Deep Learning in Computer Vision with Small Datasets

Vishwatmika Srivastava

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Augmented and Virtual Reality)

Supervisor: Michael Manzke

September 2020

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

[redacted]

Vishwatmika Srivastava

September 4, 2020

Permission to Lend and/or Copy

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Professor Michael Manzke, for providing continuous guidance throughout the research and guiding my way along the process.
I cannot thank Trinity College Dublin enough for providing me with such great platform to perform this research. The taught modules have been the building blocks for this dissertation. In addition to that, I am grateful to my Computer Vision professor, Dr. Rozenn Dayhot and Machine Learning professors, Dr. Joeran Beel and Prof. Douglas Leith teaching the subjects remarkably.
Additionally, the support of my friends and family has always been like a cherry on the cake. The motivation and support that I got from everyone is marvelous.

VISHWATMIKA SRIVASTAVA

University of Dublin, Trinity College
September 2020

Deep Learning in Computer Vision with Small Datasets

Vishwatmika Srivastava, Master of Science in Computer Science

University of Dublin, Trinity College, 2020

Supervisor: Michael Manzke

The amelioration of Deep Learning algorithms was a milestone in the history of Computer Vision, because for the first time in the history, a computer program was able to see better than human being. The Convolutional Neural Networks are known for giving better accuracy than humans for task like Image Classification and Object Detection. However, the problem with these networks is their need for data. The more data is given to a CNN, the better it will perform. Although, there are real word problems where there's impossible to get handful of data, to exemplify, If a small company wants to install a face detecting system for their employees, then the dataset would be very small for the training of the model. In such cases, it is vital to use some other techniques along with CNN to achieve good accuracy. In this research, the adverse effect of dataset size on the accuracy of the CNN classifier is studied and the aim is to find out the techniques which work best with the dataset. The datasets used here are MNIST, FMNIST and DogsVsCats (from Kaggle) are used. To verify the effects of fewer training example on the classifier, the MNIST and FMNIST dataset are used where their 10%, 50% and 100% subsets were taken. The accuracy increased with the rise in size for FMNIST, but not for MNIST, because of its trivial nature. Once this is done, two helper techniques were used to improve the classification accuracy that are: Data Augmentation and Transfer Learning. Both techniques are implemented on FMNIST and DogsVsCats. For Data Augmentation, Geometric transformation and Elastic distortion is used and an overall improvement of 2.92% on FMNIST and 16.67% on DogsVsCats was recorded. Furthermore, for Transfer Learning, MobileNetV2 and VGG19 models were used along with the CNN classifier. The accuracy for FMNIST dropped by 6.12% while DogsVsCats illustrated a rise of 24.04%. The ups and downs in the accuracy on different datasets demonstrated that the nature of dataset also plays important role in conjunction with its size. If the dataset is trivial and uncomplicated like MNIST or FMNIST, then simpler and straightforward algorithms tend to work better while the fancy algorithms result into overfitting. Complicated datasets call for advanced techniques, that's why DogsVsCats gives best results with Transfer Learning.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

The ability of humans to see objects and perceive information from it has been an excitable topic of discussion for researchers over the globe.
The computer vision researchers have been trying to develop models based on mathematics and statistics to extract 2D or 3D information from images for a long time now. The difficulty of vision comes from the fact that it is an inverse problem, where unknown solutions are discovered from insufficient information. [15].
Deep learning on the other hand is the branch of machine learning that encourages artificial intelligence by imitating the working of human brain. It lets the machine process the given data and develop patterns for decision making with time. Deep learning methods are mostly known for their demand of large datasets. In fact, the availability of large amount of data and computational power has been the leading reason for the empowerment of deep learning.
The importance of data in deep learning has never been denied because just like human brains, machines also learn by dealing with as much information as they can get. The experience to process a particular type of information enhances the decision making capacity of the algorithm. The hitch here is, that data comes at a cost. The expense to collect and manage large scale data is enormous, which increases the need of algorithms that require smaller datasets.
Furthermore, there are real world scenarios where it is nearly impossible to arrange

a huge dataset. To exemplify, a startup wants to incorporate a facial detection login system, but due to limited number of pictures of their employees, the algorithm might not work well.
Most of the work is done in the pre-processing steps for machine learning and classification algorithms, where the data is refined to make it ideal for the algorithm. In this research, the effects of small datasets on the performance of a CNN classifier is studied on two multi-class datasets MNIST [16] and FMNIST [17] and one binary dataset DogsVsCats [18]. Additionally, data augmentation has been considered to increase the amount of data available and study the outcome. The initial technique of augmentation is normal geometric transformation of images i.e. translation, rotation, scaling and shearing. Afterwards, elastic distortion is also used for the same. Both of these techniques are combined to obtain the results. Furthermore, Transfer Learning will also be studied during the course of this research to see how the technique works on different dataset.

1.2 Motivation

The effects of lack of data can be detrimental on the performance of the classifier, it can lead to various consequences like model overfitting. But again, The issues with big data are not negligible either. The hindrance regarding cost, quality, storage etc are always there.
The aim of this research is to analyze the methods stated in previous section and figure out which method works better in what condition and with which dataset. Also, how the size affects the performance of the model will also be evaluated.

1.3 Structure of the document

The upcoming document is organized in the following manner:

- **Chapter 2: Background** showcases the state-of-the-art algorithms that help in achieving the objective of the research. It also explores the underlying concepts used throughout the dissertation along with any necessary background details.

- **Chapter 3: Design** explains what techniques are used and describes the algorithms in detail.
- **Chapter 4: Implementation** illuminates the process followed throughout the research and illustrates the experiments done along with the code, libraries and environments used.
- **Chapter 5: Results and Discussion** showcases the final and intermediate results obtained throughout the research.
- **Chapter 6: Conclusions** states the final notes for the dissertation and covers the limitations and future work.

Chapter 2

Background

This chapter enlightens the basics of deep learning and computer vision that are used in this research. It also describes the relevant state of the art algorithm.
The first section covers the contextual basics for this research, followed by a description for image classification problem and the small dataset problem. Afterwards, the techniques which can be used to work with small datasets are discussed.

2.1 Contextual

2.1.1 Deep Learning

The branch of machine learning that is used to discover different levels of representations are called Deep Learning Algorithms. Hierarchical architecture is used in deep learning to learn the abstraction on high level [19]. The application of deep learning gave human-competitive results on computer vision benchmark and improved the accuracy by 30-80% [20].
There are several methods used for deep learning, but the main four categories consists of:

- Convolutional Neural Networks (CNNs).
- Restricted Botzman's Machine (RBMs).
- Autoencoders.

- Sparse Coding.

CNNs will remain the main focus of this research.

2.1.2 Computer Vision

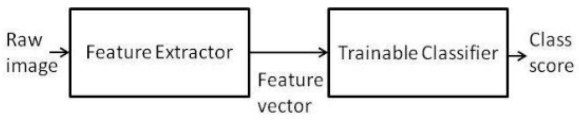
It is the field of computer science that helps the machines see. It goes hand in hand with artificial intelligence and machine learning to make systems perform better. There are several problems that CV deals with. A few of them are listed below:

- Image classification.
- Object detection.
- Image retrieval.
- Semantic segmentation.
- Human pose estimation.

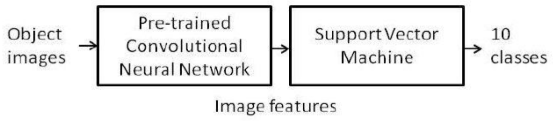
In this research, image classification will be the principle area of concentration. Image classification is the task of labelling the input images to a particular output depending on the presence of specific object class.

2.2 Image Classification

Classification is the process of producing class labels for a set of features. The classification task can be divided into two categories: Supervised and Unsupervised. The



(a) Workflow



(b) Combined structure

Figure 2.1: Image classification [2]

supervised method consists of training data that has labels assigned to it, while the unsupervised dataset is without labels and clusters are made to do the classification. The recipe of classification is straightforward: one finds a labelled dataset, builds feature and then train a classifies [21], as shown in fig(2.1a).

In this research, Convolutional Neural Networks (CNNs) will be used for this task.

2.2.1 Convoltional Neural Networks

The increased efficiency of CNNs has been a huge reason behind their success in the field of Computer Vision. In CNNs, multiple layers can be trained with gradient based learning in complex, high dimensional and non-linear mappings for large collection of data that makes them a perfect candidate for computer vision problems [22].

The basic architecture of a CNN is shown in fig(2.2). Different from Multi-layer perceptron neural networks (MLPNN) whose layers are all fully-connected layers, a CNN consists of three main layers:

- Convolutional layers.
- Pooling layers.
- Fully-connected layers.

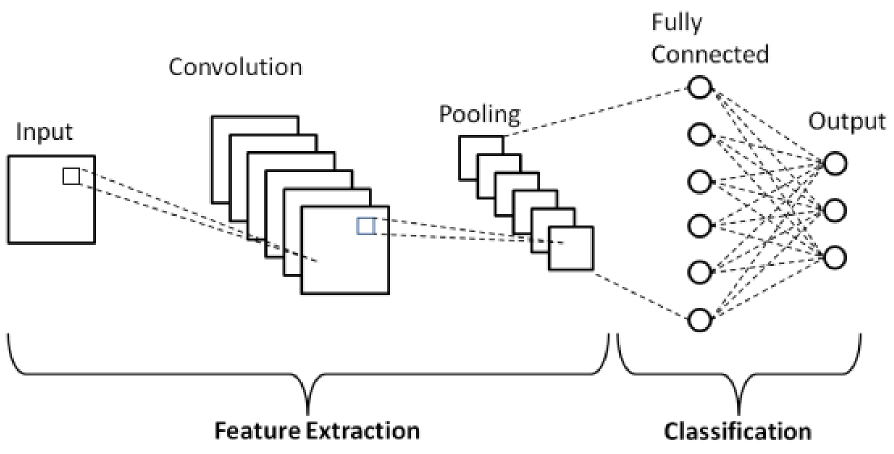


Figure 2.2: Architecture of Convolutional Neural Networks

Convolutional layers

In these layers, various feature maps are generated for the input images. The CNN creates a few kernels to convolve the image as shown in fig(2.3).

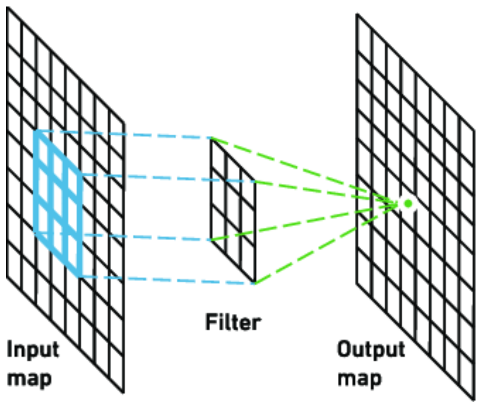


Figure 2.3: Operation of convolutional layer

These layers apply a kernel/filter to the image to create a feature map, which is again passed to either a convolution or a pooling layer to go deeper into the hierarchy in order to study the pixels.

Pooling layers

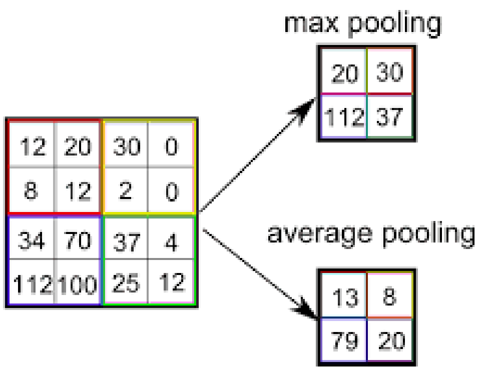


Figure 2.4: Max and Average pooling operations

These layers are used to reduce the size of the feature map. There are different types of pooling techniques that can be used, but the most famous ones are:

- Max Pooling: The maximum value amongst the window of the pixels are assigned to the resultant pixel.
- Average Pooling: The average of all the pixel values in the window is given to final pixel.

Both of these pooling techniques are illustrated in fig(2.4).

Fully connected layers

Followed by the pooling layers are fully connected layers. The task of these layers is to convert the 2D feature map into one dimension so that it can fit the output format as depicted in fig(2.5).

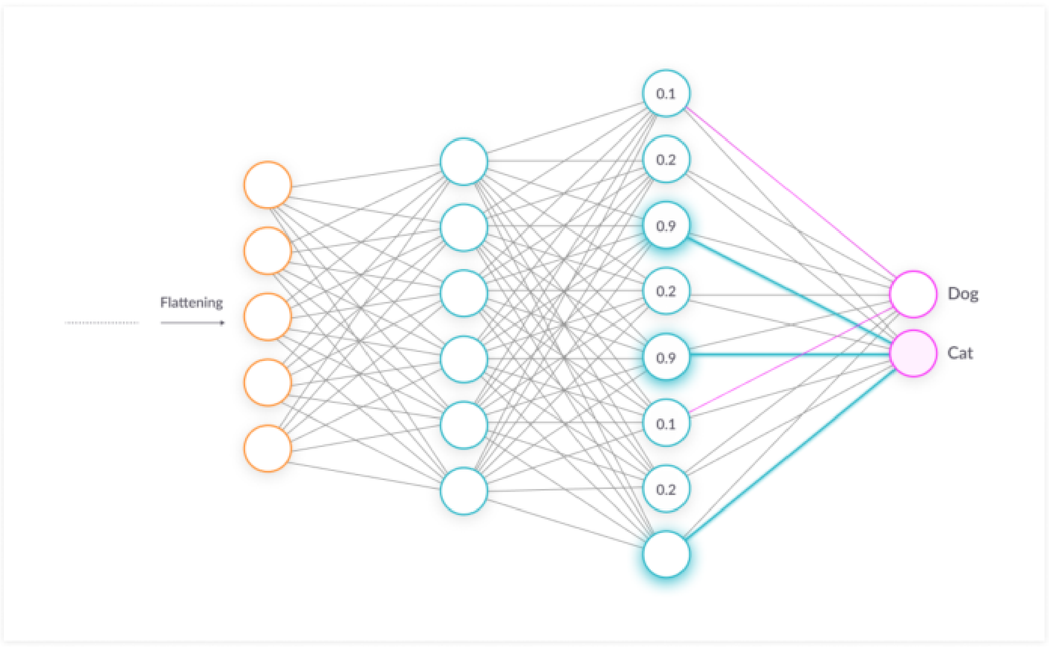


Figure 2.5: Fully connected layers after pooling

2.3 The small dataset problem

It has been well established that for a problem, with large enough data, different algorithms perform practically the same. As illustrated in fig(2.6), it is clear that as

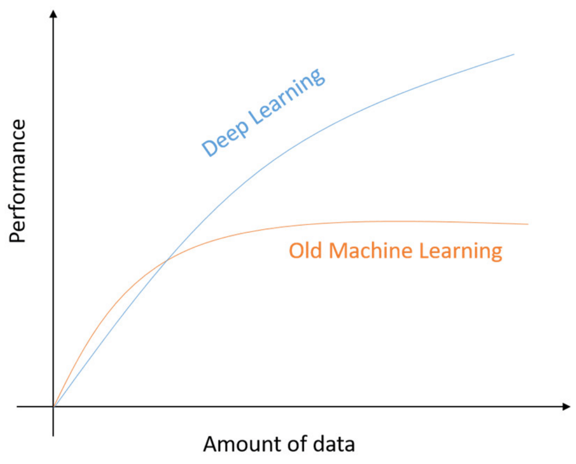


Figure 2.6: Relationship between performance and amount of data

the amount of data increases, the performance of the deep learning algorithms also increases in comparison to the traditional approaches.

