Expertise
and insight
for the future

*

Nhu Minh Kha Nguyen

Real-time fashion items classification using TensorflowJS and ZalandoMNIST dataset

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

25th August 2019

Metropolia
University of Applied Sciences

Author Title Number of pages Dates	Nhu Minh Kha Nguyen Real time fashion items classification using TensorFlowJS and ZalandoMNIST dataset 42 pages 23 March
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Supervisor

The last decade has marked a significant growth in Deep Learning, driven mainly by the rapid development of Graphics Processing Unit (GPU) which provide sufficient computing power for deep neural network training. However, AI platforms are usually centrally hosted on the computing instances provided by Cloud Service Providers and therefore leads to undesirable networked latency.

Acknowledging the problem, Google released TensorFlowJS, an open-sourced library for Machine Learning in an attempt to provide an interactive in-browser Machine Learning environment for researchers and practitioners to perform various machine learning tasks.

This thesis will uncover the fundamentals of deep learning by training deep artificial neural network models using FashionMNIST dataset and use TensorFlowJS to build a real-time hand-drawing image classification based on trained models. Traditionally, MNIST is the most popular dataset for validating an algorithm in deep learning; however, in April 2017, the pioneer of Artificial Intelligence Ian Goodfellow stated that MNIST is being overused by the Machine Learning community and that researchers and practitioners should move on to harder datasets. These facts inspired Zalando's research team to replace the obsolete MNIST with FashionMNIST – a dataset consists of 70,000 examples of Zalando's article images in 28x28 grayscale format.

To demonstrate the usability and effectiveness of in-browser Machine Learning platform, the author constructed a neural network model using Keras and train it with FashionMNIST. The parameters of the neural network were saved in text files and loaded to the browser using TensorFlowJS. The thesis project also made use of Canvas, an HTML5 feature which allows users to draw fashion items in the browser and convert the depicted image to TensorFlow input for prediction.

The outcome of the project is a functional web app which features a hand drawing area and a pie chart representing the prediction made by the neural network model.

Keywords	FashionMNIST, deep learning, artificial intelligence, neural network, Tensorflow.js

Contents

List of Abbreviations

AI	Artificial Intelligence.
ANN	Artificial Neural Network.
BN	Batch Normalization.
CL	Convolutional Layer
CNN	Convolution Neural Network.
DL	Deep Learning.
FC	Fully Connected
FCNN	Fully Connected Neural Network
ML	Machine Learning.
NN	Neural Network

1 Introduction

Artificial Intelligence is among the next big things in the software engineering field that empowers numerous applications in health care, finance, logistic, industries for good measure. Several scientists have embarked on utilising the advancement of Artificial Intelligence and high-grade dataset to change our lives profoundly. One notable field in Artificial Intelligence is Computer Vision, which rapidly emerged over the last decade thank to an enormous amount of visual data and significant development in GPU processing power. Neural networks can now have millions of trainable parameters which makes technologies like diseases self-diagnoses or self-driving car possible.

However, Computer Vision problems require real-time prediction and the AI community calls out for a client-side deep learning system. On 18th March 2018, Google released Tensorflow.js, an open-source library written in the JavaScript programming language for creating in-browser Artificial Intelligence platform. Tensorflow.js provides an environment for AI enthusiasts to train, run and deploy their AI programs in their browsers. Real-time AI has unlocked many opportunities and possibilities for AI projects, like interactive Machine Learning programs, data privacy, app sharing and decentralized computing resources.

The objectives of this thesis were to demonstrate the usability and effectiveness of in-browser Artificial Intelligence system. The thesis achieved that goal by constructing a neural network model to classify fashion item images and convert such model to JSON format for in-browser usage. The model will take a 28x28 grayscale format image as input and return the corresponding category. The process of predicting images happens on client-side, and there is no backend server involved.

The thesis is structure as follow. The first section introduces the purpose of the project, dataset and technologies. The second section provides theoretical background and applications of various Deep Learning algorithms and noteworthy neural network architectures, followed by the third section which describes the usage of Tensorflow.js in building in-browser AI system and how it’s implemented in this. The fourth section discusses the training and validating process and their implementation, while the last section analyses the result and gives information about the future work of the thesis.

2 An overview of Deep Learning

Artificial Intelligence is a vast field which concerns several areas like Statistic, Optimization, Machine Learning and Deep Learning. This section introduces DL as the state-of-the-art solution for AI problems with complex datasets like visual data, sound or natural languages and how it's related other fields of AI. Besides, this section explains the concepts of Convolutional Neural Network, optimization process of Neural Network training and then dives deeper to Mathematics theory behind their implementation.

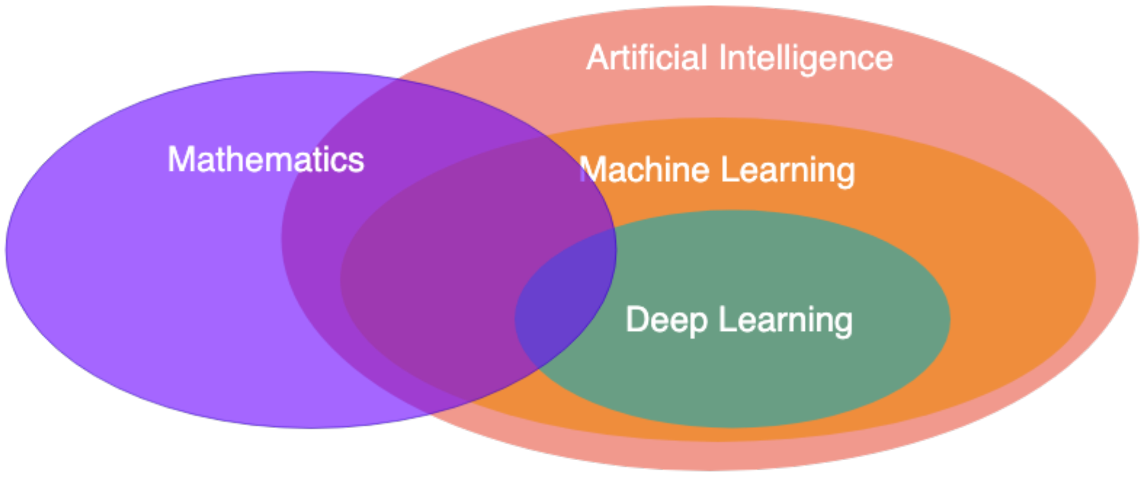


Figure 1. Relationships between Math, AI, ML and DL.

Figure 1 shows that AI, ML, and DL all involve Mathematic to some extends. Machine Learning and Deep Learning are two sub-fields of Artificial Intelligence. Machine Learning focuses on small and medium datasets, while Deep learning applies mathematical models to learn large and complex ones.

