Amrutha Sethuram, Eric Patterson, Karl Ricanek & Allen Rawls, “Improvements and performance evaluation concerning,” trong *Advances in Biometrics*, 2009.

[24] Xiao, Bing and Gao, Xinbo and Tao, Dacheng and Liu, Wei, “Biview face recognition in the shape–texture domain,” *Pattern Recognition*, tập 46, p. 1906–1919, 2013.

[25] Moustafa, Amal and Elnakib, Ahmed and Areed, Nihal, "Age-invariant face recognition based on deep features analysis," *Signal, Image and Video Processing*, vol. 14, 07 2020.