

**HMR INSTITUTE OF TECHNOLOGY & MANAGEMENT**

**HAMIDPUR, DELHI - 110036**

**Affiliated to**

**GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY**

**Sector - 16C Dwarka, Delhi - 110075, India**

**2016-2020**

**Minor Project Report**

**On**

**FINGER KNUCKLE RECOGNITION USING CONVOLUTIONAL NEURAL NETWORK**

Submitted in partial Fulfillment of the requirements for

The award of the degree of

**Bachelor of Technology**

**In**

**(ELECTRONICS & COMMUNICATION ENGINEERING)**



**Guide(s): Submitted by:**

**MR. BRAJESH KUMAR SINGH 1. MUDITA SHARMA   
 ECE-7B**

**MR. PAWAN BHUTANI Roll No. : 48513302816**

**2. IRA SHARMA**

**ECE-7B**

**Roll No: 47013302816**

**DECLARATION**

We, student(s) of B.Tech (ECE 7thSem) hereby declare that the Minor Project entitled **“Finger Knuckle Recognition using Convolutional Neural Network”** which is submitted to Department of ECE (Electronics & Communication Engineering), HMR Institute of Technology & Management, Hamidpur Delhi, affiliated to Guru Gobind Singh Indraprastha University, Dwarka(New Delhi) in partial fulfilment of requirement for the award of the degree of Bachelor of Technology in ECE, has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition. The list of member(s) involved in the project is listed below: -

Sno. Student Name Enrollment No. Student’s Signature

1. Mudita Sharma 48513302816
2. Ira Sharma 47013302816

This is to certify that the above statement made by the candidate(s) is correct to the best of my knowledge.

|  |  |
| --- | --- |
| New Delhi | Mr. Brajesh Kumar Singh  Assistant Professor |
| Date: | Signature of Guide |
|  |  |

**Mr. Avadesh Kumar Sharma**

**(Head of Department)**

**(Assistant Professor of Electronics & Communication Department)**

**HMRITM Hamidpur, New Delhi-110036**

**ACKNOWLEDGEMENT**

We have taken lots of efforts in this project. The successful and final outcome of this project required lots of guidance and assistance from many people and we are extremely fortunate to have completed our project work. Whatever we have done is only due to such guidance and assistance and we should not forget to thank them. This project would not have been possible without the kind support and help of our mentors **“Mr. Brajesh Kumar Singh”** and **“Mr. Pawan Bhutani”**. We would like to extend our sincere thanks to both of them.

We are highly indebted to our teachers and mentors for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would like to express our gratitude towards our parents, teachers and friends for their kind co-operation and encouragement which helped us in completion of this project.

We are extremely grateful to everyone for providing such co-operation.

THANKYOU!!

1. Mudita Sharma (48513302816)
2. Ira Sharma (47013302816)

**ABSTRACT**

The project entitled “Finger Knuckle Recognition using Convolutional Neural Network (CNN)” works on one of the most important algorithms of deep learning i.e. Convolutional Neural network or CNN. We have prepared a working code of this project using Jupyter Notebook and Google Colab which is helpful in line by line execution of the code.

This report starts with an introduction to the basic concept of the Machine Learning as well as deep learning. It mainly focuses on the main algorithm used in this project which is Convolutional Neural Network. It briefly explains all its layers and their uses.

It explains the Dataset and its usage in the code. It also informs us about the extraction of dataset from the official websites.

The code is explained step by step, including all the functions, libraries and algorithms used. Screenshots of the output makes the project report more interactive and easy to understand.

We’ve used many histograms and pictures in the code, which is the best way to explain the users about various aspects and factors of our code.

We have given its application and how will it be implemented in the future and then finally concluded the report with a briefly explained summary. We ended the report with bibliography which contains all the useful resources or websites that helped us while making the project.

We hope that the readers like the project!!

**CONTENTS**

Declaration 3

Acknowledgement 4

Abstract 5

Content 6-7

**Chapter 1 - Introduction** **8-12**

1.1 General Introduction 8

1.2 Machine Learning 9-10

1.3 Deep Learning Vs Machine Learning 11-12

**Chapter 2 – Convolutional Neural Network (CNN) 13-24**

* 1. Convolutional Neural Network 13-14
  2. Architecture of CNN 14
  3. Convolutional Layer 15-16
  4. Pooling Layer 16-17
  5. Fully Connected 17-18
  6. Non-Linearity layers 19-20
  7. Applications of CNN 20
  8. Advantages of CNN 21
  9. Disadvantages of CNN 22
  10. Benefits of using CNN 23-24

**Chapter 3 – Dataset 25-26**

3.1 Dataset Explanation 25

3.2 Sample Dataset 25-26

3.3 Histogram 26

**Chapter 4 – Methodology 27-84**

4.1 Code 27-57

4.2 Code explanation 58-84

**Chapter 5 – Results 85-87**

5.1 Model 85

5.2 Training epochs 86

5.3 Accuracy & Loss Graphs 86-87

**Chapter 6 – Conclusion 88**

**Chapter 7 – Bibliography 89**

**Chapter-1 Introduction**

**1.1 General Introduction**

Biometrics are part of the cutting edge of technology. Put simply, biometrics are any metrics related to human features. The most common examples of a biometric recognition system is the iPhone’s fingerprint, facial recognition, finger knuckle recognition technology. As an emerging technology, biometric systems can add great convenience by replacing passwords and helping law enforcement catch criminals. Biometric identifiers also act as access control in secure environments, both physical and digital.

Finger-knuckle-print is one of the emerging biometric traits. The region of interest is the area where the maximum information is centered, for a finger knuckle it is the area surrounding the knuckle region. The finger knuckle print refers to the inherent skin patterns that are formed at the joints in the finger back surface. Recently it has been found that the finger knuckle print is highly rich in textures and can be used to uniquely identify a person.

Automated security is one of the major concerns of modern times. Secure and reliable authentication systems are in great demand. A biometric trait like Finger Knuckle Print (FKP) of a person is unique and secure. In the recent years, hand based biometrics is extensively used for personal recognition. Finger Knuckle has unique bending and makes a distinctive biometric identifier.

The determination is based on Deep Learning and Convolutional Neural Network (CNN). Here we have used a Training Dataset of about 500 images out of which we have a total of 100 classes each containing 5 images. Using these images we have trained our model. Finally the testing is done on a set of 5 image on which final prediction is made.

In order to use the best technique of Machine learning, we have compared the three algorithms i.e. Convolutional Neural Network, Support Vector Machine and Random Forest. On comparison, we found that CNN works out the best for Image Preprocessing and its prediction.

This project can be very useful as this can help in accurate recognition of faces which is helpful for security.

**1.2 Machine Learning**

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. **Machine learning focuses on the development of computer programs** that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. **The primary aim is to allow the computers learn automatically** without human intervention or assistance and adjust actions accordingly.

### Some machine learning methods

Machine learning algorithms are often categorized as supervised or unsupervised.

* **Supervised machine learning algorithms**can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.
* **Unsupervised machine learning algorithms**are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn’t figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.
* **Semi-supervised machine learning algorithms** fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn’t require additional resources.
* **Reinforcement machine learning algorithms**is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

**1.3 Deep Learning vs. Machine Learning**In practical terms, deep learning is just a subset of machine learning. It technically is machine learning and functions in a similar way (hence why the terms are sometimes loosely interchanged), but its capabilities are different.

Basic machine learning models do become progressively better at whatever their function is, but they still some guidance. If an ML algorithm returns an inaccurate prediction, then an engineer needs to step in and make adjustments. But with a deep learning model, the algorithms can determine on their own if a prediction is accurate or not.

Let’s go back to the flashlight example: it could be programmed to turn on when it recognizes the audible cue of someone saying the word “dark”. Eventually, it could pick up any phrase containing that word. Now if the flashlight had a deep learning model, it could maybe figure out that it should turn on with the cues “I can’t see” or “the light switch won’t work”. A deep learning model is able to learn through its own method of computing – its own “brain”, if you will.

How does deep learning work?

A deep learning model is designed to continually analyze data with a logic structure similar to how a human would draw conclusions. To achieve this, deep learning uses a layered structure of algorithms called an artificial neural network (ANN). The design of an ANN is inspired by the biological neural network of the human brain. This makes for machine intelligence that’s far more capable than that of standard machine learning models.

It’s a tricky prospect to ensure that a deep learning model doesn’t draw incorrect conclusions (which is probably what keeps Elon up at night), but when it works as it’s intended to, functional deep learning is a scientific marvel and the potential backbone of true artificial intelligence.

A great example of deep learning is [Google’s AlphaGo](https://deepmind.com/research/alphago/). Google created a computer program that learned to play the abstract board game called Go, a game known for requiring sharp intellect and intuition. By playing against professional Go players, AlphaGo’s deep learning model learned how to play at a level not seen before in artificial intelligence, and all without being told when it should made a specific move (as it would with a standard machine learning model). It caused quite a stir when AlphaGo defeated multiple world-renowned “masters” of the game; not only could a machine grasp the complex and abstract aspects of the game, it was becoming one of the greatest players of it as well.

**To recap the differences between the two:**

* **Machine learning uses algorithms to parse data, learn from that data, and make informed decisions based on what it has learned**

* **Deep learning structures algorithms in layers to create an “artificial neural network” that can learn and make intelligent decisions on its own**

* **Deep learning is a subfield of machine learning. While both fall under the broad category of artificial intelligence, deep learning is what powers the most human-like artificial intelligence**

**Chater-2 Convolutional Neural Network (CNN)**

**2.1 Convolutional Neural Network**  
In [deep learning](https://en.wikipedia.org/wiki/Deep_learning), a convolutional neural network (CNN, or ConvNet) is a class of [deep neural networks](https://en.wikipedia.org/wiki/Deep_neural_network), most commonly applied to analyzing visual imagery.

CNNs are [regularized](https://en.wikipedia.org/wiki/Regularization_(mathematics)) versions of [multilayer perceptrons](https://en.wikipedia.org/wiki/Multilayer_perceptron). Multilayer perceptrons usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of [neural networks](https://datascience.hubs.vidyard.com/watch/CYfbzzj57RPfCwoMnEHD4M) that specializes in processing data that has a grid-like topology, such as an image. A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what color each pixel should be.

