Classification of Hat-, Skirt- and Sunglass-Images Using Transfer Learning from FMNIST Dataset

Thesis submitted in partial fulfillment of the requirement for the award of the degree of Masters of Science
in
Mathematics and Computing

Submitted by

Rajenki Das
Roll No. 301603020

Supervised by

Dr. R. K. Sharma

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

SCHOOL OF MATHEMATICS
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY,
PATIALA - 147004, PUNJAB

July 2018

Certificate

I hereby certify that the work, which is being presented in the thesis, entitled **"Classification of Hat-, Skirt- and Sunglass-Images Using Transfer Learning from FMNIST Dataset"** in partial fulfillment of the requirements for the award of the degree of **Masters of Science** in **Mathematics and Computing** and submitted to the School of Mathematics, Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. R. K. Sharma**, Professor, Department of Computer Science Engineering and references to other research works have been duly listed in the reference section.

The matter presented in this thesis has not been submitted elsewhere for the award of any other degree or diploma from any Institution.

Date: 27.07.2018

Rajenki

(Rajenki Das)

Roll No. 301603020

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

R K Sharma

(R. K. Sharma) 27. July. 2018

Professor, Department of Computer Science and Engineering

TIET, Patiala

Acknowledgment

I undertook the journey for the thesis on a state-of-the art-subject of Machine Learning because of the motivation and assistance I have received from many.

I would like to thank my supervisor, **Dr. R. K. Sharma**, Professor, Computer Science Engineering Department (CSED), TIET for his patient guidance and helping me to conceptualize the thesis. I have learnt a lot under his mentorship and I have been very fortunate to have a guide like him who would address my queries promptly and assess my work thoroughly. Without his encouragement and constructive criticism, this thesis could never be completed and delivered on time.

I am profoundly indebted to **Dr. Ashutosh Aggarwal**, Lecturer, CSED, TIET and **Aravind R.** and **Shebin X.P.** (Data Scientists, Institute of Analytics (USA)) for their help, support and valuable feedback. Their help was very useful for me to initiate my project and believe in it.

I would thank the entire School of Mathematics, both teaching and non-teaching staff and fellow students for their cooperation. A special thanks to the current and former Heads of the Department, **Dr. Satish Kumar Sharma** and **Dr. A. K. Lal**, and Post Graduation coordinator **Dr. Paramjeet Singh** for their valuable support.

I thank my parents for being my pillars throughout the thesis journey. Their love, support and feedback helped me sail through the challenging times.

Last but not the least, I thank all my friends for infusing me with positivity. A special thanks to Ms. Simran Lamba and my TIET mates Ms. Shweta Upadhyay, Ms. Simer Wadhwa, Mr. Soham Banerjee, Ms. Twinkle Goyal and Mr. Utkarsh Atri.

Thank you, all!

Rajenki

(Rajenki Das)

Abstract

Arranging datasets which are good and large enough is one of the prime challenges in Machine Learning. To perform a target task, conventionally, the related dataset should be big enough for getting trained and tested. But it is not always possible to fulfil such requirements of datasets, hence knowledge from one model built on a dataset can be transferred to another model with the new dataset. This process of transference is transfer learning.

In this thesis, an improved dataset of a widely used dataset MNIST which is used for benchmarking algorithms has been referenced. With the help of neural networks, the dataset has been trained. Subsequently, another dataset has been built which is similar to the parent dataset but has entirely different classes.

The target task is to classify the new classes. Weights have been transfered from one model to another for the application of transfer learning. Comparisions have been made on the basis of neural networks, model accuracies and model losses. Ultimately, these new transfer learning approaches demonstrate the benefits of transfer learning.

Contents

List of Figures

List of Abbreviations

Short Form	Expanded Form
ANN	Artificial Neural Network
API	Application Program Interface
CNN	Convolutional Neural Network
DBT	Discriminability-based Transfer
FMNIST	Fashion-MNIST
GPU	Graphic Processing Unit
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
MTL	Multi Task Learning
NN	Neural Network
ReLU	Rectified Linear Unit
TL	Transfer Learning

Chapter 1

Introduction

Today, machine learning is widely used in almost every sphere of science and technology. Some of the applications include image recognition, speech recognition, recommendation systems etc. In layman's language, machine learning is simply training a device to perform certain actions to get desired results. Formally it can be defined in the following two ways:
1. In 1959, Arthur Samuel coined the term "Machine Learning" defining it as "Field of study that gives computers the ability to learn without being explicitly programmed."
2. In 1998, Tom Mitchell defined Machine Learning as " Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some tast T and some performance measure P, if its performance on T, as measured by P, improves with experience E." (Mitchell et al. (1997))

Primarily, there are two types of machine learning algorithms: Supervised and Unsupervised. In supervised learning, the goal is to determine a relation between the inputs or attribute variables and the output or dependent attribute. The main idea behind supervised learning is depicted in the image below. [Figure 1.1]

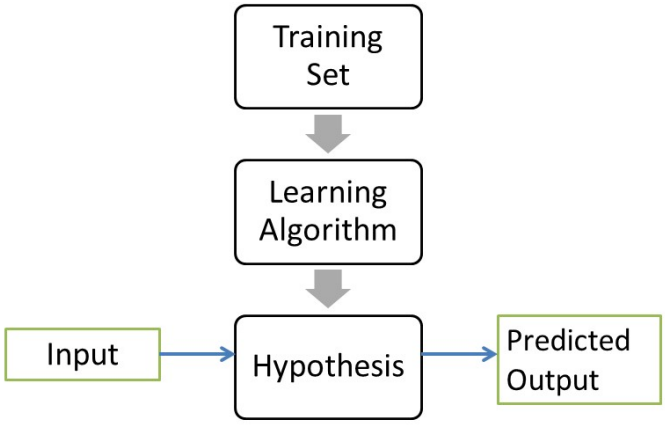

Figure 1.1: Supervised Learning

Supervised learning can be further classified to regression and classification. In simple terms, the output of a regression model is continuous valued while that of a classification model is discrete valued.

In an unsupervised learning, we start with unlabelled data, unlike supervised learning.
In this thesis, we are concerned only with the problem of classification.

With the increasing usage of machine learning, the concept of deep learning has emerged. Deep learning is a subtype of machine learning which uses the concept of neural networks. Before moving further, we discuss neural networks briefly.

Neural networks, or artificial neural networks (ANNs) or artificial neural systems, are systems that try to replicate the fundamental unit of a human brain, i.e., a neuron. In a neural network, input layer does the role of dendrites of a neuron while the output layer acts as an axon. Therefore, a neural network primarily consists of three layers, input, hidden and output. The hidden layers are called so because these don't produce the desired output directly, also for most of the cases, the output of these layers are kept hidden.

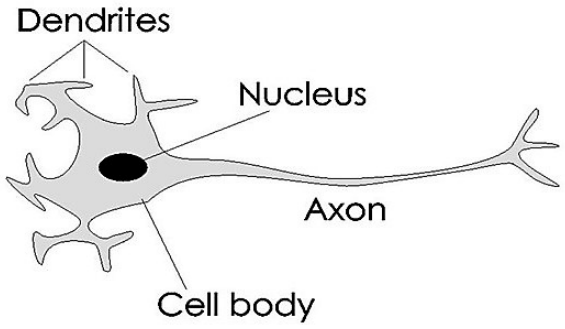

Figure 1.2: Human brain neuron

The functional understanding of a human brain neuron is translated into an artificial neural network. The schematic diagram is described below in Figure 1.3. Here, x_1, x_2, ..., x_n represent the inputs given and w_1, w_2, ..., w_n represent weights. These weighted inputs are summed and passed through activation function f which produces the output y.

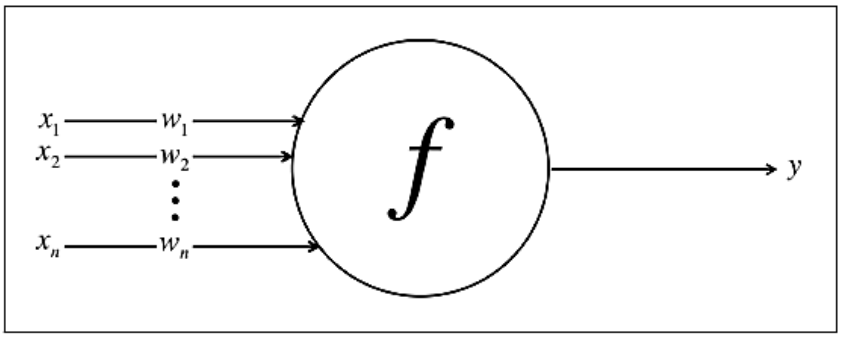

Figure 1.3: Schematic diagram of an artificial neural network

There are different types of neural networks. Convolutional Neural Networks (CNNs) are also a class of neural networks. CNNs have been discussed in section 1.1.

Everytime it is not possible to develop or find a large enough dataset to build a model from scratch. In real life, just like we apply our previously learned information to solve a new problem, similarly "transfer learning" works by learning from a pre-trained model to perform on a new related problem. Because of its analogy to instances in the real life, it has the ability to solve numerous issues of ML. Transfer learning is a

boon to the world of machine learning. Transfer learning has been further discussed in section 1.2.

The crux of the thesis is applying transfer learning on a pre-trained dataset. First, to build a dataset, images have been processed using MATLAB. Next, the models are built on both Fashion MNIST and a new dataset, using a tool called Keras. A brief introduction of the Fashion MNIST dataset is provided in section 1.3. The tools used have been discussed in section 1.4.

The final section of this chapter, section 1.5, gives a summary of the successive chapters in this thesis.

1.1 Convolutional Neural Networks

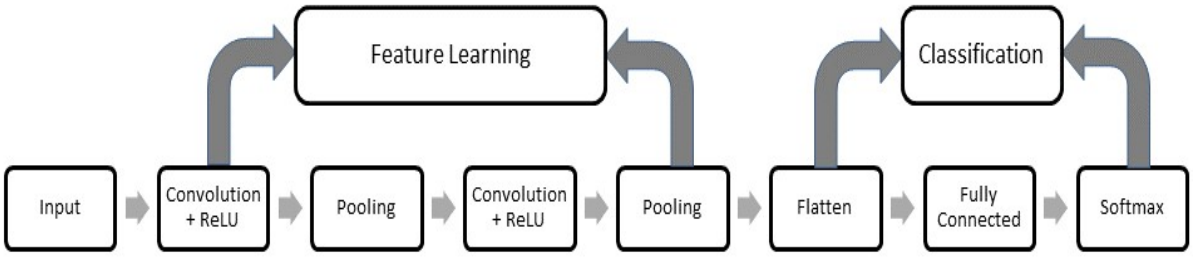

Figure 1.4: Sample CNN structure

Convolutional Neural Network (CNN) or ConvNet is a type of Neural Network that processes those data which have a grid-like topology (Goodfellow et al. (2016)). The name is derived from the mathematical operation "convolution", denoted by $*$, which is defined as:

$$(x * w)(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da \tag{1.1}$$

So, in a CNN, instead of simple matrix multiplication, convolution takes place in at least one layer of the neural network. In terms of CNN, in equation (1.1), 'x' can be called as input, 'w' as neuron or kernel and the output will be the feature map. Mostly, the first layer of the CNN is always a convolutional one. Once, convolution is done and the output is passed through an activation function (mostly ReLU), we move to the next layer i.e. Pooling layer.
There are three stages in a Pooling layer:

Stage 1 Multiple number of convolutions are performed simultaneously which produce a set of linear activations.

Stage 2 A non linear activation function acts on each of these linear activations.

Stage 3 The output of the layer is modified using appropriate pooling function, which helps in reducing the size of the representation.

After several processes of convolution and pooling, there comes the dense/ fully connected layers. Neurons in a fully connected layer is connected to each of the previous activations. These layers constitute the last few layers which help in giving the final output. For example, in the case of a classification problem, outputs from the convolutional and pooling layers are run through the fully connected layers and a final activation function in the end, usually softmax for multi-class, to determine the probabilities of each class.

Often, during training, problem of over-fitting arises i.e. although the training data fits the model, the predictions are not accurate. In the case of training a high number of features, the generalised model may not be good enough. Hence, to solve the problem of over-fitting, "regularization" is done. A layer such as DropOut may be added to perform regularization.

ReLU and Softmax functions are two of the widely used activation functions. The descriptions of each of these functions are given in the following.

1.1.1 ReLU Activation Function

Rectified Linear Unit or ReLU is defined as:

$$f(x) = \begin{cases} x, x \in [0, \infty) \\ 0, x \in (-\infty, 0) \end{cases}$$

The graph of ReLU function is depicted in Figure 1.5.

