Facial Key-point Detection

By

Adhishree Bansal (171233)

Aarushi Bhardwaj(171260)

Under the Supervision of
(Assistant Prof.: Dr. Aman Sharma)
to

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & INFORMATION TECHNOLOGY

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT, 173234, HIMACHAL PRADESH, INDIA

DECLARATION

We hereby declare that the work presented in this report entitled “**Facial keypoint Detection**” in partial fulfillment of the requirements for the award of degree of **Bachelor of Technology in Computer Science and Engineering Information Technology** submitted in the department of Computer Science and engineering and Information Technology, Jaypee university of Information Technology, Waknaghat is an authentic record of my own work carried out over a period under supervision of Dr. Aman Sharma.

Adhi Bansal

[illegible]

Adhishree Bansal (171233)

Aarushi Bhardwaj (171260)

This is to certify that the statement made above is true and in my best knowledge.

[illegible]

(Supervisor Signature)

Dr. Aman Sharma

CERTIFICATE

This is to certify that the work in this Project titled as **“Facial keypoint detection”** is entirely written, successfully completed and demonstrated by the following students themselves as a fulfillment of requirements for Bachelor’s of Engineering in Computer Science.

Aarushi Bhardwaj	(171260)
Adhishree Bansal	(171233)

ACKNOWLEDGEMENT

In the present world of ever evolving technology and cut throat competition in every field, there is a race of existence in those having the will to come forward and succeed. Most importantly, we would like to express our sincere gratitude to our advisor **Dr. Aman Sharma** for the continuous support in the study and research. It was under his guidance and effort that we were able to implement this project successfully.

Beside the hard work put by the team and the mentor, we would also like to express our gratitude to the panel for bestowing us with an opportunity to present this project and providing us with feedback which will assist us in future.

We would also like to extend our acknowledgement to our esteemed institute Jaypee University of Information Technology for exposing and motivating us to work in various fields.

TABLE OF CONTENTS

List of Figures

ABSTRACT

Facial key points include centers and corners of the eyes, eyebrows, nose and mouth, among other facial features. Our methodology involves four steps to producing our output predictions. The first step involves number of training time data augmentation techniques to expand the number of training examples, and generalizability of our model. For the second stage we will discuss a number of convolution neural networks we trained, with the best architectures derived from former LeNet and VGG Net models. Attest time, we will discuss our approach of again using data augmentation techniques to produce a composite of images from a single test image, allowing for averaged predictions. Lastly, we use a weighted ensemble of models to make our final key points predictions. We evaluate our deferent model architectures, with and without data augmentation techniques based on their Root Mean Squared Error scores they produce on the test set. 3D face recognition has become a trending research direction in both industry and academia. It inherits advantages from traditional 2D face recognition, such as the natural recognition process and a wide range of applications. Moreover, 3D face recognition systems could accurately recognize human faces even under dim lights and with variant facial positions and expressions, in such conditions 2D face recognition systems would have immense difficulty to operate. This paper summarizes the history and the most recent progresses in 3D face recognition research domain. The frontier research results are introduced in three categories: pose-invariant recognition, expression-invariant recognition, and occlusion-invariant recognition.

CHAPTER 1
(INTRODUCTION)

1.1 INTRODUCTION

With the fast development in computer vision area, more and more research works and industry applications are focused on facial key point detection. Detecting key points in given face image would act as a fundamental part of many applications, including facial expression on classification, facial alignment, tracking faces in videos and also applications in medical diagnosis. Thus, how to detect facial key points both fast and accurately to use it as a preprocessing procedure has become a big challenge.

There are two main challenges for facial key points detection, one is that facial features have great variance between different people and different external factors, the other is that we have to reduce time complexity to achieve real time key points detection.

Our primary motivation for this project was out interest in applying deep learning to significant problems with relevant uses. The applications of this research are numerous and significant, including facial expression analysis, biometric or facial recognition, medical diagnosis of facial disfigurement and even tracking of line of sight. The idea we are we are working towards solving an open and important problem with all these possible applications greatly inspired us. We specially chose the Facial key point detection Kaggle competition because it gave us ample opportunity to experiment with a wide variety of approaches and neutral net models to solve an otherwise straightforward problem of localization. The competition element also allowed us to benchmark our results against the greater community, and compare the electiveness of our methods against alternatives. Lastly, we were motivated by the inherent challenges associated with the problem. Detecting facial key points is a challenging problem given variation in both facial features as well as image conditions. Facial features dicer according to size, position, pose and expression, while image condition varies with illumination and viewing angle. These abundant variations, in combination with the necessity for highly accurate coordinate predictions (e.g. the exact corner of an eye) lead us to believe it would be a deep and interesting topic. In human identification scenarios, facial metrics are more naturally accessible than many other biometrics, such as iris, fingerprint, and palm print. Face recognition is also highly valuable in human computer interaction, access control, video surveillance, and many other applications. Although 2D face recognition research made significant progresses in recent years, its accuracy is still highly depended on light conditions and human poses. When the light is dim or the face poses are not properly aligned in the camera view, the recognition accuracy will suffer. The fast evolution of 3D sensors reveals a new path for face recognition that could overcome the fundamental limitations of 2D technologies. The geometric information contained in 3D facial data could substantially improve the recognition accuracy under conditions that are difficult for 2D technologies. Many researchers have turned their focuses to 3D face recognition and made this research area a new trend.

A general work flow for 3D face recognition is shown below. The work flow could be decomposed into two phases and five stages. In the training phase, 3D face data are acquired and then preprocessed to obtain "clean" 3D faces. Then the data are processed by feature extraction algorithms to find the features that could be used to differentiate faces. The features of each face are then stored into the feature database. In the testing phase, the target face goes through the acquisition, preprocessing, and feature extraction stages that are identical to the stages in the training phase. In the feature matching stage, the features of the target face are compared with the faces stored in the feature database and calculate the match scores. When a match score is sufficiently high, we would claim that the target face is recognized.

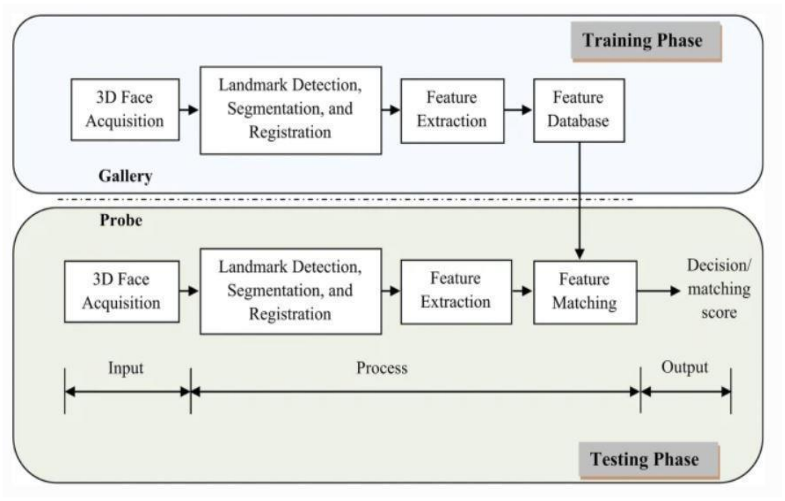


Figure1: General Flowchart

1.2 OBJECTIVES

In recent times, it is majorly seen in most of the data science projects which include a lot of AI-generated people, whether it is in papers, blogs, or videos.

In this project, facial key points also known as facial landmarks are the small magenta dots shown on each of the faces. In each training and testing image, there is an single face and 68 key-points, with coordinates (x, y) for the face. These key-points mark important areas of the face: in the yes, corners of the mouth, the nose, etc.

1.3 PROBLEM STATEMENT

This task is a currently active, designated to kaggle competition. The problem is essentially to predict the near exact coordinate locations of points on an image of a face. Given 15 facial points, there are 30 numerical values that we need to predict: a 2D (x, y) coordinate representation for each facial point. The loss is calculated as the total root mean square error that is RMSE, an elective measure of the derivations in distances between the 15 real and predicted facial point coordinates.

Our approach involves the use of deep learning and convolution neural networks to predict the x and y coordinates of a given key point class in animate containing a single face. Therefore, as shown below in the figure, our grey scale image from the dataset (96x96x1), and our output is 30 floating point values between 0.0 and 96.0, indicating where on the image the corresponding x and y coordinates are for a given key point feature. Below are example inputs (first row) and the output (second row).

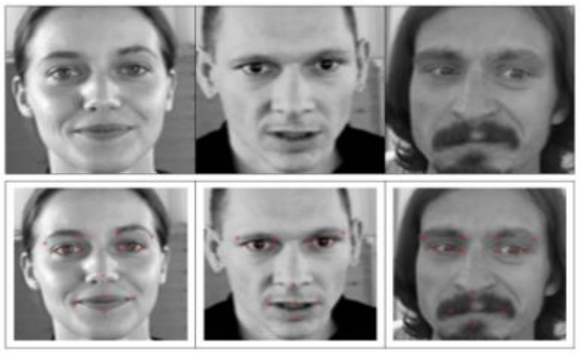


Figure2

1.4 METHODOLOGY

How does CNN works?

Consider any image, as each image represents some pixel value, we analyze the influence of nearby pixels in an image by using something called a filter, also called as weights, kernel or features. Filters are tensor which keeps track of spatial information and learns to extract features like edge detection, smooth curve, etc of objects in something called a convolution layer. The major part is to detect edges in the images and these are detected by the filters. It helps to filter out the unwanted information to amplify images. These are high pass filters where the changes occur in intensity very quickly like fro black to white pixel and vice versa.