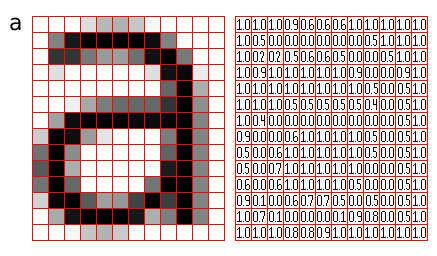
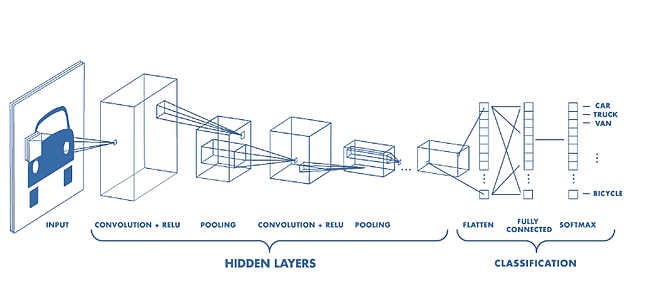


Figure 1: Representation of image as a grid of pixels ([Source](http://pippin.gimp.org/image_processing/images/sample_grid_a_square.png))

The human brain processes a huge amount of information the second we see an image. Each neuron works in its own receptive field and is connected to other neurons in a way that they cover the entire visual field. Just as each neuron responds to stimuli only in the restricted region of the visual field called the receptive field in the biological vision system, each neuron in a CNN processes data only in its receptive field as well. The layers are arranged in such a way so that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along. By using a CNN, one can [enable sight to computers](https://www.datascience.com/blog/computer-vision-in-artificial-intelligence).

**2.2 Convolutional Neural Network Architecture**

A CNN typically has three layers: a convolutional layer, pooling layer, and fully connected layer.

  
Figure 2: Architecture of a CNN

**2.3 Convolution Layer**

The convolution layer is the core building block of the CNN. It carries the main portion of the network’s computational load.

This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image, but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

During the forward pass, the kernel slides across the height and width of the image producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

If we have an input of size W x W x D and Dout number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

Screen Shot 2019-03-05 at 10.14.59 AM

This will yield an output volume of size Woutx Wout x Dout.

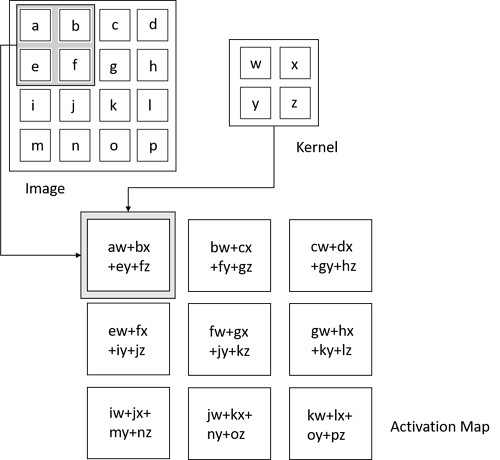


Figure 3: Convolution Operation

**2.4 Pooling Layer**The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

There are several pooling functions such as the average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. However, the most popular process is max pooling, which reports the maximum output from the neighborhood.

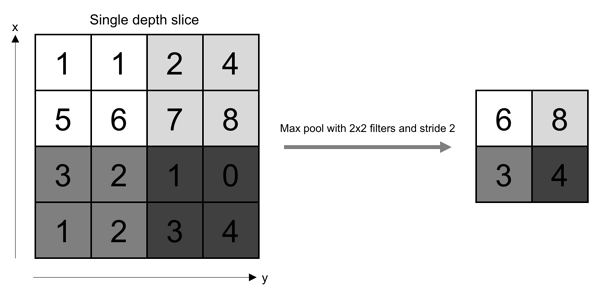
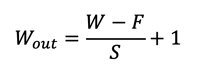


Figure 4: Pooling Operation

If we have an activation map of size W x W x D, a pooling kernel of spatial size F, and stride S, then the size of output volume can be determined by the following formula:



This will yield an output volume of size Woutx Wout x D.

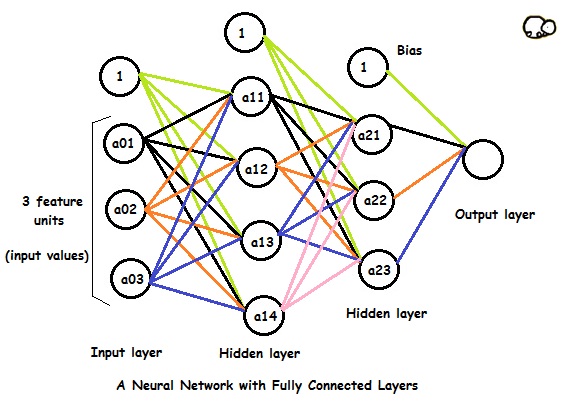
In all cases, pooling provides some translation invariance which means that an object would be recognizable regardless of where it appears on the frame.

**2.5 Fully Connected Layer**Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect.

The FC layer helps map the representation between the input and the output.

Fully Connected layers in a neural networks are those layers where all the inputs from one layer are connected to every activation unit of the next layer. In most popular machine learning models, the last few layers are full connected layers which compiles the data extracted by previous layers to form the final output. It is the second most time consuming layer second to Convolution Layer.

The diagram below clarifies the statement.

[](https://iq.opengenus.org/content/images/2019/03/fc.jpg)

In the above model:

* The first/input layer has 3 feature units and there are 4 activation units in the next hidden layer.
* The 1's in each layer are bias units.
* a01, a02 and a03 are input values to the neural network.They are basically features of the training example.
* The 4 activation units of first hidden layer is connected to all 3 activation units of second hidden layer The weights/parameters connect the two layers.

**2.6 Non-Linearity Layers**

Since convolution is a linear operation and images are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map.

There are several types of non-linear operations, the popular ones being:

**1. Sigmoid**

The sigmoid non-linearity has the mathematical form σ(κ) = 1/(1+e¯κ). It takes a real-valued number and “squashes” it into a range between 0 and 1.

However, a very undesirable property of sigmoid is that when the activation is at either tail, the gradient becomes almost zero. If the local gradient becomes very small, then in backpropagation it will effectively “kill” the gradient. Also, if the data coming into the neuron is always positive, then the output of sigmoid will be either all positives or all negatives, resulting in a zig-zag dynamic of gradient updates for weight.

**2. Tanh**

Tanh squashes a real-valued number to the range [-1, 1]. Like sigmoid, the activation saturates, but—unlike the sigmoid neurons—its output is zero centered.

**3. ReLU**

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function ƒ(κ)=max (0,κ). In other words, the activation is simply threshold at zero.

In comparison to sigmoid and tanh, ReLU is more reliable and accelerates the convergence by six times.

Unfortunately, a con is that ReLU can be fragile during training. A large gradient flowing through it can update it in such a way that the neuron will never get further updated. However, we can work with this by setting a proper learning rate.

**4. Softmax**

**Softmax** extends this idea into a multi-class world. That is, Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. This additional constraint helps training converge more quickly than it otherwise would.

**2.7 Applications of Convolutional Neural Network**

**1. Object detection:** With CNN, we now have sophisticated models like [R-CNN](https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.pdf), [Fast R-CNN](https://arxiv.org/pdf/1504.08083.pdf), and [Faster R-CNN](https://arxiv.org/pdf/1506.01497.pdf) that are the predominant pipeline for many object detection models deployed in autonomous vehicles, facial detection, and more.

**2. Semantic segmentation:** In 2015, a group of researchers from Hong Kong developed a CNN-based [Deep Parsing Network](https://arxiv.org/pdf/1509.02634.pdf) to incorporate rich information into an image segmentation model. Researchers from UC Berkeley also built [fully convolutional networks](https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf) that improved upon state-of-the-art semantic segmentation.

**3. Image captioning:** CNNs are used with recurrent neural networks to write captions for images and videos. This can be used for many applications such as activity recognition or describing videos and images for the visually impaired. It has been heavily deployed by YouTube to make sense to the huge number of videos uploaded to the platform on a regular basis.

4. CNNs have wide applications in image and video recognition, recommender systems and natural language processing. In this article, the example that I will take is related to Computer Vision.

**2.8 Advantages of CNN**

1. The usage of CNNs is motivated by the fact that they can capture / are able to learn relevant features from an image /video at different levels similar to a human brain. This is **feature learning!**Conventional neural networks cannot do this.
2. Another main feature of CNNs is **weight sharing**. Let’s take an example to explain this. Say you have a one layered CNN with 10 filters of size 5x5. Now you can simply calculate parameters of such a CNN, it would be 5\*5\*10 weights and 10 biases i.e. **5\* 5\*10 + 10 = 260 parameters**. Now let’s take a simple one layered NN with 250 neurons, here the number of weight parameters depending on the size of images is ‘250 x K’ where size of the image is P X M and K = (P \*M). Additionally, you need ‘M’ biases. For the MNIST data as input to such a NN we will have (**250\*784+1 = 19601) parameters.**Clearly, CNN is more efficient in terms of memory and complexity. Imagine NNs and CNNs with billions of neurons, then CNNs would be less complex and saves memory compared to the NN.
3. In terms of performance, CNNs outperform NNs on conventional image recognition tasks and many other tasks. Look at the Inception model, Resnet50 and many others for instance.
4. For a completely new task / problem CNNs are very good **feature extractors.**This means that you can extract useful attributes from an already trained CNN with its trained weights by feeding your data on each level and tune the CNN a bit for the specific task. E.g. : Add a classifier after the last layer with labels specific to the task. This is also called **pre-training** and CNNs are very efficient in such tasks compared to NNs. Another advantage of this pre-training is we avoid training of CNN and save memory, time. The only thing you have to train is the classifier at the end for your labels.

**2.9 Disadvantages of CNN**

In Machine Learning, a convolutional neural network is a class of deep, feed forward artificial neural networks that has successfully been applied to analyzing visual imagery. A Convolutional Neural Networks has some drawbacks some are listed below:

* Hyperparameter tuning is non-trivial.
* Need a large dataset.
* The scale of a net's weights (and of the weight updates) is very important for performance. When the features are of the same type (pixels, word counts, etc), this is not a problem. However, when the features are heterogeneous--like in many Kaggle datasets--your weights and updates will all be on different scales (so you need to standardize your inputs in some way).
* cost effective
* A convolution is a significantly slower operation than, say maxpool, both forward and backward. If the network is pretty deep, each training step is going to take much longer.
* If you don't have a good GPU they are quite slow to train (for complex tasks).

**2.10 Benefits of using CNN**

The main motivation behind the emergence of CNNs in deep learning scenarios has been to address many of the limitations that traditional neural networks faced when applied to those problems. When used in areas like image classification, traditional fully-connected neural networks simply don’t scale well due to their disproportionally large number of connections. CNNs bring a few new ideas that contribute to improve the efficiency of deep neural networks. Let’s explore a few of those some of the fundamental principles leveraged by CNNs:

**1.  Sparse Representations**

Let’s assume that you are working on an image classification problem that involves the analysis of large pictures that are millions of pixels in size. A traditional neural network will model the knowledge using matrix multiplication operations that involve every input and every parameter which results easily in tens of billions of computations. Remember that CNNs are based on convolution operations between and input and a kernel tensors? Well, it turns out that the kernel in convolution functions tends to be drastically smaller than the input which simplifies the number of computations required to train the model or to make predictions. In our sample scenario, a potential CNN algorithm will focus only on relevant features of the input image requiring fewer parameters to use in the convolution. The result could be a few billion operations smaller and more efficient than traditional fully-connected neural networks.