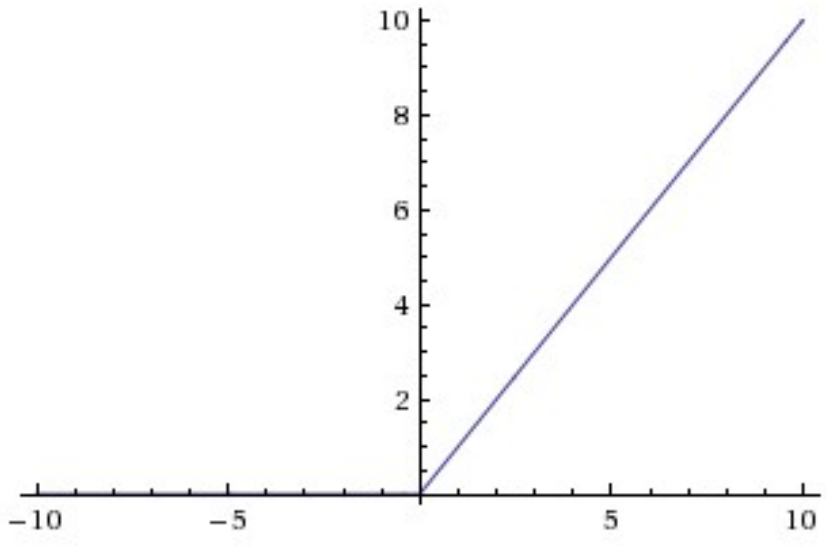

Figure 1.5: Rectified Linear Unit Graph

1.1.2 Softmax Activation Function

To define Softmax, one must understand sigmoid function first. Sigmoid function is defined as the following:

$$f(x) = \frac{1}{1 + e^{-x}}$$

The graph of Sigmoid function is depicted in Figure 1.6.

Sigmoid value lies between 0 and 1, therefore, it is used to predict the probability

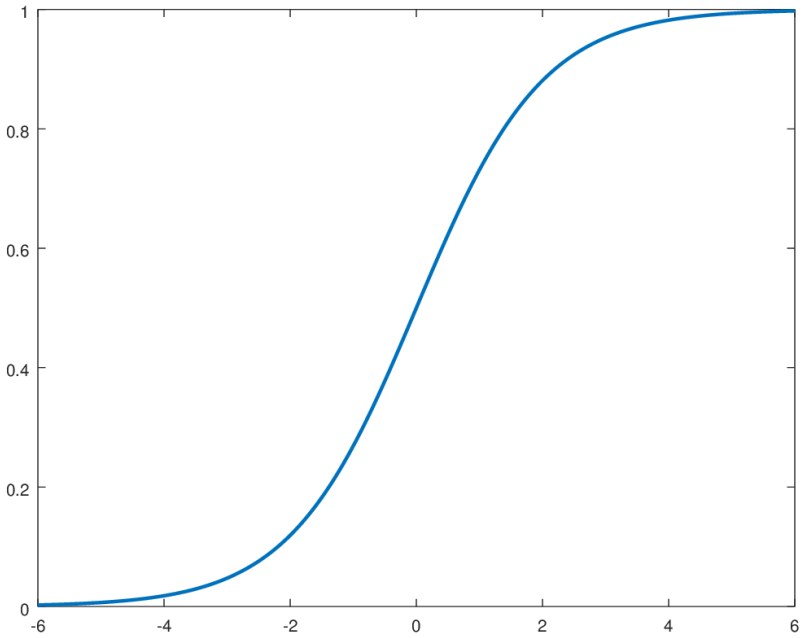

Figure 1.6: Sigmoid Graph

of a class. While sigmoid is used for binary classification, softmax function is used for predicting the probabilities of multiple classes, generally more than 2. Softmax function helps in giving the probability of each class, i.e., suppose an image is fed and the task is to identify it amongst apple (class 1), banana (class 2) and mango (class 3), then when the image passed through a suitable neural network, softmax will tell how much chances are there of the image belonging to any of the three classes by assigning a probability to each class.

Softmax function is defined as:

$$f(x)_i = \frac{e^{x_i}}{\sum_{k=0}^{N} e^{x_k}}$$

where N is the total number of classes.

1.2 Transfer Learning

Given source domain Dom_s and corresponding learning task $Task_s$. Also, given a target domain Dom_t and a corresponding learning task $Task_t$, TL aims in improving the learning of the target predictive function $f_t(.)$ in Dom_t, using the knowledge in Dom_s and $Task_s$, where $Dom_s \neq Dom_t$ or $Task_s \neq Task_t$. (Pan et al. (2010))

The major three research concerns try finding answers to what, how and when to transfer. "What to transfer" is about which part of knowledge gained can be transferred across tasks or domains. "When to transfer" is about the situations in which transferring should be carried out. "How to transfer" is about the way in which transferring can be done. Transferring knowledge does not always, obviously, improve performance. There are cases, when performance of learning can be adversely affected thereby, leading to a situation called negative transfer learning.

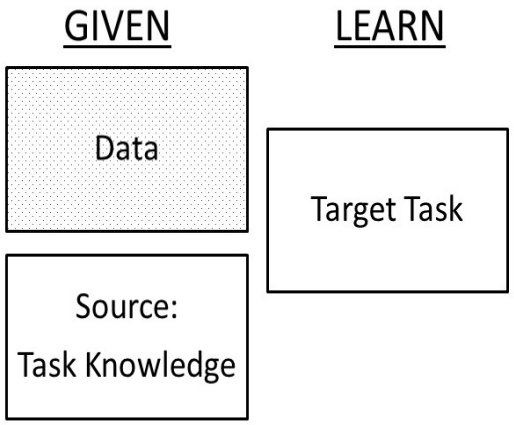

Figure 1.7: Transfer Learning

Transfer learning can be categorized to three settings: inductive, transductive and unsupervised. In inductive TL, the target and the source tasks are different, irrespective of the source and target domains being same or not.
In transductive TL, the target and the source tasks are the same, while the domains are not same.
An unsupervised TL is similar to inductive TL in the way that here also, the tasks are

different, But the difference is that in an unsupervised TL, the tasks are related.

Why Transfer Learning?

1. Practically, very few people train CNN right from scratch because it is not easy to get a dataset which is large enough. Arranging or building datasets is a major issue. Hence, pre-trained network weights are used as either initializations or fixed-feature extractors. This solves a significant portion of the problems.
2. Very deep NNs are expensive to train. Even the most expensive models can take weeks to get trained using many machines equipped with very expensive GPUs.
3. Determining the flavour/ hyper parameters/ topology/ training method for deep learning is not an easy task.

1.3 FMNIST Dataset

Fashion MNIST dataset was developed by Zalando, an e-commerce company in 2017. FMNIST dataset is very similar to the MNIST dataset which contains images of hand-written numeric digits with the corresponding labels. Both the datasets are same in terms of the structure, i.e., they have the same image and data formats. Also, in both the datasets, the proportion of training is to testing is the same.

FMNIST is a set of 28x28 labeled images of various fashion items. It contains 60,000 training examples and 10,000 test examples. There are 10 classes viz t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. FMNIST was built in order to replace MNIST for benchmarking algorithms.

Here is a glimpse of the dataset in Figure 1.8:
Every three rows of the Figure 1.8 represent a single class, i.e., the first three rows are for t-shirt, the second set of three rows, row number 4 to 6, is for trouser and so on.
The order followed, from top to bottom, is: t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot.

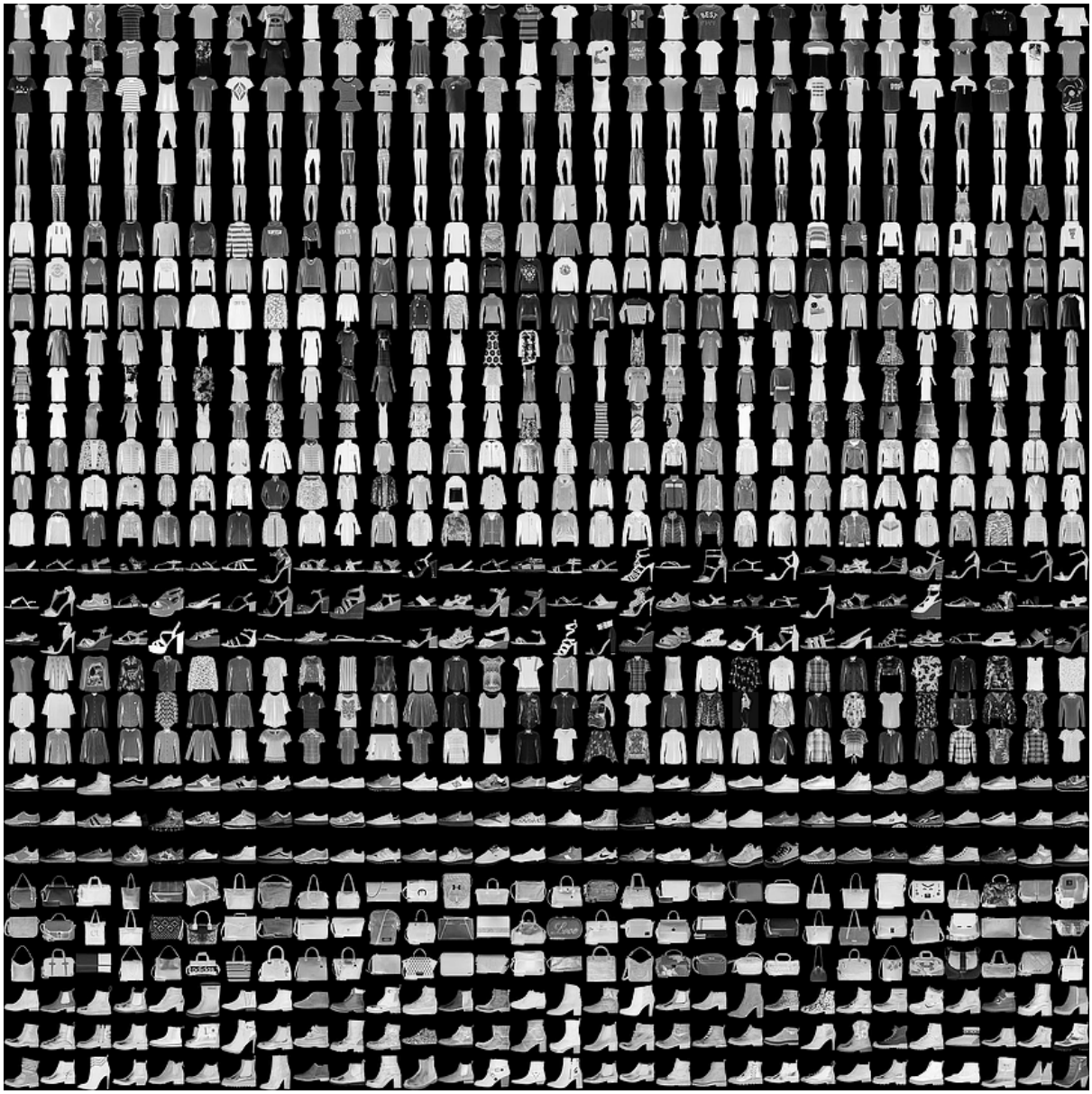

Figure 1.8: FMNIST Dataset (Courtesy: Zalando Research)

1.4 Tools Used

Primarily, two tools have been used for the project: Keras and MATLAB. Keras has been used for conversion of .png files to .csv file. Also, most importantly, all the machine learning processes of training, testing, loading weights of one model to another have been carried out using Keras.

On the other hand, MATLAB has been functional in the pre-processing of those images which have been used to build the dataset for the target task.

1.4.1 Keras

Keras, meaning "horn" in Greek, is a reference to an image from Greek and Latin literature. Keras was initially built as a part of a project work ONEIROS i.e. Open-ended Neuro-Electronic Intelligent Robot Operating System.

Keras is an open source NN library, written in Python. It is a high-level API which can run on top of many other libraries. Few of such libaries are CNTK, Tensorflow or Theano. The primary author of Keras is François Chollet who is a Google engineer. Keras was developed to enable fast experimentation.

Core data structure in Keras is a model which is a way to organize layers. The simplest model is the sequential one, linear stack of layers, which obeys the rule of "Last In First Out". There are many other complex architectures as well.

In this thesis, Keras has been used with Tensorflow as the backend. TensorFlow is an open source ML library. It was developed by the Google Brain team.

In the following figure, Figure 1.9, a screenshot of a programme running on Keras with TensorFlow as the backend, has been shown.

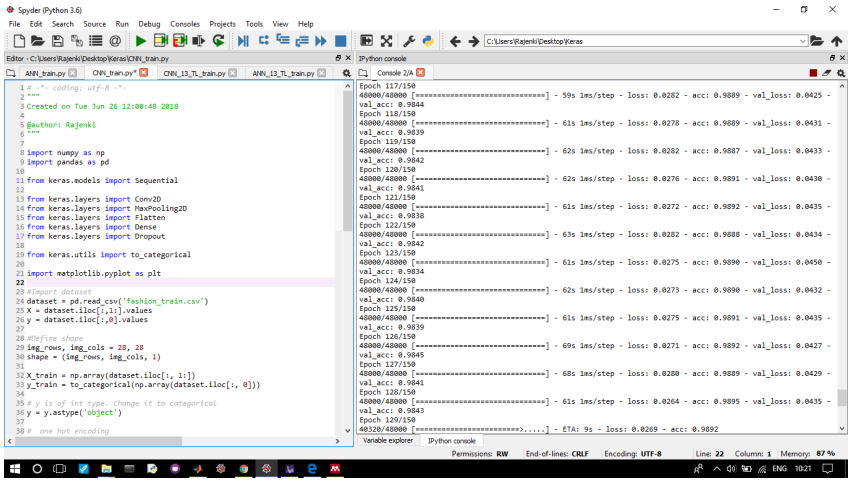

Figure 1.9: Training on Keras

1.4.2 MATLAB

MATLAB, shortened for Matrix Laboratory, combines numerical computing environment with a programming language which simplifies expressing matrices and arrays. One can manipulate matrices, plot functions, implement algorithms on given data, create user interfaces for several other purposes. Developed by MathWorks, an American organisation, MATLAB is used across academia and various industries.