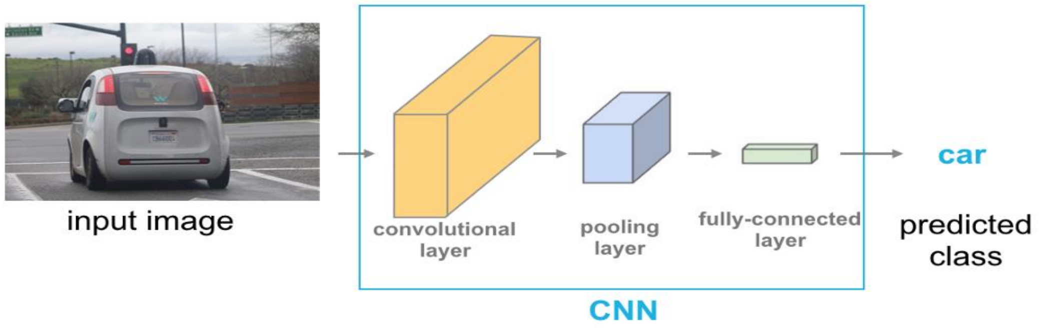


Figure3: Basic CNN structure

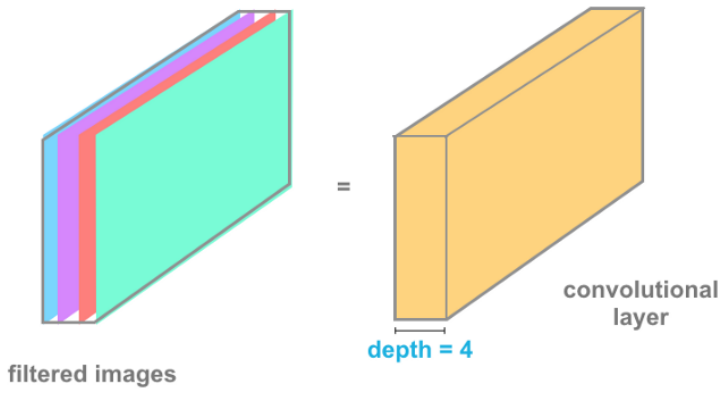
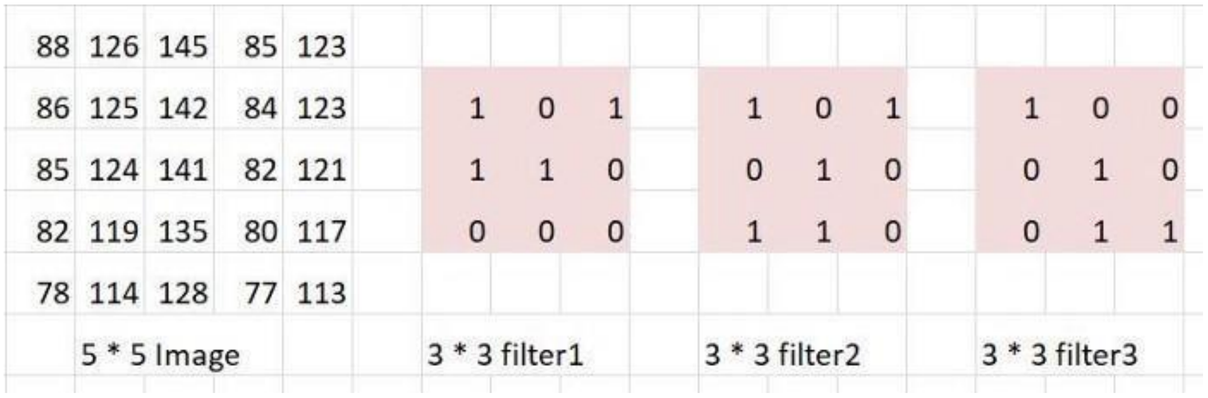


Figure4: Filtered images – flowchart

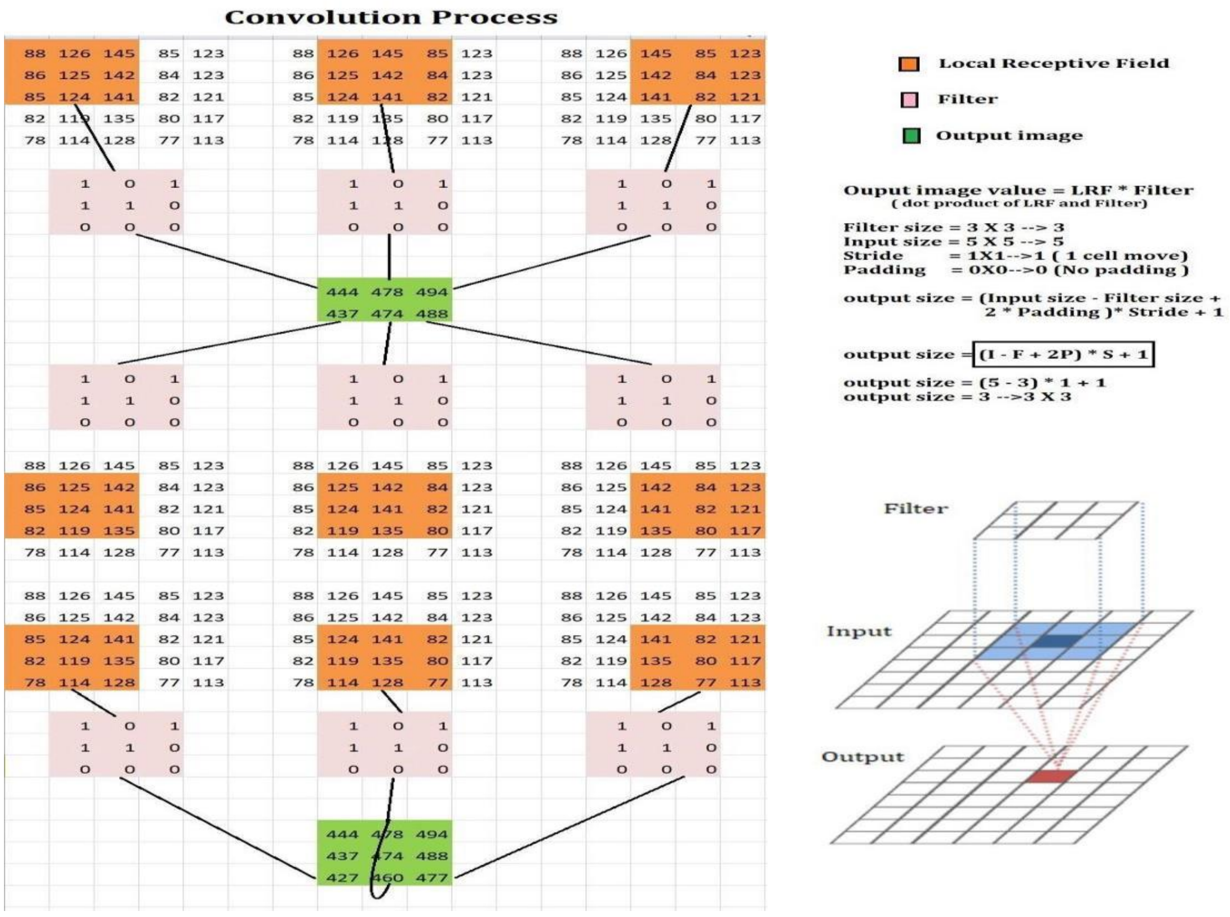
Let us consider an image size 5x5 size and 3 filters (since each filter will be used for each colour channel RGB) of 3x3 size.



88	126	145	85	123											
86	125	142	84	123		1	0	1		1	0	1		1	0	0
85	124	141	82	121		1	1	0		0	1	0		0	1	0
82	119	135	80	117		0	0	0		1	1	0		0	1	1
78	114	128	77	113											
	5 * 5 Image					3 * 3 filter1				3 * 3 filter2				3 * 3 filter3		

Figure5

For simplicity, we took 0 and 1 for filters, usually, these are continuous values. The filter convolute with the image to detect patterns and features.



Ouput image value = LRF * Filter
(dot product of LRF and Filter)

Filter size = 3 X 3 --> 3
Input size = 5 X 5 --> 5
Stride = 1X1-->1 (1 cell move)
Padding = 0X0-->0 (No padding)

output size = (Input size - Filter size + 2 * Padding)* Stride + 1

output size = $\boxed{(I - F + 2P) * S + 1}$

output size = (5 - 3) * 1 + 1
output size = 3 -->3 X 3

444	478	494
437	474	488
427	460	477

Figure6: Convolution process

Then the convolution of 5 x 5 image matrixes multiplies with 3 x 3 filter matrixes which is called "feature Map". We apply the dot product to scalar value and then move the filter by the stride over the entire image.

Sometimes filter does not fit perfectly in the input image. In that case we pad the image with zeros as shown below. This is called padding.

Next, we need to reduce the size of the image, if they are too large. Pooling layers section would reduce the number parameters when the images are too large.

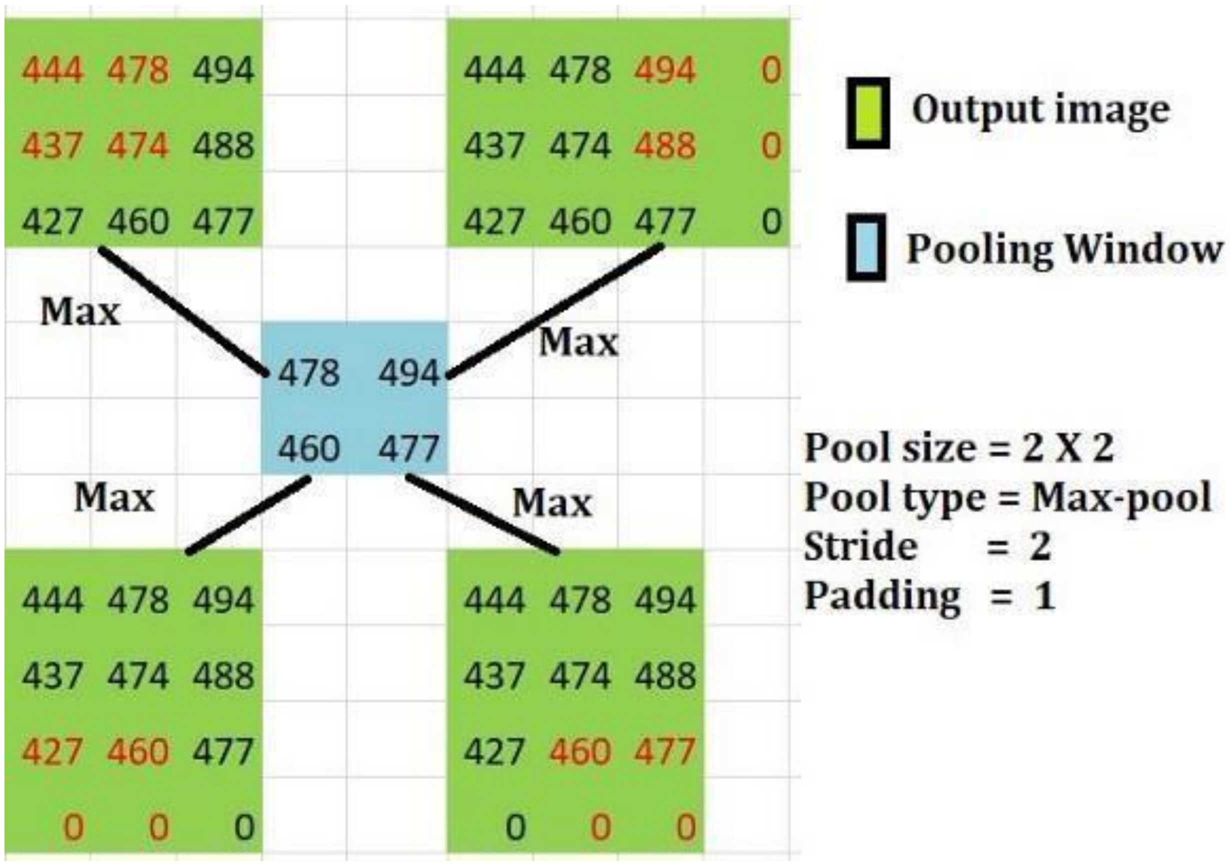


Figure7: Pooling the layers

As shown in the above image, the padding is applied so that the filter perfectly fits the given image. Adding pooling layer then decrease the size of the image and hence decrease the complexity and computations.

Next step is Normalization. Usually an activation function ReLu is used. Relu stands for rectified linear unit for a non linear operation.

The output is f(x) = max(0,x). The purpose of Relu is to add non linearity to the convolution network. In usual cases, the real world data want our network to learn non linear values.

A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. Here we are assuming that we have negative values since dealing with the real world data. In case, if there is no negative value, you can skip this part.