**2. Parameter Sharing**

Another important optimization technique used in CNNs is known as parameter sharing. Conceptually, parameter sharing simply refers to the fact that CNNs tend to reuse the same parameters across different functions in the deep neural network. More specifically, parameter sharing entails that the weight parameters will be used on every position of the input which will allow the model to learn a single set of weights once instead of a different set for every function. Parameter sharing in CNNs typically results on massive savings in memory compared to traditional models.

1. **Equivariance**

Equivariance is my favorite property of CNNs and one that can be seen as a specific type of parameter sharing. Conceptually, a function can be considered equivariance if, upon a change in the input, a similar change is reflected in the output. Using a mathematically nomenclature, a function f(x) is considered equivariant to a function g() if f(g(x))= g(f(x)). It turns out that convolutions are equivariant to many data transformation operations which means that we can predict how specific changes in the input will be reflected in the output.

**Chapter-3 Dataset**

**3.1 Dataset Explanation**

There has been increasing interest in studying finger knuckle patterns to establish human identity in wide range of commercial and forensic applications. Prior efforts have however focused on evaluating major finger knuckle which are formed on the finger dorsal surface joining proximal phalanx and middle phalanx bones. This database is established to investigate the possible use of ‘minor’ finger knuckle patterns which are formed on the finger surface joining distal phalanx and middle phalanx bones. The ‘minor’ finger knuckle patterns can either be used as independent biometric patterns or employed to improve the performance from the major finger knuckle patterns. The objective in this work has been to establish a large scale (over 500 different people) knuckle image database for the research and make it available in the public domain to help initiate research efforts in comparing, exploring relationship and combining both knuckle patterns which can often be extracted simultaneously.

**Brief Description**

The Hong Kong Polytechnic University contactless finger knuckle images database (Version 1.0) is contributed from the male and female volunteers. This database has been largely acquired in The Hong Kong Polytechnic University campus and IIT Delhi Campus during 2006-2013 using a contactless setup that simply uses a hand held camera. This database has 2515 finger dorsal images from the middle finger of 503 subjects, all the images are in bitmap (\*.bmp) format. In this dataset about 88% of the subjects are younger than 30 years. This database also provides two session finger knuckle images acquired after very long interval (4 to 7 years) to ascertain stability of knuckle crease and curved lines.

In our project we have used only 100 classes to predict the results.

Each class has 5 images.

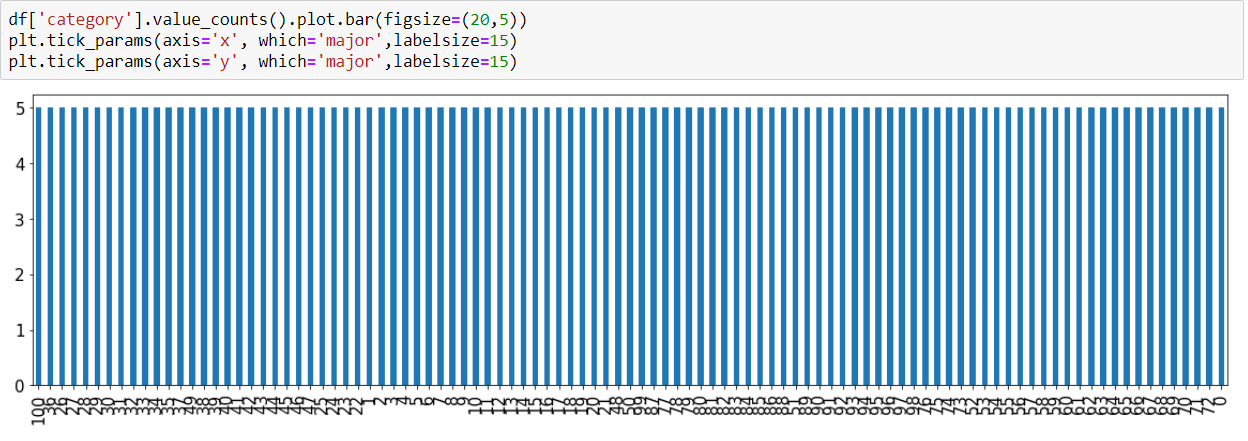
**3.2 Sample Dataset**

Our dataset looks like:



**3.3 Histogram**

This histogram shows the total number of classes i.e. 100 and the images contained in each of them i.e. 5



**Chapter-4 Methodology**

**4.1 Code**

Import numpy as np

import pandas as pd

from keras.preprocessing.image import ImageDataGenerator, load\_img

from keras.utils import to\_categorical

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

import random

import os

print(os.listdir(r"C:\Users\mudit\MUDITA\minor project\dataset"))

print(os.listdir(r"C:\Users\mudit\MUDITA\minor project\dataset\train"))

FAST\_RUN = False

IMAGE\_WIDTH=128

IMAGE\_HEIGHT=128

IMAGE\_SIZE=(IMAGE\_WIDTH, IMAGE\_HEIGHT)

IMAGE\_CHANNELS=3

filenames = os.listdir(r"C:\Users\mudit\MUDITA\minor project\dataset\train")

categories = []

for filename in filenames:

    category = filename.split('\_')[0]

   if category =='0':

        categories.append(0)

    elif category =='1':

        categories.append(1)

    elif category == '2':

        categories.append(2)

    elif category == '3':

        categories.append(3)

    elif category == '4':

        categories.append(4)

    elif category == '5':

        categories.append(5)

    elif category == '6':

        categories.append(6)

    elif category == '7':

        categories.append(7)

    elif category == '8':

        categories.append(8)

    elif category == '9':

        categories.append(9)

    elif category == '10':

        categories.append(10)

    elif category == '11':

        categories.append(11)

    elif category == '12':

        categories.append(12)

    elif category == '13':

        categories.append(13)

    elif category == '14':

        categories.append(14)

    elif category =='15':

        categories.append(15)

    elif category == '16':

        categories.append(16)

    elif category == '17':

        categories.append(17)

    elif category == '18':

        categories.append(18)

    elif category == '19':

        categories.append(19)

    elif category == '20':

        categories.append(20)

    elif category == '21':

        categories.append(21)

    elif category == '22':

        categories.append(22)

    elif category == '23':

        categories.append(23)

    elif category == '24':

        categories.append(24)

    elif category == '25':

        categories.append(25)

    elif category == '26':

        categories.append(26)

    elif category == '27':

        categories.append(27)

    elif category == '28':

        categories.append(28)

    elif category =='29':

        categories.append(29)

    elif category == '30':

        categories.append(30)

    elif category == '31':

        categories.append(31)

    elif category == '32':

        categories.append(32)

    elif category == '33':

        categories.append(33)

    elif category == '34':

        categories.append(34)

    elif category == '35':

        categories.append(35)

    elif category == '36':

        categories.append(36)

    elif category == '37':

        categories.append(37)

    elif category == '38':

        categories.append(38)

    elif category == '39':

        categories.append(39)

    elif category == '40':

        categories.append(40)

    elif category == '41':

        categories.append(41)

    elif category == '42':

        categories.append(42)

    elif category =='43':

        categories.append(43)

    elif category == '44':

        categories.append(44)

    elif category == '45':

        categories.append(45)

    elif category == '46':

        categories.append(46)

    elif category == '47':

        categories.append(47)

    elif category == '48':

        categories.append(48)

    elif category == '49':

        categories.append(49)

    elif category == '50':

        categories.append(50)

    elif category =='51':

        categories.append(51)

    elif category == '52':

        categories.append(52)

    elif category == '53':

        categories.append(53)

    elif category == '54':

        categories.append(54)

    elif category == '55':

        categories.append(55)

    elif category == '56':

        categories.append(56)

    elif category == '57':

        categories.append(57)

    elif category == '58':

        categories.append(58)

    elif category == '59':

        categories.append(59)

    elif category == '60':

        categories.append(60)

    elif category =='61':

        categories.append(61)

    elif category == '62':

        categories.append(62)

    elif category == '63':

        categories.append(63)

    elif category == '64':

        categories.append(64)

    elif category == '65':

        categories.append(65)

    elif category == '66':

        categories.append(66)

    elif category == '67':

        categories.append(67)

    elif category == '68':

        categories.append(68)

    elif category == '69':

        categories.append(69)

    elif category == '70':

        categories.append(70)

    elif category =='71':

        categories.append(71)

    elif category == '72':

        categories.append(72)

    elif category == '73':

        categories.append(73)

    elif category == '74':

        categories.append(74)

    elif category == '75':

        categories.append(75)

    elif category == '76':

        categories.append(76)

    elif category == '77':

        categories.append(77)

    elif category == '78':

        categories.append(78)

    elif category == '79':

        categories.append(79)

    elif category == '80':

        categories.append(80)

    elif category =='81':

        categories.append(81)

    elif category == '82':

        categories.append(82)

    elif category == '83':

        categories.append(83)

    elif category == '84':

        categories.append(84)

    elif category == '85':

        categories.append(85)

    elif category == '86':

        categories.append(86)

    elif category == '87':

        categories.append(87)

    elif category == '88':

        categories.append(88)

    elif category == '89':

        categories.append(89)

    elif category == '90':

        categories.append(90)

    elif category =='91':

        categories.append(91)

    elif category == '92':

        categories.append(92)

    elif category == '93':

        categories.append(93)

    elif category == '94':

        categories.append(94)

    elif category == '95':

        categories.append(95)

    elif category == '96':

        categories.append(96)

    elif category == '97':

        categories.append(97)

    elif category == '98':

        categories.append(98)

    elif category == '99':

        categories.append(99)

    else:

        categories.append(100)

df = pd.DataFrame({

        'filename': filenames,

        'category': categories

    })

df.head()

df.head()

df['category'].value\_counts().plot.bar(figsize=(20,5))

plt.tick\_params(axis='y', which='major',labelsize=15)

plt.tick\_params(axis='y', which='major',labelsize=15)

from PIL import Image, ImageFilter

import random

import numpy as np

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(IMAGE\_WIDTH, IMAGE\_HEIG

HT, IMAGE\_CHANNELS)))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(98, activation='softmax'))