2.3.1 Effects of small datasets on model performance

The two terms that are needed to be understood before discussing the performance are:

- Bias: It represents the measure of how underfitting the model is. It is the difference between observed value and predicted value [4].
- Variance: A high variance shows that model overfits the data. It is the difference in performance of the training set and the test set[4].

Ideally, the model must minimize both bias and variance in order to achieve decent performance i.e. the model not only fits the training data well but also generalizes well on the the test data.
A small datasets tends to usually tend to make the model overfit. Overfitting occurs when models adjust excessively to the training data and fails to generalize on new data. The smaller the dataset, the better model can fit the training data and more the chances of overfitting.

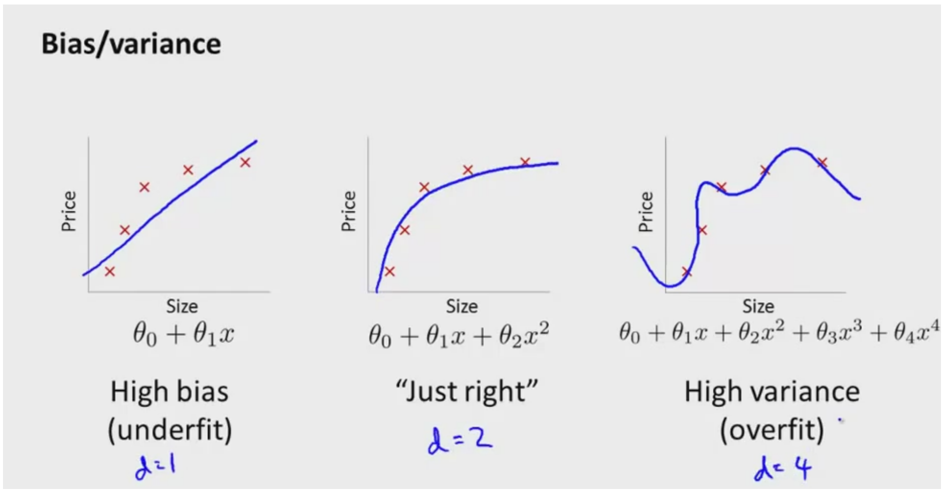


Figure 2.7: Bias vs Variance [3]

2.3.2 Issues with large datasets

Irrespective of all the good things large datasets bring with them, there are some problems that cannot be ignored. Also, these setbacks encourage the use of smaller datasets, provided the accuracy. Few of the drawbacks of huge datasets are listed below:

- Storage: Dataset sizes vary from 10 megabyte to 10 terabyte depending on the amount of data stored. Also, the amount of RAM consumed to process such amount of data is plenty.

- Cost: The real cost of data lies in its operational efforts and overall management or its integration withing the system. Big enough data can cost as much as 1 million dollars [23].

- Data quality: A very large amount of data could sometimes lead to study design or data quality issues such as errors of measurement, missing data. This brings challenges like data analysis and pre-processing [24].

- Metrology: Large datasets usually comprise of data which is complex, which steers the issue of measurement [24]. The data might be in different formats, quantity and quality. One prime example are the public surveys, where different users might enter discrete information for same fields. Standardization techniques are necessary in such cases.

- Representativeness: Big Data collected through a variety of formats catch units of phenomena that differ from the units or phenomena that are not collected [25].

2.3.3 Measurement of data quantity

The talk about big data, large or small datasets has been done since the dawn of deep learning, but how much exactly is a large dataset is not clear. The amount of data mostly depend on the problem at hand and the architecture being used to solve the problem.

As a rough rule of thumb, the model should train on at least an order of magnitude more examples than trainable parameters. Simple models on large data sets generally beat fancy models on small data sets.

An implementation of [14] on FMNIST [17] and it's subsets shows the effects of amount of data used as illustrated in fig(2.8).

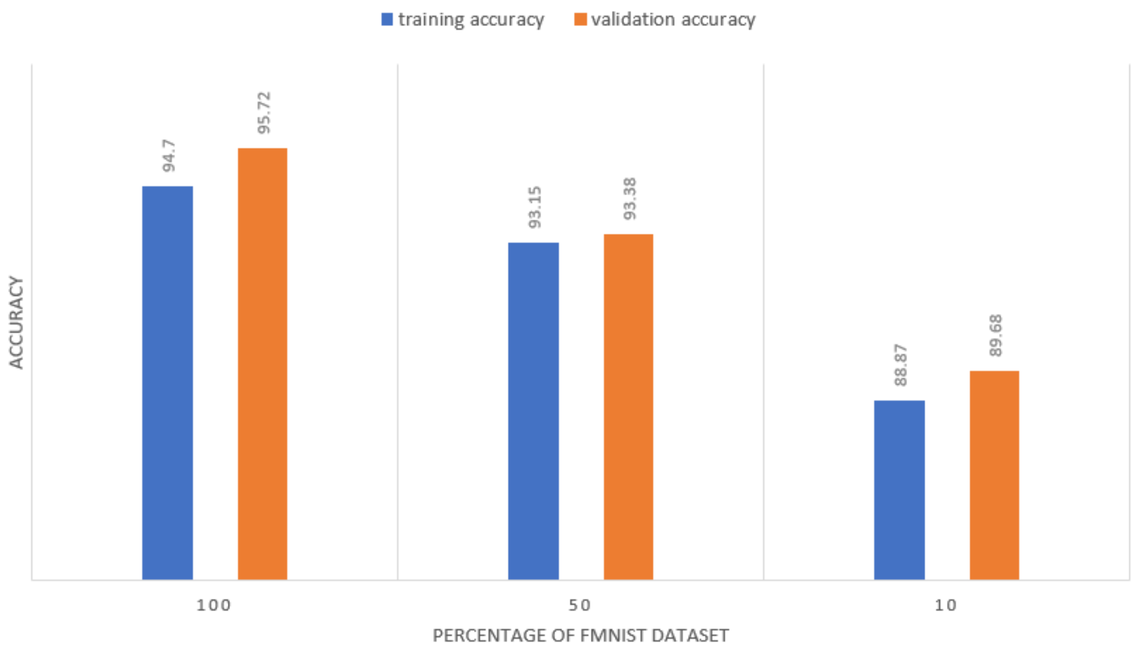


Figure 2.8: Effects of data quantity on CNN accuracy

In the figure(2.8), the decline in both training and validation accuracy is seen from using 100%, 50% and 10% of the dataset.

2.3.4 Consequences of small datasets

- Lack of Generalization: When a model is trained, it is not supposed to work well on the training set only, but also on the test and validation set. A network might overfit if the training set contains accidental regularities [26], which is more probable in small datasets because of the lack of diversity.
- Data Imbalance: This challenge arises when there are comparatively more or less examples from a particular class. For example, if there are two classes in the dataset and the distribution ratio is 80:20, then the record is skewed.
- Difficulty in optimization: It is the process of minimizing the loss of the function by finding the exact parameters, but due to less data the algorithm might not converge [27].

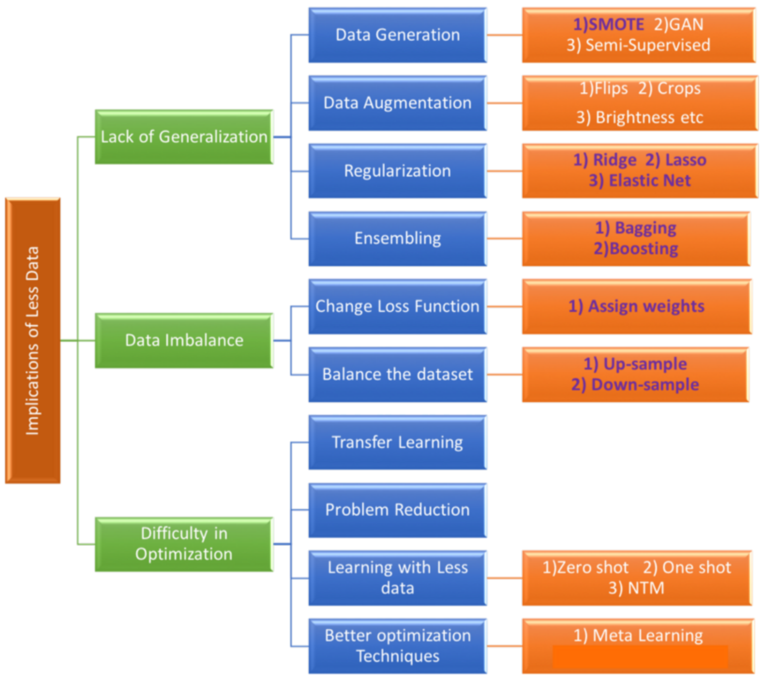


Figure 2.9: Different ways to deal with small datasets. Source:[4]

2.3.5 Methods of working with small datasets

There are two levels to deal with lesser amount of data in deep learning. The first one is where the data is modified before sending it to the classification algorithm and secondly the algorithm itself is created in such a way that it can produce better results with comparatively less amount of data.

Data level methods

In these methods, new data entries are introduced in the dataset. There are various ways to do this, like modifying the given data to create its alteration (data augmentation) or generating new data (data generation). In this research, data augmentation will be studied.

Algorithm level methods

These are more complex and require extensive domain specific knowledge. Such an approach might be expensive as per time and processing complexity. Transfer learning will be analyzed during the course of this research.

2.4 Data Augmentation using Image Transformations

Data augmentation is the process of generating samples by transforming training data, with the target of improving the accuracy and robustness of classifiers [28]. The training data is modified a little such that it is not completely distorted and the label information is intact.
Traditional transformations consist of using a combination of affine transformations to manipulate the training data [29]. All of the transformations are affine transformations of the original image that take the form [30]:

$$\vec{y} = W\vec{x} + \vec{b} \tag{2.1}$$

where y is the resultant vector, x is the original image, W is the affine matrix and b is the translation vector.

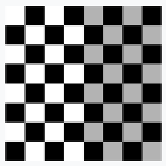
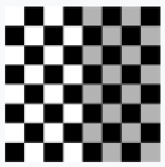
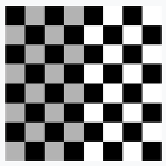
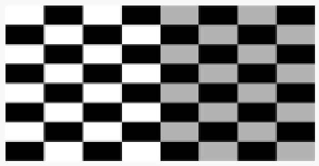
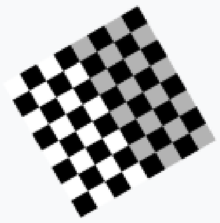
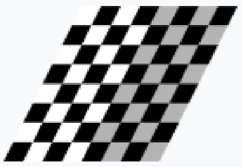
Transformation name	Affine matrix	Example
Identity (transform to original image)	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	
Translation	$\begin{bmatrix} 1 & 0 & v_x > 0 \\ 0 & 1 & v_y = 0 \\ 0 & 0 & 1 \end{bmatrix}$	
Reflection	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	
Scale	$\begin{bmatrix} c_x = 2 & 0 & 0 \\ 0 & c_y = 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	
Rotate	$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	 where $\theta = \frac{\pi}{6} = 30°$
Shear	$\begin{bmatrix} 1 & c_x = 0.5 & 0 \\ c_y = 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	

Figure 2.10: Affine Transformations for Images. Source:[5]

Few of the techniques used for image transformation are translation, rotation, scaling, shearing etc. Using equation(2.1), all of these transformations can be done, as illustrated in fig(2.10).

2.5 Data Augmentation using Elastic Distortion

In addition to affine transformations, elastic deformations improve the results for image classification tasks [31]. In the case of handwriting recognition, we postulate that the distribution has some invariance with respect to not only affine transformations, but also elastic deformations corresponding to uncontrolled oscillations of the hand muscles, dampened by inertia [31].

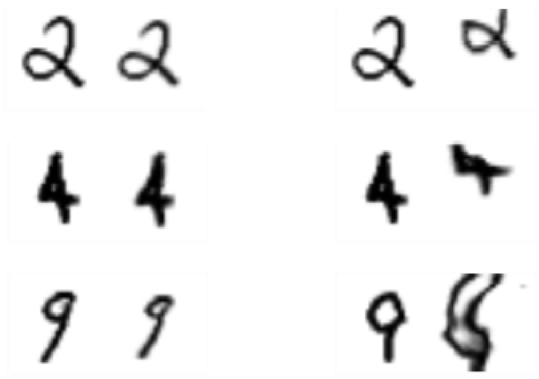


Figure 2.11: Data warping using elastic deformations. Source:[6]

In fig(2.11), Original MNIST digits are compared to warped digits. Left has $\alpha = 1.2$ and Right is $\alpha = 8$ [6].
The image deformations are created by generating displacement fields, that is Δx(x,y) = rand(-1,+1) and Δy(x,y) = rand(-1,+1), where rand(-1,+1) is a random number between -1 and +1, generated with a uniform distribution. The fields Δ x and Δ y are then convolved with a Gaussian of standard deviation σ (in pixels). For intermediate σ values, the displacement fields look like elastic deformation, where σ is the elasticity coefficient. The displacement fields are then multiplied by a scaling factor α that controls the intensity of the deformation [31].

2.6 Transfer Learning

Transfer learning refers to the approach of using the learning from one task on to another task without the requirement of learning from scratch [32] This technique has been widely used in computer vision tasks and has been instrumental in the wide application of deep learning in the industry.
Transfer learning is machine learning with an additional source of information apart from the original training set: knowledge from one or more related tasks [7], as exhibited in fig.(2.12).

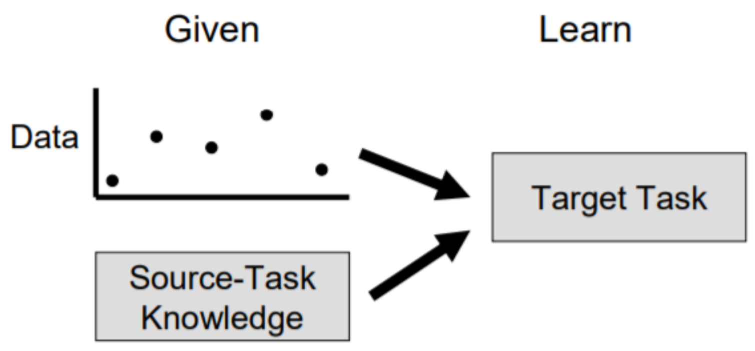


Figure 2.12: Transfer Learning. Source:[7]

It is now well established that the initial layers of models such as ResNet [33] trained on Imagenet [34] data learn to identify edges and corners in the image and later layers build on top of these features to learn more complicated structures. The final layer learns to classify the image into 1 of the 1000 categories.
These pre-trained networks can be further used as initial feature extractors or the model can be fine tuned as per the current classification task.
The main advantage of CNN for image classification is that the entire system is trained end-to-end, from raw pixels to ultimate categories, which alleviates the requirement for manually designing a suitable feature extractor [35]. As described in section(2.2.1), CNNs has different layers that have their own functionalities, but the overall structure of CNNs can be divided into two parts as shown in fig(2.13).

According to fig(2.13), the convolutional base, which has a number of convolution and pooling layers is used to extract features from the images.