$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases} \quad \begin{array}{cc} -478 & 494 \\ 460 & -477 \end{array} \qquad \begin{array}{cc} 0 & 494 \\ 460 & 0 \end{array}$$

Figure8

The final step is to flatten our matrix and feed the values to fully connected layer.

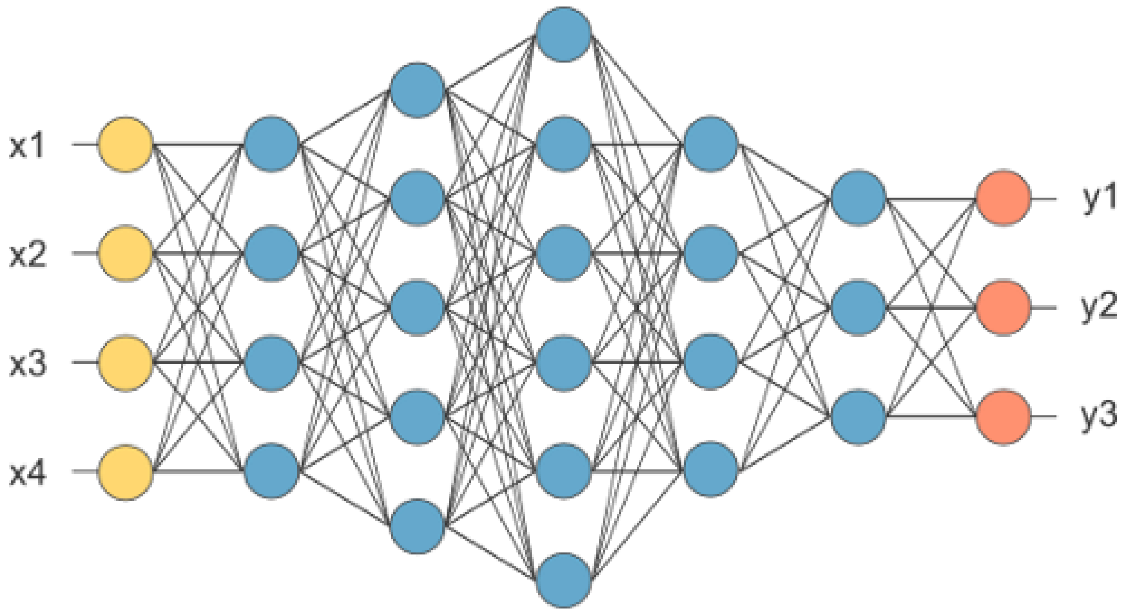


Figure9

Next, we need to train the model in the same way we train other neural networks. Using the certain number of epochs and then back-propagate to update weights and calculate the loss.

Overall Structure of CNN

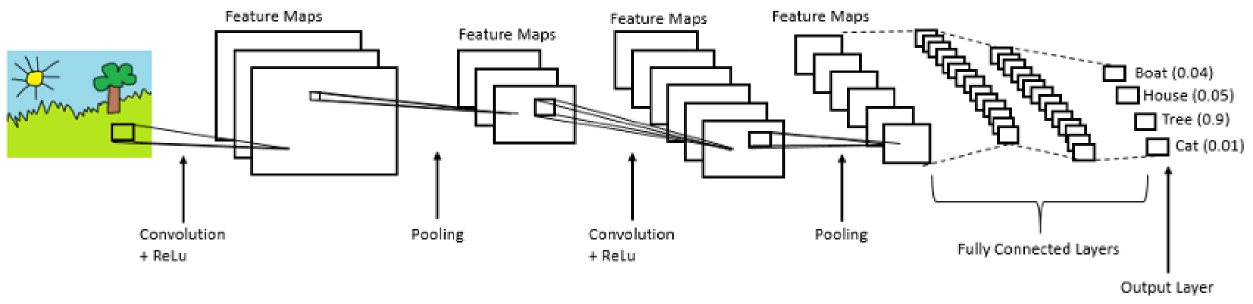


Figure10 : structure of CNN

Using the above diagram we know, that there are:

1. Convolution layer where convolution takes place
2. Pooling layer where the pooling process happens.
3. Normalization usually with use of ReLu
4. Fully connected layer

1.5 SCOPE

3D face recognition technology has been applied in many fields, such as access control and automatic driving. The iPhone X uses Face ID, technology that unlocks the phone by using infrared and visible light scans to uniquely identify your face. It works in a variety of conditions and is extremely secure. In the world of autonomous driving, the autopilot needs to manage the hand-over between the automated and the manual modes. To have a smooth hand-over, it is important to make sure that the driver is alert and ready to take control of the car before the autopilot is disengaged. To have a smooth transition between modes of operation, Omron introduces 3D facial recognition technology that detects a drowsy or distracted driver. Considering the fact that one out of every six car accidents is attributed to a drowsy or distracted driver, the technology can have a huge impact even on the safety of manual driving.

CHAPTER 2

(LITERATURE REVIEW)

What we have tried to accomplish here is to create models such as Multi-layer Perceptron and Convolutional Neural Network (CNN) in order to detect facial keypoints and then to be the judge of how well they perform, how image augmentation is done, how to create data loading and processing and at last how to train and deploy models with the use of PyTorch.

2.1) MULTI-LAYER PERCEPTRON NEURAL NETWORKS

Artificial neural networks are a fascinating area of study, although they can be intimidating when just getting started. There are a lot of specialized terminologies used when describing the data structures and algorithms used in the field. In this review we'll understand the terminology and processes used in the field of multi-layer perceptron artificial neural networks:

- The building blocks of neural networks including neurons, weights and activation functions.
- How the building blocks are used in layers to create networks.
- How networks are trained from example data.

We are going cover the following topics:

1. Multi-Layer Perceptrons.
2. Neurons, Weights and Activations.
3. Networks of Neurons.
4. Training Networks.

1. Multi-Layer Perceptrons

The field of artificial neural networks is often just called neural networks or multi-layer perceptrons after perhaps the most useful type of neural network. A perceptron is a single neuron model that was a precursor to larger neural networks. It is a field that investigates how simple models of biological brains can be used to solve difficult computational tasks like the predictive modeling tasks we see in machine learning. The goal is not to create realistic models of the brain, but instead to develop robust algorithms and data structures that we can use to model difficult problems. The power of neural networks comes from their ability to learn the representation in your training data and how to best relate it to the output variable that you want to predict. In this sense neural networks learn a mapping. Mathematically, they are capable of learning any mapping function and have been proven to be a universal approximation algorithm. The predictive capability of neural networks comes from the hierarchical or multi-layered structure of the networks. The data structure can pick out (learn to represent) features at different scales or resolutions and combine them into higher-order features. For example from lines, to collections of lines to shapes.

2. Neurons

The building block for neural networks are artificial neurons. These are simple computational units that have weighted input signals and produce an output signal using an activation function.

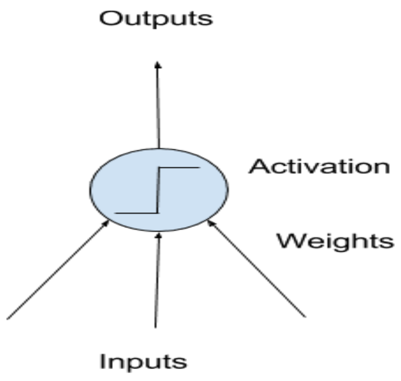


Figure11: Model of a Simple Neuron

Neuron Weights

You may be familiar with linear regression, in which case the weights on the inputs are very much like the coefficients used in a regression equation. Like linear regression, each neuron also has a bias which can be thought of as an input that always has the value 1.0 and it too must be weighted. For example, a neuron may have two inputs in which case it requires three weights. One for each input and one for the bias. Weights are often initialized to small random values, such as values in the range 0 to 0.3.

Activation

The weighted inputs are summed and passed through an activation function, sometimes called a transfer function. An activation function is a simple mapping of summed weighted input to the output of the neuron. It is called an activation function because it governs the threshold at which the neuron is activated and strength of the output signal. Historically simple step activation functions were used where if the summed input was above a threshold, for example 0.5, then the neuron would output a value of 1.0, otherwise it would output a 0.0. Traditionally non-linear activation functions are used. This allows the network to combine the inputs in more complex ways and in turn provide a richer capability in the functions they can model. Non-linear functions like the logistic also called the sigmoid function were used that output a value between 0 and 1 with an s-shaped distribution, and the hyperbolic tangent function also called tanh that outputs the same distribution over the range -1 to +1. More recently the rectifier activation function has been shown to provide better results.

3. Networks of Neurons

Neurons are arranged into networks of neurons. A row of neurons is called a layer and one network can have multiple layers. The architecture of the neurons in the network is often called the network topology.

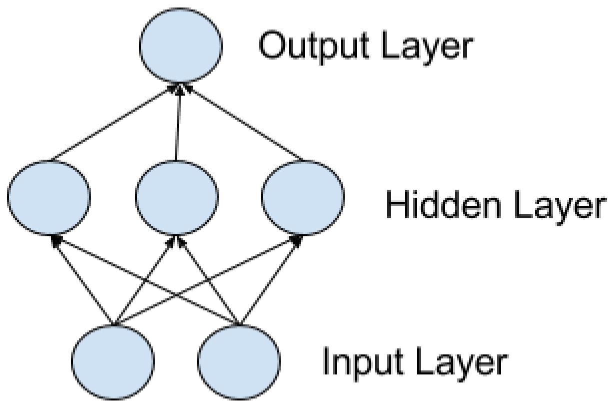


Figure12: Model of a simple network

Input or Visible Layers

The bottom layer that takes input from your dataset is called the visible layer, because it is the exposed part of the network. Often a neural network is drawn with a visible layer with one neuron per input value or column in your dataset. These are not neurons as described above, but simply pass the input value though to the next layer.

Hidden Layers

Layers after the input layer are called hidden layers because they are not directly exposed to the input. The simplest network structure is to have a single neuron in the hidden layer that directly outputs the value. Given increases in computing power and efficient libraries, very deep neural networks can be constructed. Deep learning can refer to having many hidden layers in your neural network. They are deep because they would have been unimaginably slow to train historically, but may take seconds or minutes to train using modern techniques and hardware.

Output Layer

The final hidden layer is called the output layer and it is responsible for outputting a value or vector of values that correspond to the format required for the problem.

The choice of activation function in the output layer is strongly constrained by the type of problem that you are modeling. For example:

- A regression problem may have a single output neuron and the neuron may have no activation function.
- A binary classification problem may have a single output neuron and use a sigmoid activation function to output a value between 0 and 1 to represent the probability of predicting a value for the class 1. This can be turned into a crisp class value by using a threshold of 0.5 and snap values less than the threshold to 0 otherwise to 1.
- A multi-class classification problem may have multiple neurons in the output layer, one for each class (e.g. three neurons for the three classes in the famous iris flowers classification problem). In this case a softmax activation function may be used to output a probability of the network predicting each

of the class values. Selecting the output with the highest probability can be used to produce a crisp class classification value.

4. Training Networks

Once configured, the neural network needs to be trained on your dataset.

Data Preparation

You must first prepare your data for training on a neural network. Data must be numerical, for example real values. If you have categorical data, such as a sex attribute with the values “male” and “female”, you can convert it to a real-valued representation called a one hot encoding. This is where one new column is added for each class value (two columns in the case of sex of male and female) and a 0 or 1 is added for each row depending on the class value for that row.