#model.add(Dense(101, activation='softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

model.summary()

from keras.callbacks import EarlyStopping, ReduceLROnPlateau

earlystop = EarlyStopping(patience=10)

learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc',

                                            patience=2,

                                            verbose=1,

                                            factor=0.5,

                                            min\_lr=0.00005)

callbacks = [earlystop, learning\_rate\_reduction]

df["category"] = df["category"].replace({0:'0',1: '1',2: '2',3:'3',4: '4',5 : '5',6: '6',7: '7',8: '8',9: '9',10: '10',11: '11',12: '12',13: '13',14: '14',15: '15',16: '16',17: '17',18:'18',19: '19',20: '20',21: '21',22: '22',23: '23',24: '24' ,25: '25',26: '26',27: '27',28: '28',29: '29',30: '30', 31: '31',32: '32',33:'33',34: '34',35 : '35',36: '36',37: '37',38: '38',39: '39',40: '40',41: '41',42: '42',43: '43',44: '44',45: '45',46: '46',47: '47',48: '48',49: '49',50: '50',51: '51',52: '52',53:'53',54: '54',55 : '55',56: '56',57: '57',58: '58',59: '59',60: '60',61: '61',62: '62',63:'63',64: '64',65 : '65',66: '66',67: '67',68: '68',69: '69',70: '70',71: '71',72: '72',73:'73',74: '74',75 : '75',76: '76',77: '77',78: '78',79: '79',80: '80',81: '81',82: '82',83:'83',84: '84',85 : '85',86: '86',87: '87',88: '88',89: '89',90: '90',91: '91',92: '92',93:'93',94: '94',95 : '95',96: '96',97: '97',98: '98',99: '99',100: '100'})

train\_df, validate\_df = train\_test\_split(df, test\_size=0.50, random\_state=42)

train\_df = train\_df.reset\_index(drop=True)

validate\_df = validate\_df.reset\_index(drop=True)

train\_df['category'].value\_counts().plot.bar(figsize=(20,5))

plt.tick\_params(axis='x', which='major',labelsize=15)

plt.tick\_params(axis='y', which='major',labelsize=15)

validate\_df['category'].value\_counts().plot.bar(figsize=(20,5))

plt.tick\_params(axis='x', which='major',labelsize=15)

plt.tick\_params(axis='y', which='major',labelsize=15)

total\_train = train\_df.shape[0]

total\_validate = validate\_df.shape[0]

batch\_size=10

train\_datagen = ImageDataGenerator

(

  rotation\_range=15,

    rescale=1./255,

    shear\_range=0.1,

    zoom\_range=0.2,

    horizontal\_flip=True,

    width\_shift\_range=0.1,

    height\_shift\_range=0.1

)

train\_generator = train\_datagen.flow\_from\_dataframe(

    train\_df,

    r"C:\Users\mudit\MUDITA\minor project\dataset\train",

    x\_col='filename',

    y\_col='category',

    target\_size=IMAGE\_SIZE,

    class\_mode='categorical',

    batch\_size=batch\_size

)

validation\_datagen = ImageDataGenerator(rescale=1./255)

validation\_generator = validation\_datagen.flow\_from\_dataframe(

    validate\_df,

    r"C:\Users\mudit\MUDITA\minor project\dataset\train",

    x\_col='filename',

    y\_col='category',

    target\_size=IMAGE\_SIZE,

    class\_mode='categorical',

    batch\_size=batch\_size

)

example\_df = train\_df.sample(n=1).reset\_index(drop=True)

example\_generator = train\_datagen.flow\_from\_dataframe(

    example\_df,

    r"C:\Users\mudit\MUDITA\minor project\dataset\train",

    x\_col='filename',

    y\_col='category',

    target\_size=IMAGE\_SIZE,

    class\_mode='categorical'

)

plt.figure(figsize=(12, 12))

for i in range(0,15):

    plt.subplot(5, 3, i+1)

    for X\_batch, Y\_batch in example\_generator:

        image = X\_batch[0]

        plt.imshow(image)

        break

plt.tight\_layout()

plt.show()

from PIL import ImageFile

ImageFile.LOAD\_TRUNCATED\_IMAGES=True

epochs=3 if FAST\_RUN else 50

history = model.fit\_generator(

    train\_generator,

    epochs=epochs,

    validation\_data=validation\_generator,

    validation\_steps=total\_validate//batch\_size,

    steps\_per\_epoch=total\_train//batch\_size,

    callbacks=callbacks

)

model.save\_weights("model.h5")

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))

ax1.plot(history.history['loss'], color='b', label="Training loss")

ax1.plot(history.history['val\_loss'], color='r', label="validation loss")

ax1.set\_xticks(np.arange(1, epochs, 1))

ax1.set\_yticks(np.arange(0, 5, 0.1))

ax2.plot(history.history['accuracy'], color='b', label="Training accuracy")

ax2.plot(history.history['val\_accuracy'], color='r',label="Validation accuracy")

ax2.set\_xticks(np.arange(1, epochs, 1))

legend = plt.legend(loc='best', shadow=True)

plt.tight\_layout()

plt.show()

import matplotlib.pyplot as plt

test\_filenames = os.listdir(r"C:\Users\mudit\MUDITA\minor project\dataset\test")

test\_df = pd.DataFrame({

    'filename': test\_filenames

})

nb\_samples = test\_df.shape[0]

test\_gen = ImageDataGenerator(rescale=1./255)

test\_generator = test\_gen.flow\_from\_dataframe(

    test\_df,

    r"C:\Users\mudit\MUDITA\minor project\dataset\test" ,

    x\_col='filename',

    y\_col=None,

    class\_mode=None,

    target\_size=IMAGE\_SIZE,

    batch\_size=batch\_size,

    shuffle=False

)

predict = model.predict\_generator(test\_generator, steps=np.ceil(nb\_samples/batch\_size))

test\_df['category'] = np.argmax(predict,axis=-1)

label\_map = dict((v,k) for k,v in train\_generator.class\_indices.items())

test\_df['category'] = test\_df['category'].replace(label\_map)

df["category"] = df["category"].replace({0:'0',1: '1',2: '2',3:'3',4: '4',5 : '5',6: '6',7: '7',8: '8',9: '9',10: '10',11: '11',12: '12',13: '13',14: '14',15: '15',16: '16',17: '17',18:'18',19: '19',20: '20',21: '21',22: '22',23: '23',24: '24' ,25: '25',26: '26',27: '27',28: '28',29: '29',30: '30', 31: '31',32: '32',33:'33',34: '34',35 : '35',36: '36',37: '37',38: '38',39: '39',40: '40',41: '41',42: '42',43: '43',44: '44',45: '45',46: '46',47: '47',48: '48',49: '49',50: '50',51: '51',52: '52',53:'53',54: '54',55 : '55',56: '56',57: '57',58: '58',59: '59',60: '60',61: '61',62: '62',63:'63',64: '64',65 : '65',66: '66',67: '67',68: '68',69: '69',70: '70',71: '71',72: '72',73:'73',74: '74',75 : '75',76: '76',77: '77',78: '78',79: '79',80: '80',81: '81',82: '82',83:'83',84: '84',85 : '85',86: '86',87: '87',88: '88',89: '89',90: '90',91: '91',92: '92',93:'93',94: '94',95 : '95',96: '96',97: '97',98: '98',99: '99',100: '100'})

test\_df['category'].value\_counts().plot.bar(figsize=(20,5))

sample\_test = test\_df

#sample\_test

plt.figure(figsize=(12, 24))

text\_labels=[]

for index, row in sample\_test.iterrows():

    filename = row['filename']

    category = row['category']

    img = load\_img(r'C:\Users\mudit\MUDITA\minor project\dataset\test/'+filename, target\_size=IMAGE\_SIZE)

    plt.subplot(6, 5,index+1)

    plt.imshow(img)

    if category =='0':

        text\_labels.append(0)

    elif category =='1':

        text\_labels.append(1)

    elif category == '2':

        text\_labels.append(2)

    elif category == '3':

        text\_labels.append(3)

    elif category == '4':

        text\_labels.append(4)

    elif category == '5':

        text\_labels.append(5)

    elif category == '6':

        text\_labels.append(6)

    elif category == '7':

        text\_labels.append(7)

    elif category == '8':

        text\_labels.append(8)

    elif category == '9':

        text\_labels.append(9)

    elif category == '10':

        text\_labels.append(10)

    elif category == '11':

        text\_labels.append(11)

    elif category == '12':

        text\_labels.append(12)

    elif category == '13':

        text\_labels.append(13)

    elif category == '14':

        text\_labels.append(14)

    elif category =='15':

        text\_labels.append(15)

    elif category == '16':

        text\_labels.append(16)

    elif category == '17':

        text\_labels.append(17)

    elif category == '18':

        text\_labels.append(18)

    elif category == '19':

        text\_labels.append(19)

    elif category == '20':

        text\_labels.append(20)

    elif category == '21':

        text\_labels.append(21)

    elif category == '22':

        text\_labels.append(22)

    elif category == '23':

        text\_labels.append(23)

    elif category == '24':

        text\_labels.append(24)

    elif category == '25':

        text\_labels.append(25)

    elif category == '26':

        text\_labels.append(26)

    elif category == '27':

        text\_labels.append(27)

    elif category == '28':

        text\_labels.append(28)

    elif category =='29':

        text\_labels.append(29)

    elif category == '30':

        text\_labels.append(30)

    elif category == '31':

        text\_labels.append(31)

    elif category == '32':

        text\_labels.append(32)

    elif category == '33':

        text\_labels.append(33)

    elif category == '34':

        text\_labels.append(34)

    elif category == '35':

        text\_labels.append(35)

    elif category == '36':

        text\_labels.append(36)

    elif category == '37':

        text\_labels.append(37)

    elif category == '38':

        text\_labels.append(38)

    elif category == '39':

        text\_labels.append(39)

    elif category == '40':

        text\_labels.append(40)

    elif category == '41':

        text\_labels.append(41)

    elif category == '42':

        text\_labels.append(42)

    elif category =='43':

        text\_labels.append(43)

    elif category == '44':

        text\_labels.append(44)

    elif category == '45':

        text\_labels.append(45)

    elif category == '46':

        text\_labels.append(46)

    elif category == '47':

        text\_labels.append(47)

    elif category == '48':

        text\_labels.append(48)

    elif category == '49':

        text\_labels.append(49)

    elif category == '50':

        categories.append(50)

    elif category =='51':

        text\_labels.append(51)

    elif category == '52':

        text\_labels.append(52)

    elif category == '53':

        text\_labels.append(53)

    elif category == '54':

        text\_labels.append(54)

    elif category == '55':

        text\_labels.append(55)

    elif category == '56':

        text\_labels.append(56)

    elif category == '57':

        text\_labels.append(57)

    elif category == '58':

        text\_labels.append(58)

    elif category == '59':

        text\_labels.append(59)

    elif category == '60':

        text\_labels.append(60)

    elif category =='61':

        text\_labels.append(61)

    elif category == '62':