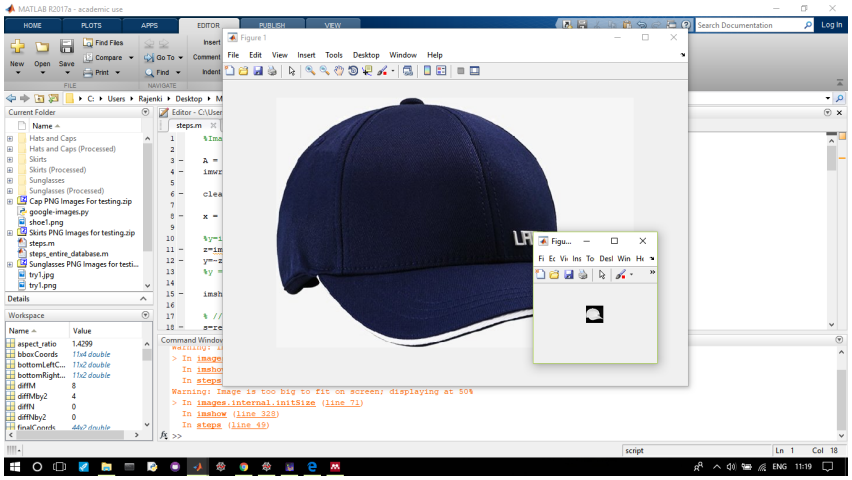

Figure 1.10: Image processing on MATLAB

1.5 Basic Components of the Models Built

Generally while training a model for machine learning purposes, the dataset is split into training and test/validation sets. In many cases, datasets are also split in three parts which includes training once and testing twice. In this case, we often say that a cross-validation takes place. Paramaters that help in analyzing the models:

1. Train accuracy: When training takes place, in every iteration, a subset of the training set is considered by the programme itself and then trained i.e. in case of supervised learning, if labels match with the predictions made during training, the train accuracy increases. This keeps running for the specified number of iterations. So, in short, the training process helps the model to familiarize with the dataset.

2. Test/ Validation accuracy: Simply training is not enough to determine the performance of an algorithm, only when it performs more or less equally well on an data external to the training dataset, we can say that the algorithm is good. So once training is done, the model is tested on the other sub-dataset which is different from the training dataset. This dataset can be termed as test or validation dataset. The test/ validation accuracy can be processed in two ways: 1) during training, after every iteration, the model is passed through the test/ validation set thereby giving a test accuracy for every step or, 2) once the entire training process is complete, the model is re-run on the test/ validation dataset. When the datasets are split into three segments, then generally, testing is done in both the ways.

3. Train loss: As accuracies get calculated, losses are also determined as per the given loss function. Similar to train accuracy, with every step, the train loss is calculated using the train dataset. Every time, the model is not able to correctly predict the label, loss increases.

4. Test/ Validation loss: It is similar to test/ validation accuracy, except that here, the loss is considered depending on the specified loss function.

In a machine learning model, the accuracy and loss are inversely proportional to each other. A model is said to perform well or the algorithm applied is said to be learning well, if the accuracies are high and the losses are minimal. The objective in Machine Learning always remains to minimize the loss and maximize the accuracy.

Specific to this thesis, the following parameters were applied using Keras, while writing the code for the models.

1. Sequential Model: It is a linear stack of neural network layers. The first layer of this model receives the information regarding input shape.

2. Kernel Initializer: It is a parameter for specifying the way in which the initial random weights can be set in the neural network layers of a Keras model.

3. Activation: It can be used in both activation and forward layers. Few of the activation functions available are Softmax, ReLU and Tanh (Hyperbolic tan function).

4. Loss: It is also a compilation component in Keras model. The aim is to optimize the loss function i.e. minimize it. Few of the ptions available to us are "mean_squared_error", "mean_absolute_error", "binary_crossentropy" and "categorical_crossentropy".

5. Optimizer: It is a compilation component in a Keras model. Optimizer means an optimization algorithm for the model. Options available are Stochastic Gradient Descent (SGD), Adam etc.

6. Metrics: This is defined in a Keras model to specify the way of analyzing the model.

7. Epoch: It is the number of iterations over the dataset given.

8. Batch Size: It is the number of training examples considered in an epoch.

9. Shuffle: Whether after every epoch, training data needs to be shuffled or not.

1.6 Chapterwise Summary

- Chapter 2 summarises the literature survey done in the field of transfer learning. Literature mostly consists of published articles, books, tech-reports over the years.
- Chapter 3 states the problem in hand, the procedure followed to collect data and process it. The dataset obtained from this chapter has been used in the next one.
- Chapter 4 presents the models built in order to achieve transfer learning. The architecture of every model is provides. Also, the accuracy and loss graphs are provided for the purpose of comparision.
- Chapter 5 consists of the conclusion and the scope of the problem in future.

Chapter 2

Literature Survey

As stated by Haskell (2000), one keeps transferring the previous learning and experiences for learning a new skill with more efficiency and speed. Similary, transfer learning is basically about transference of knowledge from one model to another. One can train a model in two ways. One is by storing the entire dataset as it is and train it from scratch. Another way could be by extracting the information from one network and using it in another. Both the methods have their own advantages and disadvantages, but in a real life scenario, the second case can significantly cut down both time and cost.

Pratt et al. (1991) addressed the problem of transfer learning from one task to another related task. Their major concern was to improve the neural network learning speed. According to Rumelhart et al. (1985), back propagation neural networks perform competitively. Pratt et al. (1991) were able to speed up back-propagation neural networks on two tasks. They could do so with the help of transfer learning where the weights were taken from smaller neural networks on subtasks. The speeding up of back-propagation was found out to be quite high in magnitude after the transference.

Neural networks gained lot of popularity due to their simplicity and efficiency. To run a simple neural network, one may require only to collect and pre-process training data. However, changes are made for getting good performances. Pratt (1992) showed how the speed of an NN can be improved by transfering information from a

trained model to another. Moreoever, while applying TL, the neural networks were modified using an algorithm called discriminability-based transfer (DBT) for improving the transference more. Ultimately, it was seen that DBT was able to improve the speed of the learning algorithm. The results were better than random initialization.

Although generally large problems are broken down to simpler problems and then solved, this procedure can backfire as argued by Waibel et al. (1989). Caruana (1997) took up the problem of training a task with several related sub tasks. In such case of learning, tasks are being trained simultaneously. This is called as Multi Task Learning (MTL). MTL has good potential in real domains. It is an inductive transfer which tries to boost the generalization performance. In an MTL model, the tasks are run in parallel on a single learner with a common representation. But the single learned model is not used in predicting the multiple tasks. This is so because the idea is that while learning multiple tasks simultaneously, the information gained by one task can be beneficial to the co-tasks.
With this discussion, it may look that applying MTL is tougher than applying a simple TL ,i.e., a sequential learning, but as mentioned by Caruana (1997), MTL becomes relatively easy when the tasks are serial. On a sequential transfer, parallel transfer can be applied easily but the vice versa may not be true. However, as per need, both types of transfer can be used together.

There can be problem when one has plenty of labeled data and the target task is related but different. In such a case, transfer learning can be applied even if the features or distributions in the domains (souce and target) are not same. Pan et al. (2008) demonstrated the use of a latent space. They found out that the latent feature space which is low-dimensional can bridge the gap between the source and target domains while transferring knowledge.

Pan et al. (2010) reviewed the progress on TL for problems related to supervised learning (classification, regression) and unsupervised learning (clustering). They proposed TL is classified into: inductive TL, transductive TL and unsupervised TL. Most works

focus on the first two settings, but unsupervised TL has the potential to attract more attention in the coming years.

On the basis of "what to transfer", TL techniques can be categorised into four contexts: instance-transfer, feature-representation-transfer, parametre-transfer and relational-kn owledge-transfer. These approaches have assumed that the selected source domain must be related to the target domain.

Pan et al. (2010) pointed out some relevant issues related to transfer learning. First of all, how to avoid negative transfer learning, i.e., the performance falls when TL is applied. As mentioned by Torrey and Shavlik (2010), the parameters that determine the performance of transfer learning are how much the source and the target tasks are related and how much advantage of this relationship can the transfer method take. If any of these factors falls short, then the performance of the transfer method may drop thereby leading to negative transfer learning. Most of the TL algorithms assume the source and the target domains to be related to each other. When the assumption fails, it leads to a negative transfer.

Secondly, there is a lot of focus on improving the generalization across source and target domains with different distributions. To attain this, the source and the target domains are assumed to have same feature space. But this assumption does not remain true all the time. There are cases when the feature spaces may vary, hence the heterogeneous transfer problem becomes another research concern.

Last but not the least, TL approaches have been mostly applied to small scale problems with limited variety. In future, the horizon of the applications should be expanded.

Till now, most of the discussions focussed on transfered learning in a supervised learning scenario. Bengio (2012) assessed the case where unsupervised learning can aid transfer learning. Representations from unsupervised learning can be used for performing another task. These representations can be learnt in two ways: one, by complete unsupervised leaning or by, taking help of labels from other tasks. Such a setup is known as self-taught learning which is also a type of transfer learning. Several levels of representations were learnt in order to address this problem.

Till now, the major discussion has been on transferring features or other information across models. Now, the transferability of features is assessed within a network, i.e., from a base neural network layer to a final layer. There are deep neural networks where the features of the first layer are specific to the particular task or dataset. Also, there are other deep NNs where the features are general to many tasks and datasets. To achieve the target task, the general features must become specific as they progress through the layers. Yosinski et al. (2014) talked about the issues that can lead to negative transferability within the network: 1) specializing the higher layer neurons to their original task at the cost of their performance on the target task. 2) the optimization challenges that arise due to the splitting of networks between co-adapted neurons. It was observed that the transferability declines when the the base and the target tasks become more distant, but such transferability can be better than using random features for learning.
They made another remark that if a network is initialized with transferred features from any number of layers, it has the ability to enhance the generalization which is long-lasting, i.e., the effect remains even if fine-tuning is done on the new task.

According to Liu et al. (2010), in transfer learning a lot of research has been done on low-rank constraint since only the apt knowledge gets shared between the local spaces in the source and target domains. Ding et al. (2015) devised a new method for transfer learning which is called as Deep Low-Rank Coding (DLRC).
DLRC works with the main idea of learning the deep neural network structure of feature representation and transfering knowledge. This is done via an iterative structured low-rank constraint that tries to deal the case where the source and target domain layers don't match. Such is taken care of layer by layer.
In addition to the above, a marginal denoising regularizer is employed that guides the learned single-layer low-rank approach by seeking an improved transformation matrix.
Finally, many of the single layer row rank codings are stacked so that robust cross-domain features are learnt with more bias to the target domain. Ding et al. (2015) experimented the specified technique on several benchmarks and the results verified

the efficacy of their proposed algorithm.

A large number of training is based on supervised learning which includes labelling of data. Shi et al. (2017) proposed to solve the problem with the help of weakly supervised approach which helps in automatated annotation on the basis of binary (weak) labels that indicate if the object is contained in a training image or not. Such a method has its own challenges. Hence Shi et al. (2017) proposed to address this problem with a new perspective. This problem was considered as a transfer learning problem instead. They used transfer learning based on the ability to learn to rank. Here, transference for automatic annotation of object location occurs from an auxiliary dataset to a dataset with unrelated object categories. They were able to show how their method outperformed the other existing approaches that aimed to annote objects in VOC dataset.

With the advancement of Deep CNNs for image classification, Nguyen et al. (2018) proposed a new deep NN architecture based on transfer learning. This novel architecture was used for microscopic image classification. In this network, they concatenated the features which were extracted from three pre-trained deep convolutional neural networks. Once, the concatenation was done, the features were used in training two fully-connected layers for classification purpose. When they experimented on datasets like 2D-Hela (Boland and Murphy (2001)) and PAP-smear datasets (Jantzen et al. (2005)) which are used for medical purposes, the new proposed structure demonstrated huge gains in the performance compared to other NN architectures which only take advantage of features from single CNN and other classification techniques i.e. where the features are not concatenated.

Chapter 3

Problem Statement, Data Collection and Preparation

3.1 Problem Statement

The target task is classifying new three classes by using the knowledge of classification of the ten objects of FMNIST dataset.
Here, FMNIST is the first dataset taken. To this 3 more classes are added:
Class Label 10: Hats and Caps
Class Label 11: Skirts
Class Label 12: Sunglasses

This new dataset has been bifurcated into two parts: train and test in 80:20 ratio. The training set has 120 images for each class, i.e., a total of 360 images. The testing set has 30 images for each class, i.e., a total of 90 images.