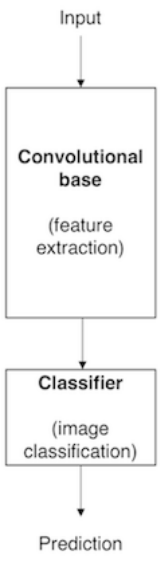


Figure 2.13: Simplified Architecture of CNN. Source:[8]

Whereas, classifier section mostly consists of fully connected layers which classify the images into their target classes according to the extracted features.

2.6.1 Deep CNNs and Small Datasets

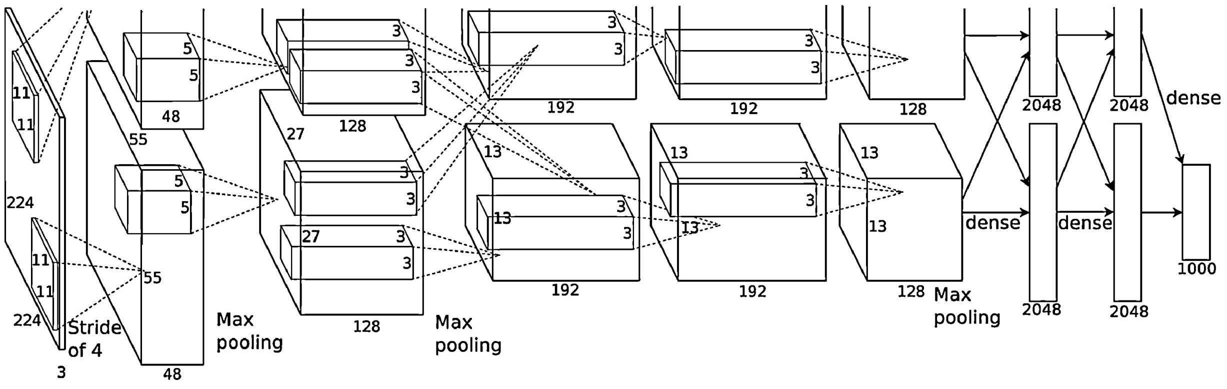


Figure 2.14: AlexNet from [9]

The main disadvantages of CNNs are:

1. They need a large labelled set to train on, in order to learn the weight parameters.

2. The GPU must be powerful to accelerate the process.

Sometimes the lack of these aforementioned necessities makes it tough to achieve the accuracy that CNNs are famous for. A popular CNN architecture [9] consists of five convolutional layers, three fully connected layers and one softmax classifier (fig.2.14) has approximately more than 60 million parameters. Deeper networks with better performances have larger number of parameters to learn.
Training such big networks on only thousands of training samples results in overfitting and consequently, lack of generalization on new data.
Transfer learning proposes a solution to this issue. The idea is to drop the classifier section of the pre-trained model and fine tune it on the target dataset.

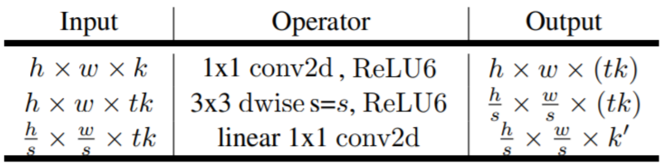
2.6.2 Pre-Trained models

In practice, very few people have the luxury of accessing very high-speed GPUs and powerful hardware to train a very deep network from scratch in a reasonable time. Therefore, pretraining a deep network (e.g., CNN) on large-scale datasets (e.g., ImageNet) is very common [36].
A pretrained model is already trained on a large amount of data and it can be used as a feature extractor or fine tuned to do a similar task as it was trained on. There are plenty of models available for this purpose. Some of the commonly used ones are:

- MobileNetV2 [11]
- VGG [37]
- ResNet [33]

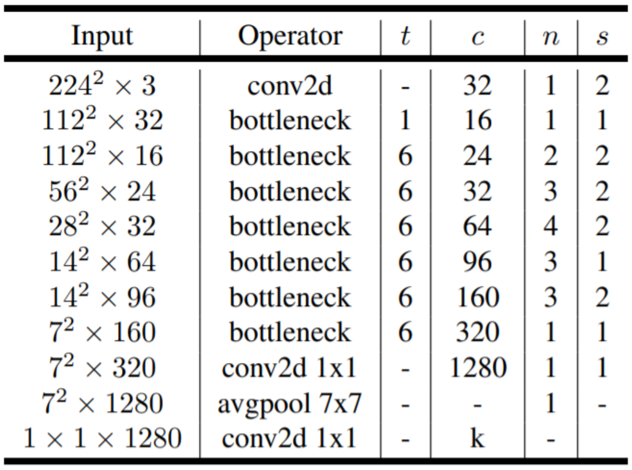
The architecture of MobileNetV2 [11] contains the initial fully convolution layer with 32 filters as shown in fig(2.15), followed by 19 residual bottleneck layers described in fig(2.16).

The models can be deeper than MobileNet. For example, Resnet [33] has up to 152 layers. These models take images as input and classifies them into approximately 1000 object categories.



Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Figure 2.15: MobileNetV2 Convolution layers



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 2.16: MobileNetV2 Bottleneck layers

2.6.3 Pre-trained models and Transfer Learning

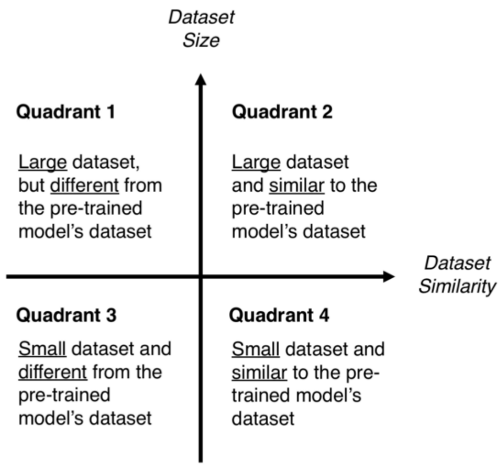
A pre-trained model can be re-purposed as per the requirements by dropping it's original classifier and adding a new one according to the current needs.

According to fig(2.17), it is clear that there are three approaches to deal with the situation.

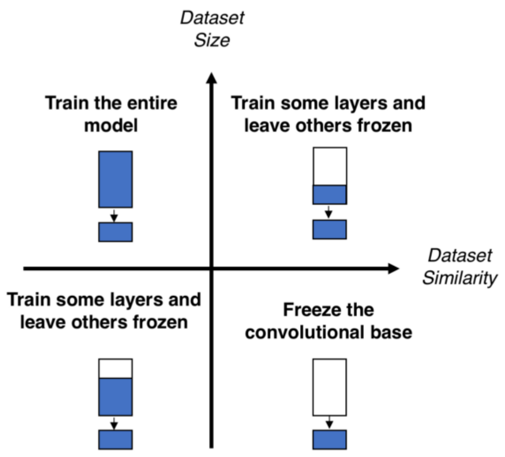
If the dataset is large and completely different from the original dataset the model was trained on, then the entire model can be trained again on this new dataset.

Additionally, some layers can be trained while leaving the others intact, if the dataset is large and similar to the original data. Same approach can be taken for a small dataset which is different from the initial dataset. This allows for the model to adjust with the new information while taking advantage from the previous knowledge.

Lastly, if the dataset is small but similar to the primary data, then only changing the classifier can do the trick.


Dataset Size
Quadrant 1
Large dataset, but different from the pre-trained model's dataset
Quadrant 2
Large dataset and similar to the pre-trained model's dataset
Dataset Similarity
Quadrant 3
Small dataset and different from the pre-trained model's dataset
Quadrant 4
Small dataset and similar to the pre-trained model's dataset

(a) Size-Similarity Matrix. Source:[8]


Dataset Size
Train the entire model
Train some layers and leave others frozen
Dataset Similarity
Train some layers and leave others frozen
Freeze the convolutional base

(b) Decision Map for the fine tuning pre-trained models. Source:[8]

Figure 2.17: Tuning a pre-trained model

2.6.4 What, When and How to "Transfer"

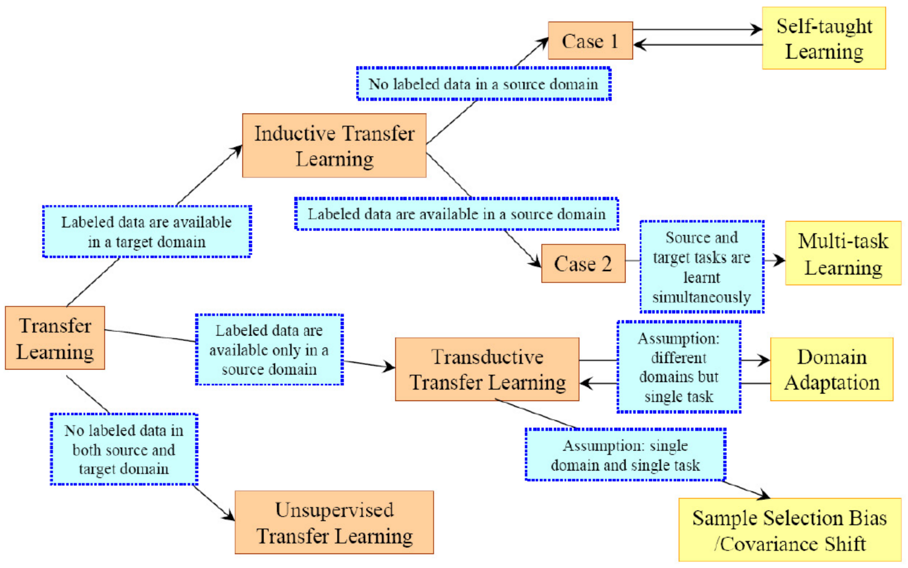

Case 1
Self-taught Learning
No labeled data in a source domain
Inductive Transfer Learning
Labeled data are available in a target domain
Labeled data are available in a source domain
Case 2
Source and target tasks are learnt simultaneously
Multi-task Learning
Transfer Learning
Labeled data are available only in a source domain
Transductive Transfer Learning
Assumption: different domains but single task
Domain Adaptation
No labeled data in both source and target domain
Assumption: single domain and single task
Unsupervised Transfer Learning
Sample Selection Bias /Covariance Shift

Figure 2.18: An Overview of Different Settings of Transfer. Source.[10]

- What: Some knowledge is specific to a specific domain (see Appendix:A.1) while other is general and can be used generally in all domains. Hence, the knowledge which is common between source and target domain can be transferred [10].
- When: The cases where source and target domains are not similar or related to each other, it is important to consider this question. Otherwise the consequence might lead to **negative transfer**.
- How: The techniques are divided into three parts according to the nature of datasets as illustrated in fig(2.18).

Chapter 3

Design

This chapter propose the high level designs for the architectures applied in this research including the geometric and elastic transformation architecture(for data augmentation), classification architecture and the transfer learning architecture along with the qualitative and quantitative metrices used to measure the accuracy of the system . It also covers the deep learning frameworks used and the whole workflow of the research process which will provide clarity to Chapter 4.

3.1 Geometric Transformations

Some of the most common geometric and photo-metric transformations applied on images to generate additional data while preserving the context of the image are rotation, translation, shearing, scaling, zooming etc. **tf.keras.preprocessing.image** tool from Tensorflow [38] is used in order to apply these transformations to the dataset for data augmentation.

The **tf.keras.preprocessing.image** [39] tool has multiple functions that allows to:

- Generate batches of Numpy [40] image tensor data with real-time augmentation, which is looped over the images.

- Perform random rotation, spatial shear, shift or zoom on the Numpy image tensor using functions like [41].

- Append the newly created images with labels to the dataset to increase data size.

Additionally, Elastic Distortion for the images are done using the functions of Numpy Library.
The general workflow of the data augmentation process is given in the fig(3.1). Using this workflow ensures that the dataset size in not small and can be increased as much as needed.

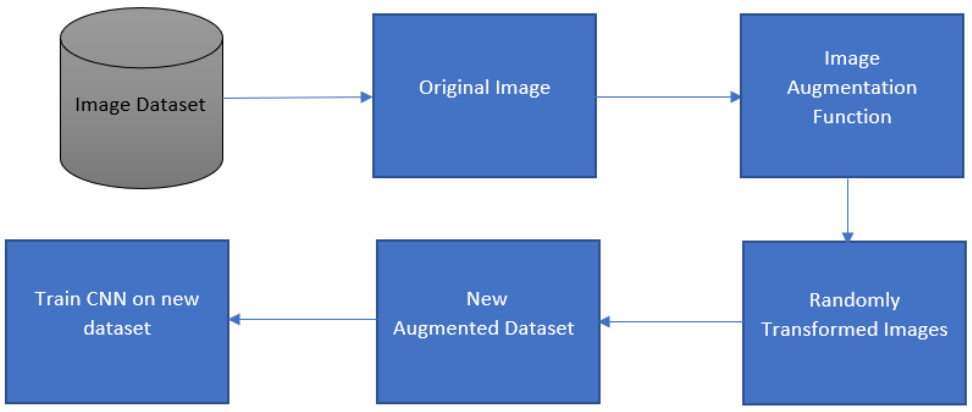


Figure 3.1: Data Augmentation Workflow

3.2 Transfer Learning Architecture

As explained in Chapter 2, Transfer Learning is done using pre-trained models. The models used in this research are MobileNetV2 [11] and VGG19 [37]. Both architectures are shown below:

- MobileNetV2:
  - In MobileNetV2, there are two types of blocks. One is residual block with stride of 1. Another one is block with stride of 2 for downsizing as shown in figure(3.2)
  - There are 3 layers for both types of blocks.

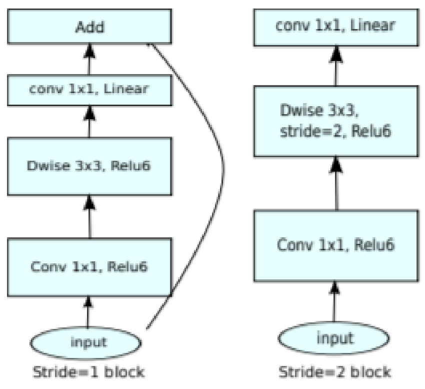
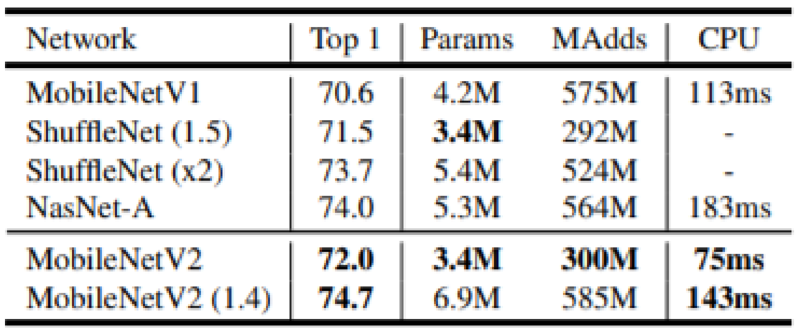


Figure 3.2: Convolutional Blocks for MobileNetV2. Source:[11]

- The first layer is 1x1 convolution with ReLU6.
- The second layer is the depthwise convolution.
- The third layer is another 1x1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain as shown in fig(2.15)
- And there is an expansion factor t. And t=6 for all main experiments.
- If the input got 64 channels, the internal output would get 64xt=64x6=384 channels.



Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Figure 3.3: Performance on ImageNet, comparison for different networks. Source:[11]

.