        text\_labels.append(63)

    elif category == '64':

        text\_labels.append(64)

    elif category == '65':

        text\_labels.append(65)

    elif category == '66':

        text\_labels.append(66)

    elif category == '67':

        text\_labels.append(67)

    elif category == '68':

        text\_labels.append(68)

    elif category == '69':

        text\_labels.append(69)

    elif category == '70':

        text\_labels.append(70)

    elif category =='71':

        text\_labels.append(71)

    elif category == '72':

        text\_labels.append(72)

    elif category == '73':

        text\_labels.append(73)

    elif category == '74':

        text\_labels.append(74)

    elif category == '75':

        text\_labels.append(75)

    elif category == '76':

        text\_labels.append(76)

    elif category == '77':

        text\_labels.append(77)

    elif category == '78':

        text\_labels.append(78)

    elif category == '79':

        text\_labels.append(79)

    elif category == '80':

        text\_labels.append(80)

    elif category =='81':

        text\_labels.append(81)

    elif category == '82':

        text\_labels.append(82)

    elif category == '83':

        text\_labels.append(83)

    elif category == '84':

        text\_labels.append(84)

    elif category == '85':

        text\_labels.append(85)

    elif category == '86':

        text\_labels.append(86)

    elif category == '87':

        text\_labels.append(87)

    elif category == '88':

        text\_labels.append(88)

    elif category == '89':

        text\_labels.append(89)

    elif category == '90':

        text\_labels.append(90)

    elif category =='91':

        text\_labels.append(91)

    elif category == '92':

        text\_labels.append(92)

    elif category == '93':

        text\_labels.append(93)

    elif category == '94':

        text\_labels.append(94)

    elif category == '95':

        text\_labels.append(95)

    elif category == '96':

        text\_labels.append(96)

    elif category == '97':

        text\_labels.append(97)

    elif category == '98':

        text\_labels.append(98)

    elif category == '99':

        text\_labels.append(99)

    else:

        text\_labels.append(100)

    plt.xlabel(filename + '   this is  ' + category)

plt.tight\_layout()

plt.show()

**4.2 Code Explanation**

1. **import numpy as np**

[NumPy](https://en.wikipedia.org/wiki/NumPy)is the most popular mathematical library in Python. It makes working and computing large, multi-dimensional arrays and matrices super easy and fast. It has a large collection of high-level mathematical functions to operate on these arrays.

1. **import pandas as pd**

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

1. **from keras.preprocessing.image import ImageDataGenerator, load\_img**

Keras Preprocessing is the data preprocessing and data augmentation module of the Keras deep learning library. It provides utilities for working with image data, text data, and sequence data.

1. **from keras.utils import to\_categorical**

np.utils.to\_categorical is used to convert array of labeled data(from 0 to nb\_classes-1) to one-hot vector.

1. **from sklearn.model\_selection import train\_test\_split**

Split arrays or matrices into random train and test subsets.

Quick utility that wraps input validation. and next(ShuffleSplit().split(X, y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

1. **import matplotlib.pyplot as plt**

[Matplotlib](https://matplotlib.org/) is a plotting library for the Python. It can be used for plotting lines, bar-chart, graphs, histograms and even displaying Images.

1. **import random**

Randomwill help us create random numbers which will be used when we split or shuffle our data set.

1. **import os**

osis an inbuilt python package for accessing your computer and file system. It can be used to display content in directories, create new folders and even delete folders.

1. **print(os.listdir(r"C:\Users\mudit\MUDITA\minor project\dataset"))**

it prints all the files present in this directory. Here we have, test and train folders inside this directory.

1. **FAST\_RUN = False**

**IMAGE\_WIDTH=128**

**IMAGE\_HEIGHT=128**

**IMAGE\_SIZE=(IMAGE\_WIDTH, IMAGE\_HEIGHT)**

**IMAGE\_CHANNELS=3**

This is defining all the constants used in this code i.e. the complete size of image, including its width, height and its channels i.e. RGB (Red Green Blue) which is helpful in showing the colorful image.

1. **filenames = os.listdir(r"C:\Users\mudit\MUDITA\minor project\dataset\train")**

**categories = []**

**for filename in filenames:**

**category = filename.split('\_')[0]**

**if category =='0':**

**categories.append(0)**

**elif category =='1':**

**categories.append(1)**

**elif category == '2':**

**categories.append(2)**

**elif category == '3':**

**categories.append(3)**

**elif category == '4':**

**categories.append(4)**

**elif category == '5':**

**categories.append(5)**

**elif category == '6':**

**categories.append(6)**

**elif category == '7':**

**categories.append(7)**

**elif category == '8':**

**categories.append(8)**

**elif category == '9':**

**categories.append(9)**

**elif category == '10':**

**categories.append(10)**

**elif category == '11':**

**categories.append(11)**

**elif category == '12':**

**categories.append(12)**

**elif category == '13':**

**categories.append(13)**

**elif category == '14':**

**categories.append(14)**

**elif category =='15':**

**categories.append(15)**

**elif category == '16':**

**categories.append(16)**

**elif category == '17':**

**categories.append(17)**

**elif category == '18':**

**categories.append(18)**

**elif category == '19':**

**categories.append(19)**

**elif category == '20':**

**categories.append(20)**

**elif category == '21':**

**categories.append(21)**

**elif category == '22':**

**categories.append(22)**

**elif category == '23':**

**categories.append(23)**

**elif category == '24':**

**categories.append(24)**

**elif category == '25':**

**categories.append(25)**

**elif category == '26':**

**categories.append(26)**

**elif category == '27':**

**categories.append(27)**

**elif category == '28':**

**categories.append(28)**

**elif category =='29':**

**categories.append(29)**

**elif category == '30':**

**categories.append(30)**

**elif category == '31':**

**categories.append(31)**

**elif category == '32':**

**categories.append(32)**

**elif category == '33':**

**categories.append(33)**

**elif category == '34':**

**categories.append(34)**

**elif category == '35':**

**categories.append(35)**

**elif category == '36':**

**categories.append(36)**

**elif category == '37':**

**categories.append(37)**

**elif category == '38':**

**categories.append(38)**

**elif category == '39':**

**categories.append(39)**

**elif category == '40':**

**categories.append(40)**

**elif category == '41':**

**categories.append(41)**

**elif category == '42':**

**categories.append(42)**

**elif category =='43':**

**categories.append(43)**

**elif category == '44':**

**categories.append(44)**

**elif category == '45':**

**categories.append(45)**

**elif category == '46':**

**categories.append(46)**

**elif category == '47':**

**categories.append(47)**

**elif category == '48':**

**categories.append(48)**

**elif category == '49':**

**categories.append(49)**

**elif category == '50':**

**categories.append(50)**

**elif category =='51':**

**categories.append(51)**

**elif category == '52':**

**categories.append(52)**

**elif category == '53':**

**categories.append(53)**

**elif category == '54':**

**categories.append(54)**

**elif category == '55':**

**categories.append(55)**

**elif category == '56':**

**categories.append(56)**

**elif category == '57':**

**categories.append(57)**

**elif category == '58':**

**categories.append(58)**

**elif category == '59':**

**categories.append(59)**

**elif category == '60':**

**categories.append(60)**

**elif category =='61':**

**categories.append(61)**

**elif category == '62':**

**categories.append(62)**

**elif category == '63':**

**categories.append(63)**

**elif category == '64':**

**categories.append(64)**

**elif category == '65':**

**categories.append(65)**

**elif category == '66':**

**categories.append(66)**

**elif category == '67':**

**categories.append(67)**

**elif category == '68':**

**categories.append(68)**

**elif category == '69':**

**categories.append(69)**

**elif category == '70':**

**categories.append(70)**

**elif category =='71':**

**categories.append(71)**

**elif category == '72':**

**categories.append(72)**

**elif category == '73':**

**categories.append(73)**

**elif category == '74':**

**categories.append(74)**

**elif category == '75':**

**categories.append(75)**

**elif category == '76':**

**categories.append(76)**

**elif category == '77':**

**categories.append(77)**

**elif category == '78':**

**categories.append(78)**

**elif category == '79':**

**categories.append(79)**

**elif category == '80':**

**categories.append(80)**

**elif category =='81':**

**categories.append(81)**

**elif category == '82':**

**categories.append(82)**

**elif category == '83':**

**categories.append(83)**

**elif category == '84':**

**categories.append(84)**

**elif category == '85':**

**categories.append(85)**

**elif category == '86':**

**categories.append(86)**

**elif category == '87':**

**categories.append(87)**

**elif category == '88':**

**categories.append(88)**

**elif category == '89':**

**categories.append(89)**

**elif category == '90':**

**categories.append(90)**

**elif category =='91':**

**categories.append(91)**

**elif category == '92':**

**categories.append(92)**

**elif category == '93':**

**categories.append(93)**

**elif category == '94':**

**categories.append(94)**

**elif category == '95':**

**categories.append(95)**

**elif category == '96':**

**categories.append(96)**

**elif category == '97':**

**categories.append(97)**

**elif category == '98':**

**categories.append(98)**

**elif category == '99':**

**categories.append(99)**

**else:**

**categories.append(100)**

**df = pd.DataFrame({**

**'filename': filenames,**

**'category': categories**

**})**

Here training data is being prepared. In filenames we have given the complete path of the folder containing training images. Here we have made an array called category which stores 101 classes i.e.(0 to 100). Then an if else statement is used in order to distinguish between the hundred categories. If the name of image matches with the respective category then category is set to “0”,”1”,”2”…”99” otherwise it is set to “100”.

1. **df.head()**

it prints the details of filename and category of top five images in the training dataset.

1. **df.tail()**

it prints the details of filename and category of bottom five images in the training dataset.

1. **df['category'].value\_counts().plot.bar**

this command gives the count of the two classes or categories in the training dataset using bar plot

1. **from PIL import Image, ImageFilter**

The **ImageFilter** module contains definitions for a pre-defined set of filters, which can be be used with the [**Image.filter()**](https://pillow.readthedocs.io/en/5.1.x/reference/Image.html#PIL.Image.Image.filter) method.

The current version of the library provides the following set of predefined image enhancement filters:

* BLUR
* CONTOUR
* DETAIL
* EDGE\_ENHANCE
* EDGE\_ENHANCE\_MORE
* EMBOSS
* FIND\_EDGES
* SHARPEN
* SMOOTH
* SMOOTH\_MORE

1. **import random**

Randomwill help us create random numbers which will be used when we split or shuffle our data set.

1. **import numpy as np**

[NumPy](https://en.wikipedia.org/wiki/NumPy)is the most popular mathematical library in Python. It makes working and computing large, multi-dimensional arrays and matrices super easy and fast. It has a large collection of high-level mathematical functions to operate on these arrays.