2.1 Neural Network

Researches in Deep Learning focus on generating a hierarchy of concepts that enable the machine to gain knowledge of complicated concept. To achieve that goal, Deep

Learning utilises Neural Network which combines several computational units to build up a more powerful instance.

The neural circuit system inspired computer scientist to build the concept of neural network. Each neural network consists of multiple computing units which is similar to synapse in animal brain. Both gain knowledge of outside world and perform various tasks through examples, usually without being programmatically configured. However, the development of ANN now focuses on performing a particular task, resulting in deviations from biological one.

2.1.1 Artificial Neural Network

Many Deep Learning fields have heavily adopted Artificial Neural Network, such as real-time translation, speech to text conversion, and vision system. One novelistic application of ANN is the Generative Adversarial [1] proposed by Ian Goodfellow et al. in 2014. The GAN allows two ANNs to train by opposing each other, simulating the concepts of two-player games. By applying GAN, Deep Learning system can generate realistic photographs, audio, text, and speech.

Mathematically, ANN is represented as a composition of multiple different functions altogether. Assume that there existed the function $y = f^*(x)$ that map input vector x into category y, the ultimate goal of an ANN is to optimize the parameter θ so that $f(x, \theta)$ can approximate $f^*(x)$. A quintessential example of ANN is Deep Feedforward Network (DFN) [2] which is of extreme importance to Deep Learning development since they form the skeleton of how ANN works.

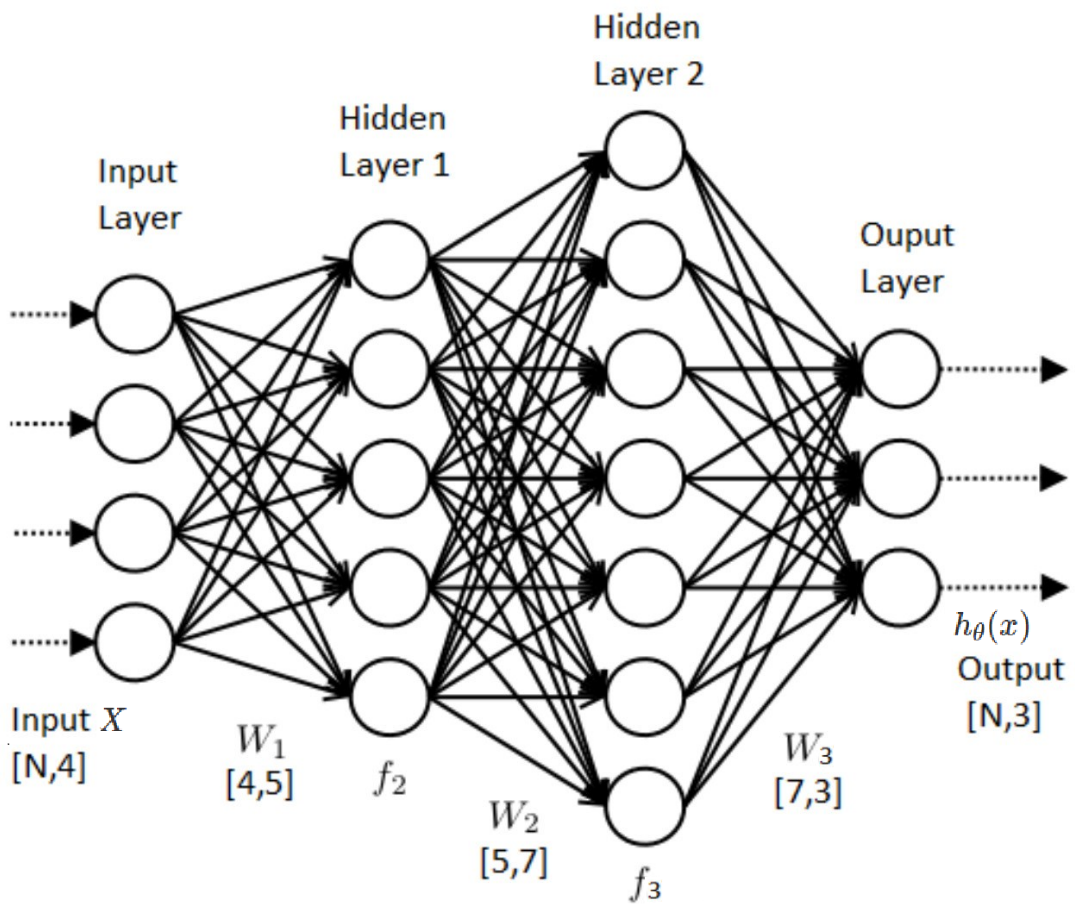


Figure 2. ANN with four layers. [3]

Figure 2 demonstrate an ANN with 1 input layers, 2 hidden layers of different size and the output layers. ANN is often referred to as Fully Connected Neural Network since a node of a layer connects to all nodes of the layer right after it.

2.1.2 Convolution Neural Network

Overfitting usually occurs when large-size image datasets are fed to the training Deep Feedforward Neural Network process [4]. For AI problem that use high-resolution images as input, the number of trainable parameters of an ANN can quickly reach over 10 million. In order for a FCNN to process a grayscale picture of dimension 1920x1080, its first layer would require at least 2,073,600 parameters. The number of computing nodes exponentially increases as the size of input grows and significantly slows down the training process.

To solve this issue, Convolutional Neural Network [5] introduces a concept called **kernel** (also called **filter**). The mechanism of kernel is based on the convolution operation in Mathematics. The convolution operation allows CNN to capture sub-feature of the inputs; therefore, the output of CNN remains unchanged despite small transformations of the inputs, such as flipping and shifting [4, p. 267].