3.2 Data Collection

3.2.1 Pre-trained Dataset

FMNIST dataset was downloaded from `https://www.kaggle.com/zalando-research/fashionmnist` in the form of .csv files of training and testing data.

Number of columns in each of the files is 785, where the first column represents the label number and the remaining 784 (=28*28) columns denote the pixel values. First row in each file consists of the headings given to the columns. There are total number of 60,001 rows in training and 10,001 in testing files.

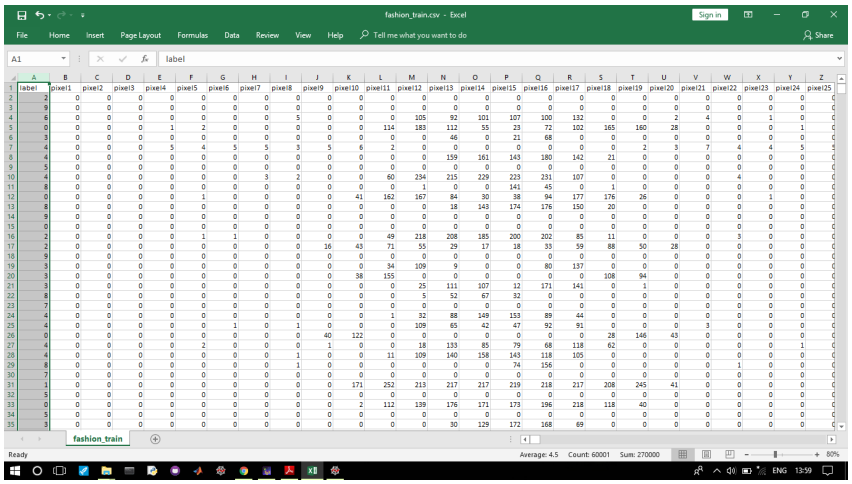

Figure 3.1: FMNIST train .csv file

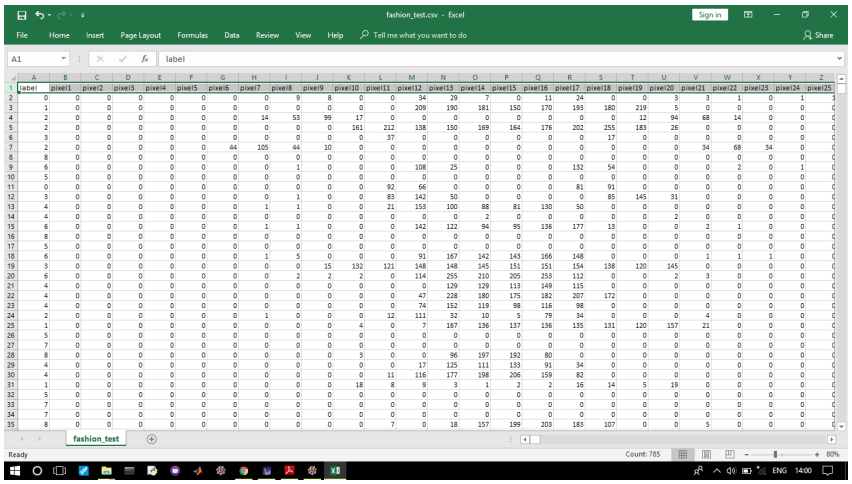

Figure 3.2: FMNIST test .csv file

This dataset has been trained to make it the pre-trained dataset for the target task.

Format of the files:

1) Each row represents a separate image.

2) The first column represents the class label:

- 0: T-Shirt/ Top
- 1: Trouser
- 2: Pullover
- 3: Dress
- 4: Coat
- 5: Sandal
- 6: Shirt
- 7: Sneaker
- 8: Bag
- 9: Ankle boot

3) The rest of the columns (column no: 2 to 785) represent pixel numbers.
4) Each pixel value is from 1 to 255.

3.2.2 New Dataset

To build a new dataset for transfer learning, images were extracted from `https://www.zalando.co.uk`. 150 images were downloaded for each class. The dataset was intentionally kept small since the objective was to load weights from another model and see the performance of transfer learning. To match the format of the images in the pre-trained dataset, white colour images were not downloaded since the contrast is low with respect to the background.

One sample picture for each class of the original, unprocessed images is given below in Figure 3.3.

Figure 3.3: Hat/Cap, Skirt, Sunglass (From Left to Right)

3.3 Pre-processing of Images

The images in FMNIST have a particular format. Here, to achieve transfer learning, it is important that the new dataset should also have the same format as that of FMNIST dataset. As per Xiao et al. (2017), the following steps were executed for developing FMNIST dataset. So, in order to replicate the image formatting, the same steps were executed one by one using MATLAB. There are 7 specific steps as follows:

1. Convert the downloaded images to .png format.
2. Trim edges which are close to colour of the corner pixels.
3. Resize longest edge to 28.
4. Sharpen the pixels using Gaussian operator of radius and standard deviation of 1.
5. Extend shortest edge to 28. Put the image to centre.
6. Negate intensities of the image.
7. Convert to 8-bit grayscale pixels.

Samples of the processed images for each class are shown in the following figures: (Figure: 3.2, 3.3 and 3.4)

Figure 3.4: Class lable 10 (Processed)

Figure 3.5: Class label 11 (Processed)

Figure 3.6: Class label 12 (Processed)

In the sample shown, every image is processed to be of size 28x28 pixels.

Once all the images were processed, they were converted to three .csv files, one for each class, using Keras. Later, all these three files were merged and training and testing datasets were creating in 80:20 ratio.
Number of columns in each of the files is 785 again. There are total number of 361 rows in training and 91 in testing files.

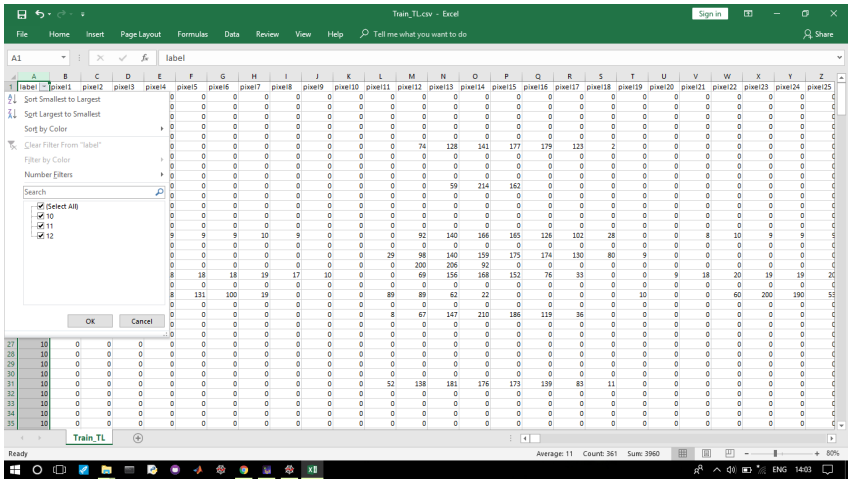

Figure 3.7: Three new classes .csv file (train)

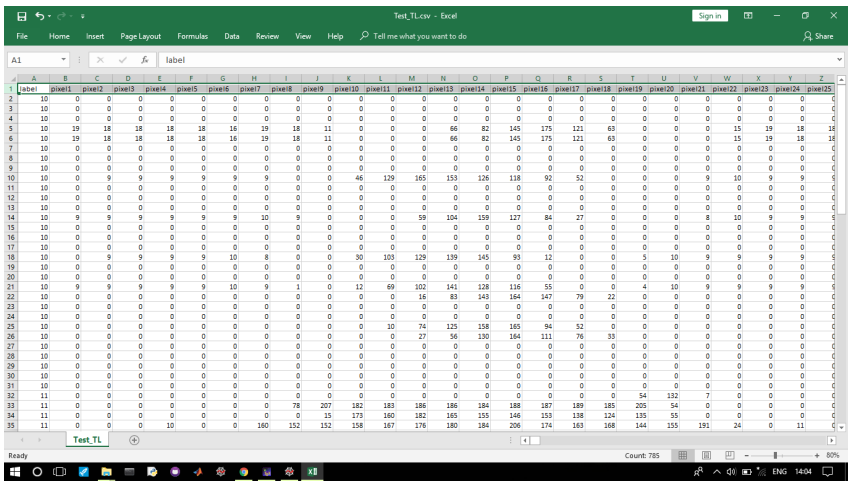

Figure 3.8: Three new classes .csv file (test)

3.4 Chapter Summary

In this chapter, first the target task was described. Next, the content and structure of the FMNIST dataset was explained. As a part of the data collection, data was collected by downloading images from Zalando's website. The new dataset contains images belonging to three classes. After data collection, data preparation followed. In order to use the new dataset further, the following steps were carried out:

1. All the images were processed to match the image format of FMNIST. It was done in 7 steps with the help of MATLAB.
2. The images were put together and converted to a .csv file using Keras.

Chapter 4

Methodology

In this thesis, transfer learning has been attained by mainly two steps:
1) Build a model on the parent/ source dataset, which is with 10 classes.
2) Use the weights of the parent model on the new task of classifying 3 new classes.

A gist of all the number and type of models together is shown below. There are 5 models for each ANN (ANN in this thesis has been used for a conventional Artificial Neural Network) and CNN

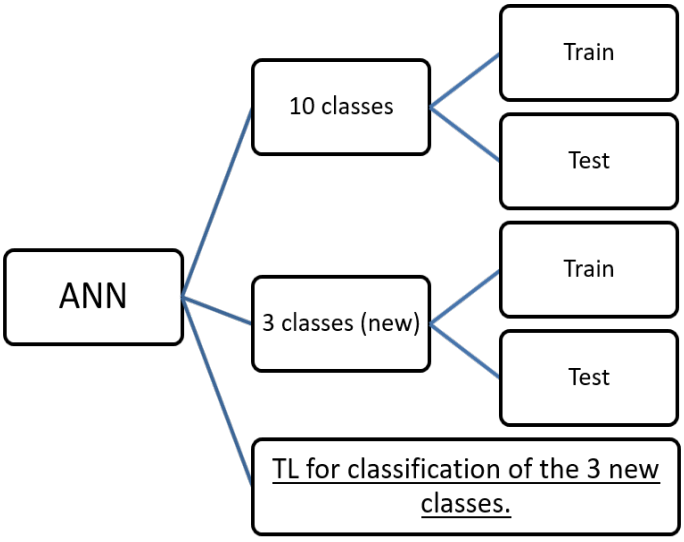

Figure 4.1: Models built (ANN)

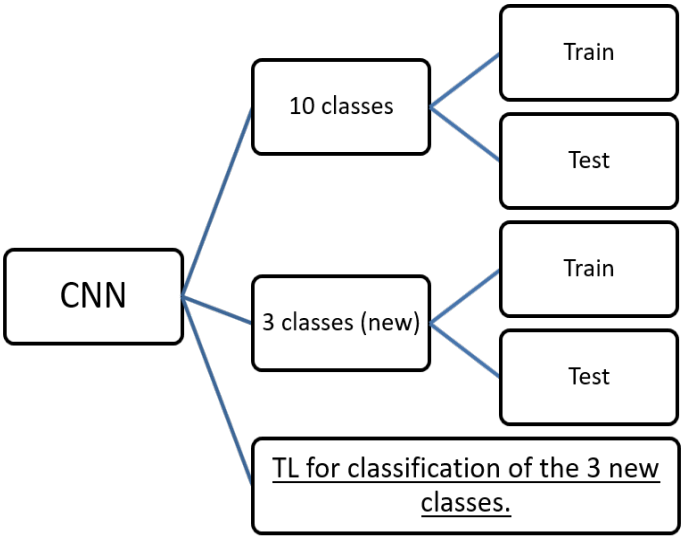

Figure 4.2: Models built (CNN)

In every model built, the parameteres are more or less the same. The following are kept fixed throughout the models:

1. Train: Test is 80: 20 ratio.
2. Model used is Sequential.
3. Kernel Initializer: It is 'uniform' in ANN models.
4. Activation: Expect for the last layer, all other layers use ReLU. In the last layer, softmax functIon is used.
5. Loss: It is specified to be 'categorical_crossentropy'.
6. Optimizer: In every model, 'adam' is kept as the optimizer.
7. Number of epochs: 150.
8. Batch Size: 128.
9. Regularization: In CNNs, a DropOut layer has been added.
10. Shuffle: True

The models are analyzed on the basis of:
Train Accuracy, Test/ Validation Accuraacy, Train Loss, Test/ Validation Loss.

The architecture of every model is defined by:
Type of Layer, Output Shape and Number of Parameters.

4.1 Models using ANN

4.1.1 10 Classes

The training architecture of the model is as follows:

Layer (type)	Output Shape	Number of Parameters
dense_33 (Dense)	(None, 256)	200960
dense_34 (Dense)	(None, 128)	32896
dense_35 (Dense)	(None, 10)	1290

Total number of parameters: 235,146

Trainable paramaters: 235,146

Non-trainable parameters: 0

Loss and accuracy during the training are given below:

Number of Epochs	Loss (%)	Accuracy (Ratio)
10	0.2400	0.9112
50	0.0656	0.9743
100	0.0343	0.9876
150	0.0202	0.9930

Test accuracy: 90.01 %.