The overall architecture is shown in fig(2.16) and the detailed model summary is given in Appendix(A.2)
The performance of MobileNetV2 on ImageNet [34], in comparison to different networks is shown in figure(3.3)

- VGG19:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_5 (InputLayer)         [(None, 150, 150, 3)]     0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080    
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 37, 37, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0         
_________________________________________________________________
```

```
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
_________________________________________________________________
```

The base models for these architectures are frozen, so that the previously calculated parameters can be used to extract features from the data, and after that a classifier is added to sort that images into their respective classes.
The basic workflow of the Transfer Learning is shown in fig(3.4), where all the steps can be seen chronologically.

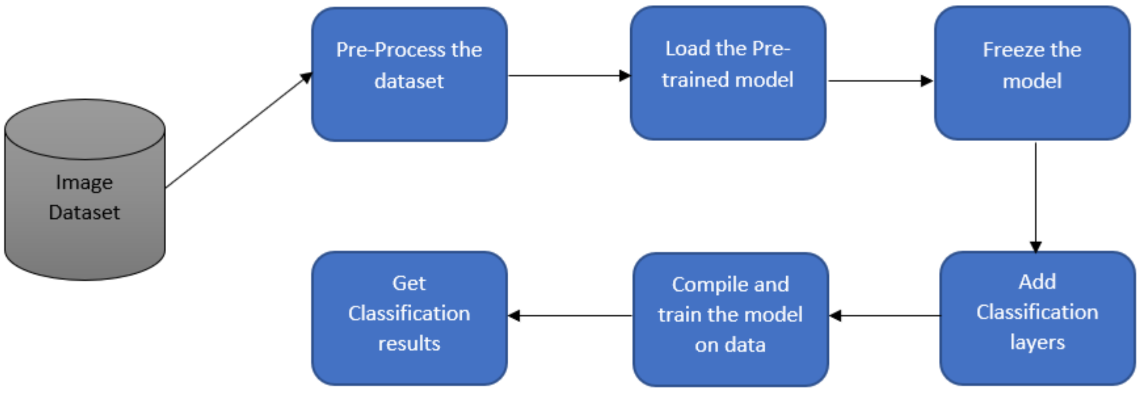


Image Dataset
Pre-Process the dataset
Load the Pre-trained model
Freeze the model
Add Classification layers
Compile and train the model on data
Get Classification results

Figure 3.4: Transfer Learning Workflow

3.3 Architecture of Classification model

Generally, CNN is used for image classification as the neurons in the hidden layers have learnable parameters that extract the important features from input.

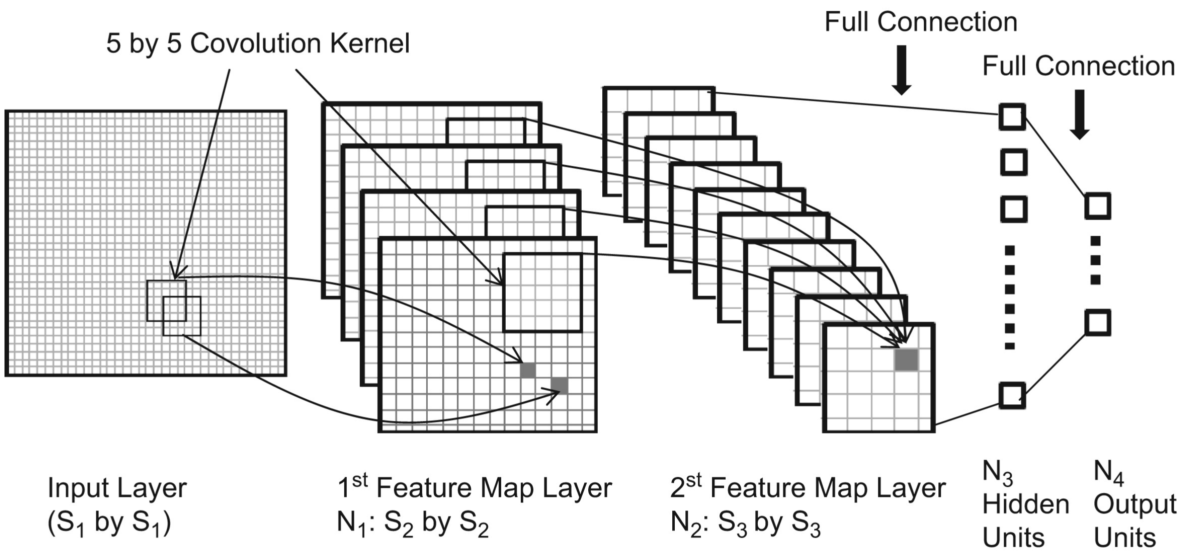

5 by 5 Covolution Kernel
Full Connection
Full Connection
Input Layer (S_1 by S_1)
1st Feature Map Layer N_1: S_2 by S_2
2st Feature Map Layer N_2: S_3 by S_3
N_3 Hidden Units
N_4 Output Units

Figure 3.5: CNN model used in classification architecture. Source:[12]

The architecture shown in fig(3.5), is what the CNN model looks like, with first convolutional layer having filter of 5x5x1(1 because the image is in grayscale). This in turn will produce a 2-dimensional activation map that consists of responses of the filter at given region.

Consequently, the pooling layer reduces the size of input images as per the results of a

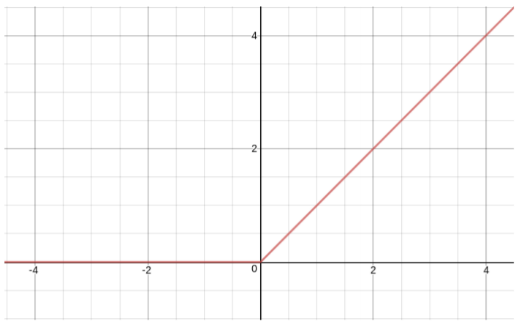


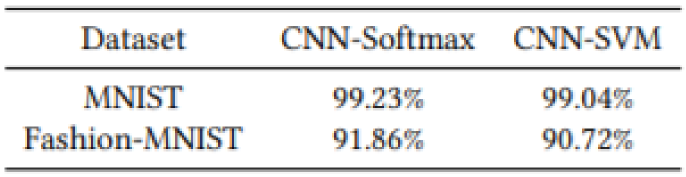
Figure 3.6: The Rectified Linear Unit (ReLU) activation function produces 0 as an output when x $<$ 0, and then produces a linear output with slope of 1 when x $>$ 0. Source:[13]

convolution filter. Lastly, an activation function is used for introducing non-linearities in the computation. Without such, the model will only learn linear mappings. The commonly-used activation function these days is the ReLU function (figure.3.6) [42]. ReLU is commonly-used over tanh and sigmoid for it was found out that it greatly accelerates the convergence of stochastic gradient descent compared the other two functions [9].

The technical architecture of CNN given in [13] as described above has following steps:

```
(1) INPUT: 32 x 32 x 1
(2) CONV5: 5 x 5 size, 32 filters, 1 stride
(3) ReLU: max(0,h\theta(x))
(4) POOL: 2 x 2 size, 1 stride
(5) CONV5: 5 x 5 size, 64 filters, 1 stride
(6) ReLU: max(0,h\theta(x))
(7) POOL: 2 x 2 size, 1 stride
(8) FC: 1024 Hidden Neurons
(9) DROPOUT: p = 0.5
(10) FC: 10 Output Classes
```

Furthermore, The accuracy of CNN with Softmax and SVM is shown in figure(3.7) clearly demonstrates that the difference between accuracy is not very large but CNN-Softmax still performs better than CNN-SVM.



Dataset	CNN-Softmax	CNN-SVM
MNIST	99.23%	99.04%
Fashion-MNIST	91.86%	90.72%

Figure 3.7: Test accuracy of CNN-Softmax and CNN-SVM on image classification. Source:[13]

The workflow given in [14], as shown in fig(3.8) is performed on MNIST dataset. The same workflow is used for other datasets for the classification task.

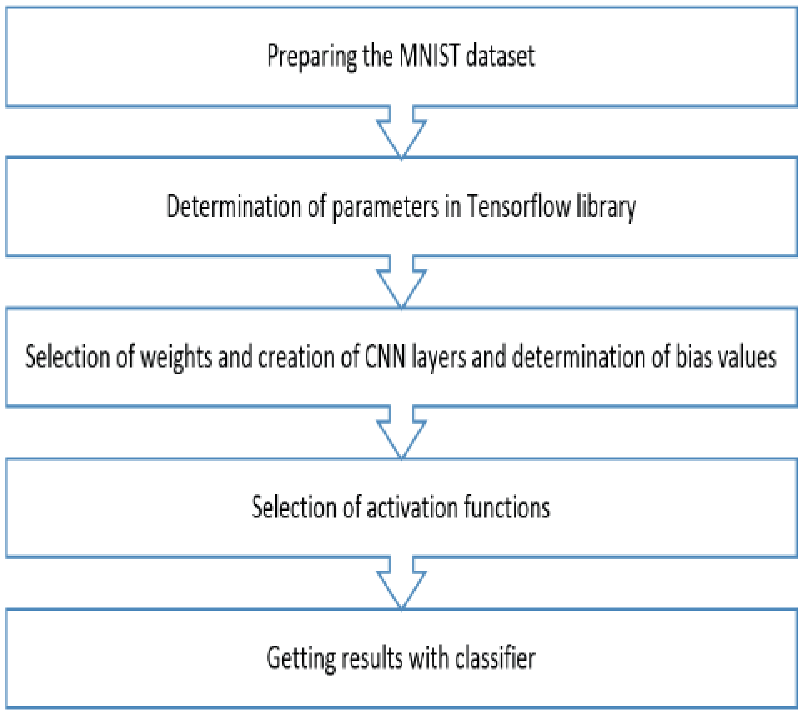
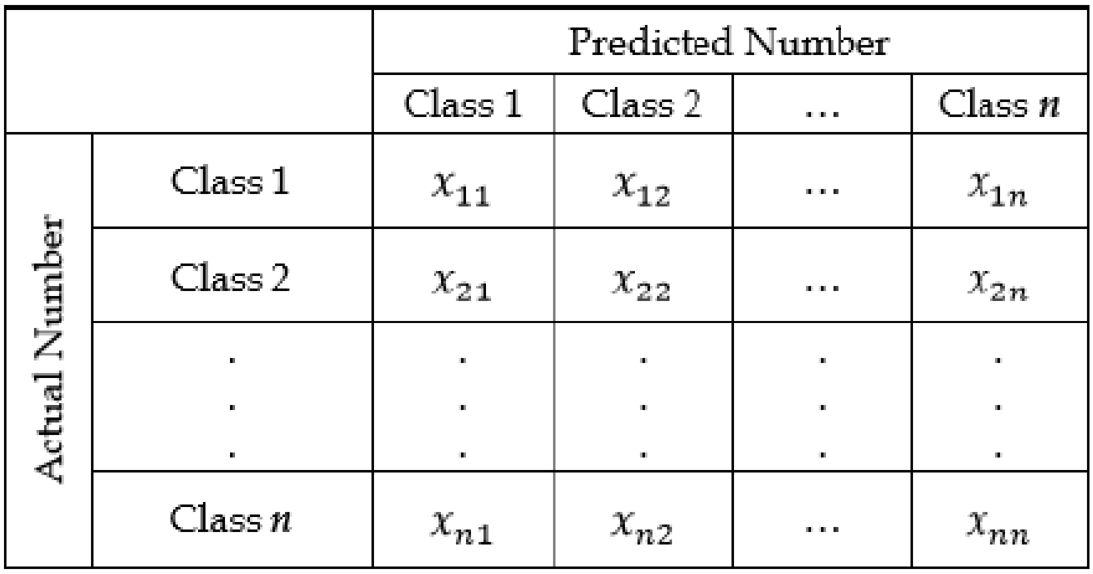

Preparing the MNIST dataset
Determination of parameters in Tensorflow library
Selection of weights and creation of CNN layers and determination of bias values
Selection of activation functions
Getting results with classifier

Figure 3.8: Workflow for CNN classification with MNIST. Source:[14]

3.4 Performance metrics for the model

To properly understand the accuracy of classification done by the model, it is mandatory to have some metrices to test it on.
A summary of classification results can be stored in a matrix called **Confusion Matrix** also known as error matrix, which stores the number of correct and incorrect predictions broken down by each class. The minority classes are called the "positive classes" and the majority classes are called the "negative classes".
Consider a multi-class classification problem on a dataset containing n classes. The



		Predicted Number			
		Class 1	Class 2	...	Class n
Actual Number	Class 1	x_{11}	x_{12}	...	x_{1n}
	Class 2	x_{21}	x_{22}	...	x_{2n}
	⋮	⋮	⋮	⋮	⋮
	Class n	x_{n1}	x_{n2}	...	x_{nn}

Table 3.1: Confusion Matrix for n classes. Source:[1]

generalized format of the confusion matrix for the same can be observed in Table 3.1, where the actual values and predicted values are compared to see how many predictions are right or how confused the classifier is during the classification process.
In general, the results obtained can be divided into four classes:

- **True Positives:** The cases where the predicted class and actual class is same are known as true positives. The true positive rate (TPR) can be calculated using

Equation(3.1).

$$TPR = \sum_{j=1}^{n} x_{jj} \tag{3.1}$$

- **False Negatives:** The cases in which data samples from positive classes get misclassified. The total number of cases of False Negatives for each class i or false negative rate(FNR) can be calculated as Equation(3.2).
  In essence, it is the sum of all elements in a column, except for the element which belongs to the diagonal (or True Positive).

$$FNR_i = \sum_{j=1, j\neq i}^{n} x_{ij} \tag{3.2}$$

- **False Positives:** The cases in which a sample from a negative class gets misclassified under other labels. The total number of cases of False Positives for each class i or false positive rate(FPRi) can be calculated using Equation(3.3).
  In essence, it is the sum of all elements in a row, except for the element which belongs to the diagonal (or True Positive).

$$FPR_i = \sum_{j=1, j\neq i}^{n} x_{ji} \tag{3.3}$$

- **True Negatives:** The cases in which a data sample belonging to a negative class is correctly classified. The total number of cases of True Negatives for each class i or true negative rate(TNRi) can be calculated as Equation(3.4).
  In essence, it is the sum of all the values of the confusion matrix, excluding the row and column belonging to the class i.

$$TNR_i = \sum_{j=1, j\neq i}^{n} \sum_{k=1, k\neq i}^{n} x_{jk} \tag{3.4}$$

3.4.1 Accuracy

The accuracy of classification algorithms is given by Equation(3.5). Accuracy represents the ratio of the number of correct predictions to the total number of predictions.

$$Accuracy = \frac{TPR + TNR}{TPR + TNR + FPR + FNR} \tag{3.5}$$

3.4.2 Precision

Precision measures how many predictions were correct, i.e., the ratio of how many of the predicted labels are actually present in the ground truth of the dataset. High Precision indicates an example labelled as positive is indeed positive (a small number of FP). Precision can be calculated using Equation(3.6).

$$Precision = \frac{TPR}{TPR + FPR} \tag{3.6}$$

3.4.3 Recall

Recall, or True Positive Rate is a ratio of how many predicted labels are the same as actual labels to the total number of actual labels or data samples for a class i. High Recall indicates the class is correctly recognized (a small number of FN). Recall can be found using Equation(3.7).

$$Recall = \frac{TPR}{TPR + FNR} \tag{3.7}$$

3.4.4 F1-Score or F-measure

Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more.
The F-Measure will always be nearer to the smaller value of Precision or Recall. it can be calculated using Equation(3.8).

$$F1 - Score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{3.8}$$

3.5 Deep Learning Frameworks

Plenty of deep learning frameworks are available for implementing machine learning and computer vision tasks like Tensorflow [38], Keras [43], PyTorch [44] etc. This section briefly describes the structure and working of the frameworks employed in this research implementation.

3.5.1 Tensorflow

TensorFlow. is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used

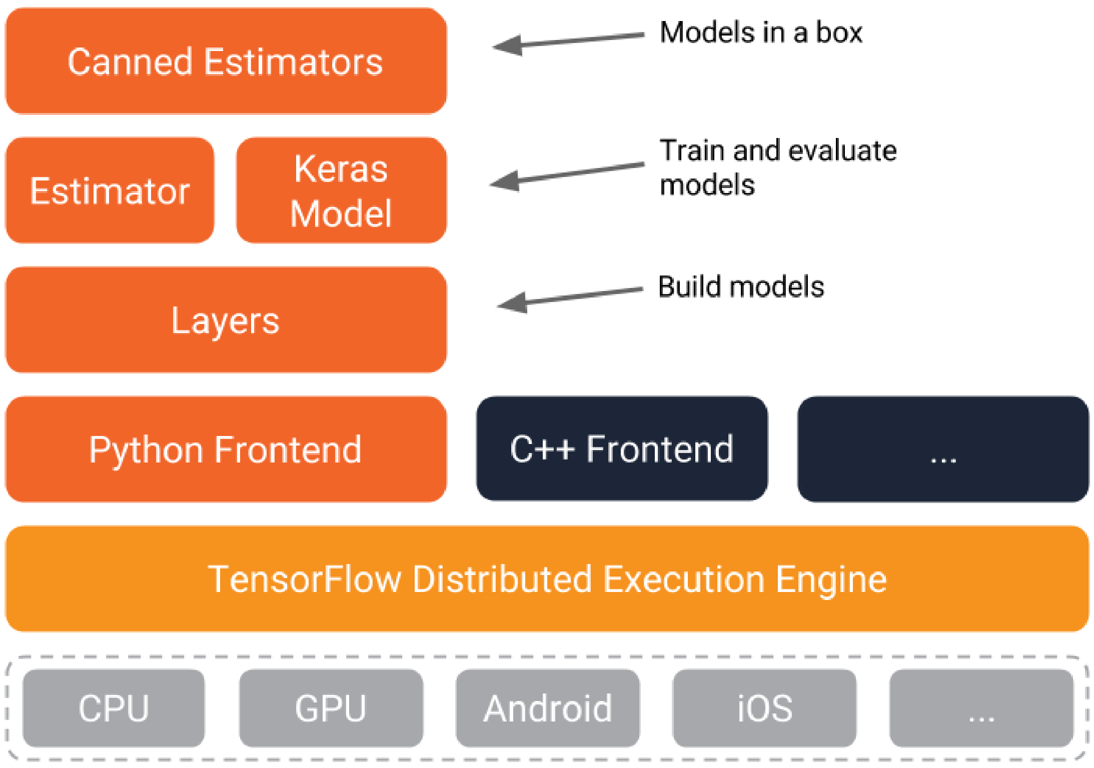


Figure 3.9: Tensorflow programming stack

for machine learning applications such as neural networks. It is used for both research and production at Google. It operates by developing a static computational graph for the operations, wherein nodes represent the functions or operators and the edges are data (tensors). The programming stack for tensorflow can be seen in figure(3.9).

3.5.2 Numpy

Numpy is a part of Python and is used to apply high-level mathematical functions on multi-dimensional arrays and matrices. It is used to create large arrays for all the images used in the research and then apply operations on them accordingly.

3.6 Workflow

The last section of this chapter shows the overall workflow of this dissertation i.e. how different components are connected to each other and used to evaluate the performance of each algorithm and the complete architecture.

It is shown in figure(3.10). The first step is to simply record the CNN+Softmax

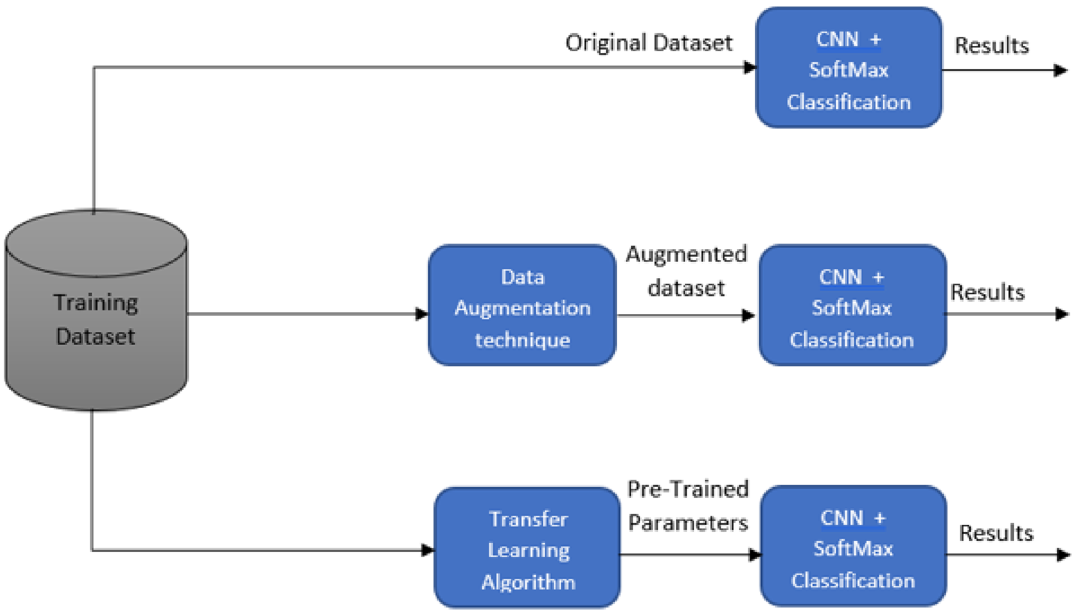

Original Dataset
CNN + SoftMax Classification
Results
Training Dataset
Data Augmentation technique
Augmented dataset
CNN + SoftMax Classification
Results
Transfer Learning Algorithm
Pre-Trained Parameters
CNN + SoftMax Classification
Results

Figure 3.10: Overall workflow of the dissertation

accuracy on the original dataset. After that the dataset is augmented using geometric

transformations and elastic distortions to see how the performance gets better.
Finally, Transfer learning algorithms are used to get the pre-learned parameter and apply them on the classifier. Different datasets are used in this process to see how the react to these techniques and which algorithm is better for what kind of data.
These results would then be compared and contrasted in Chapter 5 to gain a thorough understanding of the effect of transfer learning and data augmentation using different methods.

Chapter 4

Implementation

This chapter consumes the design specifications explained in Chapter(3) and outline the details of implementation used throughout the research. It also specifies the datasets and setup used for the dissertation. These implementations are used in next chapter to draw conclusions from the experiments.

4.1 Technical Setup

4.1.1 Libraries and Languages

For the implementation purposes of this research, **Python 3** is used as the programming language. Additionally, as discussed in chapter 3, Tensorflow is the main deep learning framework used here. Keras is also used, but since it is a part of Tensorflow library, it's not explained separately. Other useful libraries are **Numpy, Matlpotlib and Pandas**.

4.1.2 Environment

For general implementation, Google Colab [45] is used. Google Colaboratory more commonly referred to as "Google Colab" or just simply "Colab" is a research project for prototyping machine learning models on powerful hardware options such as GPUs and TPUs. It provides a serverless Jupyter notebook environment for interactive development. Google Colab is free to use like other G Suite products [46].

4.2 Datasets

For the purpose of implementation, testing and evaluation of algorithms used in this research the following datasets are used.

Name	Dimensions			Size		Classes
	Width	Height	Depth	Training	Test	
MNIST	28	28	1	60000	10000	10
FMNIST	28	28	1	60000	10000	10

Table 4.1: Dataset summaries

MNIST [16] and FMNIST [17] are the standard datasets that are used as benchmark in machine learning and computer vision. The characteristics are explained in the following section.
A few reasons to use this database are:

- The datasets are not too big, which makes them perfect for the testing. Also, if required we can use a subset of the databases.
- These databases are balanced which makes them secure from the class imbalance problem [47].

4.2.1 MNIST dataset

The MNIST dataset available at [16] is a collection of 70000 images of handwritten digits. The images are divided into 60000 training examples and 10000 test set images. The dimension of these images are 28x28 with a depth of 1, which connotes that the images are greyscale.
It is elite for pattern recognition on real world data without any pre-processing requirements. The dataset is shown in fig(4.1), which illustrates that all the images have handwritten numbers between 0-9.
It is derived from NIST dataset [48], which is a larger collection of black and white images of characters, digits and other symbols. The images of digits are converted into greyscale and centered into 28x28 image by computing the center of mass of the pixels and translating the image such that this point remains at the center.

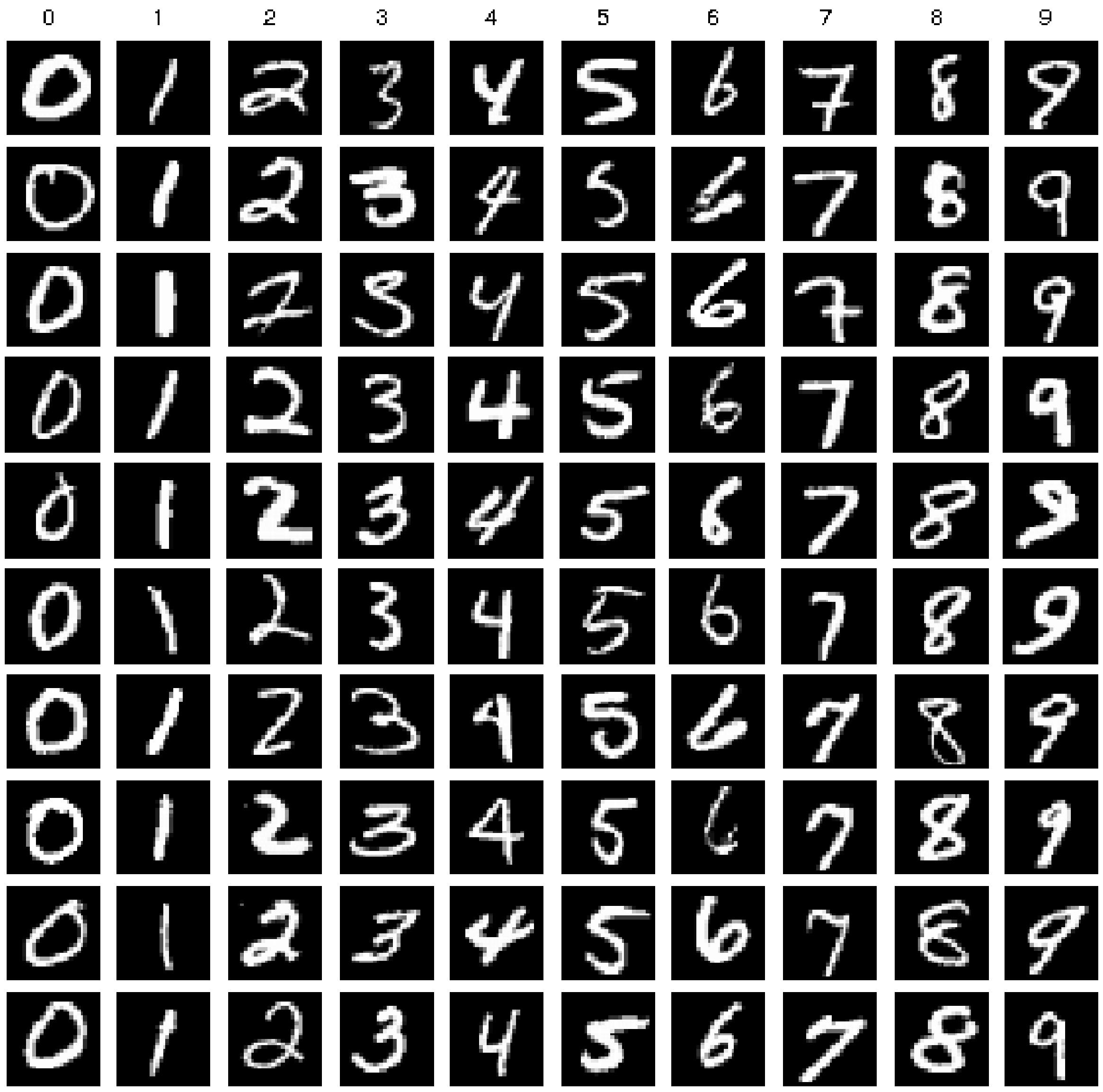
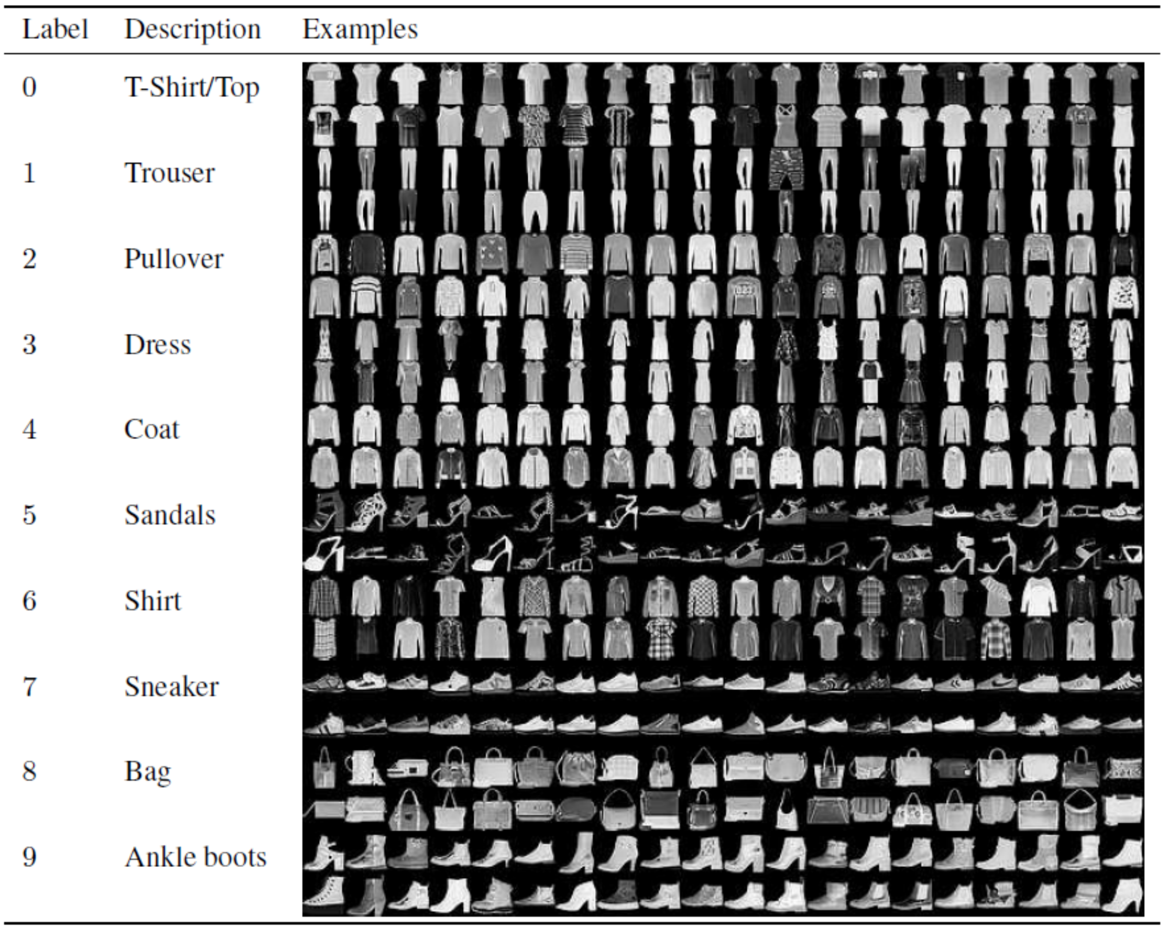


Figure 4.1: MNIST dataset with labels

4.2.2 FMNIST dataset

The Fashion MNIST [17] is dataset contains 28x28 greyscale images of clothing articles as depicted in fig(4.2). Sometimes the simplicity of MNIST dataset makes it prone to false positives sometimes, therefore FMNIST is also used for the evaluation of this research.

It was created by Zolando [49] to serve as **direct drop in replacement** for the original MNIST dataset. It shares the same size and training-test split as MNIST.



Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 4.2: Fashion MNIST with labels

As mentioned before, this dataset is perfectly balanced and each class contains exactly 6,000 train instances and 1,000 test instances.

4.2.3 Dogs Vs. Cats Dataset

Since, the last two dataset are greyscale and very simple in terms of background details and objects present in the images and might give good results even with a simple algorithms or result in overfiiting when presented to complex algorithms, therefore another dataset is used for this study. This dataset is called Dogs vs. Cats [18] and is imported from Kaggle [50].

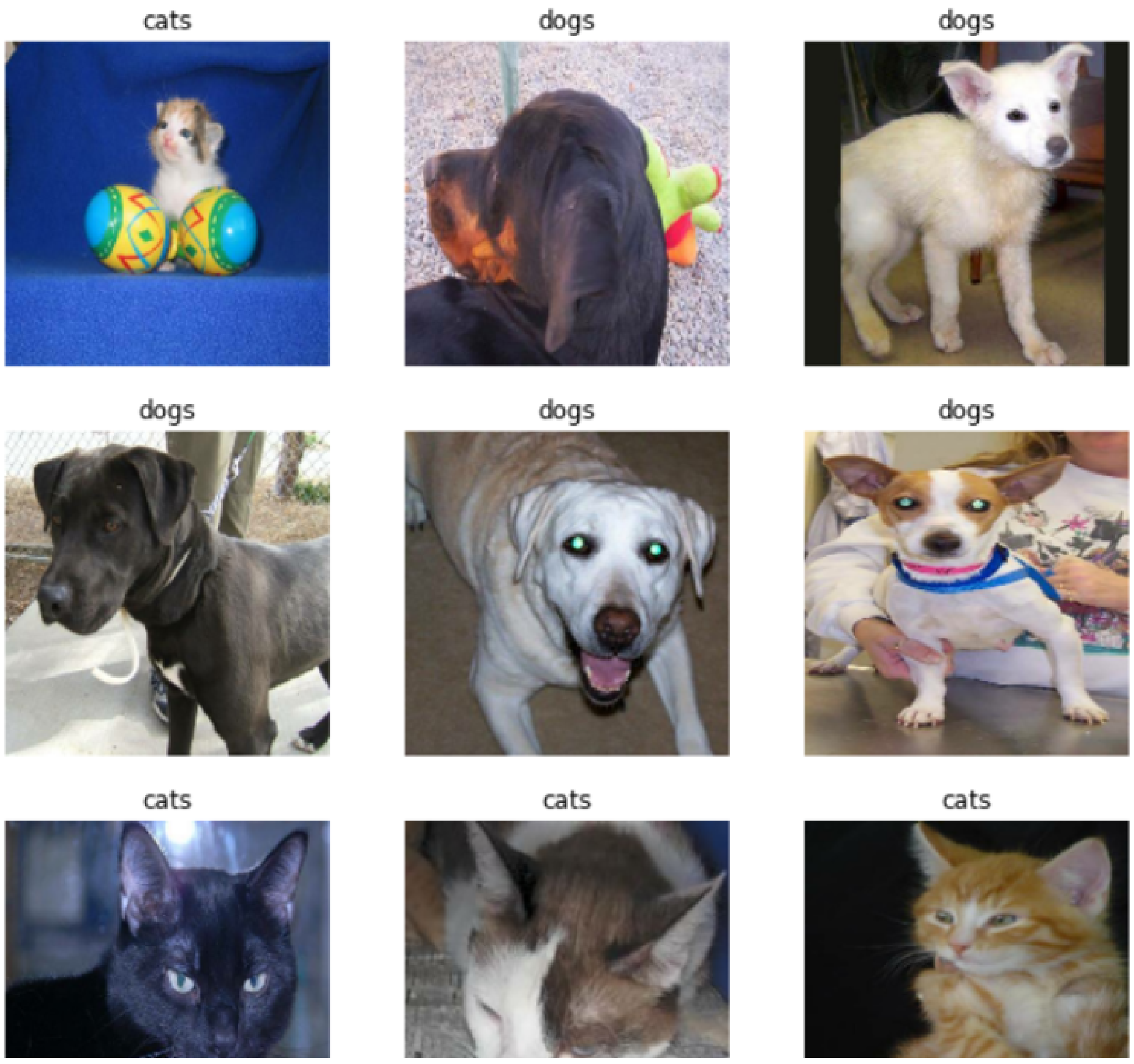


Figure 4.3: Cats and Dogs Dataset with labels

There are two folders given with this dataset, train and test. The train folder contains 25,000 images of dogs and cats. Each image in this folder has the label as part of the filename. The test folder contains 12,500 images, named according to a numeric id. For each image in the test set, a probability that the image is a dog (1 = dog, 0 = cat) is predicted.

4.3 Data Augmentation

In order to increase the amount of data available for training, the data set is augmented using the following ways:

4.3.1 Image Transformation

As described in Chapter(3), the **tf.keras.preprocessing.image** class is used for image trasformations. The functions used to do the same can be seen in code shown below in Listing(4.1).

The function **augment_data** takes in the dataset images along with their labels and the augmentation factor which is basically the number of times an image will be transformed. The additional arguments for this function takes in whether user wants to do rotation, shearing, shifting, zoom or all of them on the images.

This function returns the augmented dataset with labels. Each image will have $4 * augmentation\_factor$ more variations of itself in the resulting set.