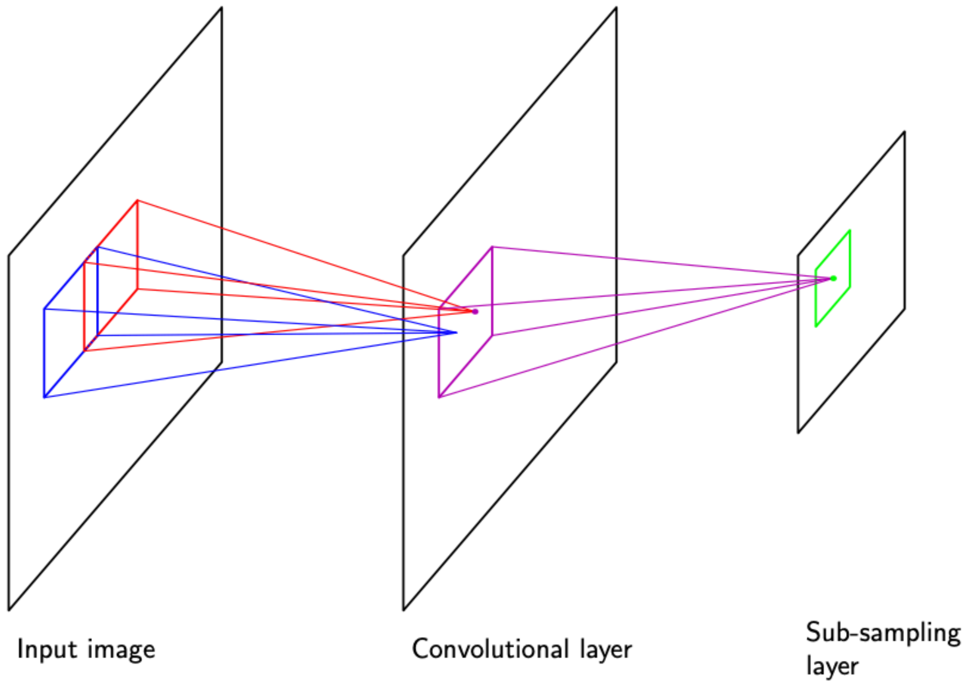


Figure 3. CNN filtering [4, p. 268]

The figure above illustrates the process of capturing sub-feature in Convolutional Neural Network using convolution operation. Each subregion of the image is filtered, and the most notable feature of that subregion is extracted and passed to the next layer of the network. The convolution operation following the below formula, with I and K are two-dimensional input and kernel respectively:

$$S(i,h) = (I * K)(i,h) = (x+a)^t = \sum_g \sum_t I(g,t)K(i-g,h-t) \text{ [2, p. 328]}$$

In Convolutional Neural Network, many kernels together form a special layer called the convolutional layer. This particular layer responsible for performing the core functionality of the CNN: detecting features of subregions of the input image like edge and colour scheme for good measure.

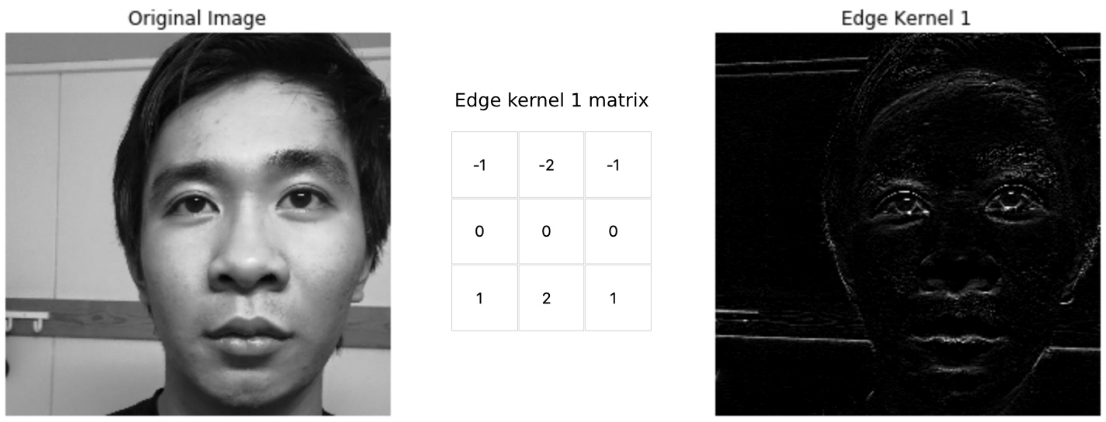


Figure 4. Visualization of edge detection using bottom Sobel.

Figure 4 shows the effect of image filtering using bottom Sobel. The image on the right-hand side displays horizontal lines very clearly while erasing vertical lines.

Convolutional Neural Network reduces the number of required parameters significantly by applying identical convolution layer to all subregions of the input image. Also, the kernel that learns useful features for the training process can be re-used to detect such features virtually everywhere without the need to re-train. These mechanisms in the translation invariance property of CNN, meaning that it can identify patterns regardless of their location inside the input image [6, p. 565].

2.1.3 Pooling Layer

Many Convolutional Neural Network models, LeNet-5 [7] for example, decrease the dimensions of convolution layers by using subsampling methods. The training process of a convolutional network generally involves three stages, which are producing linear activations, detecting feature and pooling respectively [2, p. 335]. In CNN training process, the occurrence of the feature is more considerable than its whereabouts, meaning that CNN can perform better if it is invariant to translation transformation of the input image. Pooling layers helps CNN to achieve this as long as the translation is sufficiently small.

Pooling layer works by perform statistical operations on a particular location of the net (also known as the receptive field). For example, max pooling [8] operation selects the largest value in the receptive field and pass it through the next layer. There are several other pooling types: average subsampling, max pooling and weighted average sample, to name a few. Among all, max pooling yield superior performance in practice [9].

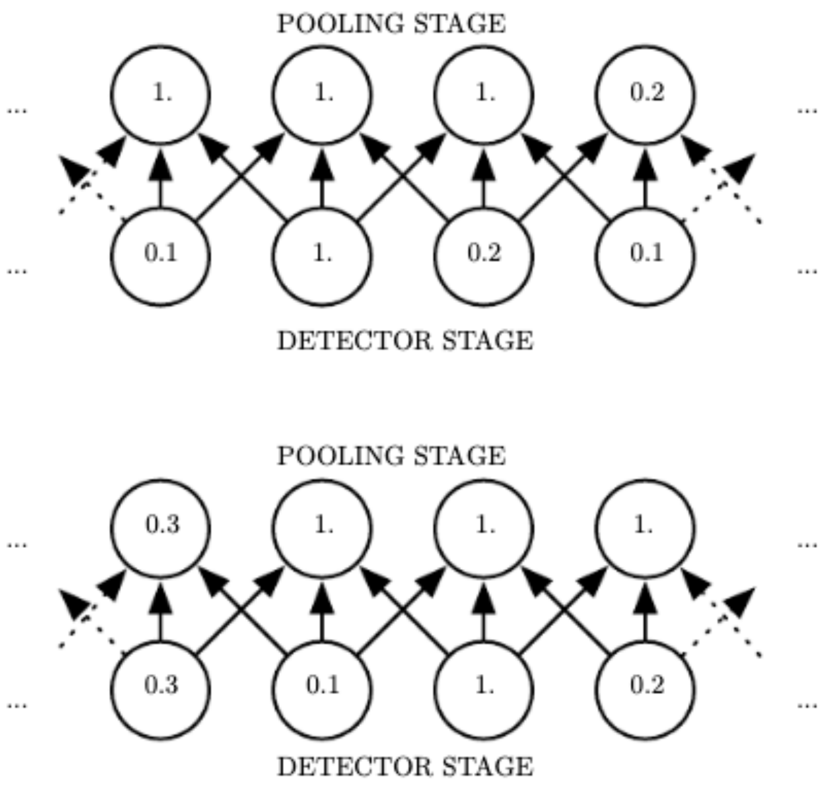


Figure 5. An illustration of max pooling operation

Figure 5 demonstrates the max pooling operation performed on two different input. The top rows represent the output of pooling layer, and the bottom row represent the input. The later input is indeed the first one shifted by 1 pixel to the right. As can be seen from the figure, only the first and last value of the lower top row have changed because max pooling only selects the most significant value within a receptive field regardless of its exact location.