This same one hot encoding can be used on the output variable in classification problems with more than one class. This would create a binary vector from a single column that would be easy to directly compare to the output of the neuron in the network’s output layer, that as described above, would output one value for each class.

Neural networks require the input to be scaled in a consistent way. You can rescale it to the range between 0 and 1 called normalization. Another popular technique is to standardize it so that the distribution of each column has the mean of zero and the standard deviation of 1.

Scaling also applies to image pixel data. Data such as words can be converted to integers, such as the popularity rank of the word in the dataset and other encoding techniques.

Stochastic Gradient Descent

The classical and still preferred training algorithm for neural networks is called stochastic gradient descent. This is where one row of data is exposed to the network at a time as input. The network processes the input upward activating neurons as it goes to finally produce an output value. This is called a forward pass on the network. It is the type of pass that is also used after the network is trained in order to make predictions on new data.

The output of the network is compared to the expected output and an error is calculated. This error is then propagated back through the network, one layer at a time, and the weights are updated according to the amount that they contributed to the error. This clever bit of math is called the back propagation algorithm.

The process is repeated for all of the examples in your training data. One round of updating the network for the entire training dataset is called an epoch. A network may be trained for tens, hundreds or many thousands of epochs.

Weight Updates

The weights in the network can be updated from the errors calculated for each training example and this is called online learning. It can result in fast but also chaotic changes to the network.

Alternatively, the errors can be saved up across all of the training examples and the network can be updated at the end. This is called batch learning and is often more stable.

Typically, because datasets are so large and because of computational efficiencies, the size of the batch, the number of examples the network is shown before an update is often reduced to a small number, such as tens or hundreds of examples. The amount that weights are updated is controlled by a configuration parameter called the learning rate. It is also called the step size and controls the step or change made to network weight for a given error. Often small weight sizes are used such as 0.1 or 0.01 or smaller.

The update equation can be complemented with additional configuration terms that you can set.

- Momentum is a term that incorporates the properties from the previous weight update to allow the weights to continue to change in the same direction even when there is less error being calculated.
- Learning Rate Decay is used to decrease the learning rate over epochs to allow the network to make large changes to the weights at the beginning and smaller fine tuning changes later in the training schedule.

Prediction

Once a neural network has been trained it can be used to make predictions. You can make predictions on test or validation data in order to estimate the skill of the model on unseen data. You can also deploy it operationally and use it to make predictions continuously.

The network topology and the final set of weights is all that you need to save from the model. Predictions are made by providing the input to the network and performing a forward-pass allowing it to generate an output that you can use as a prediction.

2.2) A BRIEF HISTORY OF CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks, also called ConvNets, were first introduced in the 1980s by Yann LeCun, a postdoctoral computer science researcher. LeCun had built on the work done by Kunihiko Fukushima, a Japanese scientist who, a few years earlier, had invented the neocognitron, a very basic image recognition neural network.

The early version of CNNs, called LeNet (after LeCun), could recognize handwritten digits. CNNs found a niche market in banking and postal services and banking, where they read zip codes on envelopes and digits on checks.

But despite their ingenuity, ConvNets remained on the sidelines of computer vision and artificial intelligence because they faced a serious problem: They could not scale. CNNs needed a lot of data and compute resources to work efficiently for large images. At the time, the technique was only applicable to images with low resolutions.

If you come from a digital signal processing field or related area of mathematics, you may understand the convolution operation on a matrix as something different. Specifically, the filter (kernel) is flipped prior to being applied to the input. Technically, the convolution as described in the use of convolutional neural networks is actually a "*cross-correlation"*. Nevertheless, in deep learning, it is referred to as a "*convolution*" operation.

2.3) CONVOLUTION IN COMPUTER VISION

The idea of applying the convolutional operation to image data is not new or unique to convolutional neural networks; it is a common technique used in computer vision.

Historically, filters were designed by hand by computer vision experts, which were then applied to an image to result in a feature map or output from applying the filter then makes the analysis of the image easier in some way.

For example, below is a hand crafted 3×3 element filter for detecting vertical lines:

0.0, 1.0, 0.0

0.0, 1.0, 0.0

0.0, 1.0, 0.0

Applying this filter to an image will result in a feature map that only contains vertical lines. It is a vertical line detector. You can see this from the weight values in the filter; any pixel values in the center vertical line will be positively activated and any on either side will be negatively activated. Dragging this filter systematically across pixel values in an image can only highlight vertical line pixels.

A horizontal line detector could also be created and also applied to the image, for example:

0.0, 0.0, 0.0

1.0, 1.0, 1.0

0.0, 0.0, 0.0

Combining the results from both filters, e.g. combining both feature maps, will result in all of the lines in an image being highlighted. A suite of tens or even hundreds of other small filters can be designed to detect other features in the image. The innovation of using the convolution operation in a neural network is that the values of the filter are weights to be learned during the training of the network.

The network will learn what types of features to extract from the input. Specifically, training under stochastic gradient descent, the network is forced to learn to extract features from the image that minimize the loss for the specific task the network is being trained to solve, e.g. extract features that are the most useful for classifying images as dogs or cats.

Power of Learned Filters

Learning a single filter specific to a machine learning task is a powerful technique. Yet, convolutional neural networks achieve much more in practice.

Multiple Filters

Convolutional neural networks do not learn a single filter; they, in fact, learn multiple features in parallel for a given input. For example, it is common for a convolutional layer to learn from 32 to 512 filters in parallel for a given input.

This gives the model 32, or even 512, different ways of extracting features from an input, or many different ways of both "learning to see" and after training, many different ways of "seeing" the input data. This diversity allows specialization, e.g. not just lines, but the specific lines seen in your specific training data.

Multiple Channels

Color images have multiple channels, typically one for each color channel, such as red, green, and blue. From a data perspective, that means that a single image provided as input to the model is, in fact, three images. A filter must always have the same number of channels as the input, often referred to as "depth". If an input image has 3 channels (e.g. a depth of 3), then a filter applied to that image must also have 3 channels (e.g. a depth of 3). In this case, a 3×3 filter would in fact be 3x3x3 or [3, 3, 3] for rows, columns, and depth. Regardless of the depth of the input and depth of the filter, the filter is applied to the input using a dot product operation which results in a single value. This means that if a convolutional layer has 32 filters, these 32 filters are not just two-dimensional for the two-dimensional image input, but are also three-dimensional, having specific filter weights for each of the three channels. Yet, each filter results in a single feature map. Which means that the depth of the output of applying the convolutional layer with 32 filters is 32 for the 32 feature maps created.

Multiple Layers

Convolutional layers are not only applied to input data, e.g. raw pixel values, but they can also be applied to the output of other layers. The stacking of convolutional layers allows a hierarchical decomposition of the input. Consider that the filters that operate directly on the raw pixel values will learn to extract low-level features, such as lines. The filters that operate on the output of the first line layers may extract features that are combinations of lower-level features, such as features that comprise multiple lines to express shapes. This process continues until very deep layers are extracting faces, animals, houses, and so on.This is exactly what we see in practice. The abstraction of features to high and higher orders as the depth of the network is increased.

2.4) THE LIMITS OF CONVOLUTIONAL NEURAL NETWORKS

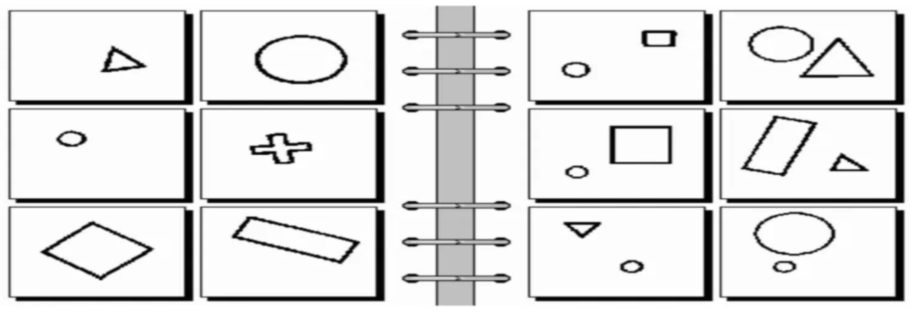
Despite their power and complexity, convolutional neural networks are, in essence, pattern-recognition machines. They can leverage massive compute resources to ferret out tiny and inconspicuous visual patterns that might go unnoticed to the human eye. But when it comes to understanding the meaning of the contents of images, they perform poorly.

Consider the following image. A well-trained ConvNet will tell you that it’s the image of a soldier, a child and the American flag. But a person can give a long description of the scene, and talk about military service, tours in a foreign country, the feeling of longing for home, the joy of reuniting with the family, etc. Artificial neural networks have no notion of those concepts.

These limits become more evident in practical applications of convolutional neural networks. For instance, CNNs are now widely used to moderate content on social media networks. But despite the vast repositories of images and videos they're trained on, they still struggle to detect and block inappropriate content. In one case, Facebook's content-moderation AI banned the photo of a 30,000-year-old statue as nudity. Also, neural networks start to break as soon as they move a bit out of their context. Several studies have shown that CNNs trained on ImageNet and other popular datasets fail to detect objects when they see them under different lighting conditions and from new angles.

ObjectNet, a dataset that better represents the different nuances of how objects are seen in real life. CNNs don't develop the mental models that humans have about different objects and their ability to imagine those objects in previously unseen contexts.

Another problem with convolutional neural networks is their inability to understand the relations between different objects. Consider the following image, which is known as a "Bongard problem," named after its inventor, Russian computer scientist Mikhail Moiseevich Bongard. Bongard problems present you with two sets of images (six on the left and six on the right), and you must explain the key difference between the two sets. For instance, in the example below, images in the left set contain one object and images in the right set contain two objects. It's easy for humans to draw such conclusions from such small amounts of samples. If I show you these two sets and then provide you with a new image, you'll be able to quickly decide whether it should go into the left or right set.



Bongard problems are easy for humans to solve, but hard for computer vision systems.

But there’s still no convolutional neural network that can solve Bongard problems with so few training examples. In one study conducted in 2016, AI researchers trained a CNN on 20,000 Bongard samples and tested it on 10,000 more. CNN's performance was much lower than that of average humans.

The peculiarities of ConvNets also make them vulnerable to adversarial attacks, perturbations in input data that go unnoticed to the human eye but affect the behavior of neural networks. Adversarial attacks have become a major source of concern as deep learning and especially CNNs have become an integral component of many critical applications such as self-driving cars.

Does this mean that CNNs are useless? Despite the limits of convolutional neural networks, however, there’s no denying that they have caused a revolution in artificial intelligence. Today, CNNs are used in many computer vision applications such as facial recognition, image search and editing, augmented reality, and more. In some areas, such as medical image processing, well-trained ConvNets might even outperform human experts at detecting relevant patterns. As advances in convolutional neural networks show, our achievements are remarkable and useful, but we are still very far from replicating the key components of human intelligence.

2.5) PYTORCH

PyTorch is the premier open-source deep learning framework developed and maintained by Facebook. At its core, PyTorch is a mathematical library that allows you to perform efficient computation and automatic differentiation on graph-based models. Achieving this directly is challenging, although thankfully, the modern

PyTorch API provides classes and idioms that allow you to easily develop a suite of deep learning models.