4.1.2 3 Classes (New)

The training architecure of the model is as follows:

Layer (type)	Output Shape	Number of Parameters
dense_50 (Dense)	(None, 256)	200960
dense_51 (Dense)	(None, 128)	32896
dense_52 (Dense)	(None, 13)	1677

Total number of parameters: 235,533

Trainable paramaters: 235,533

Non-trainable parameters: 0

Loss and accuracy during the training are given below:

Number of Epochs	Loss (%)	Accuracy (Ratio)
10	0.1778	0.9444
50	0.0034	1.0000
100	6.2937e-04	1.0000
150	2.6214e-05	1.0000

Test accuracy is: 100.00 %.

4.1.3 TL for Classification of the New Classes

Architecture of the transfer learning model looks like the following. Except for the last ones, all other layers of the 10 class training (source) and transfer learning models are same. :

Layer (type)	Output Shape	Number of Parameters
dense_50 (Dense)	(None, 256)	200960
dense_51 (Dense)	(None, 128)	32896
dense_55 (Dense)	(None, 13)	182

Total number of parameters: 234,038

Trainable paramaters: 234,038

Non-trainable parameters: 0

Both training and testing accuracies are compared.

Number of Epochs	Train Accuracy (Ratio)	Test Accuracy (Ratio)
10	0.9222	0.9000
50	0.9917	0.9556
100	0.9917	0.9556
150	0.9944	0.9556

In Figure 4.3, both training and validation accuracies are plotted for the ANN model in which TL is applied.

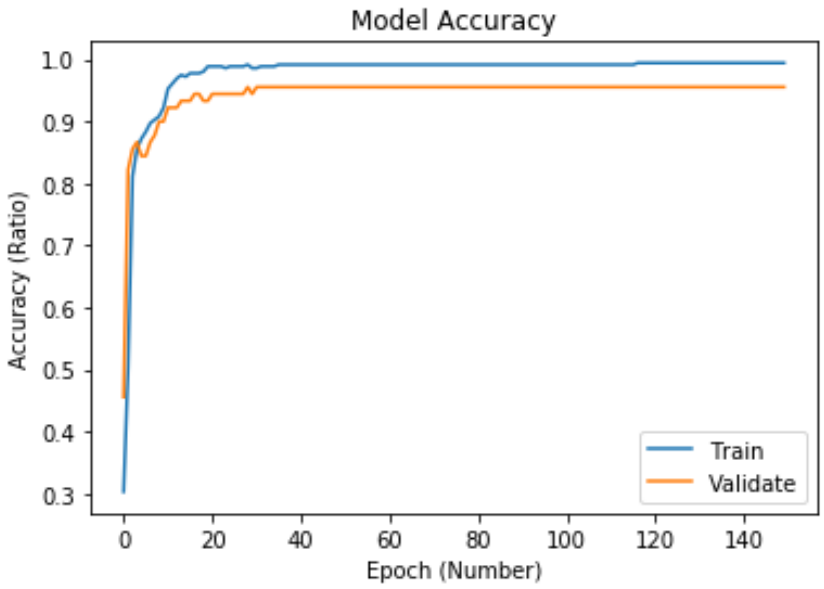

Figure 4.3: Transfer learning accuracy using ANN

In the following, both training and testing losses are compared.

Number of Epochs	Train Loss (%)	Test Loss (Ratio)
10	2.4433	2.4395
50	2.0428	2.0491
100	1.6219	1.6398
150	1.2752	1.3047

In Figure 4.4, both training and validation accuracies are plotted for the ANN model in which TL is applied.

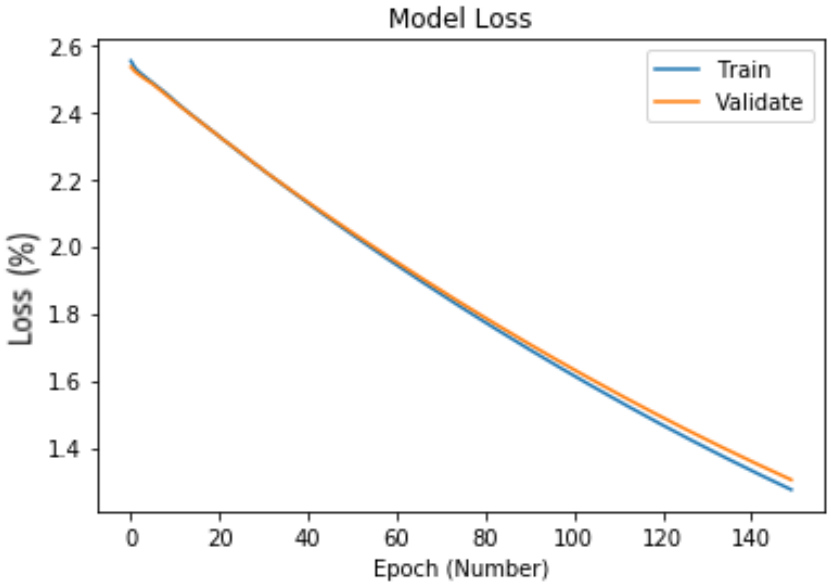

Figure 4.4: Transfer learning loss using ANN

4.2 Models using CNN

4.2.1 10 Classes

The training architecure of the model is as follows:

Layer (type)	Output Shape	Number of Parameters
conv2d_9 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2D_9 (MaxPooling2D)	(None, 13, 13, 32)	0
dropout_9 (Dropout)	(None, 13, 13, 32)	0
conv2D_10 (Conv2D)	(None, 11, 11, 32)	9248
max_pooling2D_10 (MaxPooling2D)	(None, 5, 5, 32)	0
dropout_10 (Dropout)	(None, 5, 5, 32)	0
flatten_5 (Flatten)	(None, 800)	0
dense_45 (Dense)	(None, 128)	102528
dense_46 (Dense)	(None, 10)	1290

Total number of parameters: 113,386

Trainable paramaters: 113,386

Non-trainable parameters: 0

Loss and accuracy during the training are given below:

Number of Epochs	Loss (%)	Accuracy (Ratio)
10	0.2823	0.8960
50	0.1703	0.9349
100	0.1397	0.9469
150	0.1285	9508

Test accuracy is: 92.76 %

4.2.2 3 Classes (New)

The training architecure of the model is as follows:

Layer (type)	Output Shape	Number of Parameters
conv2d_11 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2D_11 (MaxPooling2D)	(None, 13, 13, 32)	0
dropout_11 (Dropout)	(None, 13, 13, 32)	0
conv2D_12 (Conv2D)	(None, 11, 11, 32)	9248
max_pooling2D_12 (MaxPooling2D)	(None, 5, 5, 32)	0
dropout_12 (Dropout)	(None, 5, 5, 32)	0
flatten_6 (Flatten)	(None, 800)	0
dense_53 (Dense)	(None, 128)	102528
dense_54 (Dense)	(None, 13)	1677

Total number of parameters: 113,773

Trainable paramaters: 113,773

Non-trainable parameters: 0

Loss and accuracy during the training are given below:

Number of Epochs	Loss (%)	Accuracy (Ratio)
10	0.2992	0.8806
50	0.0473	0.9833
100	0.0170	0.9972
150	0.0129	0.9944

Test accuracy is: 100.00 %.

4.2.3 TL for Classification of the New Classes

Architecture of the transfer learning model looks like the following. Except for the last ones, all other layers of the 10 class training (parent) and transfer learning models are same:

Layer (type)	Output Shape	Number of Parameters
conv2d_9 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2D_9 (MaxPooling2D)	(None, 13, 13, 32)	0
dropout_9 (Dropout)	(None, 13, 13, 32)	0
conv2D_10 (Conv2D)	(None, 11, 11, 32)	9248
max_pooling2D_10 (MaxPooling2D)	(None, 5, 5, 32)	0
dropout_10 (Dropout)	(None, 5, 5, 32)	0
flatten_5 (Flatten)	(None, 800)	0
dense_45 (Dense)	(None, 128)	102528
dense_46 (Dense)	(None, 13)	143

Total number of parameters: 112,239

Trainable paramaters: 112,239

Non-trainable parameters: 0

Both training and testing accuracies are compared.

Number of Epochs	Train (Ratio)	Test (Ratio)
10	0.3611	0.3333
50	0.9889	0.9889
100	1.0000	0.1.0000
150	1.0000	1.0000

In Figure 4.5, both training and validation accuracies are plotted for the CNN model in which TL is applied.

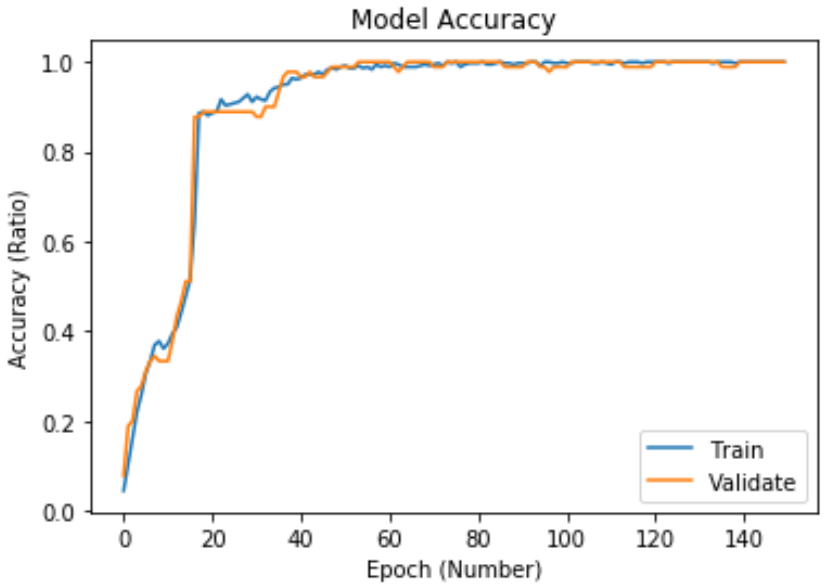

Figure 4.5: Transfer learning accuracy using CNN

After plotting accuracies of the CNN model with TL, both training and testing losses are also compared.

Number of Epochs	Train (%)	Test (%)
10	2.2329	2.2082
50	1.7480	1.7568
100	1.3167	1.3354
150	1.0034	1.0301

In Figure 4.6, both training and validation losses are plotted for the CNN model in which TL is applied.