1. **from keras.models import Sequential**

**from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization**

**model = Sequential()**

**model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(IMAGE\_WIDTH, IMAGE\_HEIGHT, IMAGE\_CHANNELS)))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2, 2)))**

**model.add(Dropout(0.25))**

**model.add(Conv2D(64, (3, 3), activation='relu'))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2, 2)))**

**model.add(Dropout(0.25))**

**model.add(Conv2D(128, (3, 3), activation='relu'))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2, 2)))**

**model.add(Dropout(0.25))**

**model.add(Flatten())**

**model.add(Dense(512, activation='relu'))**

**model.add(BatchNormalization())**

**model.add(Dropout(0.5))**

**model.add(Dense(98, activation='softmax'))**

**#model.add(Dense(101, activation='softmax'))**

**model.compile(loss='categorical\_crossentropy', optimizer='rmsprop', metrics=['accuracy'])**

**model.summary()**

these commands are used for building the model.

* **Input Layer**: It represent input image data. It will reshape image into single dimension array. Example your image is 64x64 = 4096, it will convert to (4096, 1) array.
* **Conv Layer**: This layer will extract features from image.
* **Pooling Layer**: This layer reduce the spatial volume of input image after convolution.
* **Fully Connected Layer**: It connect the network from a layer to another layer
* **Output Layer**: It is the predicted values layer.

Here we create a sequential model. This tells keras to stack all layers sequentially.

1. Here we create the first layer by calling the **.add()** function on the model we created and pass the type of layer we want — a **Conv2D**layer. This first layer is called the **input layer** and has some important parameters we need to set.  
   **\*\*** **filter size**[**32**]: This is the size of the output dimension (i.e. the number of output filters in the convolution)  
   **\*\*** **kernel\_size [3,3]:**This specifies the height and width of the 2D convolution window.  
   **\*\* activation** [‘**relu**’]: We select an activation function also called non-linearity to be used by our neural network.

R***eLU***(Rectified Linear Unit) is the most common activation function used today, other variations are ***leaky***R***eLU***and eLU***.***

1. Here we add a [***MaxPool2D***](https://keras.io/layers/pooling/#maxpooling2d)layer. Its function is to reduce the spatial size of the incoming features and therefore helps reduce the amount of parameters and computation in the network, thereby helping to reduce overfitting.

**Overfitting** happens when our model memorizes the training data. The model will perform excellently at training time but fail at test time.

1. Here we add a **Flatten**layer. A conv2D layers extract and learn spatial features which is then passed to a dense layer after it has been flattened. This is the work of the flatten layer.
2. Here we add a Dropout layer with value 0.5. Dropout randomly drops some layers in neural networks and then learns with the reduced network. This way, the network learns to be independent and not reliable on a single layer. Bottom-line is that it helps in overfitting.  
   ***0.5***means to randomly drop half of the layers.
3. Last layer has an output size of **2** because we have landslide and non landslide classes and the activation function used is **softmax**

We can preview the arrangement and parameter size of our convnet by calling the keras function **.summary()** on the model object.

We pass three parameters to the model.compile() command

1. L**oss [‘binary\_crossentropy’] :**We specify a loss function that our optimizer will minimize. In this case since we’re working with a 101 class problem.
2. Remember the ***optimizers***we defined earlier? we’ll going to use one of them called the ***rmsprop***. This is not a fixed choice, it is part of a process called ***hyper-parameter tuning*** which may be the difference between a world class model and a naive one.
3. Here we specify which metric we want to use in measuring our model’s performance after training. We want to know if our model is doing well or not
4. **from keras.callbacks import EarlyStopping, ReduceLROnPlateau**

Reduce learning rate when a metric has stopped improving.

Inherits From: [Callback](https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback)

1. **earlystop = EarlyStopping(patience=10)**

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training. You can pass a list of callbacks (as the keyword argument callbacks) to the .fit() method of the Sequential or Model classes. The relevant methods of the callbacks will then be called at each stage of the training.

EARLYSTOPPING is used to stop training when a monitored quantity has stopped improving.

1. **learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc',**

**patience=2,**

**verbose=1,**

**factor=0.5,**

**min\_lr=0.00005)**

Reduce learning rate when a metric has stopped improving.

1. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced

**Arguments**

* **monitor**: quantity to be monitored.
* **factor**: factor by which the learning rate will be reduced. new\_lr = lr \* factor
* **patience**: number of epochs with no improvement after which learning rate will be reduced.
* **verbose**: int. 0: quiet, 1: update messages.
* **mode**: one of {auto, min, max}. In min mode, lr will be reduced when the quantity monitored has stopped decreasing; in max mode it will be reduced when the quantity monitored has stopped increasing; in auto mode, the direction is automatically inferred from the name of the monitored quantity.
* **min\_delta**: threshold for measuring the new optimum, to only focus on significant changes.
* **cooldown**: number of epochs to wait before resuming normal operation after lr has been reduced.
* **min\_lr**: lower bound on the learning rate.

1. **callbacks = [earlystop, learning\_rate\_reduction]**

Abstract base class used to build new callbacks.

1. **df["category"] = df["category"].replace({0:'0',1: '1',2: '2',3:'3',4: '4',5 : '5',6: '6',7: '7',8: '8',9: '9',10: '10',11: '11',12: '12',13: '13',14: '14',15: '15',16: '16',17: '17',18:'18',19: '19',20: '20',21: '21',22: '22',23: '23',24: '24' ,25: '25',26: '26',27: '27',28: '28',29: '29',30: '30', 31: '31',32: '32',33:'33',34: '34',35 : '35',36: '36',37: '37',38: '38',39: '39',40: '40',41: '41',42: '42',43: '43',44: '44',45: '45',46: '46',47: '47',48: '48',49: '49',50: '50',51: '51',52: '52',53:'53',54: '54',55 : '55',56: '56',57: '57',58: '58',59: '59',60: '60',61: '61',62: '62',63:'63',64: '64',65 : '65',66: '66',67: '67',68: '68',69: '69',70: '70',71: '71',72: '72',73:'73',74: '74',75 : '75',76: '76',77: '77',78: '78',79: '79',80: '80',81: '81',82: '82',83:'83',84: '84',85 : '85',86: '86',87: '87',88: '88',89: '89',90: '90',91: '91',92: '92',93:'93',94: '94',95 : '95',96: '96',97: '97',98: '98',99: '99',100: '100'})**

**train\_df, validate\_df = train\_test\_split(df, test\_size=0.50, random\_state=42)**

**train\_df = train\_df.reset\_index(drop=True)**

**validate\_df = validate\_df.reset\_index(drop=True)**

**train\_df['category'].value\_counts().plot.bar(figsize=(20,5))**

**plt.tick\_params(axis='x', which='major',labelsize=15)**

**plt.tick\_params(axis='y', which='major',labelsize=15)**

**validate\_df['category'].value\_counts().plot.bar(figsize=(20,5))**

**plt.tick\_params(axis='x', which='major',labelsize=15)**

**plt.tick\_params(axis='y', which='major',labelsize=15)**

**total\_train = train\_df.shape[0]**

**total\_validate = validate\_df.shape[0]**

**batch\_size=10**

these commands are used for preparing the data.

Because we will use image genaretor with class\_mode="categorical". We need to convert column category into string. Then image\_generator will convert it one-hot encoding which is good for our classification.

1. **train\_datagen = ImageDataGenerator(**

**rotation\_range=15,**

**rescale=1./255,**

**shear\_range=0.1,**

**zoom\_range=0.2,**

**horizontal\_flip=True,**

**width\_shift\_range=0.1,**

**height\_shift\_range=0.1**

**)**

**train\_generator = train\_datagen.flow\_from\_dataframe(**

**train\_df,**

**r"C:\Users\mudit\MUDITA\minor project\dataset\train",**

**x\_col='filename',**

**y\_col='category',**

**target\_size=IMAGE\_SIZE,**

**class\_mode='categorical',**

**batch\_size=batch\_size**

**)**

these are used for training the generator.

Arguments specific to flow\_from\_dataframe:

* **directory** — (str)Path to the directory which contains all the images.  
  set this to   
  **None** if your x\_col contains absolute\_paths pointing to each image files instead of just filenames.
* **x\_col**— (str) The name of the column which contains the filenames of the images.
* **y\_col**— (str or list of str) If class\_mode is not “raw” or not “input” you should pass the name of the column which contains the class names.  
  None, if used for test\_generator.
* **class\_mode**— (str) Similar to flow\_from\_directory, this accepts “categorical”(default), ”binary”, ”sparse”, ”input”, None and also an extra argument “raw”.  
  If class\_mode is set to “  
  **raw**” it treats the data in the column or list of columns of the dataframe as raw target values(which means you should be sure that data in these columns must be of numerical datatypes), will be helpful if you’re building a model for regression task like predicting the angle from the images of steering wheel or building a model that needs to predict multiple values at the same time.  
  For Test generator  
  *:*Set this to **None***,*to return only the images.
* **batch\_size:** For train and valid generator you can keep this according to your needs but for test generator:Set this to some number that divides your total number of images in your test set exactly.   
  Actually, you should set the “batch\_size” in both train and valid generators to some number that divides your total number of images in your train set and valid respectively, but this doesn’t matter before because even if batch\_size doesn’t match the number of samples in the train or valid sets and some images gets missed out every time we yield the images from generator, but it would be sampled the very next epoch you train.  
  But for the test set, you should sample the images exactly once, no less or no more. If Confusing, just set it to 1(but maybe a little bit slower).
* **shuffle:** Set this to *False(For Test generator only, for others set True),*because you need to yield the images in “order”, to predict the outputs and match them with their unique ids or filenames.
* **drop\_duplicates:**If you’re for some reason don’t want duplicate entries in your dataframe’s x\_col, set this to False, default is True.
* **validate\_filenames:**whether to validate image filenames in x\_col. If True, invalid images will be ignored. Disabling this option can lead to speed-up in the instantiation of this class if you have a huge amount of files**,**default is True.

1. **validation\_datagen = ImageDataGenerator(rescale=1./255)**

**validation\_generator = validation\_datagen.flow\_from\_dataframe(**

**validate\_df,**

**r"C:\Users\mudit\MUDITA\minor project\dataset\train",**

**x\_col='filename',**

**y\_col='category',**

**target\_size=IMAGE\_SIZE,**

**class\_mode='categorical',**

**batch\_size=batch\_size**

)

These are the commands for validation generator. They are similar to the training generator as explained above.