```
def augment_data(dataset, dataset_labels, augementation_factor=1,
    use_random_rotation=True, use_random_shear=True, use_random_shift=
    True, use_random_zoom=True):
  augmented_image = []
  augmented_image_labels = []

  for num in range (0, dataset.shape[0]):

    for i in range(0, augementation_factor):
      # original image:
      augmented_image.append(dataset[num])
      augmented_image_labels.append(dataset_labels[num])

      if use_random_rotation:
        augmented_image.append(tf.keras.preprocessing.image.
    random_rotation(dataset[num], 20, row_axis=0, col_axis=1,
    channel_axis=2))
        augmented_image_labels.append(dataset_labels[num])

      if use_random_shear:
```

```python
        augmented_image.append(tf.keras.preprocessing.image.random_shear(dataset[num], 0.2, row_axis=0, col_axis=1, channel_axis=2))
        augmented_image_labels.append(dataset_labels[num])

      if use_random_shift:
        augmented_image.append(tf.keras.preprocessing.image.random_shift(dataset[num], 0.2, 0.2, row_axis=0, col_axis=1, channel_axis=2))
        augmented_image_labels.append(dataset_labels[num])

      if use_random_zoom:
        augmented_image.append(tf.keras.preprocessing.image.random_zoom(dataset[num], (0.9, 0.9), row_axis=0, col_axis=1, channel_axis=2))
        augmented_image_labels.append(dataset_labels[num])

  return np.array(augmented_image), np.array(augmented_image_labels)
```

Listing 4.1: Function for Image Transformation

The individual functions and their usage are described below:

- **tf.keras.preprocessing.image.random_rotation:** Performs a random rotoation on Numpy Tensor and returns rotated numpy image tensor. Input arguments are a 3d image tensor, rotation range in degrees, index of axis or rows, columns and channels in the input tensor [51].

- **tf.keras.preprocessing.image.random_shear:** Performs a random spatial shear of a Numpy image tensor and returns sheared Numpy image tensor. Input arguments are a 3d input tensor, transformation intensities in degrees, index of axis for rows, columns and channels in the input tensor. Additonal arguments are fill_mode, cval and interpolation_order which are not used in this function [52].

- **tf.keras.preprocessing.image.random_shift:** Performs a random spatial shift of a Numpy image tensor and returns shifted Numpy image tensor. Input arguments are a 3d input tensor, width and height shift range as a float fraction of them, index of axis for rows, columns and channels in the input tensor [53].

- **tf.keras.preprocessing.image.random_zoom:** Performs a random spatial zoom of a Numpy image tensor and returns zoomed Numpy image tensor and raises a value error if zoom_range isn't a tuple. Input arguments are a 3d input tensor, zoom_range for width and height which is a tuple of floats, index of axis for rows, columns and channels in the input tensor [54].

Apart from these, the **append** function is used to add the transformed Numpy tensor to the end of the array and **np.array** is used to return all of the images as a single Numpy tensor and same is done for the labels.

4.3.2 Elastic Distortion

This Elastic Distortion method is based on [31], as dsecribed in Chpater(2) the values of α and σ are used in the implementation.
The function described in Listing(4.2) takes in the Numpy tensor image along with the alpha_range and sigma for the distortion. It returns the distorted Numpy tensor image. This function is called on every image to create a new dataset with one original and one distorted image, i.e. the dataset is doubled in size.

```python

def elastic_transform(image, alpha_range, sigma, random_state=None):
    if random_state is None:
        random_state = np.random.RandomState(None)

    if np.isscalar(alpha_range):
        alpha = alpha_range
    else:
        alpha = np.random.uniform(low=alpha_range[0], high=alpha_range
   [1])

    shape = image.shape
    dx = gaussian_filter((random_state.rand(*shape) * 2 - 1), sigma) *
    alpha
    dy = gaussian_filter((random_state.rand(*shape) * 2 - 1), sigma) *
    alpha

    x, y, z = np.meshgrid(np.arange(shape[0]), np.arange(shape[1]), np
.arange(shape[2]), indexing='ij')
    indices = np.reshape(x+dx, (-1, 1)), np.reshape(y+dy, (-1, 1)), np
.reshape(z, (-1, 1))

    return map_coordinates(image, indices, order=1, mode='reflect').
reshape(shape)
```

Listing 4.2: Function for Image Elastic Distortion

The function used in the above code are described as follows:

- **np.random.RandomState:** RandomState exposes a number of methods for generating random numbers drawn from a variety of probability distributions. In addition to the distribution-specific arguments, each method takes a keyword argument size that defaults to None. If size is None, then a single value is generated and returned. If size is an integer, then a 1-D array filled with generated values is returned. If size is a tuple, then an array with that shape is filled and returned.[55]
- **np.random.uniform:** Draws samples from a uniform distribution.
  Samples are uniformly distributed over the half-open interval [low, high) (includes low, but excludes high). In other words, any value within the given interval is equally likely to be drawn by uniform.[40]
- **gaussian_filter:** Implements a multidimensional gaussian filter. The input arguments are the input array, sigma which is the standard deviation for the kernel.[55]
- **np.meshgrid:** Returns coordinate matrices from coordinate vectors.
  Make N-D coordinate arrays for vectorized evaluations of N-D scalar/vector fields over N-D grids, given one-dimensional coordinate arrays x1, x2,. . . , xn. [40]
- **map_coordinates:** Map the input array to new coordinates by interpolation.
  The array of coordinates is used to find, for each point in the output, the corresponding coordinates in the input. The value of the input at those coordinates is determined by spline interpolation of the requested order.
  The shape of the output is derived from that of the coordinate array by dropping

the first axis. The values of the array along the first axis are the coordinates in the input array at which the output value is found.[55]

The above two techniques for data augmentation are used individually and together to record the effects on the accuracy which is discussed in Chapter(5).

4.4 Classification Model

The classification model is used to finally classify the images and put labels on them. It is created using the code shown in Listing(4.3).

```
model = models.Sequential()
model.add(layers.Conv2D(32, (5, 5), activation='relu', input_shape
    =(28,28,1)))
model.add(layers.MaxPool2D((2,2)))
model.add(layers.Conv2D(64, (5,5), activation='relu'))
model.add(layers.MaxPool2D((2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(10))
```

Listing 4.3: CNN model for classification

The two classes used here are **tf.keras.model** and **tf.keras.layers**. The **model class** groups layers into an object with training and inference features while the **layers class** is the class from which all layers inherit.
A layer is a callable object that takes as input one or more tensors and that outputs one or more tensors. It involves computation, defined in the call() method, and a state (weight variables), defined either in the constructor _init_() or in the build() method [43]. The functions used in the above code segment are explained below:

- **models.Sequential:** It is used to group a linear stack of layers into a model. It provides training and inference features on this model.

- **model.add:** Adds a layer instance on top of the layer stack. It takes in layer instance as an argument.

- **layers.Conv2D:** Creates a 2D convolutional layer (e.g. spatial convolution over images). This layer is used to make a convolutional kernel which is further used to create tensor o f outputs.
- **layers.MaxPool2D:** Max pooling operation for 2D spatial data. It decreases the size of the feature by taking the maximum value given it the pool_size window. Window is shifted by strides in each dimension.
  The resulting output is in the shape of (input_shape - pool_size + 1)/strides.
- **layers.Flatten:** Flatten the input, i.e. it changes the input size from 2D to 1D. To illustrate, if the input had size (None, 4, 4) the resultant would have the size of (None, 16). It does not affect the batch size.
- **layers.Dense:** It is used to implement the fully connected neural network layers. The operation: output = activation(dot(input, kernel) + bias) is implemented by Dense layer, where element-wise activation is performed on the input which is input and kernel and bias is the optional bias vector.

The resultant network is shown in the table below which describes the output shape of each layer and the number of paramenters each layer will learn.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 24, 24, 32)        832

_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 12, 12, 32)        0

_________________________________________________________________
conv2d_21 (Conv2D)           (None, 8, 8, 64)          51264

_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 4, 4, 64)          0

_________________________________________________________________
flatten_8 (Flatten)          (None, 1024)              0

_________________________________________________________________
dense_22 (Dense)             (None, 256)               262400
```

```
_________________________________________________________________
dense_23 (Dense)             (None, 10)                2570
=================================================================
Total params: 317,066
Trainable params: 317,066
Non-trainable params: 0
_________________________________________________________________
```

As it's clear from the table that pooling layers do not have any parameters to learn because they are only used to reduce the size of the feature map. The size of final result is (None, 10) which implies the probability for the 10 classes.

4.5 Transfer Learning

The libraries and functions written to implement Transfer Learning process are briefly described in this section.

- **Preprocessing the data:** The preprocessing of the data is an important step in this course of action because if the input is not compatible with the specified input size and feature of the model, then the model will not be compiled. The preprocessing function is defined below which changes the image size to 160x160.

```python
IMG_SIZE = 160 # All images will be resized to 160x160

def format_example(image, label):
    image = tf.cast(image, tf.float32)
    image = (image/127.5) - 1
    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
    image = tf.image.grayscale_to_rgb(image)
    return image, label

```

Listing 4.4: Pre-processing dataset

The above **format_example** function takes in the image and it's label and returns them after converting the image accordingly.