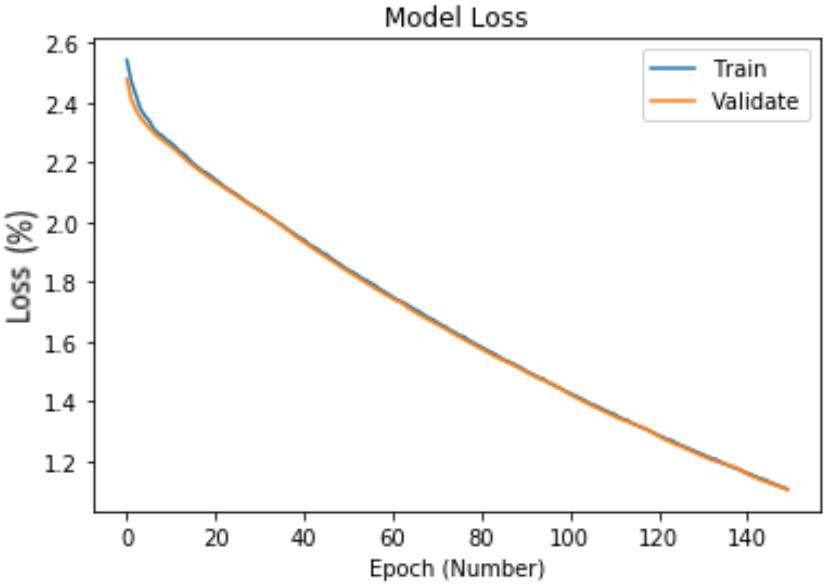

Figure 4.6: Transfer learning loss using CNN

4.3 Comparision of Transfer Learning between ANN and CNN

4.3.1 Training

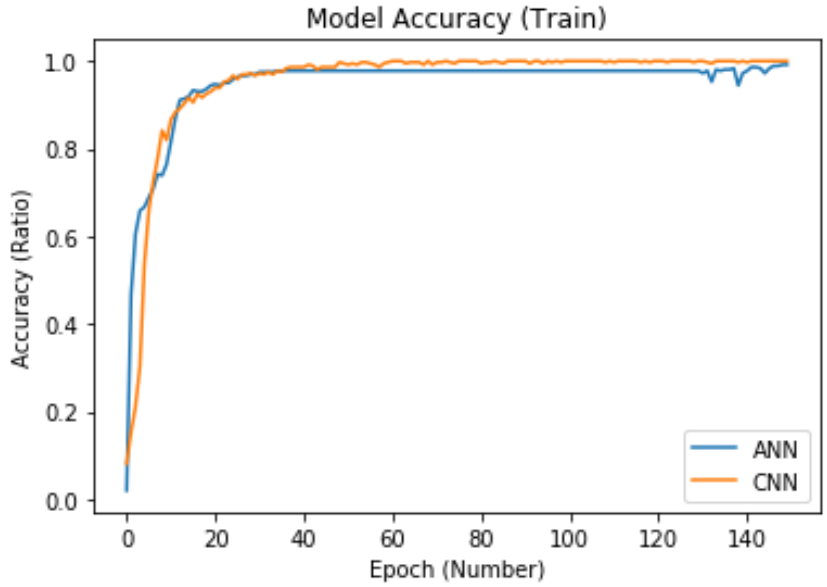

Figure 4.7: ANN vs CNN train accuracy

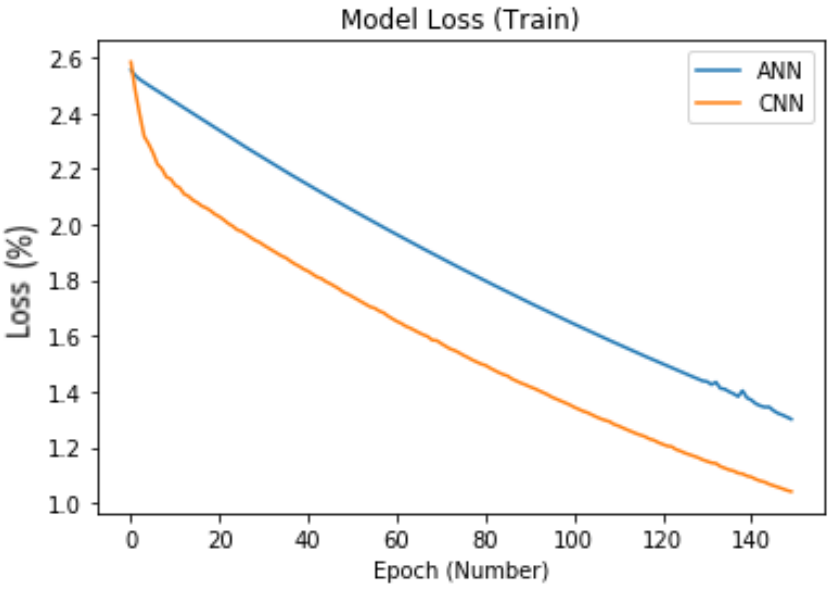

Figure 4.8: ANN vs CNN train loss

4.3.2 Testing

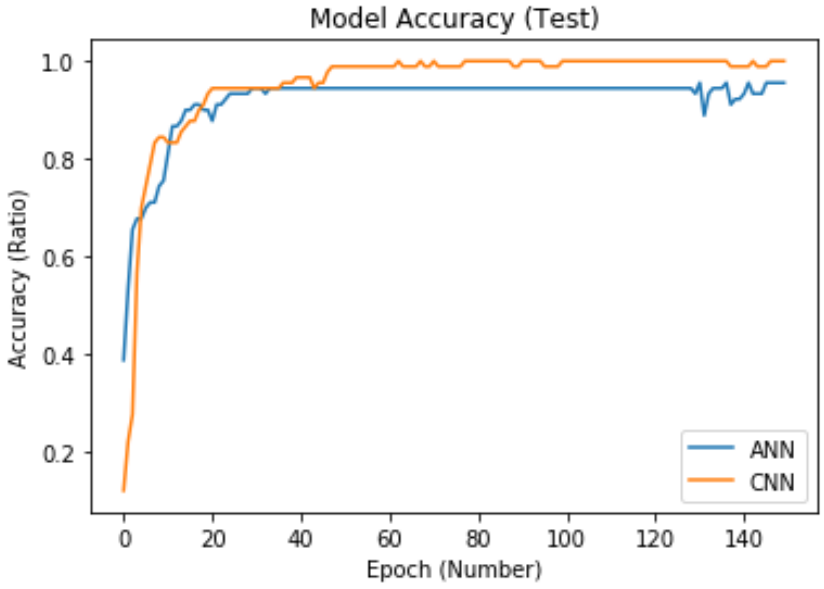

Figure 4.9: ANN vs CNN test accuracy

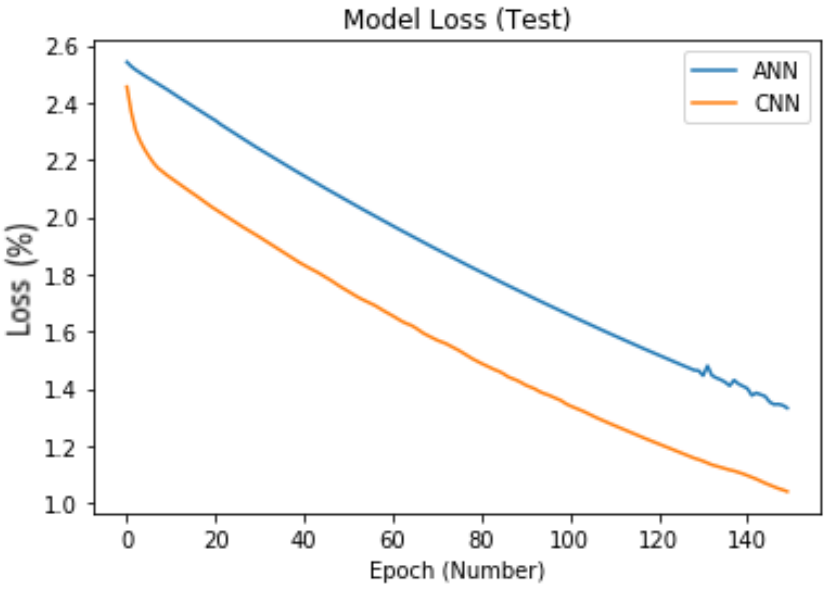

Figure 4.10: ANN vs CNN test loss

4.4 Pre and Post Transfer Learning

The required models have been built as described in the sections preceding this. Now, it is important to compare the performances of the models. The target task is classifying the three new classes viz hat/cap, skirt, sunglass. Now, we compare the models when transfer learning is applied and when transfer learning is not applied. When TL is applied, weights of the FMNIST 10 class model are loaded. When TL is not applied, the model is built from scratch where every layer is distinctly defined. Models were trained using ANN and CNN. We analyze the performance for both ANN and CNN pre and post TL in the subsequent subsections.

In the following, we check what happens to the model accuracy and loss for both training and test with and without transfer learning.

4.4.1 ANN

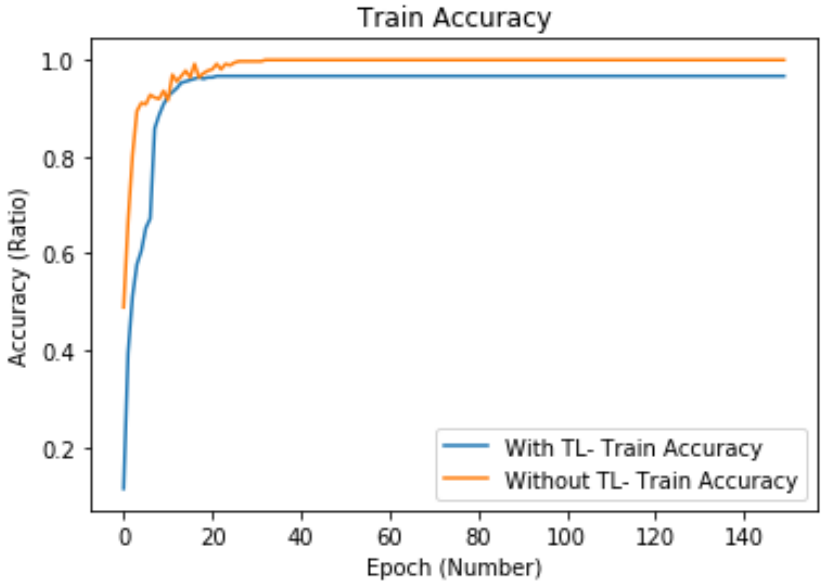

Figure 4.11: With vs without TL (Train Accuracy (ANN))

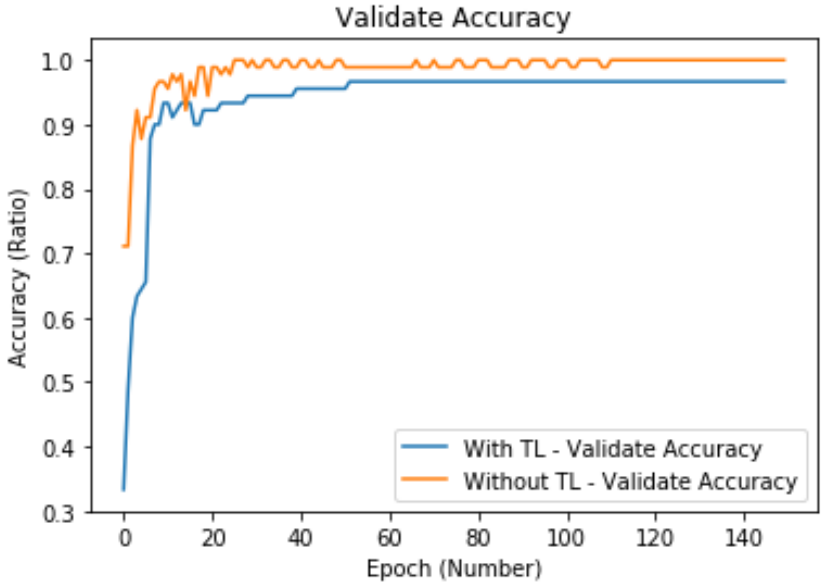

Figure 4.12: With vs without TL (Test Accuracy (ANN))

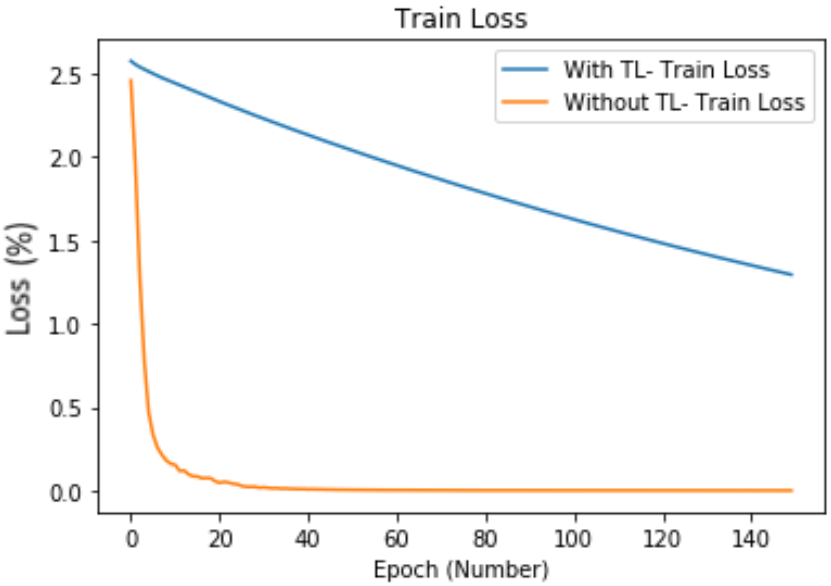

Figure 4.13: With vs without TL (Train Loss (ANN))

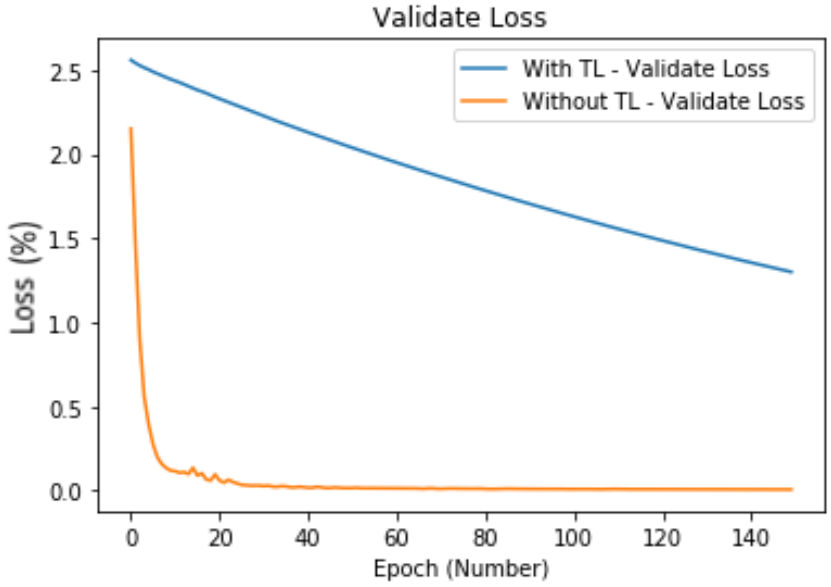

Figure 4.14: With vs without TL (Test Loss (ANN))