1. **example\_df = train\_df.sample(n=1).reset\_index(drop=True)**

**example\_generator = train\_datagen.flow\_from\_dataframe(**

**example\_df,**

**r"C:\Users\mudit\MUDITA\minor project\dataset\train",**

**x\_col='filename',**

**y\_col='category',**

**target\_size=IMAGE\_SIZE,**

**class\_mode='categorical'**

**)**

**plt.figure(figsize=(12, 12))**

**for i in range(0,15):**

**plt.subplot(5, 3, i+1)**

**for X\_batch, Y\_batch in example\_generator:**

**image = X\_batch[0]**

**plt.imshow(image)**

**break**

**plt.tight\_layout()**

**plt.show()**

These commands show how our generator works. It shows any random sample image and applies augmentation on it and plots it.

1. **epochs=3 if FAST\_RUN else 50**

**history = model.fit\_generator(**

**train\_generator,**

**epochs=epochs,**

**validation\_data=validation\_generator,**

**validation\_steps=total\_validate//batch\_size,**

**steps\_per\_epoch=total\_train//batch\_size,**

**callbacks=callbacks**

)

These commands are used to fit the model.

The **validation split** variable in **Keras** is a value between [0...100]. **Keras** proportionally **split** your training set by the value of the variable. The first set is used for training and the 2nd set for **validation** after each epoch.

**Epoch**: an arbitrary cutoff, generally defined as "one pass over the entire dataset", used to separate training into distinct phases, which is useful for logging and periodic evaluation. So, in other words, a number of **epochs** means how many times you go through your training set.

An epoch usually means one iteration over all of the **training** data. For instance if you have 20,000 images and a batch size of 100 then the epoch should contain 20,000 / 100 = 200 steps.

one epoch = one forward pass and one backward pass of all the training examples.**batch size** = the number of training examples in one forward/backward pass. The higher the **batch size**, the more memory space you'll need. number of iterations = number of passes, each pass using [**batch size**] number of examples

1. **model.save\_weights("model.h5")**

This is used to save the model.

1. **fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))**

**ax1.plot(history.history['loss'], color='b', label="Training loss")**

**ax1.plot(history.history['val\_loss'], color='r', label="validation loss")**

**ax1.set\_xticks(np.arange(1, epochs, 1))**

**ax1.set\_yticks(np.arange(0, 5, 0.1))**

**ax2.plot(history.history['accuracy'],**

**color='b', label="Training accuracy")**

**ax2.plot(history.history['val\_accuracy'],**

**color='r',label="Validation accuracy")**

**ax2.set\_xticks(np.arange(1, epochs, 1))**

**legend = plt.legend(loc='best', shadow=True)**

**plt.tight\_layout()**

**plt.show()**

These commands are used to visualize the training. It gives two plots:

* 1. Between training loss and validation loss.
  2. Between training accuracy and validation accuracy.

1. **test\_filenames = os.listdir(r"C:\Users\mudit\MUDITA**

**\minor project\dataset\test")**

**test\_df = pd.DataFrame({**

**' filename': test\_filenames**

**})**

**nb\_samples = test\_df.shape[0]**

these commands are used for preparing the testing data.

1. **test\_gen = ImageDataGenerator(rescale=1./255)**

**test\_generator = test\_gen.flow\_from\_dataframe(**

**test\_df,**

**r"C:\Users\mudit\MUDITA\minor project\dataset\test" ,**

**x\_col='filename',**

**y\_col=None,**

**class\_mode=None,**

**target\_size=IMAGE\_SIZE,**

**batch\_size=batch\_size**

**shuffle=False**

**)**

Here we create the test generator in the same way as we created train generator.

1. **predict = model.predict\_generator(test\_generator, steps=np.ceil(nb\_samples/batch\_size))**

this is used for predicting the test generator

1. **test\_df['category'] = np.argmax(predict, axis=-1)**

For categoral classication the prediction will come with probability of each category. So we will pick the category that have the highest probability with numpy average max

**34 .label\_map = dict((v,k) for k,v in train\_generator.class\_indices.items())**

We will convert the predict category back into our generator classes by using train\_generator.class\_indices. It is the classes that image generator map while converting data into computer vision

**35.test\_df['category'] = test\_df['category'].replace(label\_map)**

**df["category"] = df["category"].replace({0:'0',1: '1',2: '2',3:'3',4: '4',5 : '5',6: '6',7: '7',8: '8',9: '9',10: '10',11: '11',12: '12',13: '13',14: '14',15: '15',16: '16',17: '17',18:'18',19: '19',20: '20',21: '21',22: '22',23: '23',24: '24' ,25: '25',26: '26',27: '27',28: '28',29: '29',30: '30', 31: '31',32: '32',33:'33',34: '34',35 : '35',36: '36',37: '37',38: '38',39: '39',40: '40',41: '41',42: '42',43: '43',44: '44',45: '45',46: '46',47: '47',48: '48',49: '49',50: '50',51: '51',52: '52',53:'53',54: '54',55 : '55',56: '56',57: '57',58: '58',59: '59',60: '60',61: '61',62: '62',63:'63',64: '64',65 : '65',66: '66',67: '67',68: '68',69: '69',70: '70',71: '71',72: '72',73:'73',74: '74',75 : '75',76: '76',77: '77',78: '78',79: '79',80: '80',81: '81',82: '82',83:'83',84: '84',85 : '85',86: '86',87: '87',88: '88',89: '89',90: '90',91: '91',92: '92',93:'93',94: '94',95 : '95',96: '96',97: '97',98: '98',99: '99',100: '100'})**

From our prepared data part. We map data with {1: '1', 0:’0'…100;’101’}. Now we will map the result back to our 101 classes.

**36.test\_df['category'].value\_counts().plot.bar(figsize=(20,5))**

we can visualize the result through this plot.