2.1.4 Activation function

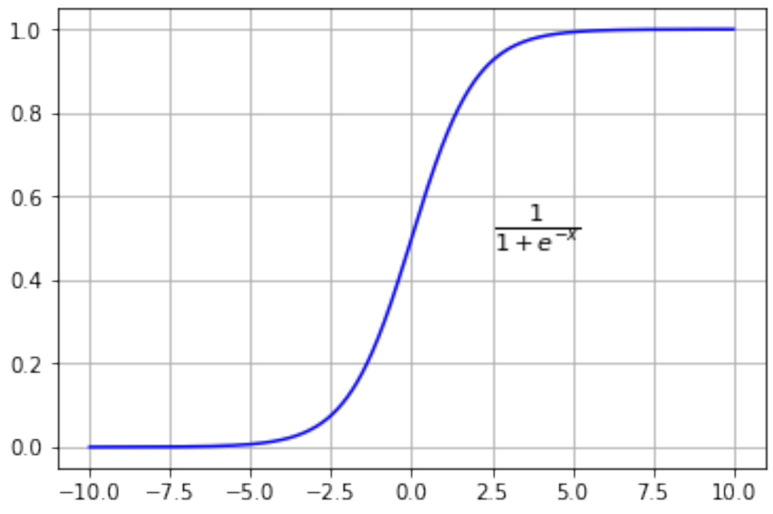
Regardless of how many layers the ANN possesses, it cannot approximate non-linear separable dataset. To tackle this issue, Artificial Neural Networks employs a mathematical concept called the activation function. For an ANN with a sufficiently large number of computing nodes, the use of activation function can enable the ANN to approximate any known complex function [10].

The choice of activation function significantly affects the overall performance of Convolutional Neural Network. This thesis will explain the concept of and give intuition about Sigmoid (also known as Logistic), Rectified Linear Unit (ReLU), TanH, and Leaky Rectified Linear Unit (Leaky ReLU). Also, the graph of each function will be provided for a better understanding.
The following Python snippet will plot Sigmoid, TanH, ReLU and Leaky ReLU respectively:

```python
from matplotlib import pylab
import pylab as plt
import numpy as np
import tensorflow as tf
x = plt.linspace(-10,10,10000)
with tf.Session() as sess:
  plt.plot(x, sess.run(tf.nn.sigmoid(x)), 'b')
  plt.text(2.5, 0.5, r'$\frac{1}{1+e^{-x}}$', fontsize=16)
  plt.grid()
  plt.show()
  plt.plot(x, sess.run(tf.nn.tanh(x)), 'b')
  plt.text(2.5, 0, r'$\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$', fontsize=16)
  plt.grid()
  plt.show()
  plt.plot(x, sess.run(tf.nn.relu(x)), 'b')
  plt.text(-5, 4, r'$max(0,x)$', fontsize=16)
  plt.grid()
  plt.show()
  plt.plot(x, sess.run(tf.nn.leaky_relu(x)), 'b')
  plt.text(-5, 4, r'$max(0.1*x,x)$', fontsize=16)
  plt.grid()
  plt.show()
```

2.1.4.1 Sigmoid function

The sigmoid function has wide application in optimization process due to its monotonicity, boundedness and differentiability. Sigmoid guarantees positive output for any given arbitrary real input. The graph of the sigmoid functions features an S-shaped curve. The graph of the sigmoid function generated by the Python snippet is:

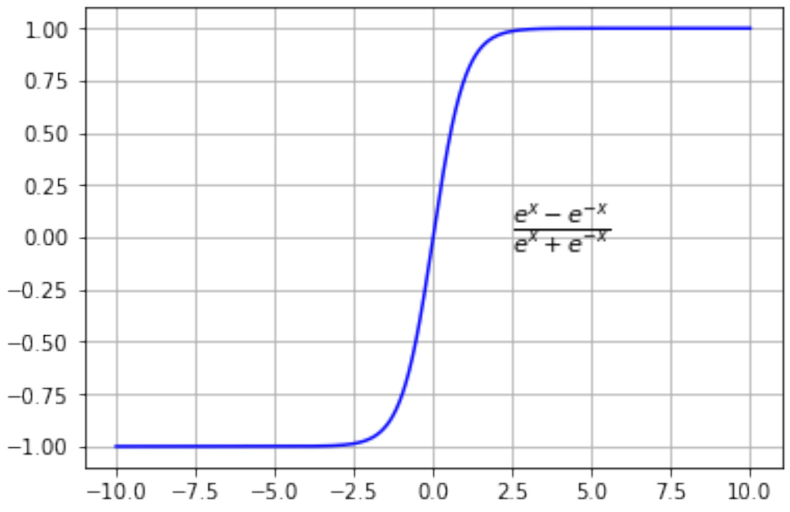


The formula of the sigmoid function is:

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

2.1.4.2 TanH function

The TanH function is a bounded, differentiable and monotonic function. Its differentiate, however, is not monotonic. Like the sigmoid function, the graph of TanH features an S-shaped curve with the range from -1 to 1. One significant difference between TanH and sigmoid is that the negative region of the input x will be mapped to the range from [-1, 0] instead of near-zero and therefore results in better performance [2, p. 195]. The graph for the TanH function generated by the Python snippet is:



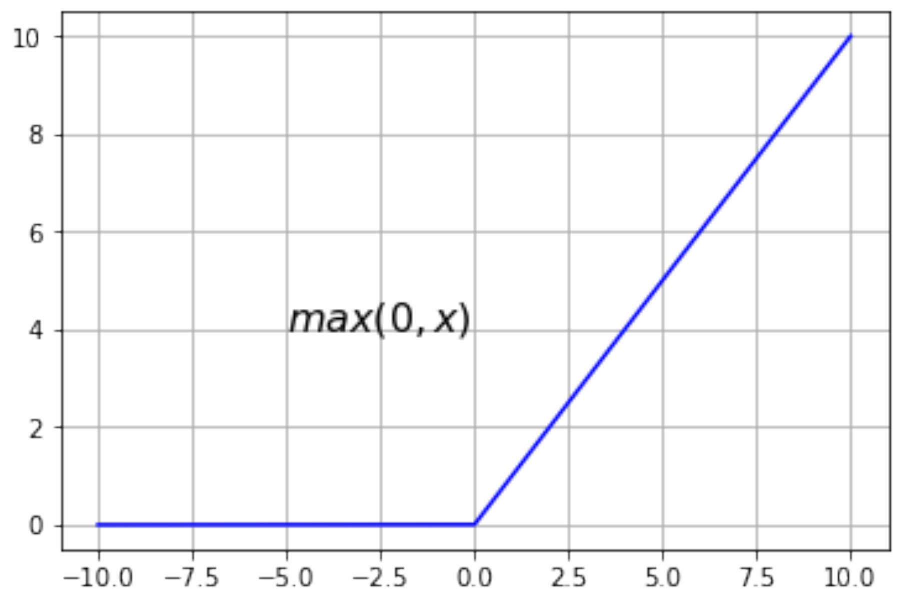
TanH function is defined as:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