4.4.2 CNN

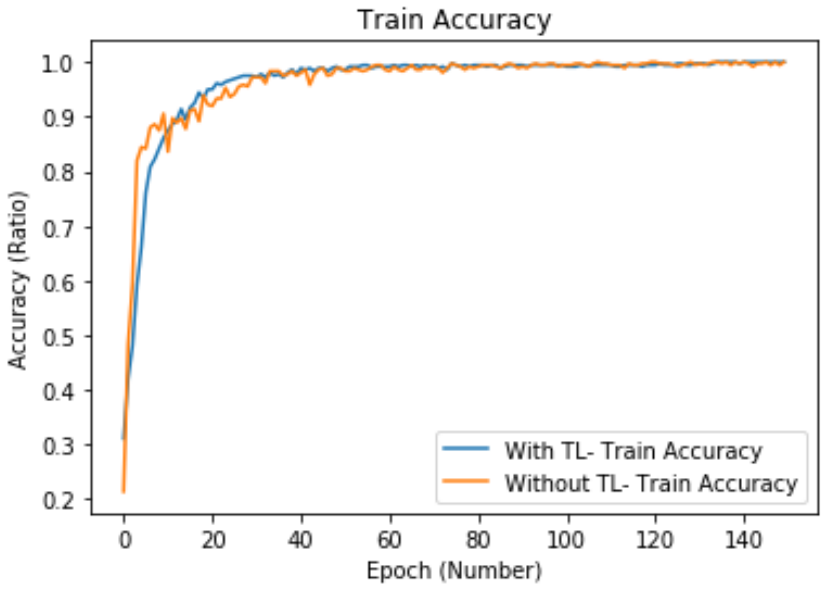

Figure 4.15: With vs without TL (Train Accuracy (CNN))

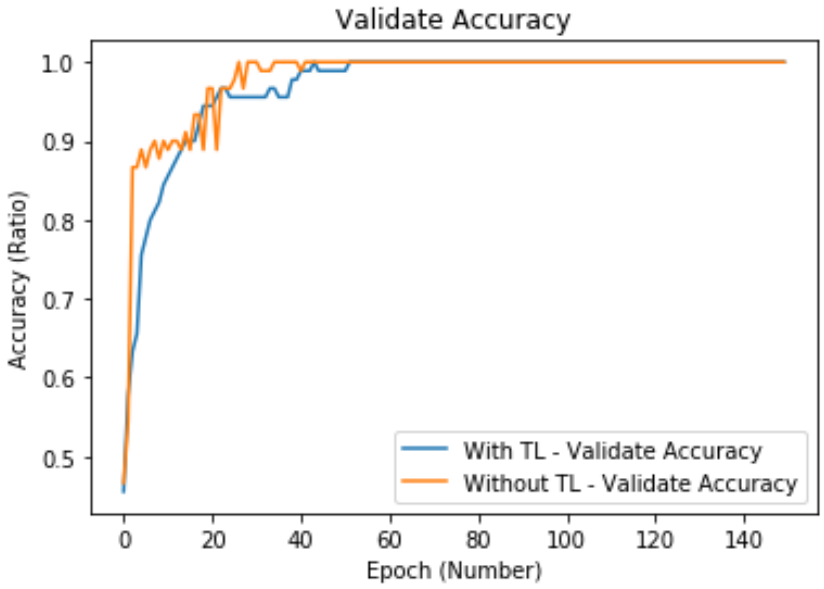

Figure 4.16: With vs without TL (Test Accuracy (CNN))

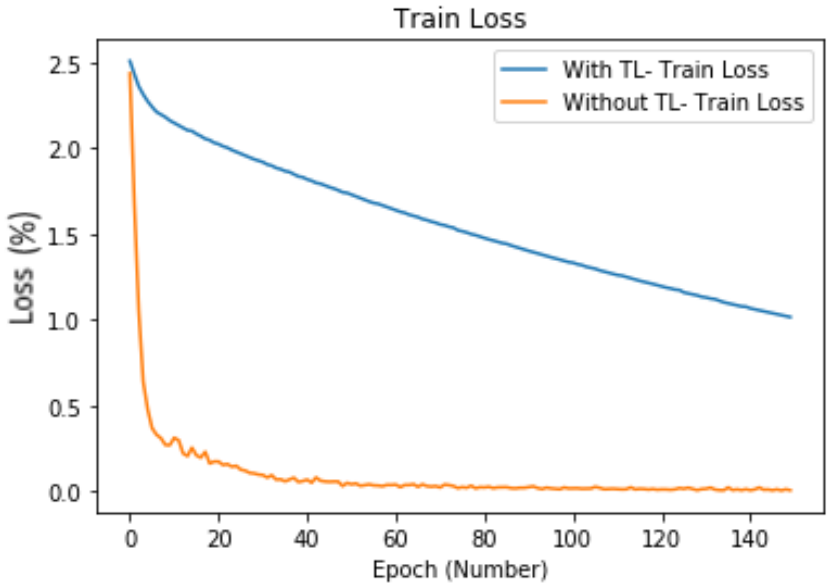

Figure 4.17: With vs without TL (Train Loss (CNN))

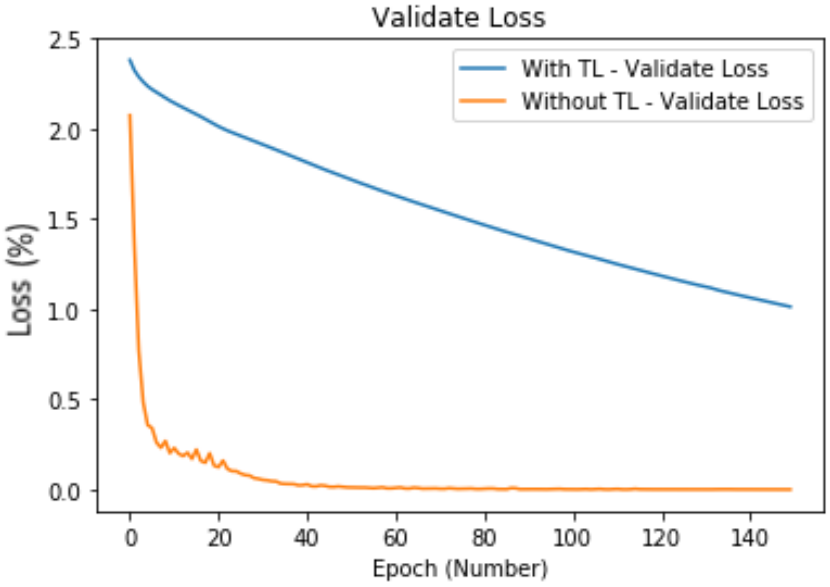

Figure 4.18: With vs without TL (Test Loss (CNN))

Chapter Summary

Two datasets were used, one, the FMNIST dataset which acted as the source and the second one was built with three new classes of fashion items, as mentioned in the previous chapter, i.e., chapter number 3.

Total 10 programmes were run for training and testing on ANN with TL, ANN without TL, CNN with TL, CNN without TL. The architecture for every model was described in terms of the type of layer, shape of the output and the number of parameters.

In order to make comparisions easy, the accuracy and loss of every model with the new dataset were plotted. Conclusions drawn from this chapter have been briefly put together in the following chapter, i.e., chapter number 5.

Chapter 5

Conclusion and Future Scope

5.1 Conclusion

For the given case taken, the concluding remarks include:

1. Training of the original 10 classes dataset works much better for the Convolutional Neural Network model (CNN) than the basic Artificial Neural Network (ANN) model.

2. When the model is trained, CNN shows better training accuracy than ANN, after 150 epochs. But both ANN and CNN have the same test accuracy.

3. For both ANN and CNN, the models have better training as well as validation accuracies when transfer learning is not applied. Also, for both ANN and CNN, the losses are less when transfer learning is not applied. They seem to go constant after around 20 epochs. In the case where transfer learning is applied, the losses decline steadily as inferred from the negative slopes.

Although, negative transfer learning seems to take place for both ANN and CNN, but the differences in accuracies and losses are not much significant. Hence, in dire cases when the datasets are small, transfer learning is an approach that can be helpful.

5.2 Future Scope

The following can be explored, which have not been covered in the thesis:

1. Optimization of transfer learning. Optimization can be done with respect to loss, accuracy, time taken etc.

2. More neural networks can be built for transfer learning and comparisions can be done to check which NN gives the best TL result.

3. More transfer learning methods can be applied like freezing only particular layers or feature extraction.

4. Cross neural network transfer learning can be done, i.e., using weights of one type of NN to another type.

5. Performance of transfer learning can be checked by using the weights on a lesser related target task.

6. Targets can be merged. For example, in this thesis, the target was to classify the new classes, but transfer learning can be applied to classify not only the new classes but also the old ones in the very same model.

Appendix

Pre-processing of images belonging to the new classes

```
A = imread('a.jpg');
imwrite(A,'a.png');

clear;

x = imread('a.png'); %step1

%y=im2bw(x);
z=im2bw(x,graythresh(x));
y=~z;
%y = y(1:end-10,:);

imshow(y);

% //Calculate all bounding boxes
s=regionprops(y, 'BoundingBox');

%// Obtain all of the bounding box co-ordinates
bboxCoords = reshape([s.BoundingBox], 4, []).';

% // Calculate top left corner
topLeftCoords = bboxCoords(:,1:2);

% // Calculate top right corner
topRightCoords = [topLeftCoords(:,1) + bboxCoords(:,3) topLeftCoords(:,2)];

% // Calculate bottom left corner
```

```
28 bottomLeftCoords = [topLeftCoords(:,1) topLeftCoords(:,2) + bboxCoords(:,4)
       ];
29
30 % // Calculate bottom right corner
31 bottomRightCoords = [topLeftCoords(:,1) + bboxCoords(:,3) ...
32      topLeftCoords(:,2) + bboxCoords(:,4)];
33
34 % // Calculating the minimum and maximum X and Y values
35 finalCoords = [topLeftCoords; topRightCoords; bottomLeftCoords;
       bottomRightCoords];
36 minX = min(finalCoords(:,1));
37 maxX = max(finalCoords(:,1));
38 minY = min(finalCoords(:,2));
39 maxY = max(finalCoords(:,2));
40
41 % Draw the rectangle on the screen
42 width = (maxX - minX + 1);
43 height = (maxY - minY + 1);
44 rect = [minX minY width height];
45
46 % // Show the image
47 imshow(x);
48 hold on;
49 rectangle('Position', rect, 'EdgeColor', 'yellow');
50
51 X1=floor(minX);
52 Y1=floor(minY);
53 X2=ceil(maxX);
54 Y2=ceil(maxY);
55
56 x_new=x(Y1:Y2,X1:X2,:);
57
58 close all;
59
60 imshow(x_new);
61
62 [M_new, N_new, P_new]=size(x_new);
```

```
63
64 aspect_ratio=N_new/M_new;
65
66 if(M_new<N_new)
67     N_reduced=28;
68     M_reduced=round(N_reduced/aspect_ratio);
69 else
70     M_reduced=28;
71     N_reduced=round(M_reduced*aspect_ratio);
72 end
73
74 x_new_resized=imresize(x_new,[M_reduced,N_reduced],'bicubic','Antialiasing'
       ,true);
75 x_new_resized_double=double(x_new_resized);
76
77 figure, imshow(x_new_resized);
78
79 H=fspecial('gaussian',3,1.0);
80 x_blur=imfilter(x_new_resized_double,H,'symmetric','same','conv');
81 x_edge_map_double=x_new_resized_double-x_blur;
82 x_sharpen_double=x_new_resized_double+x_edge_map_double;
83 x_sharpen=uint8(x_sharpen_double);
84
85 figure, imshow(x_sharpen);
86
87 [M_new_sharpen, N_new_sharpen, P_new_sharpen]=size(x_sharpen);
88
89 if(M_new_sharpen==28)
90     diffN=28-N_new_sharpen;
91     diffM=0;
92 elseif(N_new_sharpen==28)
93     diffM=28-M_new_sharpen;
94     diffN=0;
95 end
96
97 diffMby2=floor(diffM/2);
98 diffNby2=floor(diffN/2);
```

```
99
100 LBM=diffMby2+1;
101 UBM=LBM+M_new_sharpen-1;
102
103 LBN=diffNby2+1;
104 UBN=LBN+N_new_sharpen-1;
105
106
107 x_sharpen_28_double=255*ones(28,28,3);
108 x_sharpen_28_double(LBM:UBM,LBN:UBN,:)=double(x_sharpen);
109 x_sharpen_28=uint8(x_sharpen_28_double);
110
111 figure, imshow(x_sharpen_28);
112
113 x_negated=imcomplement(x_sharpen_28);
114 x_final=rgb2gray(x_negated);
115
116 figure, imshow(x_final);
```

Training an ANN for 10 classes of FMNIST

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jun 26 11:28:42 2018
4
5  @author: Rajenki
6  """
7
8  # Fashion Mnist Model file
9  # Fashion Mnist with Keras
10 import numpy as np
11 import matplotlib.pyplot as plt
12 import pandas as pd
13 from keras.utils import to_categorical
14
15 # Importing the dataset
16 dataset = pd.read_csv('fashion_train.csv')
17 X = dataset.iloc[:,1:].values
```