PyTorch is an open-source Python library for deep learning developed and maintained by Facebook. The project started in 2016 and quickly became a popular framework among developers and researchers. Torch (*Torch7*) is an open-source project for deep learning written in C and generally used via the Lua interface. It was a precursor project to PyTorch and is no longer actively developed. PyTorch includes "*Torch*" in the name, acknowledging the prior torch library with the "*Py*" prefix indicating the Python focus of the new project.

The PyTorch API is simple and flexible, making it a favorite for academics and researchers in the development of new deep learning models and applications. The extensive use has led to many extensions for specific applications (such as text, computer vision, and audio data), and may pre-trained models that can be used directly. As such, it may be the most popular library used by academics.

The flexibility of PyTorch comes at the cost of ease of use, especially for beginners, as compared to simpler interfaces like Keras. The choice to use PyTorch instead of Keras gives up some ease of use, a slightly steeper learning curve, and more code for more flexibility, and perhaps a more vibrant academic community.

2.5.1) PyTorch Deep Learning Model Life-Cycle

A model has a life-cycle, and this very simple knowledge provides the backbone for both modeling a dataset and understanding the PyTorch API.

The five steps in the life-cycle are as follows:

1. Prepare the Data.

2. Define the Model.

3. Train the Model.

4. Evaluate the Model.

5. Make Predictions.

Step 1: Prepare the Data

The first step is to load and prepare your data. Neural network models require numerical input data and numerical output data. You can use standard Python libraries to load and prepare tabular data, like CSV files. For example, Pandas can be used to load your CSV file, and tools from scikit-learn can be used to encode categorical data, such as class labels. PyTorch provides the Dataset class that you can extend and customize to load your dataset.

Step 2: Define the Model

The next step is to define a model. The idiom for defining a model in PyTorch involves defining a class that extends the Module class. The constructor of your class defines the layers of the model and the forward() function is the override that defines how to forward propagate input through the defined layers of the model. Many layers are available, such as Linear for fully connected layers, Conv2d for convolutional layers, and MaxPool2d for pooling layers. Activation functions can also be defined as layers, such as ReLU, Softmax, and Sigmoid.

Step 3: Train the Model

The training process requires that you define a loss function and an optimization algorithm.

Stochastic gradient descent is used for optimization, and the standard algorithm is provided by the SGD class, although other versions of the algorithm are available, such as Adam. Training the model involves enumerating the *DataLoader* for the training dataset.

First, a loop is required for the number of training epochs. Then an inner loop is required for the mini-batches for stochastic gradient descent.

Each update to the model involves the same general pattern comprised of:

- Clearing the last error gradient.
- A forward pass of the input through the model.
- Calculating the loss for the model output.
- Back propagating the error through the model.
- Update the model in an effort to reduce loss.

Step 4: Evaluate the model

Once the model is fit, it can be evaluated on the test dataset. This can be achieved by using the *DataLoader* for the test dataset and collecting the predictions for the test set, then comparing the predictions to the expected values of the test set and calculating a performance metric.

Step 5: Make predictions

A fit model can be used to make a prediction on new data. For example, you might have a single image or a single row of data and want to make a prediction. This requires that you wrap the data in a PyTorch Tensor data structure. A Tensor is just the PyTorch version of a NumPy array for holding data. It also allows you to perform the automatic differentiation tasks in the model graph, like calling *backward()* when training the model.

The prediction too will be a Tensor, although you can retrieve the NumPy array by detaching the Tensor from the automatic differentiation graph and calling the NumPy function.

2.5.2) How to Develop PyTorch Deep Learning Models

A Multilayer Perceptron model, or MLP for short, is a standard fully connected neural network model. It is comprised of layers of nodes where each node is connected to all outputs from the previous layer and the output of each node is connected to all inputs for nodes in the next layer. An MLP is a model with one or more fully connected layers. This model is appropriate for tabular data, that is data as it looks in a table or spreadsheet with one column for each variable and one row for each variable. There are three predictive modeling problems you may want to explore with an MLP; they are binary classification, multiclass classification, and regression.

CHAPTER 3

(SYSTEM DEVELOPMENT)

3.1) CONVOLUTIONAL NEURAL NETWORKS (CNN)

The convolutional neural network, or CNN for short, is a specialized type of neural network model designed for working with two-dimensional image data, although they can be used with one-dimensional and three-dimensional data. Central to the convolutional neural network is the convolutional layer that gives the network its name. This layer performs an operation called a "*convolution*".

In the context of a convolutional neural network, a convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel. The filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is the element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the "*scalar product*".

Using a filter smaller than the input is intentional as it allows the same filter (set of weights) to be multiplied by the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom.

This systematic application of the same filter across an image is a powerful idea. If the filter is designed to detect a specific type of feature in the input, then the application of that filter systematically across the entire input image allows the filter an opportunity to discover that feature anywhere in the image. This capability is commonly referred to as translation invariance, e.g. the general interest in whether the feature is present rather than where it was present.

The output from multiplying the filter with the input array one time is a single value. As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent a filtering of the input. As such, the two-dimensional output array from this operation is called a "feature map".

Once a feature map is created, we can pass each value in the feature map through a non linearity, such as a ReLU, much like we do for the outputs of a fully connected layer.

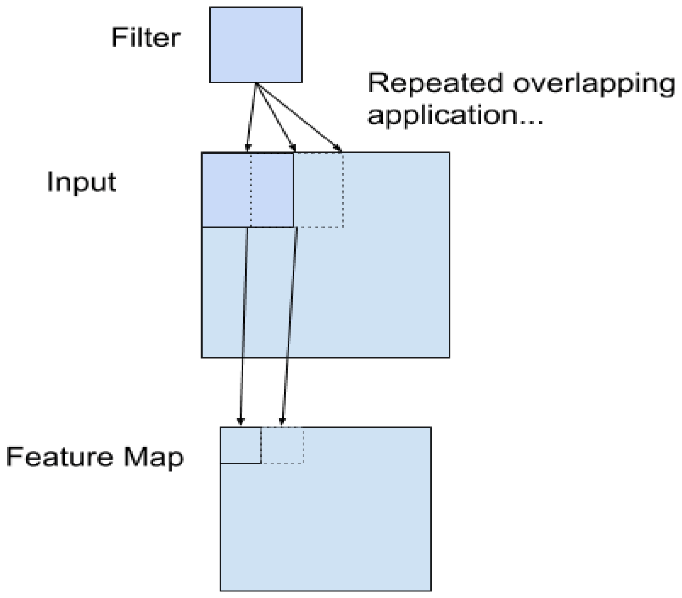


Figure13: Example of a Filter Applied to a Two-Dimensional Input to Create a Feature Map

3.2) HOW DO CNNS WORK?

Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value.

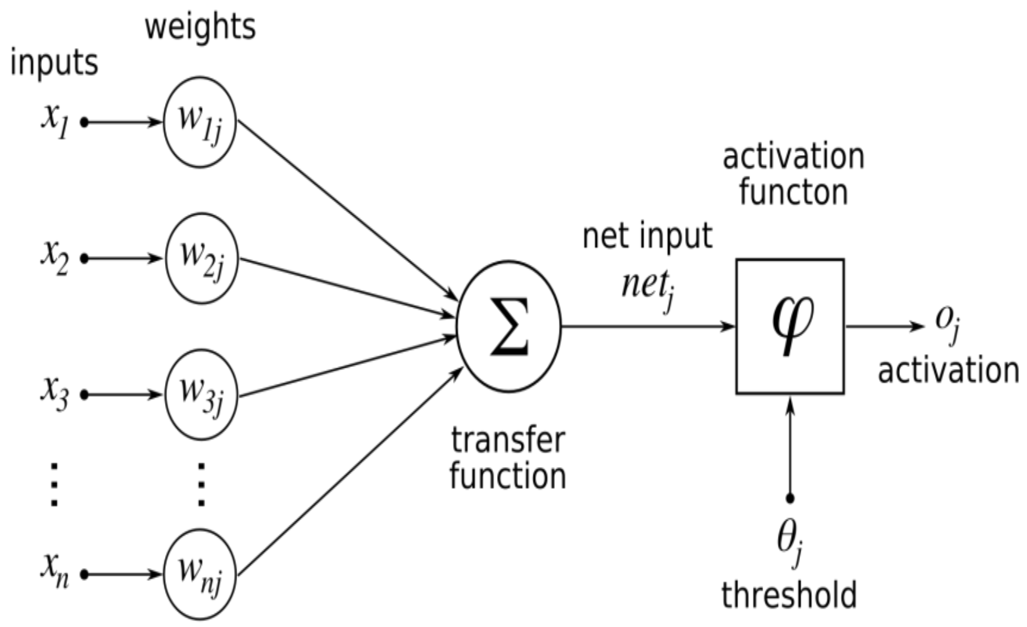


Figure14: The structure of an artificial neuron, the basic component of artificial neural networks

The behavior of each neuron is defined by its weights. When fed with the pixel values, the artificial neurons of a CNN pick out various visual features. When you input an image into a ConvNet, each of its layers generates several activation maps. Activation maps highlight the relevant features of the image. Each of the neurons takes a patch of pixels as input, multiplies their color values by its weights, sums them up, and runs them through the activation function.

The first (or bottom) layer of the CNN usually detects basic features such as horizontal, vertical, and diagonal edges. The output of the first layer is fed as input of the next layer, which extracts more complex features, such as corners and combinations of edges. As you move deeper into the convolutional neural network, the layers start detecting higher-level features such as objects, faces, and more.

The operation of multiplying pixel values by weights and summing them is called "convolution" (hence the name convolutional neural network). A CNN is usually composed of several convolution layers, but it also contains other components. The final layer of a CNN is a classification layer, which takes the output of the final convolution layer as input (remember, the higher convolution layers detect complex objects).

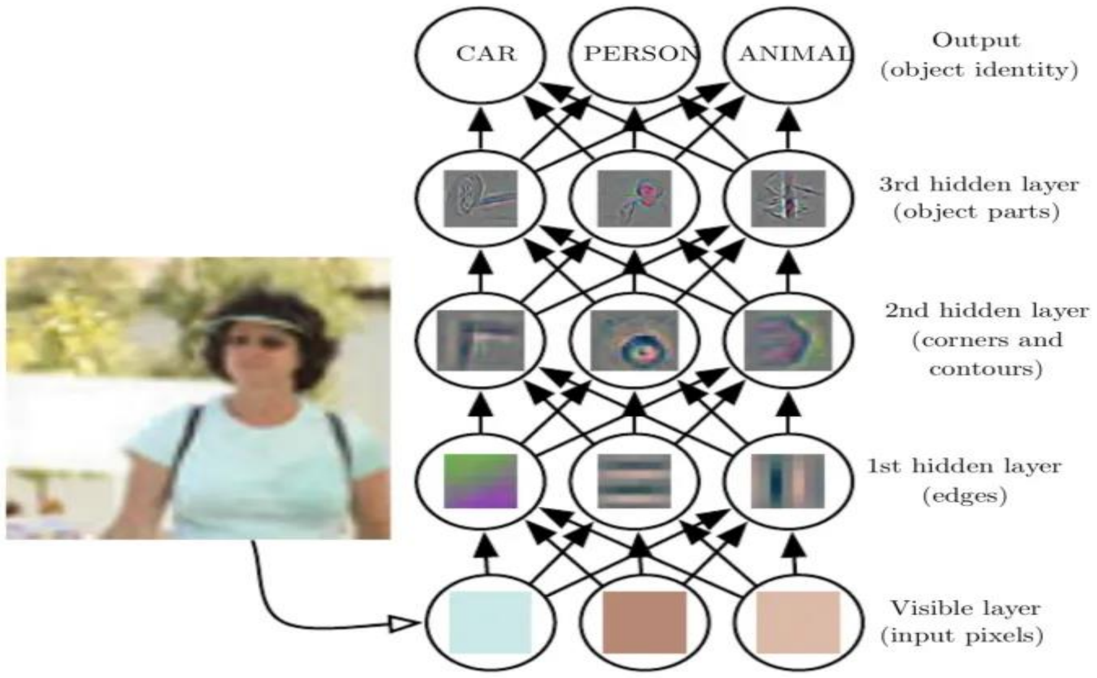


Figure15: Layers of CNN

3.3) TRAINING THE CONVOLUTIONAL NEURAL NETWORK

One of the great challenges of developing CNNs is adjusting the weights of the individual neurons to extract the right features from images. The process of adjusting these weights is called “training” the neural network.