2.1.4.3 ReLu function

Though Sigmoid and TanH functions generally yield good performance, they do have undesired disadvantages. One of them is both functions easily saturate when the input value approaches negative or positive infinity [2, p. 195]. This problem of the two hyperbolic leads to the vanishing gradient descent problem and make the training process of deep ANN ineffective.

Hahnloser et al. introduced the ReLU (an abbreviation of Refined Linear Unit) activation function in 2000 [11] in an attempt to construct an activation that behaves like a linear function yet does not prevent the ANN to learn complex datasets. The figure below demonstrate the graph for the ReLU function:



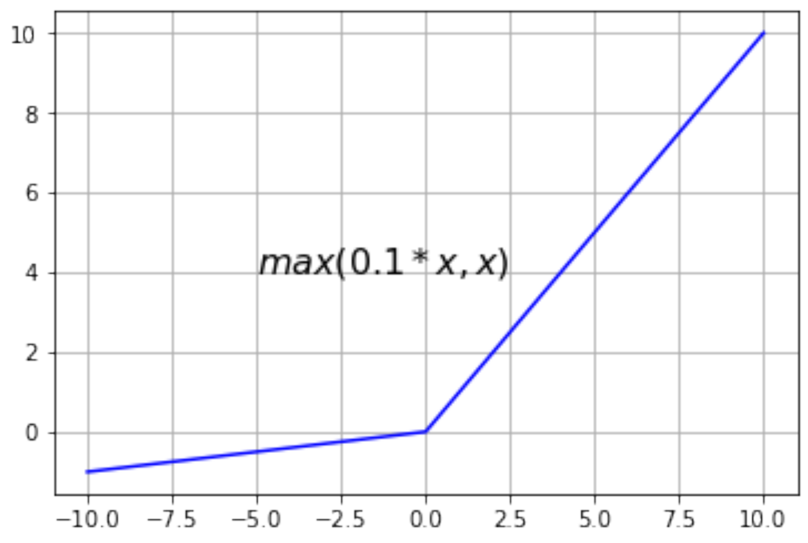
The ReLU function follows the formula:

$$f(x) = \max(0, x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

2.1.4.4 Leaky ReLU

The limitation of ReLU is that when the input is negative the gradient becomes zero and leads to the case where a node in the network never activates in the training process. Should many units in the network feed the activation function with negatives input the optimization step would be considerably slower.

The Leaky ReLU function [12] is identical to the ReLU function in the positive region of x. For negative input x, it guarantees a small positive gradient to prevent freezing the optimization process. The graph for Leaky ReLU is as follow:



The Leaky ReLU function follow the below formula:

$$f(x) = \max(0, x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

The parameter α is usually within the range $0.01 \leq \alpha \leq 0.1$ depending on the dataset and the architecture of ANN.

2.2 Optimization for Deep Learning

There are numerous cases where Artificial Intelligence algorithms need to involve optimization. For instance, solving the inference of Principal Component Analysis would require dealing with an optimization problem. Among several optimization problems that AI involves, optimizing the weight parameters of the ANN is the most complicated and arduous one [2, p. 271]

This thesis will focus on one particular optimization in Deep Learning: optimizing the parameter $\boldsymbol{\theta}$ to minimize the cost function $J(\theta)$ which is a notable measure for the overall performance of the training process.

2.2.1 Definition of optimization for deep learning

The fact that there are many intractable performance measures in Deep Learning makes direct optimization complicated. So, in contrast to traditional optimization problems which deals with minimising the cost using various mathematical models, a cost function $J(\theta)$ is minimized in the hope that the performance measures will be improved. The cost function $J(\theta)$ is defined as

$$J(\theta) = \mathrm{E}_{(x,y)\sim p_{data}} L(f(x;\theta), y) \text{ [2, p. 272]},$$

where $L(f(x;\theta), y)$ is the per training sample loss function, $f(x;\theta)$ is the hypothesis function that yield prediction for input x with distribution $p_{data}(x,y)$ given parameter θ.

2.2.2 Empirical Risk Minimization

One problem of Deep Learning is that training data comes from a fixed but unknown distribution $p_{data}(x,y)$. This means optimization process in Deep Learning will focus on minimizing empirical error instead of true error which is calculated based on all possible training data. The true distribution of training data is replaced by the empirical distribution $\hat{p}_{data}(x,y)$ and the cost function $J(\theta)$ for m training samples becomes

$$J(\theta) = \mathrm{E}_{(x,y)\sim p_{data}} L(f(x;\theta), y) = \frac{1}{m}\sum_{i=1}^{m} L(f(x^i, \theta), y^i).$$

The process of optimizing $J(\theta)$ is known as empirical risk minimization and $\mathrm{E}_{(x,y)\sim p_{data}} L(f(x;\theta), y)$ is referred to as the empirical error.

2.2.3 Loss function

Choosing the loss function has a major impact on the outcome of the training process. The competence of loss function varies according to the problem type; for instance, regression problem with numerical output type would benefit from Mean Squared Error and classification problem with multiple labels would be suitable for Categorical Cross Entropy. The purpose of this thesis was to classify fashion images, so Categorical Cross Entropy was chosen.

The formula for categorical cross entropy for N labels is as follow:

$$L(\hat{y}, y) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{N} y_{ij} * \log(\hat{y}_{ij})$$

Computing the gradient of $L(\hat{y}, y)$ is relatively easy, which is a significant advantage of Categorical Cross Entropy.

2.2.4 Backpropagation

The process of minimizing empirical risk is achieved by adopting the local message passing method in which pass error of each layer to the previous one and the weight parameters will be updated accordingly is known as backpropagation. The motivation for this method is to train the ANN such that it can learn the concept of any arbitrary mapping function [13].