```
18 y = dataset.iloc[:,0].values
19
20 img_rows, img_cols = 28, 28
21 shape = (img_rows, img_cols, 1)
22
23 X_train = np.array(dataset.iloc[:, 1:])
24 y_train = to_categorical(np.array(dataset.iloc[:, 0]))
25
26 # y is of int type. Change it to categorical
27 y = y.astype('object')
28
29 from keras.utils import np_utils
30 y = np_utils.to_categorical(y)
31
32 X_train = X_train.astype('float32')
33 X_train /= 255
34
35 # Import Keras libraries
36 from keras.models import Sequential
37 from keras.layers import Dense
38
39 # ANN
40 classifier = Sequential() # Initializing
41 classifier.add(Dense(units = 256, kernel_initializer = 'uniform',
       activation = 'relu', input_dim = 784))
42 classifier.add(Dense(units = 128, kernel_initializer = 'uniform',
       activation = 'relu'))
43 classifier.add(Dense(units = 10, kernel_initializer = 'uniform', activation
       = 'softmax'))
44 classifier.summary()
45 classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
       metrics = ['accuracy'])
46
47 # Fitting Neural Networks
48 history=classifier.fit(X_train, y_train, batch_size = 128, epochs = 150) #
       Lesser no of epochs - Basic Model
49
```

```
50 # Save model
51 classifier.save('ann_train.h5') # h5 file will now be in your directory
```

Testing an ANN for 10 classes of FMNIST

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jun 26 11:52:38 2018
4
5  @author: Rajenki
6  """
7  import numpy as np
8  import pandas as pd
9  from keras.utils import to_categorical
10 from keras.models import load_model
11
12 img_rows, img_cols = 28, 28
13 shape = (img_rows, img_cols, 1)
14
15 classifier = load_model('fashion.h5')
16
17 #Test
18 data_test = pd.read_csv('fashion_test.csv')
19
20 X_test = np.array(data_test.iloc[:, 1:])
21 y_test = to_categorical(np.array(data_test.iloc[:, 0]))
22
23 X_test = X_test.astype('float32')
24 X_test /= 255
25
26 predicted_classes = classifier.predict_classes(X_test)
27 actual_classes = np.zeros(10000)
28 y_test = y_test.astype('int')
29 for i in range(0,10000):
30     for j in range(0, 10):
31         if(y_test[i][j] == 1):
32             actual_classes[i] = (j)
33
```

```
34 print(predicted_classes)
35 print(actual_classes)
36 correct = 0
37 for k in range (0,10000):
38     if predicted_classes[k] == actual_classes[k]:
39         correct= correct+1
40
41 print("Test accuracy is:", (correct/10000)*100,"%")
```

Training a CNN for 3 new classes

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jun 26 12:21:09 2018
4
5  @author: Rajenki
6  """
7  import numpy as np
8  import pandas as pd
9  from keras.models import Sequential
10 from keras.layers import Conv2D
11 from keras.layers import MaxPooling2D
12 from keras.layers import Flatten
13 from keras.layers import Dense
14 from keras.layers import Dropout
15 from keras.utils import to_categorical
16 import matplotlib.pyplot as plt
17
18 #Import dataset
19 dataset = pd.read_csv('Train_TL.csv')
20 X = dataset.iloc[:,1:].values
21 y = dataset.iloc[:,0].values
22
23 img_rows, img_cols = 28, 28
24 shape = (img_rows, img_cols, 1)
25
26 X_train = np.array(dataset.iloc[:, 1:])
27 y_train = to_categorical(np.array(dataset.iloc[:, 0]))
```

```
28
29 # y is of int type. Change it to categorical
30 y = y.astype('object')
31
32 #  one hot encoding
33 from keras.utils import np_utils
34 y = np_utils.to_categorical(y)
35
36
37 X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
38 X_train = X_train.astype('float32')
39 X_train /= 255
40
41 #CNN
42 classifier = Sequential()
43
44 classifier.add(Conv2D(32, (3, 3), input_shape = shape, activation = 'relu')
       )
45 classifier.add(MaxPooling2D(pool_size = (2, 2)))
46 classifier.add(Dropout(0.2))
47
48 classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
49 classifier.add(MaxPooling2D(pool_size = (2, 2)))
50 classifier.add(Dropout(0.4))
51
52 classifier.add(Flatten())
53
54 classifier.add(Dense(units = 128, activation = 'relu'))
55 classifier.add(Dense(units = 13, activation = 'softmax'))
56 classifier.summary()
57 classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
       metrics = ['accuracy'])
58
59 history2 = classifier.fit(X_train, y_train, batch_size = 128, epochs = 150)
        # Lesser no of epochs - Basic Model
60
61 # Save model
```

```
62 classifier.save('cnn_13_train.h5') # h5 file will now be in your directory
63
64 #  "Accuracy"
65 plt.plot(history2.history['acc'])
66
67 plt.title('classifier accuracy')
68 plt.ylabel('accuracy')
69 plt.xlabel('epoch')
70 plt.show()
71
72 # "Loss"
73 plt.plot(history2.history['loss'])
74
75 plt.title('classifier loss')
76 plt.ylabel('loss')
77 plt.xlabel('epoch')
78 plt.show()
```

Transfer learning on CNN to classify 3 new classes

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jun 26 12:26:36 2018
4
5  @author: Rajenki
6  """
7  import numpy as np
8  import pandas as pd
9
10 from keras.models import Sequential
11 from keras.layers import Dense
12 from keras.utils import to_categorical
13 import matplotlib.pyplot as plt
14 from keras.models import load_model
15
16 #Import dataset
17 dataset = pd.read_csv('NewClasses.csv')
18 X = dataset.iloc[:,1:].values
```

```
19 y = dataset.iloc[:,0].values
20
21 img_rows, img_cols = 28, 28
22 shape = (img_rows, img_cols, 1)
23
24 X_train = np.array(dataset.iloc[:, 1:])
25 y_train = to_categorical(np.array(dataset.iloc[:, 0]))
26
27 # y is of int type. Change it to categorical
28 y = y.astype('object')
29
30 #  one hot encoding
31 from keras.utils import np_utils
32 y = np_utils.to_categorical(y)
33
34
35 X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
36 X_train = X_train.astype('float32')
37 X_train /= 255
38
39 #CNN
40 classifier = Sequential()
41 classifier = load_model('cnn_train.h5')
42 classifier.summary()
43 classifier.layers.pop()
44
45 classifier.add(Dense(units = 13, activation = 'softmax'))
46 classifier.summary()
47 classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
       metrics = ['accuracy'])
48
49 history2 = classifier.fit(X_train, y_train, batch_size = 128, epochs = 150,
        validation_split = 0.2, shuffle= True) # Lesser no of epochs - Basic
       Model
50
51 # Save model
52 classifier.save('cnn_13_tl_train.h5') # h5 file will now be in your
```

```
    directory
53
54 #  "Accuracy"
55 plt.plot(history2.history['acc'])
56 plt.plot(history2.history['val_acc'])
57 plt.title('Model Accuracy')
58 plt.ylabel('Accuracy')
59 plt.xlabel('Epoch')
60 plt.legend(['Train', 'Validate'], loc='best')
61 plt.show()
62 # "Loss"
63 plt.plot(history2.history['loss'])
64 plt.plot(history2.history['val_loss'])
65 plt.title('Model Loss')
66 plt.ylabel('Loss')
67 plt.xlabel('Epoch')
68 plt.legend(['Train', 'Validate'], loc='best')
69 plt.show()
```

Testing Image Processing and ANN Model of 10 Classes

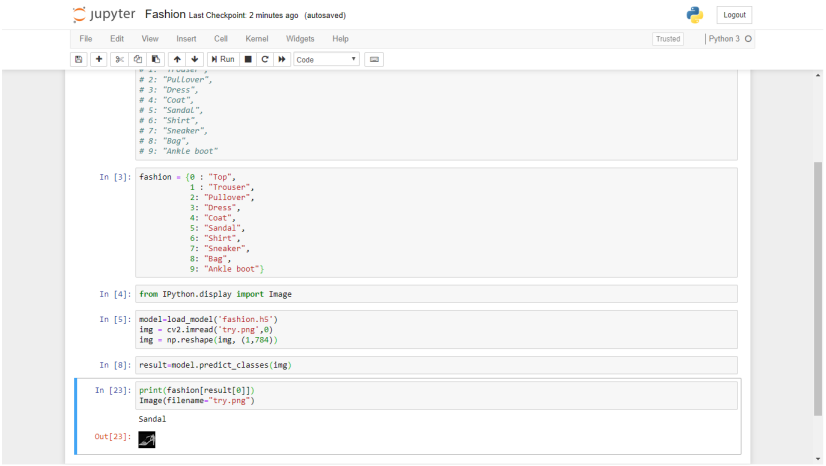

Figure 1: Testing Image Processing and ANN Model of 10 Classes

Training a model

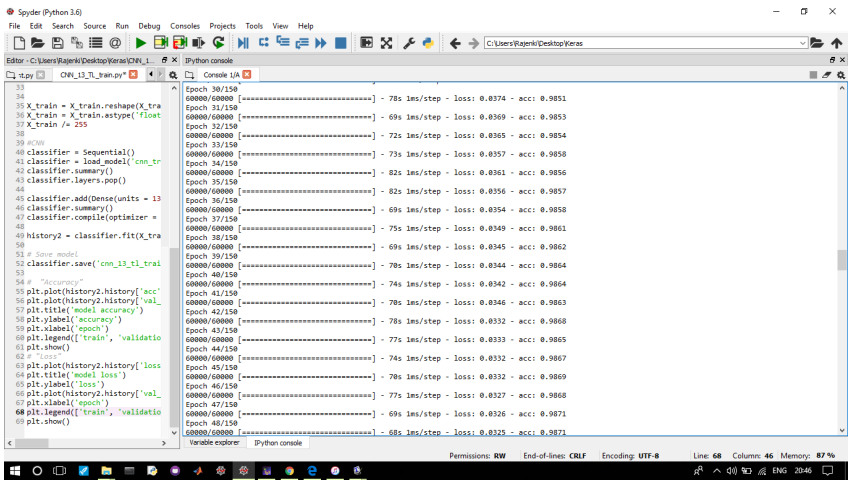

Figure 2: Training a model

Testing a model

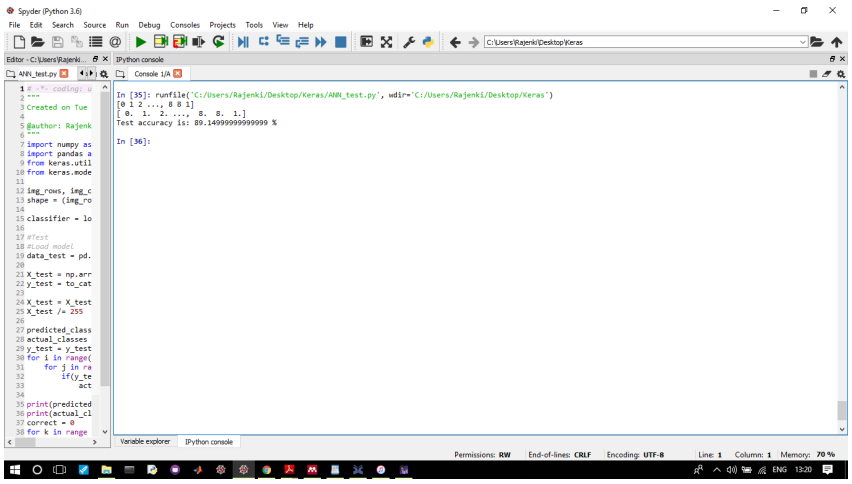

Figure 3: Testing a model

References

Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36.

Boland, M. V. and Murphy, R. F. (2001). A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of hela cells. *Bioinformatics*, 17(12):1213–1223.

Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75.

Ding, Z., Shao, M., and Fu, Y. (2015). Deep low-rank coding for transfer learning. In *IJCAI*, pages 3453–3459.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Haskell, R. E. (2000). *Transfer of learning: Cognition and instruction*. Elsevier.

Jantzen, J., Norup, J., Dounias, G., and Bjerregaard, B. (2005). Pap-smear benchmark data for pattern classification. *Nature inspired Smart Information Systems (NiSIS 2005)*, pages 1–9.

Liu, G., Lin, Z., Yan, S., Sun, J., Yu, Y., and Ma, Y. (2010). Robust recovery of subspace structures by low-rank representation. *arXiv preprint arXiv:1010.2955*.

Mitchell, T. M. et al. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877.

Nguyen, L. D., Lin, D., Lin, Z., and Cao, J. (2018). Deep cnns for microscopic image classification by exploiting transfer learning and feature concatenation. In *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, pages 1–5. IEEE.

Pan, S. J., Kwok, J. T., and Yang, Q. (2008). Transfer learning via dimensionality reduction. In *AAAI*, volume 8, pages 677–682.

Pan, S. J., Yang, Q., et al. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Pratt, L. Y. (1992). Non-literal transfer among neural network learners. *Colorado School of Mines: MCS-92-04*.

Pratt, L. Y., Mostow, J., Kamm, C. A., and Kamm, A. A. (1991). Direct transfer of learned information among neural networks. In *AAAI*, volume 91, pages 584–589.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Shi, Z., Siva, P., and Xiang, T. (2017). Transfer learning by ranking for weakly supervised object annotation. *arXiv preprint arXiv:1705.00873*.

Torrey, L. and Shavlik, J. (2010). Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global.

Waibel, A., Sawai, H., and Shikano, K. (1989). Modularity and scaling in large phonemic neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12):1888–1898.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.