**37.sample\_test = test\_df**

**#sample\_test**

**plt.figure(figsize=(12, 24))**

**text\_labels=[]**

**for index, row in sample\_test.iterrows():**

**filename = row['filename']**

**category = row['category']**

**img = load\_img(r'C:\Users\mudit\MUDITA\minor project\dataset\test/'+filename, target\_size=IMAGE\_SIZE)**

**plt.subplot(6, 5,index+1)**

**plt.imshow(img)**

**if category =='0':**

**text\_labels.append(0)**

**elif category =='1':**

**text\_labels.append(1)**

**elif category == '2':**

**text\_labels.append(2)**

**elif category == '3':**

**text\_labels.append(3)**

**elif category == '4':**

**text\_labels.append(4)**

**elif category == '5':**

**text\_labels.append(5)**

**elif category == '6':**

**text\_labels.append(6)**

**elif category == '7':**

**text\_labels.append(7)**

**elif category == '8':**

**text\_labels.append(8)**

**elif category == '9':**

**text\_labels.append(9)**

**elif category == '10':**

**text\_labels.append(10)**

**elif category == '11':**

**text\_labels.append(11)**

**elif category == '12':**

**text\_labels.append(12)**

**elif category == '13':**

**text\_labels.append(13)**

**elif category == '14':**

**text\_labels.append(14)**

**elif category =='15':**

**text\_labels.append(15)**

**elif category == '16':**

**text\_labels.append(16)**

**elif category == '17':**

**text\_labels.append(17)**

**elif category == '18':**

**text\_labels.append(18)**

**elif category == '19':**

**text\_labels.append(19)**

**elif category == '20':**

**text\_labels.append(20)**

**elif category == '21':**

**text\_labels.append(21)**

**elif category == '22':**

**text\_labels.append(22)**

**elif category == '23':**

**text\_labels.append(23)**

**elif category == '24':**

**text\_labels.append(24)**

**elif category == '25':**

**text\_labels.append(25)**

**elif category == '26':**

**text\_labels.append(26)**

**elif category == '27':**

**text\_labels.append(27)**

**elif category == '28':**

**text\_labels.append(28)**

**elif category =='29':**

**text\_labels.append(29)**

**elif category == '30':**

**text\_labels.append(30)**

**elif category == '31':**

**text\_labels.append(31)**

**elif category == '32':**

**text\_labels.append(32)**

**elif category == '33':**

**text\_labels.append(33)**

**elif category == '34':**

**text\_labels.append(34)**

**elif category == '35':**

**text\_labels.append(35)**

**elif category == '36':**

**text\_labels.append(36)**

**elif category == '37':**

**text\_labels.append(37)**

**elif category == '38':**

**text\_labels.append(38)**

**elif category == '39':**

**text\_labels.append(39)**

**elif category == '40':**

**text\_labels.append(40)**

**elif category == '41':**

**text\_labels.append(41)**

**elif category == '42':**

**text\_labels.append(42)**

**elif category =='43':**

**text\_labels.append(43)**

**elif category == '44':**

**text\_labels.append(44)**

**elif category == '45':**

**text\_labels.append(45)**

**elif category == '46':**

**text\_labels.append(46)**

**elif category == '47':**

**text\_labels.append(47)**

**elif category == '48':**

**text\_labels.append(48)**

**elif category == '49':**

**text\_labels.append(49)**

**elif category == '50':**

**categories.append(50)**

**elif category =='51':**

**text\_labels.append(51)**

**elif category == '52':**

**text\_labels.append(52)**

**elif category == '53':**

**text\_labels.append(53)**

**elif category == '54':**

**text\_labels.append(54)**

**elif category == '55':**

**text\_labels.append(55)**

**elif category == '56':**

**text\_labels.append(56)**

**elif category == '57':**

**text\_labels.append(57)**

**elif category == '58':**

**text\_labels.append(58)**

**elif category == '59':**

**text\_labels.append(59)**

**elif category == '60':**

**text\_labels.append(60)**

**elif category =='61':**

**text\_labels.append(61)**

**elif category == '62':**

**text\_labels.append(63)**

**elif category == '64':**

**text\_labels.append(64)**

**elif category == '65':**

**text\_labels.append(65)**

**elif category == '66':**

**text\_labels.append(66)**

**elif category == '67':**

**text\_labels.append(67)**

**elif category == '68':**

**text\_labels.append(68)**

**elif category == '69':**

**text\_labels.append(69)**

**elif category == '70':**

**text\_labels.append(70)**

**elif category =='71':**

**text\_labels.append(71)**

**elif category == '72':**

**text\_labels.append(72)**

**elif category == '73':**

**text\_labels.append(73)**

**elif category == '74':**

**text\_labels.append(74)**

**elif category == '75':**

**text\_labels.append(75)**

**elif category == '76':**

**text\_labels.append(76)**

**elif category == '77':**

**text\_labels.append(77)**

**elif category == '78':**

**text\_labels.append(78)**

**elif category == '79':**

**text\_labels.append(79)**

**elif category == '80':**

**text\_labels.append(80)**

**elif category =='81':**

**text\_labels.append(81)**

**elif category == '82':**

**text\_labels.append(82)**

**elif category == '83':**

**text\_labels.append(83)**

**elif category == '84':**

**text\_labels.append(84)**

**elif category == '85':**

**text\_labels.append(85)**

**elif category == '86':**

**text\_labels.append(86)**

**elif category == '87':**

**text\_labels.append(87)**

**elif category == '88':**

**text\_labels.append(88)**

**elif category == '89':**

**text\_labels.append(89)**

**elif category == '90':**

**text\_labels.append(90)**

**elif category =='91':**

**text\_labels.append(91)**

**elif category == '92':**

**text\_labels.append(92)**

**elif category == '93':**

**text\_labels.append(93)**

**elif category == '94':**

**text\_labels.append(94)**

**elif category == '95':**

**text\_labels.append(95)**

**elif category == '96':**

**text\_labels.append(96)**

**elif category == '97':**

**text\_labels.append(97)**

**elif category == '98':**

**text\_labels.append(98)**

**elif category == '99':**

**text\_labels.append(99)**

**else:**

**text\_labels.append(100)**

**plt.xlabel(filename + '   this is  ' + category)**

**plt.tight\_layout()**

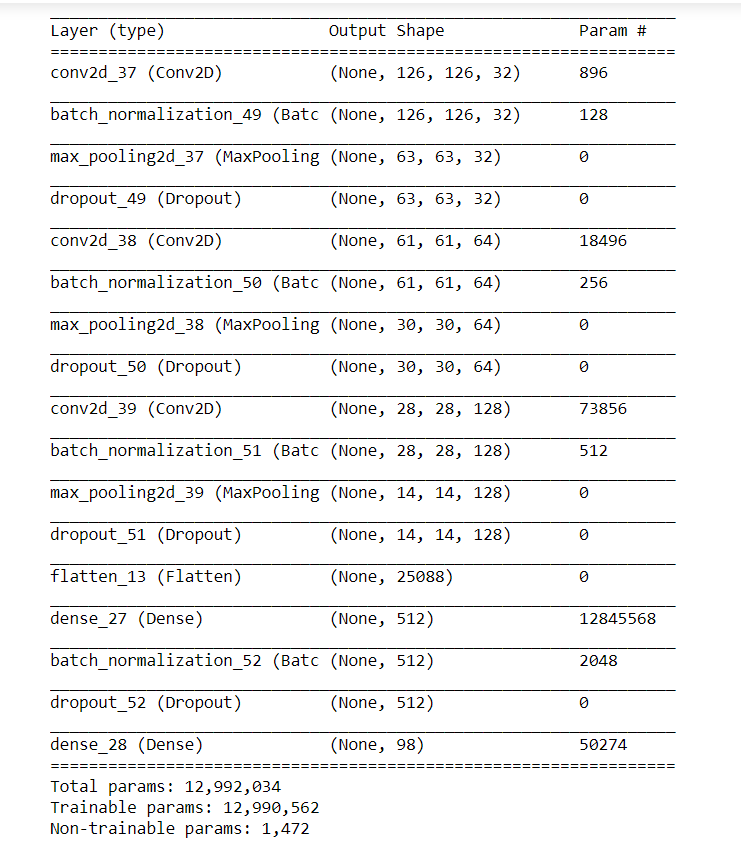
**plt.show()**

Through this code, we can see the predicted result with images and a string below it stating the filename and telling which no. of images is this.

**Chapter-5 Results**

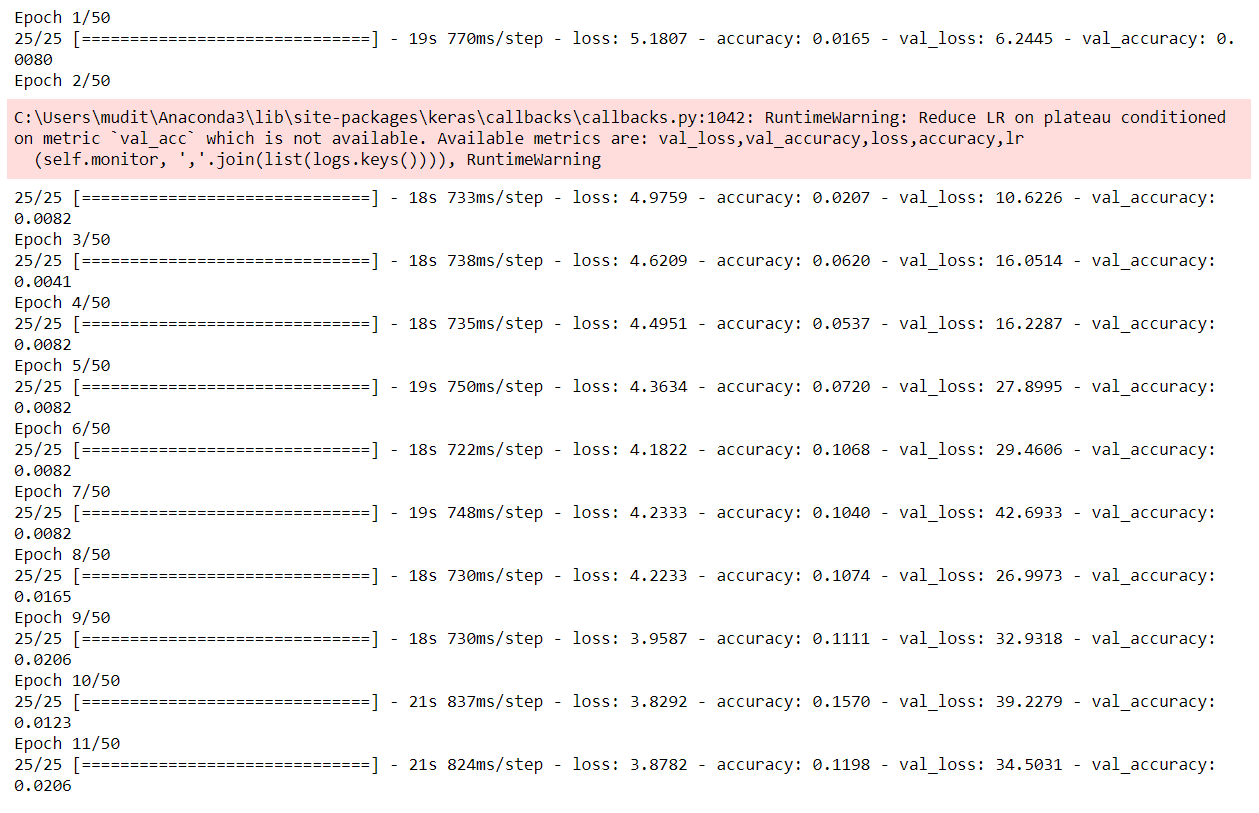
**5.1 Model**

Our model is defined as:



**5.2 Training Epochs**

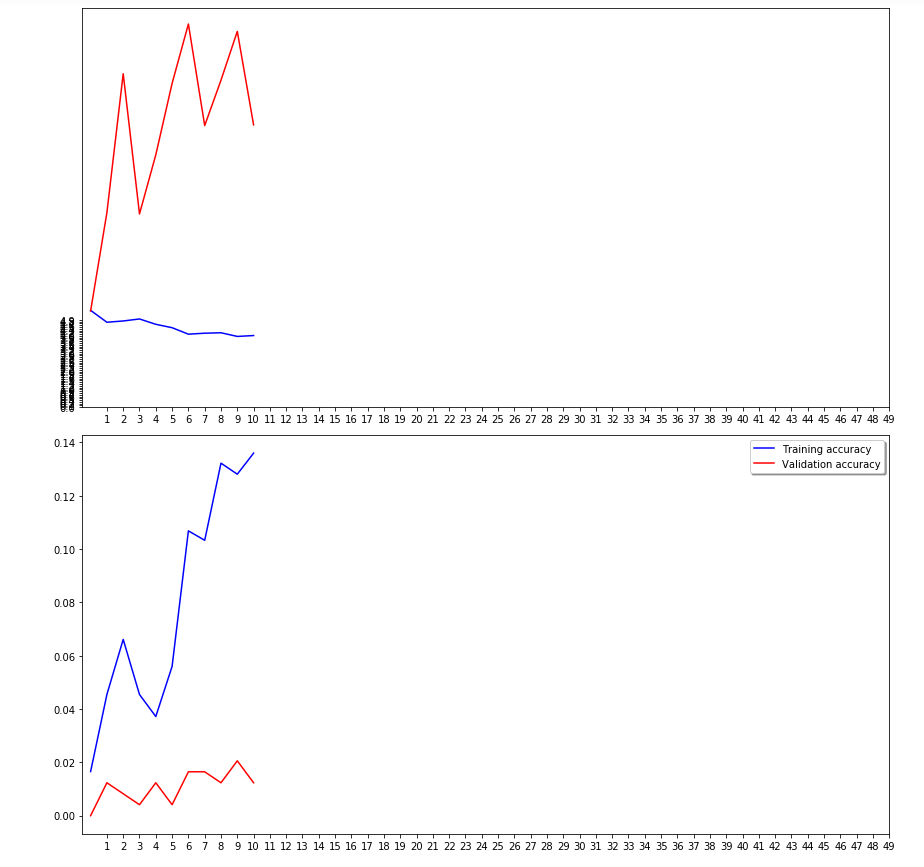
During training, epochs ran and produced the following result:



**5.3 Accuracy & Loss Graphs**  
The following figure shows two graphs

(i) Training loss vs. Validation loss

(ii)Training accuracy vs. Validation accuracy

****

**Chapter-6 Conclusion**

The objective of our project is Finger Knuckle Recognition. In this project we have prepared a program in which we give the input as an image it predicts whether the image belongs to which of the given 100 classes that we specified in our dataset. We have used Jupyter notebook and Google Colab for step by step execution and easy understanding of the code. Based on these hundred available labels we train our CNN model for the final prediction. Finally we provide a set of 5 image as input in test directory and prediction is made.

We have used the method of Convolutional Neural Network because it is one of the best methods used in image classification.

This type of project can be used for improving and increasing security, which is one of the major concerns today.

It can be used in ATMs to detect or to recognize the original user of the debit card. Hence preventing ATM theft. Also we propose to use in cars, such that the car will start only when its owner tries to start it otherwise it won’t get started, thereby minimizing the chance of theft.

We will try to improve our project by implementing these changes in future. Also we’ll try to increase our dataset so as to get a better accuracy.

We hope that the readers enjoyed reading the report!!

Thank You Everyone!!

**Chapter-7 Bibliography**

* 1. <https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8>
  2. <https://towardsdatascience.com/image-detection-from-scratch-in-keras-f314872006c9>
  3. <https://www.datascience.com/blog/convolutional-neural-network>
  4. <https://www.kaggle.com/uysimty/keras-cnn-dog-or-cat-classification>
  5. <https://towardsdatascience.com/fast-ai-season-1-episode-2-1-e9cc80d81a9d>
  6. <https://www.kaggle.com/>
  7. <https://keras.io/>
  8. <https://www.google.com/>