There are two requirements that the loss function must satisfy in order that it’s applicable for backpropagation [14]. The first requirement is that the cost function can be represented as an average of losses over individual training samples. The reason for this requirement is that backpropagation computes the gradient of the Mean Squared Error partially for each training samples. The second requirement is that the loss function can be expressed as a function whose input is the output of the ANN. The formula for categorical cross entropy satisfies all requirements.

Backpropagation update the weight parameters of the ANN using the following rule:

$$Z^{[i]} = Z^{[i]} - \gamma \frac{\partial G}{\partial W^{[i]}}$$

$$h^{[i]} = h^{[i]} - \gamma \frac{\partial G}{\partial a^{[i]}}$$

The parameter γ is the pre-defined learning rate of the ANN and i is the index of the layer.

2.2.5 Regularization

A critical problem in deep learning is to train the neural networks in a way that it performs well not only on the observed sample but also unknown ones. Many optimizing strategies in deep learning try to decrease the empirical error on test dataset with the expense of getting higher training error [2, p. 224]. Such strategies are called regularization – one of the major optimization methods in Deep Learning.

It’s generally a good practice to penalize the weight of each layer rather than regularize the bias parameter [2, p. 226]. The reason for this strategy is that each bias affect only a single variable, meaning leaving the biased unregularized would not cause variance to rise. Also, since bias parameters are introduced to combat the underfitting problem, regularizing biases may cause the ANN model to underfit complex datasets.

2.2.5.1 L^1 regularization

The formula for L^1 regularization on the model parameter $\boldsymbol{w}$ is defined as the summation of modulus of individual parameters and has the following mathematical expression:

$$\Omega(w) = \left||w|\right|_1 = \sum_i |w_i|.$$

In practice, the L^1 regularization controls the impact of $\Omega(w)$ by multiply $\Omega(w)$ with a positive hyperparameter α. With L^1 regularization the cost function becomes

$$\tilde{J}(\theta) = J(\theta) + \alpha * \left||w|\right|_1.$$

The gradient for $\tilde{J}(\theta)$ is given by

$$\nabla_w \tilde{J}(\theta) = \nabla_w J(\theta) + \alpha sign(w),$$

where $sign(w)$ denotes the sign of w applied elementwise.

2.2.5.2 L^2 regularization

L^2 regularization forces the weight parameters to degenerate overtime. It limits the complexity of the neural network by applying a penalty on the growth in the magnitude of the weight parameters. The formula of L^2 regularization is the sum up the square of individual weight parameter together. The mathematical expression of L^2 regularization is:

$$\Omega(w) = \left||w|\right|_2 = \sum_i w_i^2 .$$

Similar to the L^1 regularization, the impact of $\Omega(w)$ on the cost function is also controlled by multiplying $\Omega(w)$ with $\frac{1}{2}\alpha$. The regularized cost function $\tilde{J}(\theta)$ has the following formula:

$$\tilde{J}(\theta) = \frac{\alpha}{2}\Omega(w) + J(\theta) = \frac{\alpha}{2}w^T w + J(\theta),$$

with the corresponding gradient

$$\nabla_w \tilde{J}(\theta) = \nabla_w J(\theta) + \alpha w.$$

Applying L^2 regularization cause the weight parameters to be closer to the origin [2, p. 227]. This strategy is known to enhance the generalization of the ANN [15] and is the most common form of regularization.

2.2.6 Batch normalization

It has been known for decades that normalizing the input vector x to normal distribution $x \sim \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$ enhance the performance of neural networks [16]. Batch Normalization evolves the idea of input normalization to normalizing intermediate layers of a deep ANN, although the Batch Normalization technique is typically performed across mini-batches of the training dataset and is therefore sometimes referred to as mini-batch normalization.

2.3 Neural network architectures

The cost of training CNN models using high-resolution images has been prohibitively high and therefore not wholly applicable to real-life problems. Fortunately, the rapid development of Graphics Processing Unit with excellent support for optimizing convolution operation make it feasible to train CNN models with large dataset like ImageNet ILSVRC-2010 [17]. This section focusses on explaining the CNN architectures that achieved a considerably low error rate on the ImageNet datasets.

2.3.1 AlexNet

Alex Krizhevsky invented the AlexNet [18] in an attempt to perform image classification task on the dataset ImageNet LSVRC-2010 and achieved a remarkable performance with an error rate of 17%.

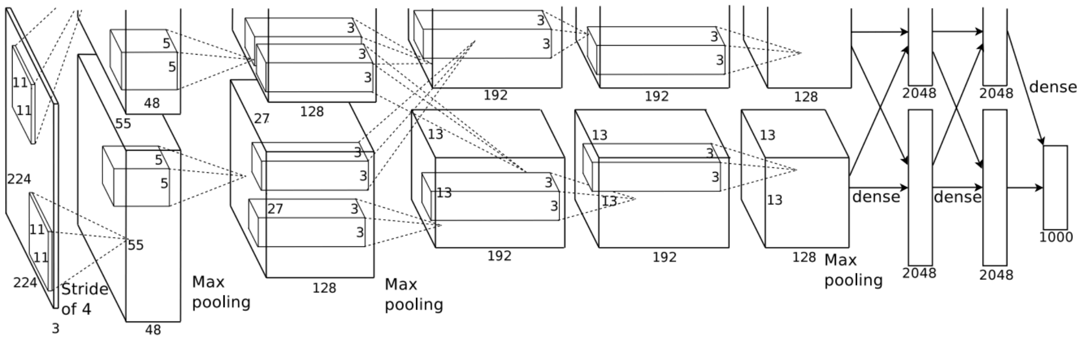


Figure 6. Illustration of the AlexNet architecture. [18]

As can be seen from figure 6, AlexNet consists of 60,000,000 parameters, 650,000 neurons, five CL, three FC layers and 1000-classes SL at last.

2.3.2 LeNet-5

LeNet-5 [19] is considered of the pioneering architecture for convolutional neural network. The network was designed by Yann LeCun et al in 1998 with the purpose of classifying the famous MNIST dataset.

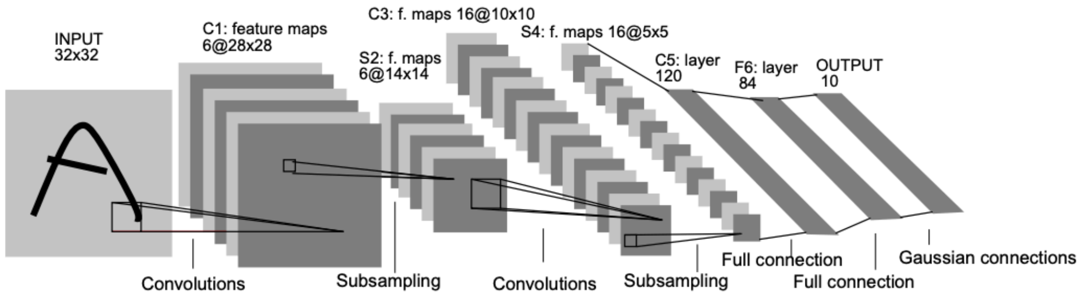


Figure 7. The illustration of the LeNet-5 architecture. [19]

Figure 7 demonstrates that LeNet-5 has two convolution – subsampling pairs and three fully connected layers. The first pair features six convolution layers of size 28 by 28 and six max pooling layers of size 14 by 14. The second one possesses sixteen convolution layers of size 10 by 10 and sixteen max pooling layers of size 5 by 5.