The first two lines of the function convert the pixel values to float and normalize them between 0 and 1, so that the values are in a workable range. Afterwards, the image is resized to 160x160, as it is the input format for the model. Lastly, the image is converted to 1 channel to 3 channels. This step is dependent on the dataset being used.

- **Loading the base model:** The pre-trained models are already available in the **tf.keras.applications** library which can be imported directly to the project.

```
IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)

# Create the base model from the pre-trained model MobileNet V2
base_model = tf.keras.applications.MobileNetV2(input_shape=
    IMG_SHAPE, include_top=False, weights='imagenet')

```

Listing 4.5: Loading Base Model

The MobileNetV2 model as described in Section(3.2) is imported here. The arguments are input shape which defines the shape of the images, include_top which is set to false because we do not need to include the classification layers of the model. The weights that are included here are of Imagenet dataset as described in Section(3.2).

- **Freezing the Convolutional Base:** Once the model is loaded, it is frozen in order to prevent the model from re-training itself on the new data and calculating new parameters.

```
base_model.trainable = False

```

Listing 4.6: Freeezing Base Model

```
=================================================================
Total params: 2,257,984
Trainable params: 0
Non-trainable params: 2,257,984
_________________________________________________________________
```

As it is seen above, the number of trainable parameters are 0, which proves the previous point of not training the data on new data.

- **Adding classification layers:** After the loading and freezing of model is done, the final fully connected classification layer can be added to the model. Here, the **tf.keras.layers** library is used.

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)

prediction_layer = tf.keras.layers.Dense(10)
prediction_batch = prediction_layer(feature_batch_average)

model = tf.keras.Sequential([
  base_model,
  global_average_layer,
  prediction_layer
])

```

Listing 4.7: Adding Classification Layers to Base Model

One pooling and one dense layer is added to the model. Here, the output of the dense model depends on the number of classes present in the dataset.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_160 (Model) (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 10)                12810
=================================================================
Total params: 2,270,794
Trainable params: 12,810
Non-trainable params: 2,257,984
```

```
_________________________________________________________________
```

The total number of trainable parameter for the 10 classes dataset is 12810 as illustrated above. The more the classes, the more the number of parameters.

- **Training the model:** Lastly, after the compilation of the model, it can be trained on the dataset. The **fit** function is used to for this task, it takes in the training and validation dataset along with the number of epochs. In this study, 10 epochs are used for each classification process.

```
history = model.fit(train_batches, epochs=initial_epochs,
    validation_data=validation_batches)

```

Listing 4.8: Training the model

The results of this process are then compared with previous classification results to draw the conclusions, which is discussed in the next chapter.

Chapter 5

Results and Discussions

This chapter establishes and discusses the results obtained in the intermediate steps of the study as well as the final values of the classification metrics obtained for each of the datasets. These results are compared and contrasted and critically analysed to derive insights regarding the research question being addressed.

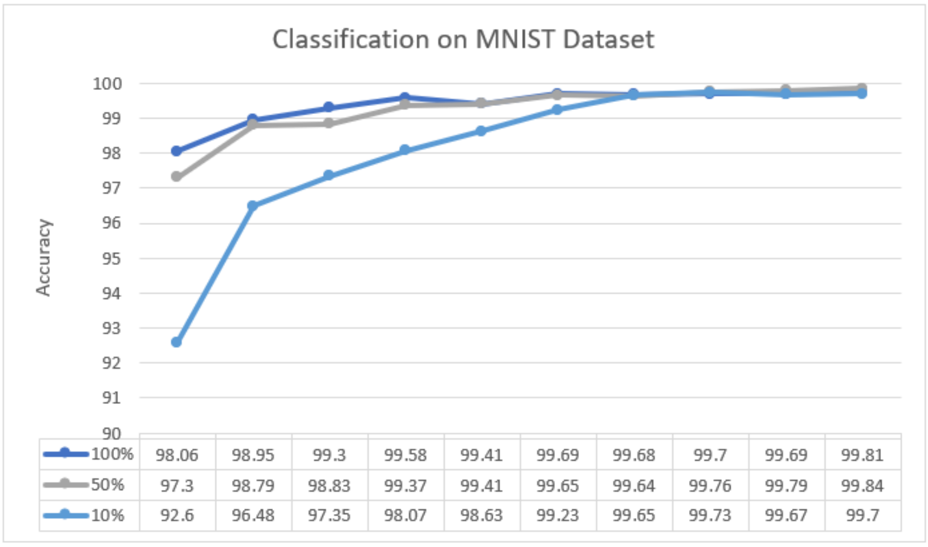
5.1 Classification

The process of classification, as described in Chapter(2) and (3) is the first step of this research. Initially, MNIST and FMNIST dataset are used and classification is performed on their 100%, 50% and 10% subset. The results are shown in fig(5.1) and fig(5.2).
The subsets for DogsVsCats dataset is not used here because it is already smaller than these two datasets and the fact that dataset size affect the accuracy of an algorithm has been proved.

5.1.1 MNIST

In fig(5.1), the accuracy of CNN+Softmax is illustrated over the 10 epochs. It is evident from the graph that the initially the accuracy was affected due to the size of the dataset, but as the epochs were advanced, the accuracy eventually caught up.

This shows that irrespective of the size of the image set, the MNIST dataset's clas-

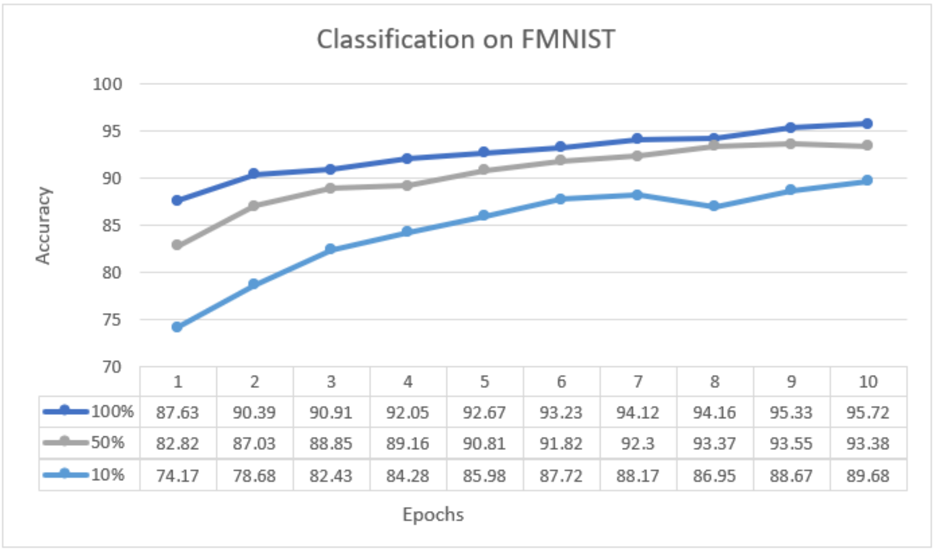


100%	98.06	98.95	99.3	99.58	99.41	99.69	99.68	99.7	99.69	99.81
50%	97.3	98.79	98.83	99.37	99.41	99.65	99.64	99.76	99.79	99.84
10%	92.6	96.48	97.35	98.07	98.63	99.23	99.65	99.73	99.67	99.7

Figure 5.1: Classification on MNIST

sification is intact. Since, the dataset has already given the accuracy of 99.7% on a simple classifier, it is not used for more advanced algorithms.

5.1.2 FMNIST



	1	2	3	4	5	6	7	8	9	10
100%	87.63	90.39	90.91	92.05	92.67	93.23	94.12	94.16	95.33	95.72
50%	82.82	87.03	88.85	89.16	90.81	91.82	92.3	93.37	93.55	93.38
10%	74.17	78.68	82.43	84.28	85.98	87.72	88.17	86.95	88.67	89.68

Figure 5.2: Classification on FMNIST

Since, the MNIST dataset classification was unaffected by the size, FMNIST dataset is used to study the effects. As shown in fig(5.2), the accuracy increases with number of epochs for each subset, but there's a huge difference between the overall initial and final accuracy. While the 100% dataset starts at 87.63% and ends at 95.72%, the 10% dataset accuracy drops to 74.17% in the first epoch and reaches to 89.68%. This marginal drop in the accuracy proves that smaller datasets need more than a CNN+Softmax classifier to perform better.

5.2 Data Augmentation

The results for the three types of augmentation (i.e. geometric transformation, elastic distortion and their combination) process used in this research are presented in this section. Here, only the results for FMNIST dataset is shown to establish the context.

The geometric transformation on FMNIST is shown in fig(5.3). The first row demon-

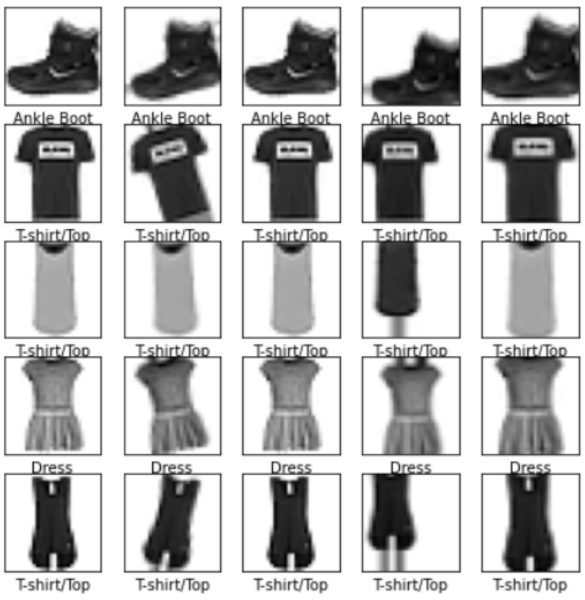


Figure 5.3: Geometric Transformation on FMNIST

strates one call to the transformation function on a single image. The first row consists of five images where first image is the original one, which is later rotated, sheared, shifted and zoomed to create new images. This way every image has it's five variations in the dataset making it five times bigger than the original dataset.

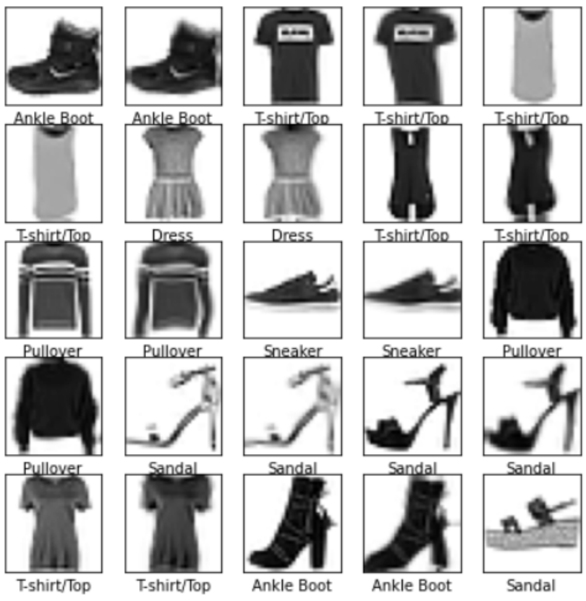


Figure 5.4: Elastic Distortion on FMNIST

Furthermore, in fig(5.4), the elastic distortion function is applied on the FMNIST dataset. It creates one new image for every input which is distorted according to the input arguments given to the function.
As it is clear from the figure every image is followed by it's elastically distorted version making the the dataset twice bigger than it's original size.
Only using this dataset is also an option to achieve good results in some cases, but in this study, a combination of geometric transformation and elastic distortion is used for the data augmentation process, which is shown next.

In fig(5.5), the results for the combination of previous two techniques are shown. As it is evident from the figure that every image has six variation to it where the first one is the original image, followed by four geometric transformed variations and lastly one distorted image.
This way the dataset is six time bigger than it's original version, making it better for the classification task. The classification results for data augmentation are clubbed with transfer learning and softmax classification to compare the final outcome and are

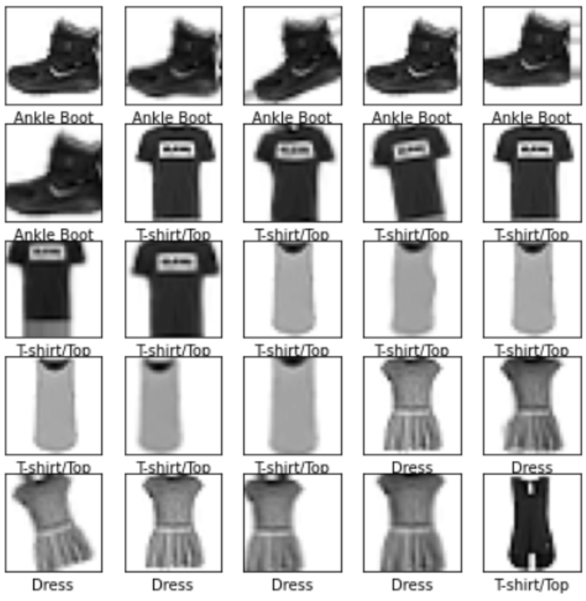
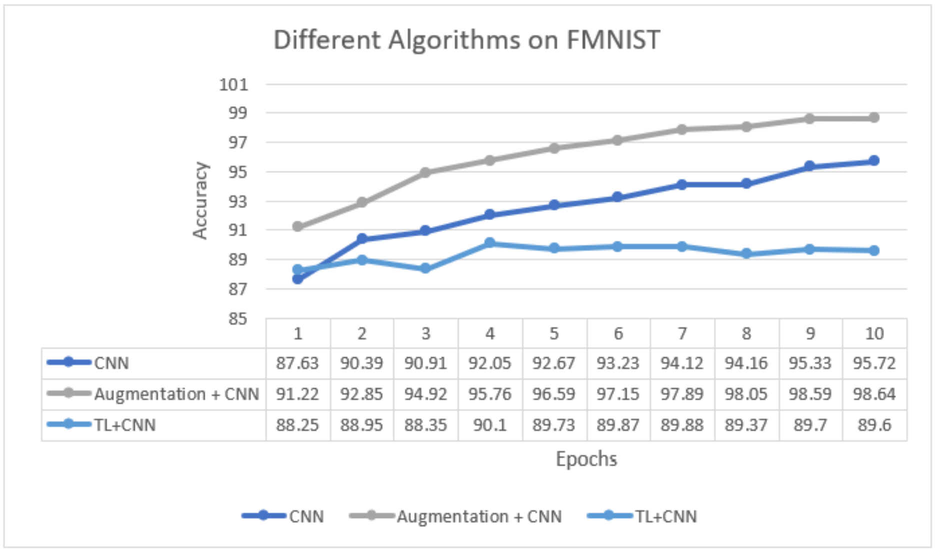


Figure 5.5: Combined Data Augmentation on FMNIST

shown in next section.