In the beginning, CNN starts off with random weights. During training, the developers provide the neural network with a large dataset of images annotated with their corresponding classes (cat, dog, horse, etc.). The ConvNet processes each image with its random values and then compares its output with the image’s correct label. If the network’s output does not match the label—which is likely the case at the beginning of the training process—it makes a small adjustment to the weights of its neurons so that the next time it sees the same image, its output will be a bit closer to the correct answer.

The corrections are made through a technique called backpropagation (or backprop). Essentially, backpropagation optimizes the tuning process and makes it easier for the network to decide which units to adjust instead of making random corrections.

Every run of the entire training dataset is called an “epoch.” The ConvNet goes through several epochs during training, adjusting its weights in small amounts. After each epoch, the neural network becomes a bit better at classifying the training images. As the CNN improves, the adjustments it makes to the weights become smaller and smaller. At some point, the network “converges,” which means it essentially becomes as good as it can.

After training the CNN, the developers use a test dataset to verify its accuracy. The test dataset is a set of labeled images that are not part of the training process. Each image is run through the ConvNet, and the output is compared to the actual label of the image. Essentially, the test dataset evaluates how good the neural network has become at classifying images it has not seen before.

If a CNN scores good on its training data but scores bad on the test data, it is said to have been “overfitted.” This usually happens when there’s not enough variety in the training data or when the ConvNet goes through too many epochs on the training dataset.

The success of convolutional neural networks is largely due to the availability of huge image datasets developed in the past decade. ImageNet, the contest mentioned at the beginning of this article, got its title from a namesake dataset with more than 14 million labeled images. There are other more specialized datasets, such as the MNIST, a database of 70,000 images of handwritten digits.

You don’t, however, need to train every convolutional neural network on millions of images. In many cases, you can use a pretrained model, such as the AlexNet or Microsoft’s ResNet, and finetune it for another more specialized application. This process is called transfer learning, in which a trained neural network is retrained a smaller set of new examples.

3.4) TRAIN TIME DATA AUGMENTATION

We decided to use a variety of data augmentation techniques to

(a) expand the size of our training data, and

(b) help generalize our model.

The most elective data augmentation techniques at train time that we ended up using were horizontal reflection, slight rotation and contrast reduction. Our procedure was executed as follows. For a given training image we would with probability 0.5 apply horizontal reflection. If the probability succeeded, we would haven ’augmented set’ of 2 images: the original image, and the horizontally reflected image. Next, we would, with probability 0.5 for each of the images in the augmented set, apply the rotation transformation (as discussed below). Every time the probability succeeded, our ’augmented set’ would be supplemented with an additional image. Lastly, on the images in the augmentation set we would apply the contrast reduction transformation with probability 0.5. With this stacking approach, for each training image there was a possibility of producing between 1 and 23= 8 images, inclusive (the final augmented set), which would all be added to the new training dataset as inputs to our models. The expected number of images in an augmented set from a single image is 1.5×1.5×1.5=3.375.Therefore, on average, our training set of 2140 raw images was transformed into 7222.5 input images.

3.5 HORIZONTAL REFLECTION (MIRROR)

The first data augmentation technique is fairly straight forward. We need only reflect the image and its keypoint labels horizontally and then remap the keypoint labels to their new representations (left center eye becomes right center eye, and vice versa).

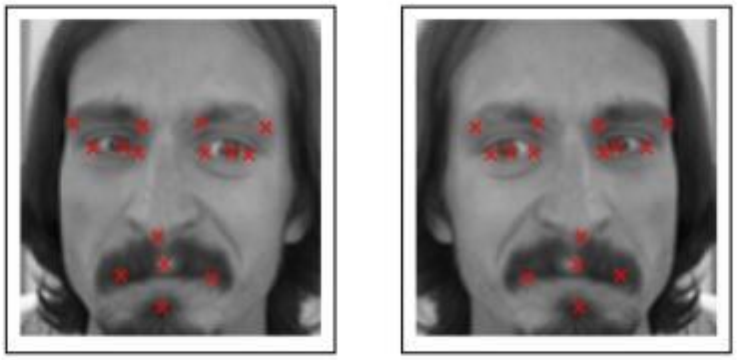


Figure16

3.6 ROTATION

For the second data augmentation we rotate the image clockwise or counterclockwise each with probability 0.5. The image pixel matrix (X) and labels are rotated using X•R, where R is the rotation matrix. The images are padded with their mean pixel value, along the edges where parts of the image were rotated out of bounds.

$$R = \begin{matrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{matrix}$$

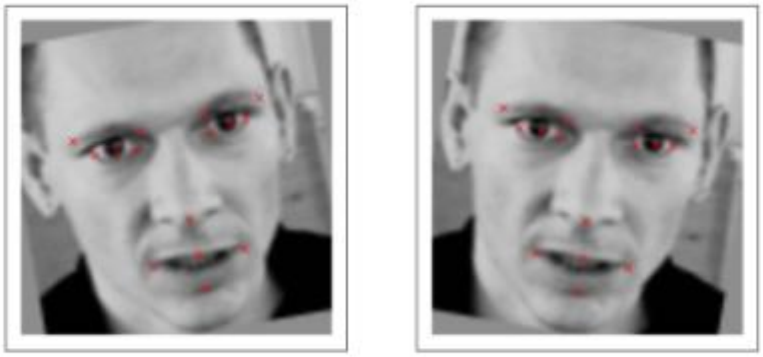


Figure17

3.7 CONTRAST JITTERING (REDUCTION)

For the third data augmentation we reduce the contrast of the grayscale image. This is done by applying the formula CR(X) below, where X is the96x96x1 input and mean(X) is the average pixel value in X. The idea is that pixel values are shifted slightly towards the images mean pixel value. The degree of shift is determined by hyper parameter which we eventually set to 0.8 after experimentation.

$$CR(X) = (\delta * X) + (1 - \delta) * mean(X)$$

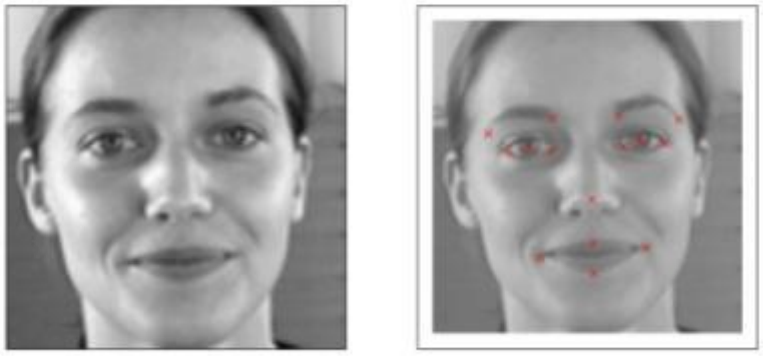


Figure18

In practice we found that each of these augmentations improved results individually and even more so in combination. We could not discern significant differences from varying the probability of application for each as long as probabilities were between 0.4 and 0.6. The impact of contrast reduction increased substantially when and was gradually less effective at higher values, and rapidly less elective at lower values. Once we found the optimal value of contrast jittering significantly improved results. This is most likely because the grayscale images within the dataset vary greatly in terms of contrast. Addition-ally, the mean pixel value varies quite significantly. The rotation angle hyper parameter produced generally good results for any angles below 12o, but otherwise didn’t vary greatly in performance.

CHAPTER 4

(PERFORMANCE ANALYSIS)

In this part, we will talk about the algorithms we used to recognize the keypoints on the faces and their task completions in order to explore their differences and find which would be the most suitable one. In addition, all the codes are also provided in case you want to run it by yourselves. However, please note that there is randomness among different experiments, which means you should not be surprised if you cannot obtain the same result.

In this section, I just focus on the specific algorithm itself. And in the next section, the contrast among different algorithms will be proposed. And we will describe our step in Python to build Neural network and Convolution neural network since they are the most challenging aspect in the experiment.

First of all, I list all the results here in tables:

	RMSE1	**RMSE2**
Knn	3.375	2.346
Linear	4.513	6.020
Lasso	3.558	2.979
Elastic	4.044	2.959
Ridge	8.464	2.609
Decision tree	3.745	4.101
Neural network	2.923	2.875
CNN	1.972	2.086

4.1) KNN

We prefer to start our discussion among the algorithms from kNN(k-NearestNeighbor) because it is very simple to understand and visualize.

KNN regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based the mean of the labels of its nearest neighbors. The advantages lie in its simplicity and effectiveness. Theoretically, the more training instances we can provide, the better the performance of Knn algorithms will be. Luckily, there is a big data set for us to train so RMSE1 has been reduced to about 3.375 with k=5. Talking about the weak points about Knn. I want to mention the Curse of Dimensionality, which significantly cut down the power of kNN algorithm. In this case, this problem is obvious because we have 30 and 8 dimensions in each data sets. Although the huge number of instances reduce the Curse of Dimensionality, to some extent, it is still rigorous.

However, Knn receives 2.346 RMSE2 on a the data set of 8 dimensions while 3.375 RMSE1 on a data set of 30 dimensions . It uncovers the Curse of Dimensionality, to some extent.

4.2) LASSO

The Lasso regression is a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer parameter values, effectively reducing the number of variables upon which the given solution is dependent. For this reason, the Lasso and its variants are fundamental to the field of compressed sensing. Under certain conditions, it can recover the exact set of non-zero weights.

Mathematically, it consists of a linear model trained with ℓ1 prior as regularizer. The objective function to minimize is:

min12nsamples||Xw−y||22+α||w||1

And it can be proved that solving for the lasso regression is equivalent to solve the following problem:

minn∑i=1(yi−β0−p∑j=1βjxij)2,p∑j=1|βj|≤s

Lasso regression enjoys a good RMSE1 with only 3.559. Some parameters may be extra since sometimes we do not need such information to recognize a face. On the other hand, parameter redundancy may cause the problem of overfitting. Lasso regression has a the ability to overcome overfitting and that is the reason why it is so popular among machine learning skills.