Because of the limit in computing power in 1998, LeNet-5 did not possess immense number of parameters to learn large datasets like ImageNet. However, the architecture ensures shift, scale and distort invariance to some degree by using local receptive field, shared weights and spatial sampling [19].

2.3.3 VGG16

VGG16 [20] was designed by Karen Simomyan and Andrew Zisserman from the Visual Geometry Group of University of Oxford. The architecture achieved the top-5 error of 6.8% on the ILSVRC-2014 dataset, securing the second place of the ILSVRC-2014 challenge.

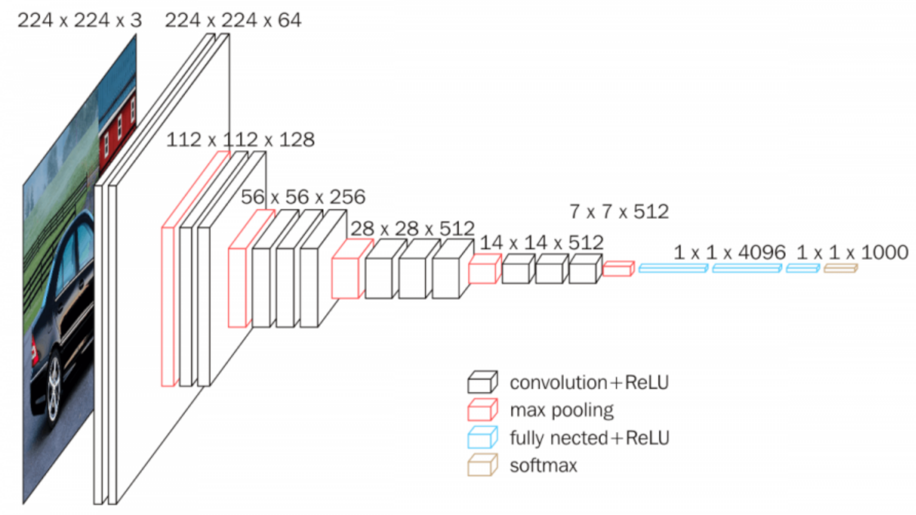


Figure 8. The illustration of VGG16 architecture

As can be seen from Figure 8, the VGG16 architecture employs several consecutives convolution layers which enhance the ability to captures important features of the image and therefore improve the overall accuracy. VGG16 is adapted to object detection and localization research field and typically serve the purpose of base neural network.

2.4 Challenges and limitation

Even though Deep Learning is regarded as a state-of-the-art solution for learning large and complex dataset, there are still various challenges and barriers to address. This section focuses on two critical issues of Deep Learning: Overfitting and Computing expenses.

2.4.1 Overfitting

Modern neural network models often have millions of neurons (trainable parameters). Having a large number of trainable parameters enable the model to learn complex data but usually leads to the overfitting problem in which the model fails to yield good performance when it processes never-seen-before data. There are several strategies to combat the overfitting problem, yet it remains one of the most challenging issue in ANN optimization process.

2.4.2 Computing expenses

Training a large neural network model usually requires high-end Graphics Processing Unit, which significantly increases the overall cost. Also, it’s quite often that the training process lasts for days, even months, so the energy cost usually accounts for a significant portion in the total budget. High expense means that reachability is low and therefore slow down the development pace.

3 Libraries and Frameworks

This thesis used various libraries and frameworks to implement the real-time fashion classification system, namely Keras, TensorFlow, and TensorFlowJS. This section introduces the essential details about these libraries and framework, as well as analyses their advantages and disadvantages.

3.1 TensorFlow

TensorFlow was initially implemented by the Google Brain team [21] for internal usages only. Google decided to make the first release of TensorFlow in 2015 as an open-sourced software framework under the Apache License 2.0. Since then, TensorFlow has been massively adopted for both research and production usages by various software companies.

TensorFlow offers various level of abstraction for implementing and training neural network models depending on developers' needs. The framework provides an outstanding performance when it comes to computational speed thanks to the C++ kernel. The kernel allows developers to scale the applications without sacrificing speed or performance.

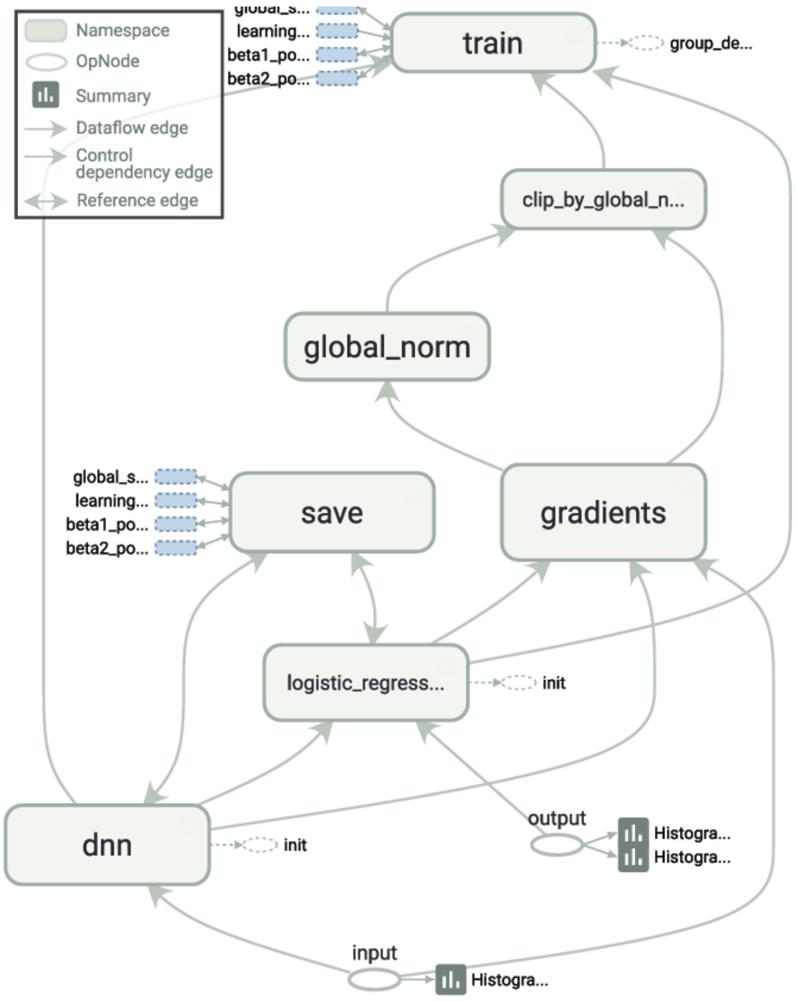


Figure 9. An architecture of TensorFlow computational graph

Figure 9 demonstrates the workflow of TensorFlow computational graph. The input data is passed to Deep Neural Network (DNN) layers and from there, it either goes through several parameters optimization processes or gets saved for later uses.