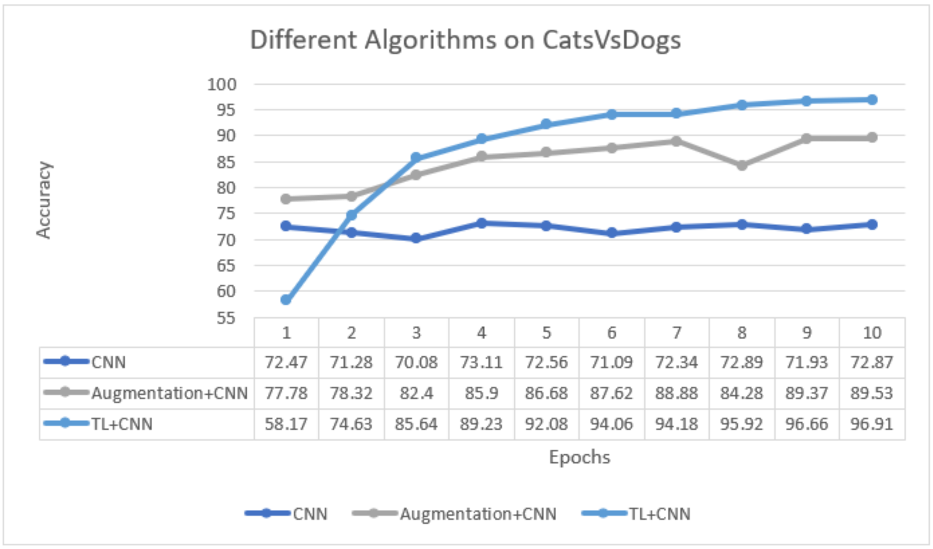
5.3 Transfer Learning



	1	2	3	4	5	6	7	8	9	10
CNN	87.63	90.39	90.91	92.05	92.67	93.23	94.12	94.16	95.33	95.72
Augmentation + CNN	91.22	92.85	94.92	95.76	96.59	97.15	97.89	98.05	98.59	98.64
TL+CNN	88.25	88.95	88.35	90.1	89.73	89.87	89.88	89.37	89.7	89.6

Figure 5.6: Performance of all algorithms on FMNIST

As described in previous chapters, Transfer learning uses pre-trained models to perform the task at hand. In this section, the results obtained from transfer learning and the previous two techniques are put together to compare the results. Here, the results for FMNIST and DogsVsCats datasets are illustrated.
In fig(5.6), CNN classification started at 87.63% and ends at 95.72% whereas the Augmentation + CNN classification improves the accuracy to 98.64%.
The interesting thing to notice here is that transfer learning was assumed to perform better that the other two algorithms, but in case of FMNIST dataset, initial the accuracy decreased to 88.25% and kept swinging near the same value for each epoch with the final accuracy at 89.6%.



	1	2	3	4	5	6	7	8	9	10
CNN	72.47	71.28	70.08	73.11	72.56	71.09	72.34	72.89	71.93	72.87
Augmentation+CNN	77.78	78.32	82.4	85.9	86.68	87.62	88.88	84.28	89.37	89.53
TL+CNN	58.17	74.63	85.64	89.23	92.08	94.06	94.18	95.92	96.66	96.91

Figure 5.7: Performance of all algorithms on CatsVsDogs

Subsequently, the performance of all algorithms is studied on the DogsVsCats dataset in fig(5.7). The accuracy for CNN classification starts at 72.47% and ends at 72.87% which leads to the fact that the algorithm was unable to generalize new data effectively. Afterwards, the Augmentation+CNN process increased the accuracy to 89.53%, which is a huge improvement from previous algorithm. Finally, transfer learning works best with the dataset. Although, the accuracy started at 58.17%, as the epochs increased the accuracy was also going up and reached till 96.91% which is a pretty good performance on the dataset.

Chapter 6

Conclusions

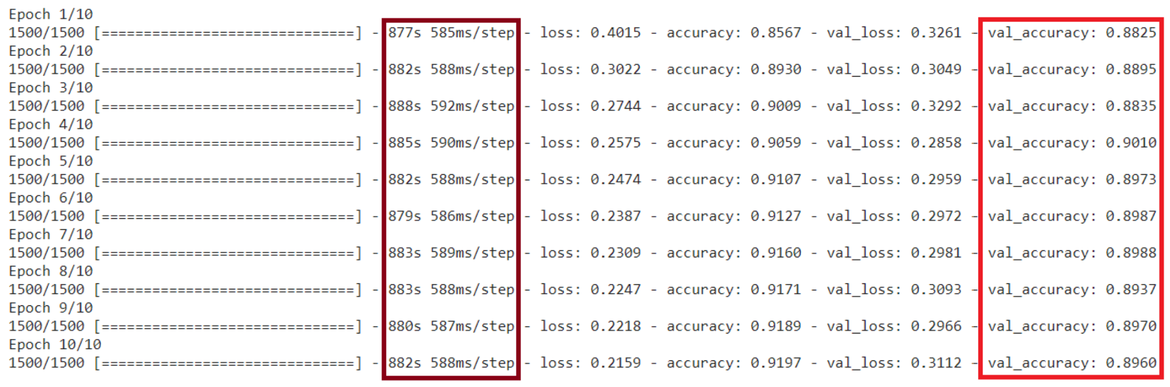
Image classification is an important task in the field of computer vision. It is used widely in tasks like labelling a tumor as malignant or benign, classifying handwritten digits or facial recognition. Image classification has become an important part of human lives without even knowing.
Deep convolutional neural networks has shown a lot of potential in image recognition domain. Some networks even deliver better results than human beings, because they can see beyond the pixels and analyze the details better. But the size of dataset plays a vital role in the performance of these models. The bigger the dataset, the better the performance.
Data augmentation is a way to increase the size of the dataset, which improves the accuracy of the classifier. But augmentation has an upper limit and it shouldn't be done after a certain point, because the pictures become unrecognizable after a threshold which affects the generalization ability of the classifier and generating such pictures is a redundant work.
On the other hand, Transfer learning is an algorithm level method which doesn't change the dataset, but uses pre-trained models and their parameters to do the classification on the current dataset. TL is a very powerful method because it uses extremely deep CNNs that are already trained on big datasets and have enough knowledge to apply on new dataset. The thing to consider before using TL is the nature and size of the dataset in comparison to the original dataset the model was trained on. TL gives better results in comparison to simple CNN classification or classification with data augmentation.

Another interesting thing to consider here is the nature of the dataset. If the dataset is simple, i.e. the background is empty or there's only one or two objects in the foreground, then basic methods will perform better than the advanced ones. For example, FMNIST dataset gives better results with data augmentation, while transfer learning results is overfitting of the model and fails to give good validation accuracy. On the other hand, DogsVsCats algorithm, which is comparatively more complex than FMNIST gives much better results using TL. Also, if the dataset is too simple, the size doesn't have much effect on the performance, e.g. MNIST.
Lastly, transfer learning takes ample amount of time and device requirements, therefore it is better to use data level method first to see if they are giving a better performance.

6.1 Limitations and Future Work

Throughout the course of this study, a few limitations were encountered, some were sorted out while the others remained. In this section those limitations and their future analysis are discussed.

- **Data Augmentation:** The approach that has been taken in this study for data augmentation is very customized i.e. all the functions were coded from scratch. This approach is good for understanding the background details but is not efficient when it comes to time complexity and output.
  Therefore, pre-made data augmentation tools like GANs or advanced ImageGenartors can be used for better results.



```
Epoch 1/10
1500/1500 [==============================] - 877s 585ms/step - loss: 0.4015 - accuracy: 0.8567 - val_loss: 0.3261 - val_accuracy: 0.8825
Epoch 2/10
1500/1500 [==============================] - 882s 588ms/step - loss: 0.3022 - accuracy: 0.8930 - val_loss: 0.3049 - val_accuracy: 0.8895
Epoch 3/10
1500/1500 [==============================] - 888s 592ms/step - loss: 0.2744 - accuracy: 0.9009 - val_loss: 0.3292 - val_accuracy: 0.8835
Epoch 4/10
1500/1500 [==============================] - 885s 590ms/step - loss: 0.2575 - accuracy: 0.9059 - val_loss: 0.2858 - val_accuracy: 0.9010
Epoch 5/10
1500/1500 [==============================] - 882s 588ms/step - loss: 0.2474 - accuracy: 0.9107 - val_loss: 0.2959 - val_accuracy: 0.8973
Epoch 6/10
1500/1500 [==============================] - 879s 586ms/step - loss: 0.2387 - accuracy: 0.9127 - val_loss: 0.2972 - val_accuracy: 0.8987
Epoch 7/10
1500/1500 [==============================] - 883s 589ms/step - loss: 0.2309 - accuracy: 0.9160 - val_loss: 0.2981 - val_accuracy: 0.8988
Epoch 8/10
1500/1500 [==============================] - 883s 588ms/step - loss: 0.2247 - accuracy: 0.9171 - val_loss: 0.3093 - val_accuracy: 0.8937
Epoch 9/10
1500/1500 [==============================] - 880s 587ms/step - loss: 0.2218 - accuracy: 0.9189 - val_loss: 0.2966 - val_accuracy: 0.8970
Epoch 10/10
1500/1500 [==============================] - 882s 588ms/step - loss: 0.2159 - accuracy: 0.9197 - val_loss: 0.3112 - val_accuracy: 0.8960
```

Figure 6.1: Execution of each Epoch in Transfer Learning for FMNIST

- **Time taken for Transfer Learning:** As it can be seen inside the brown box in fig(6.1), the execution of each epoch took approximately 880 seconds i.e. about 14.6 minutes, which is a lot of time for a dataset like FMNIST regarding the size and complexity.
  The solution to this issue can be to use different pre-trained models to see which works better with which dataset. In this research, both MobileNetV2 and VGG19 took a lot of time with FMNIST, while the MobileNetV2 model took less time with DogsVsCats dataset.

- **Over-fitting in Transfer Learning:** The red box in fig(6.1) denotes the accuracy after every epoch. It is clear that after some epochs, the validation accuracy decreases which shows that the model is overfitting over the data because the training accuracy is increasing constantly. It confirms that not all the models work well on all the dataset. The choice of pre-trained model is as important as any other step in the process.

6.2 Final Notes

Although the size of dataset matters to achieve a good classification accuracy, but with enough resources, research and computational power, this problem can be tackled. Data Augmentation and Transfer Learning both are powerful tools to use while working with small datasets, but the nature of the images is also a good point to consider before using any of these techniques.

Bibliography

[1] C. Manliguez, “Generalized confusion matrix for multiple classes,” 11 2016.

[2] Y. Shima, “Image augmentation for object image classification based on combination of pre-trained CNN and SVM,” *Journal of Physics: Conference Series*, vol. 1004, p. 012001, apr 2018.

[3] A. NG, “Machine learning by stanford,”

[4] J. P. Maheswari, “Breaking the curse of small datasets in machine learning: Part 1,” 12 2018.

[5] “Affine transformation.”

[6] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, “Understanding data augmentation for classification: When to warp?,” in *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1–6, 2016.

[7] L. Torrey and J. W. Shavlik, “Chapter 11 transfer learning,” 2009.

[8] P. Marcelino, “Transfer learning from pre-trained models,” 10 2018.

[9] A. Krizhevsky, I. Sutskever, and G. Hinton, “Imagenet classification with deep convolutional neural networks,” vol. 2, pp. 1097–1105, 2012.

[10] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[11] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” 2018.

[12] X.-X. Niu and C. Y. Suen, "A novel hybrid cnn–svm classifier for recognizing handwritten digits," *Pattern Recognition*, vol. 45, no. 4, pp. 1318 – 1325, 2012.

[13] A. F. Agarap, "An architecture combining convolutional neural network (cnn) and support vector machine (svm) for image classification," *arXiv preprint arXiv:1712.03541*, 2017.

[14] F. Ertam and G. Aydin, "Data classification with deep learning using tensorflow," pp. 755–758, 10 2017.

[15] R. Szeliski, *Computer vision: algorithms and applications*, vol. Texts in computer science. Springer, 2011.

[16] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[17] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[18] kaggle, "Dogs vs. cats dataset," 2020.

[19] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding," *Neurocomput.*, vol. 187, p. 27–48, Apr. 2016.

[20] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *CoRR*, vol. abs/1202.2745, 2012.

[21] D. Forsyth and J. Ponce, *Computer vision: a modern approach.* Pearson Education, 2nd ed., international ed ed., 2012.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[23] E. Savitz, "The big cost of data,"

[24] D. Cox, C. Kartsonaki, and R. H. Keogh, "Big data: Some statistical issues," *Statistics & Probability Letters*, vol. 136, pp. 111 – 115, 2018. The role of Statistics in the era of big data.

[25] S. Biffignandi and S. Signorelli, "From big data to information: statistical issues through examples," 10 2015.

[26] "Lecture 9: Generalization, toronto univeristy."

[27] R. Sun, "Optimization for deep learning: theory and algorithms," 2019.

[28] A. Fawzi, H. Samulowitz, D. Turaga, and P. Frossard, "Adaptive data augmentation for image classification," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3688–3692, 2016.

[29] E. J. Bjerrum, "Smiles enumeration as data augmentation for neural network modeling of molecules," 2017.

[30] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," 2017.

[31] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2*, ICDAR '03, (USA), p. 958, IEEE Computer Society, 2003.

[32] J. P. Maheswari, "Breaking the curse of small datasets in machine learning: Part 2," 12 2018.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[35] D. Han, Q. Liu, and W. Fan, "A new image classification method using cnn transfer learning and web data augmentation," *Expert Systems with Applications*, vol. 95, pp. 43 – 56, 2018.

[36] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[37] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.

[39] Google, “Tensolflow module for image preprocessing,” 07 2020.

[40] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, “The numpy array: a structure for efficient numerical computation,” *Computing in Science & Engineering*, vol. 13, no. 2, p. 22, 2011.

[41] Google, “Tensolflow image preprocessing functions,” 07 2020.

[42] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,” *Nature*, vol. 405, pp. 947–951, 2000.

[43] F. Chollet *et al.*, “Keras,” 2015.

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[45] Google, “Google colaboratory,” 2020.

[46] E. Bisong, *Google Colaboratory*, pp. 59–64. Berkeley, CA: Apress, 2019.

[47] N. Japkowicz and S. Stephen, “The class imbalance problem: A systematic study,” *Intell. Data Anal.*, vol. 6, p. 429–449, Oct. 2002.

[48] A. Kramida, Yu. Ralchenko, J. Reader, and and NIST ASD Team. NIST Atomic Spectra Database (ver. 5.7.1), [Online]. Available: `https://physics.nist.gov/asd` [2017, April 9]. National Institute of Standards and Technology, Gaithersburg, MD., 2019.

[49] “Zolando research,” 2017.

[50] “Kaggle inc.,” 2020.

[51] Google, “Tensorflow image rotate.”

[52] Google, “Tensorflow image shear.”

[53] Google, “Tensorflow image shift.”

[54] Google, “Tensorflow image zoom.”

[55] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.

Appendix A

Appendix

This chapter is used for the additional definitions used throughout the research.

A.1 Background

Domain	a domain D consists of two components: a feature space X and a marginal probability distribution P(X), where X = {x1, . . . , xn}
Task	Given a specific domain, D = {X, P(X)}, a task consists of two components: a label space Y and an objective predictive function f(·) (denoted by T = {Y, f(·)}), which is not observed but can be learned from the training data, which consist of pairs $\{x_i, y_i\}$, where $x_i \epsilon$ X and $y_i \epsilon$ Y.

A.2 Design

Model Summary for MobileNetV2: https://docs.google.com/document/d/1UhDjfSb-xhIPiQAU3sCitLQNgkhFS-NPgQZaaF5bXHs/edit?usp=sharing

A.3 Code

The Google Colab Notebooks for the implementation of this research can be found at this address:
https://drive.google.com/drive/folders/1rgqgc7cpGnTUvX_KxsljS6ELEHEZvSml?usp=sharing