The choice of the α is very important because large α will squeeze almost all the parameters to zero, which obviously cannot meet our requirements, while small α does not have the ability to fulfill the task as a lasso regression, in contrast, it may perform as a normal linear regression.

The best way to choose α is cross validation. However, cross validation is not easily to do on this dataset because it is too much big so that long times would be taken to run the code. Luckily, we almost find the best choice (α =0.1). However, you should note that since there is no Theoretical guidance to α, so this may not be the unique choice.

4.3) ELASTIC NET

Elastic Net is a linear regression model trained with ℓ1 and ℓ2 prior as regularizer. This combination allows for learning a sparse model where few of the weights are non- zero like Lasso, while still maintaining the regularization properties of Ridge. Elastic-net is useful when there are multiple features which are correlated with one another. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.

A practical advantage of trading-off between Lasso and Ridge is it allows Elastic-Net to inherit some of Ridge’s stability under rotation.

In elastic net there is two parameters to be adjusted, which significantly increases the difficulty of tuning. In our experiment the RMSE1 is 4.04. Therefore, we prefer to attribute the poor performance to parameters. Although we determine the parameters using cross validation, it is still hard to tune. Finally, the parameters are fixed with $\alpha=0.1,\rho=0.5$.

Compared with Lasso regression, Elastic Net achieves a higher test error. However, we cannot conclude that Elastic Net cannot compare with the Lasso regression. Instead, Lasso regression is strong enough to prevent the system from overfitting rather than.

4.4) RIDGE REGRESSION

Compared with linear regression ridge regression balance the error and variance by adding penalty. The parameters of ridge regression are obtained my minimize:

$$n\sum i=1(yi-\beta 0-p\sum j=1\beta jxij)2\lambda p\sum j=1\beta 2j$$

If $\lambda=0$ it will degenerate into normal linear regression. In addition, it can be proved that solving for ridge regression is equivalent to solving the following problem:

$$minn\sum i=1(yi-\beta 0-p\sum j=1\beta jxij)2,p\sum j=1\beta 2j\leq s$$

Ridge regression has closed form solution as a convex optimization problem. This property improved the speed of the code largely. However, ridge regression has huge gap between RMSE1 and RMSE2, as can be seen from the table above. This problem confused us a lot during the experiment. However, after dividing the data set again, we obtain a normal result for ridge regression with RMSE1 = 2.967 and RMSE2 = 3.104. In the table above we still report the previous RMSE because we want to mention this surprising result may due to luck but still need further investigation.

4.5) DECISION TREE

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualized.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- Use a white box model. If a given situation is ob- servable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model results may be more difficult to interpret.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called over- fitting. Mechanisms such as pruning (not currently sup- ported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

The RMSE1 of decision trees is 3.75 with k=5. We have the confidence to believe that overfitting would not occur with applying only k=5 to such a big data set. However, decision tree may ignore the correlation between characters, which is an important property in this experiment.

4.6) NEURAL NETWORK

4.6.1 Package description

As you know Keras is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

In this experiment, we first study how to use the pack- ages Keras and Theano to build our neural network, then we choose two different neural networks to achieve our goal.They have a common feature is very precise, but the calculation is very large, we spent 5 days to debug them. Unfortunately, BP neural network has not reached an ideal result, but CNN's result is very good.

4.6.2 Steps to build neural network

Now, we'll give you an idea of how to build the BP neural network. This has the same idea as handwritten number recognition, but is slightly different in some detail. This is a regression problem, not a classification problem. First we'd better to use the 'MSE' to replace the 'categorical crossentropy'. Second, we must use the linear function f(x)=x as the activation function in the last layer, because the value of the other functions will be limited to 0 and 1, can not complete the predict. The other things we have done as follow:

1. We choose the different optimizer to build our neural network, such as 'Rmsprop' and 'SGD'.

2. We use 4 layers which has 300,150,50,8 neural for training.

3. After some times attempt, we choose the nb_epoch=500, batch_size=30.

4.7 CONVOLUTION NEURAL NETWORK

4.7.1 Steps to build convolution neural network

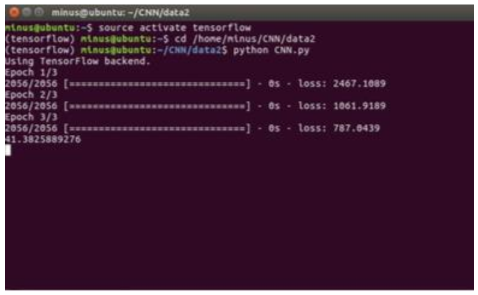
As we all know, the most popular technique in image recognition is CNN. So we spend the longest time in this method, now I will introduce the idea of our CNN.

- We use PCA method to reduce dimension from 96×96 to 16×16, This method saves 95% information, and makes the calculation faster.
- Because Keras requires the input of 4-dimensional vectors, so we need replace the 7049×16×16 image as 7649×16×16×1 image.
- We use 2 convolution layers. The first layer we use 32 filters with dimension 5×5, then the pool_size is 2×2. In the second layer, we use the 8 filter with dimension 3×3, then the pool_size is 2×2. we also use the dropout method in each layer.

- After the convolution layer, we flatten it and add a normal hidden layer with 100 neural.
- Optimizer and loss function is similar with BP network.
- After some times attempt, we choose the nb_epoch=400, batch_size=50.

We try to install the cuda and use the GPU to compute it but fail, so we only use the CPU to train this neural network, it takes a very long time. Each test time are more than 10 hours, so the number of adjusting the parameters is not enough. But fortunately, the results are pretty good!

Following is a picture of a small demo run on our own computer. However, the whole code should be run on the server.



```
minus@ubuntu: ~/CNN/data2
minus@ubuntu:~$ source activate tensorflow
(tensorflow) minus@ubuntu:~$ cd /home/minus/CNN/data2
(tensorflow) minus@ubuntu:~/CNN/data2$ python CNN.py
Using TensorFlow backend.
Epoch 1/3
2056/2056 [==============================] - 0s - loss: 2467.1089
Epoch 2/3
2056/2056 [==============================] - 0s - loss: 1061.9189
Epoch 3/3
2056/2056 [==============================] - 0s - loss: 787.0439
41.3825889276
```

Figure19: Principle Sketch of PCA

4.7.2 Analysis

The most successful approach is the convolutional neural network, which has been widely applied to image data. Convolutional neural network has three important mechanisms: (i) local receptive fields, (ii) weight sharing, and (iii)subsampling. The structure of a convolutional network is illustrated below:

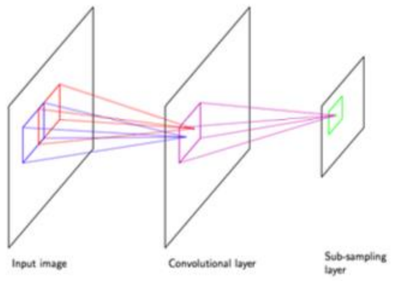


Figure20: Principle Sketch of PCA

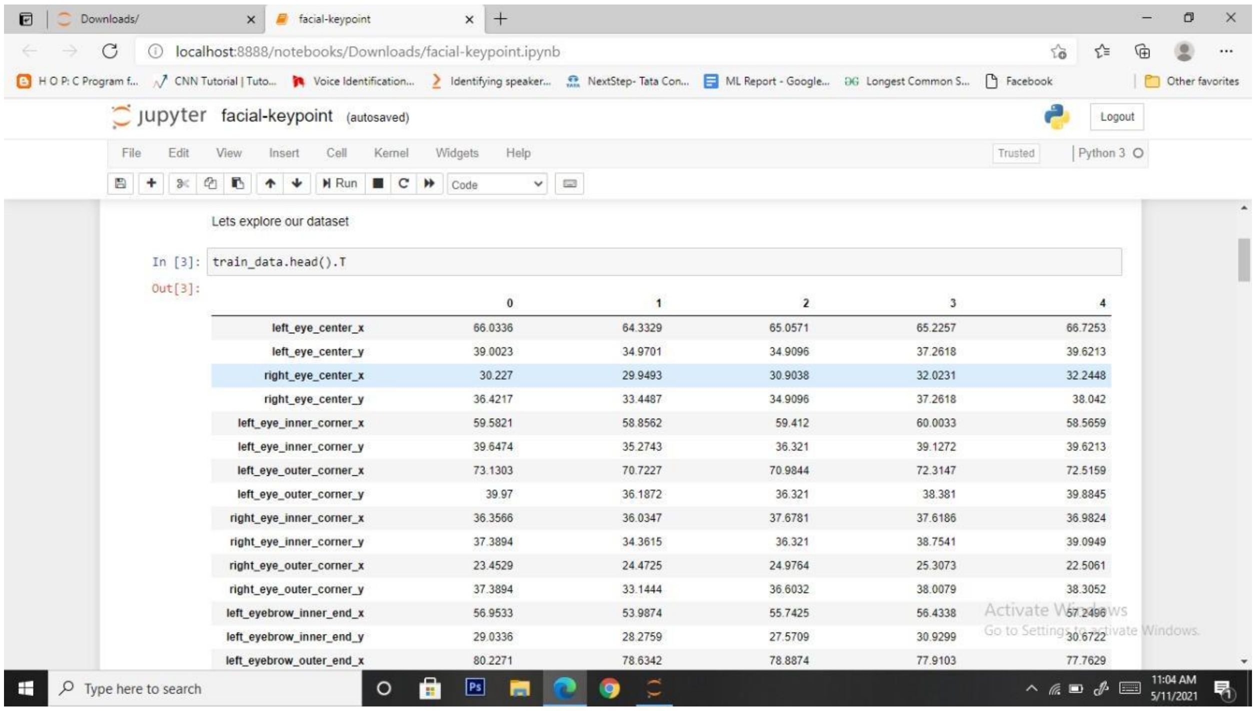
In the convolution layer the units are organized into planes, each of which is called a feature map. Units in a feature map each take inputs only from a small sub region of the image, and all of the units in a feature map are considered to share the same weight values. For instance, a feature map might consists of 100 units arranged in a 10×10 grid, with each unit taking inputs from a 5×5 pixel patch of the image. The whole feature map therefore has 25 adjustable weight parameters plus one adjustable bias parameter. Input values from a patch are linearly combined using the weights and the bias, and the result transformed by a sigmoid nonlinearity.

The whole network can be trained by error minimization using back propagation to evaluate the gradient of the error function. This involves a slight modification of the usual back propagation algorithm to ensure that the shared-weight constraints are satisfied. Due to the use of local receptive fields, the number of weights in the network is smaller than if the network were fully connected. Furthermore, the number of independent parameters to be learned from the data is much smaller still, due to the substantial numbers of constraints on the weights. Convolutional neural network is an excellent approach to image recognition because it makes the full use of local feature and shares soft weights, which is critical to images. CNN achieves the best result with 1.972 RMSE1 and 2.086 RMSE2.

CHAPTER 5

(RESULT)

3D face recognition is an important and popular area in recent years. More and more researchers are working on this field and presenting their 3D face recognition methods. In this paper, we surveyed some of the latest methods for 3D face recognition under expressions, occlusions, and pose variations. At first we summarized some various available 3D face databases. All of the above methods are tested on these databases. Almost all researchers use the following three formats of face data: point cloud, mesh and range data. All three type face data are obtained by 3D scanner. The recognition methods are mainly divided into two categories: local methods and holistic methods. Although many experiments are carried out based on the holistic method, we believe that the local method is more suitable for 3D face recognition. Compared to holistic methods, the local method has stronger robustness in terms of occlusion and can obtain better experimental results.

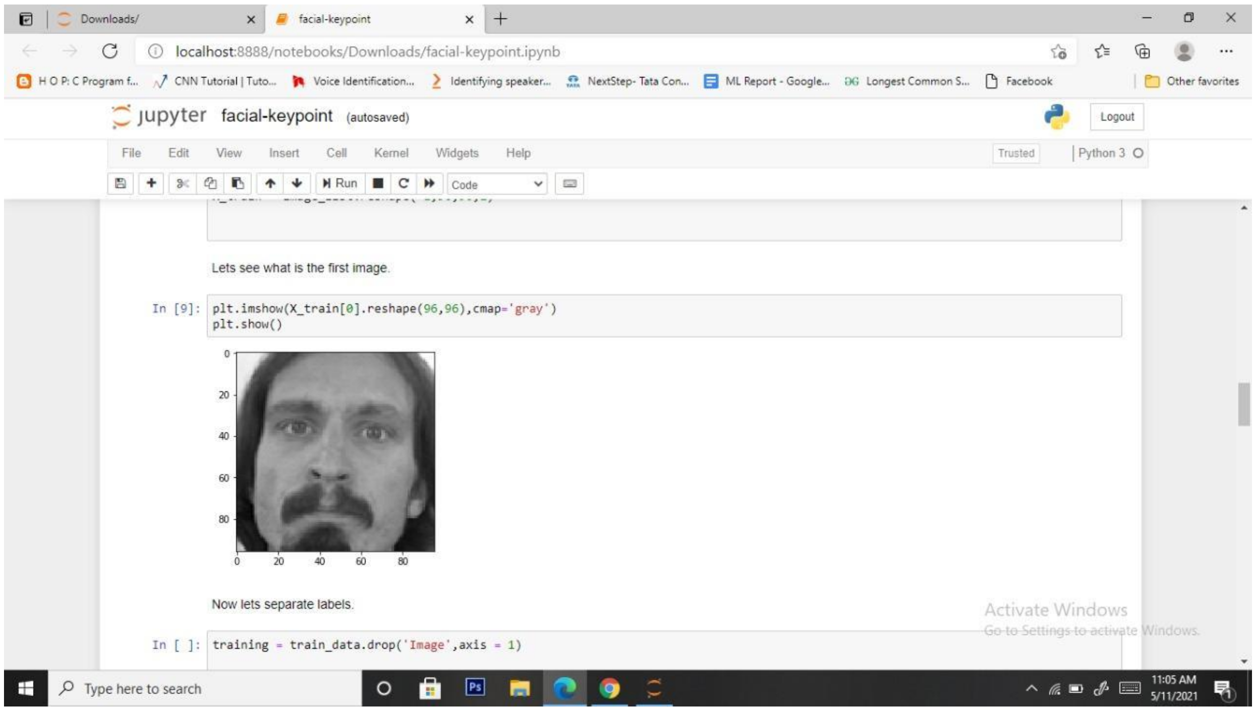

Downloads/
facial-keypoint
localhost:8888/notebooks/Downloads/facial-keypoint.ipynb
jupyter facial-keypoint (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Lets explore our dataset

```
In [3]: train_data.head().T
Out[3]:
```

	0	1	2	3	4
left_eye_center_x	66.0336	64.3329	65.0571	65.2257	66.7253
left_eye_center_y	39.0023	34.9701	34.9096	37.2618	39.6213
right_eye_center_x	30.227	29.9493	30.9038	32.0231	32.2448
right_eye_center_y	36.4217	33.4487	34.9096	37.2618	38.042
left_eye_inner_corner_x	59.5821	58.8562	59.412	60.0033	58.5659
left_eye_inner_corner_y	39.6474	35.2743	36.321	39.1272	39.6213
left_eye_outer_corner_x	73.1303	70.7227	70.9844	72.3147	72.5159
left_eye_outer_corner_y	39.97	36.1872	36.321	38.381	39.8845
right_eye_inner_corner_x	36.3566	36.0347	37.6781	37.6186	36.9824
right_eye_inner_corner_y	37.3894	34.3615	36.321	38.7541	39.0949
right_eye_outer_corner_x	23.4529	24.4725	24.9764	25.3073	22.5061
right_eye_outer_corner_y	37.3894	33.1444	36.6032	38.0079	38.3052
left_eyebrow_inner_end_x	56.9533	53.9874	55.7425	56.4338	57.2496
left_eyebrow_inner_end_y	29.0336	28.2759	27.5709	30.9299	30.6722
left_eyebrow_outer_end_x	80.2271	78.6342	78.8874	77.9103	77.7629

Figure21


Downloads/
facial-keypoint
localhost:8888/notebooks/Downloads/facial-keypoint.ipynb
jupyter facial-keypoint (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Lets see what is the first image.

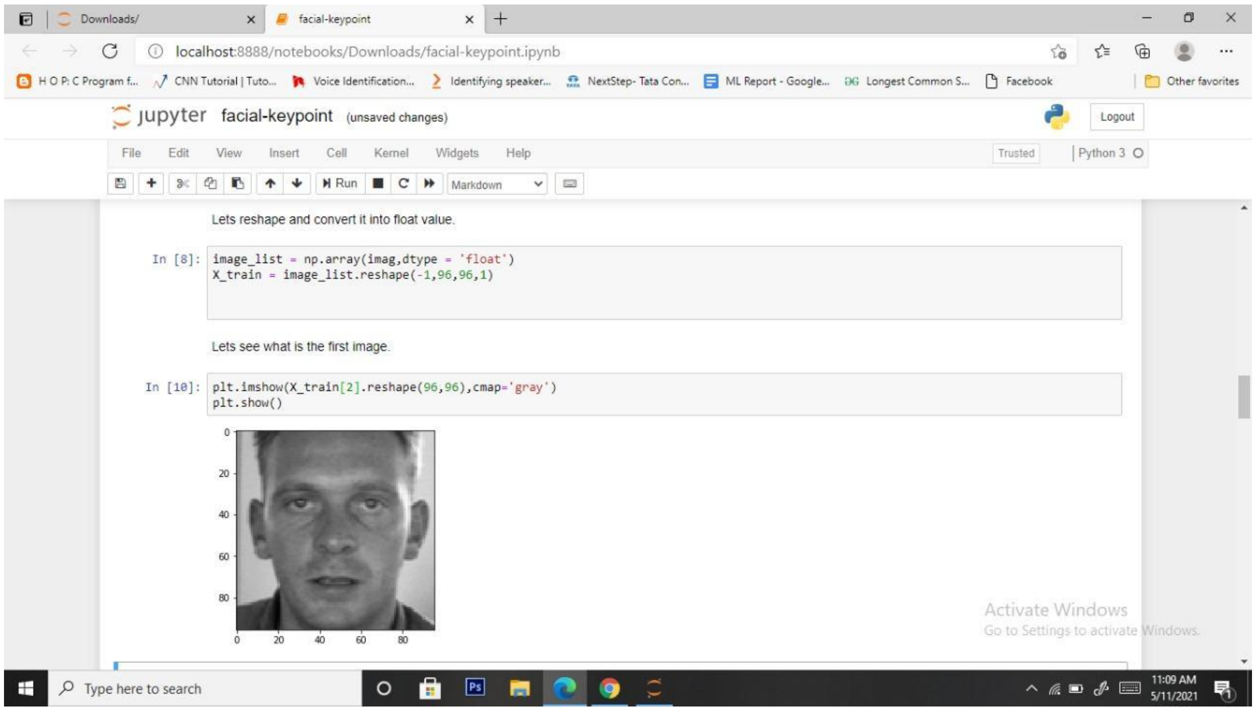
```
In [9]: plt.imshow(X_train[0].reshape(96,96),cmap='gray')
        plt.show()
```

Now lets separate labels.

```
In [ ]: training = train_data.drop('Image',axis = 1)
```

Activate Windows
Go to Settings to activate Windows.
Type here to search
11:05 AM
5/11/2021

Figure22


jupyter facial-keypoint (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Logout
Run
Markdown
localhost:8888/notebooks/Downloads/facial-keypoint.ipynb
Activate Windows
Go to Settings to activate Windows.
Type here to search
11:09 AM
5/11/2021

Lets reshape and convert it into float value.

```
In [8]: image_list = np.array(imag,dtype = 'float')
        X_train = image_list.reshape(-1,96,96,1)
```

Lets see what is the first image.

```
In [10]: plt.imshow(X_train[2].reshape(96,96),cmap='gray')
         plt.show()
```

Figure23

REFERENCES

1. Facial Keypoint Detection Competition.Kaggle, 7 May 2013. Web. 31 Dec. 2016.¡https://www.kaggle.com/c/facial-keypoints-detection¿. 1.2, 1.3.

2. Liang, Lin, et al. “Face alignment via component-based discriminative search.” Computer Vision–ECCV 2008. Springer Berlin Heidelberg, 2008. 72-85.

3. Amber, Brian, and Thomas Vetter. “Optimal landmark detection using shape models andbranch and bound.” Computer Vision (ICCV),2011 IEEE International Conference on. IEEE,2011.

4. Belhumeur, Peter N., et al. “Localizing parts offaces using a consensus of exemplars.” PatternAnalysis and Machine Intelligence, IEEE Trans-actions on 35.12 (2013): 2930-2940.

5. M. Dantone, J. Gall, G. Fanelli, and L. J. V.Gool. Real-time facial feature detection usingconditional regression forests. In Proc. CVPR,2012.

6. D. Ciresan, U. Meier, and J. Schmidhuber.Multi-column deep neural networks for imageclassification. In Proc. CVPR, 2012.

7. M. Valstar, B. Martinez, X. Binefa, and M. Pantic. Facial point detection using boosted regres-sion and graph models. In Proc. CVPR, 2010.

8. Sun, Yi, Xiaogang Wang, and Xiaoou Tang.“Deep convolutional network cascade for facialpoint detection.” Proceedings of the IEEE Con-ference on Computer Vision and Pattern Recognition. 2013.

9. Nouri, Daniel. 2015. Github. Kaggle Fa-cial Keypoints Detection tutorial. Available from https://github.com/dnouri/kfkd-tutorial/blob/master/kfkd.py

10. https://medium.com/diving-in-deep/facial-keypoints-detection-with-pytorch-86bac79141e4

f,msdmf,sd
mv

ORIGINALITY REPORT

2 % SIMILARITY INDEX	20% INTERNET SOURCES	2% PUBLICATIONS	8% STUDENT PAPERS

PRIMARY SOURCES

1	arxiv.org Internet Source	6%
2	machinelearningmastery.com Internet Source	5%
3	bdtechtalks.com Internet Source	2%
4	medium.com Internet Source	2%
5	cs231n.stanford.edu Internet Source	1%
6	www.ir.juit.ac.in:8080 Internet Source	1%
7	Submitted to La Trobe University Student Paper	1%
8	Submitted to University of Strathclyde Student Paper	<1%
9	towardsdatascience.com Internet Source	<1%