3.2 Keras

Keras is a framework for architecting ANN model implemented in Python. It is capable of utilizing TensorFlow, Theano, Microsoft Cognitive Toolkit, or PlaidML as its backend depending on the configuration.

The initial released of Keras was on 27 March 2015. The library was originally developed by François Chollet [22] and actively maintained by various developers as it is an open-sourced project. Two years after Keras's first release, Google's TensorFlow team announced an official module to support the Keras functional API in TensorFlow.

3.3 TensorFlow.js

TensorFlow.js is an open-sourced framework designed to build ML models in the JavaScript programming language. The Google's TensorFlow team announced the official release of TensorFlow.js for Artificial Intelligence application implementation in March 2018 [23]. The library allows deploying pre-trained TensorFlow model to the browser for running, re-training and modifying.

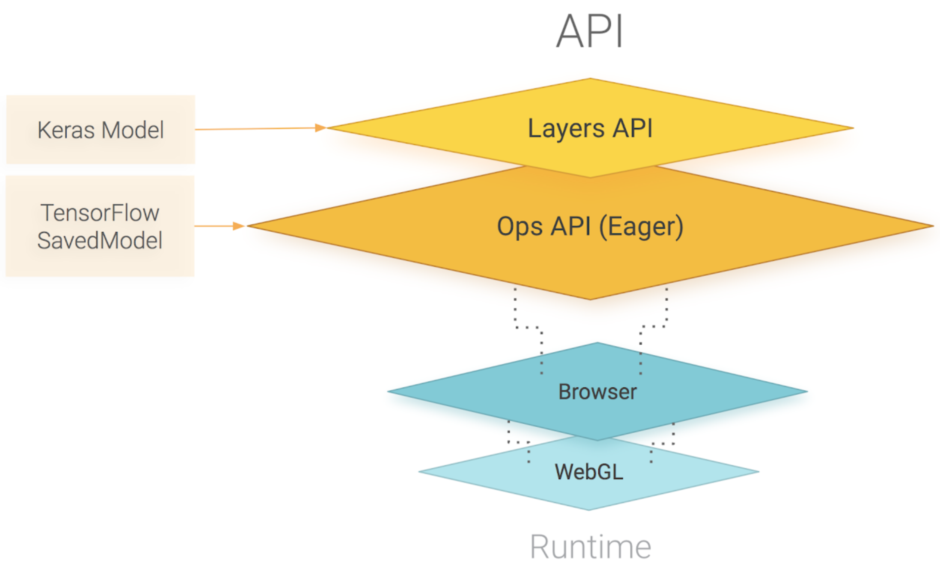


Figure 10. Architecture of TensorFlow.js application [23]

As can be seen from Figure 10, TensorFlow.js relies on WebGL for running execution and provides a high-level APIs for Deep Learning models building. TensorFlow.js also supports pre-built models implemented in Keras and SavedModels.

4 Training and validation

Understanding the characteristics of the dataset is a crucial part of any AI project. This section introduces the basic features of the dataset and visualization to make it more understandable. Additionally, this section covers basic details about Keras as a Neural Network Application Program Interface, the detailed architecture of the CNN, and the training process along with sample codes.

4.1 Fashion MNIST

The MNIST Database of Handwritten Digit Images was published in 2012 and has been actively used as the base dataset to validate new models in Deep Learning research. However, with the fast-paced growth of DL, the AI community calls out for a more complex dataset as MNIST is too easy and does not represent modern Deep Learning task.

In August 2017, Zalando published the Fashion MNIST dataset [24] in an attempt to replace the traditional handwritten digits dataset. The new dataset preserves all the accessibility of MNIST like size, format and number of classes. The Fashion MNIST is significantly more complex than the MNIST since the samples were captured using digital cameras (thus possess more features).

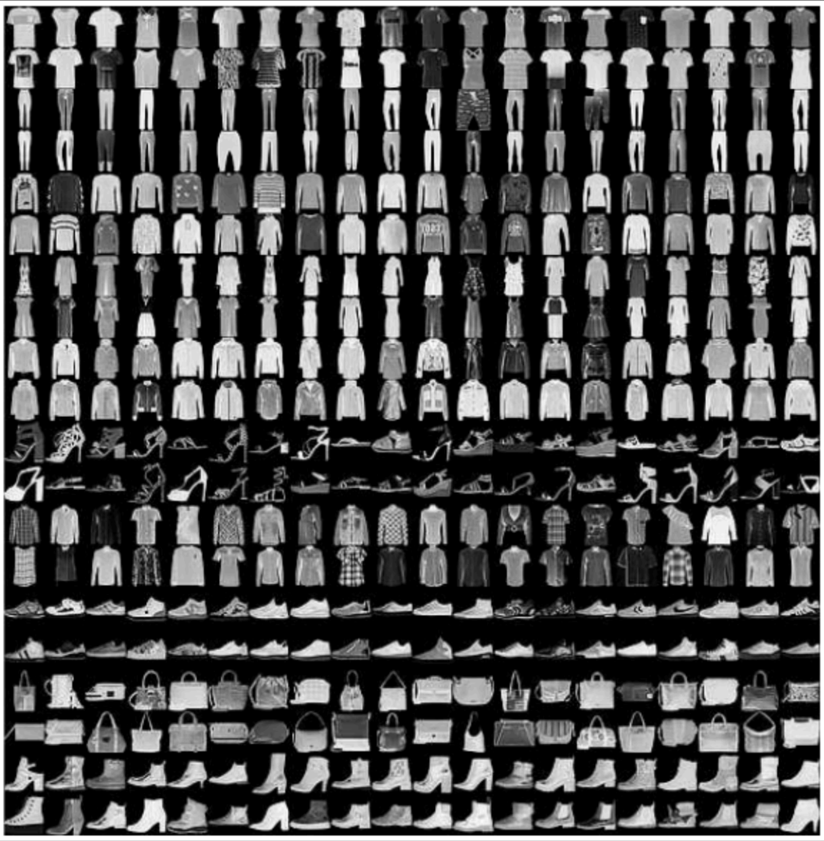
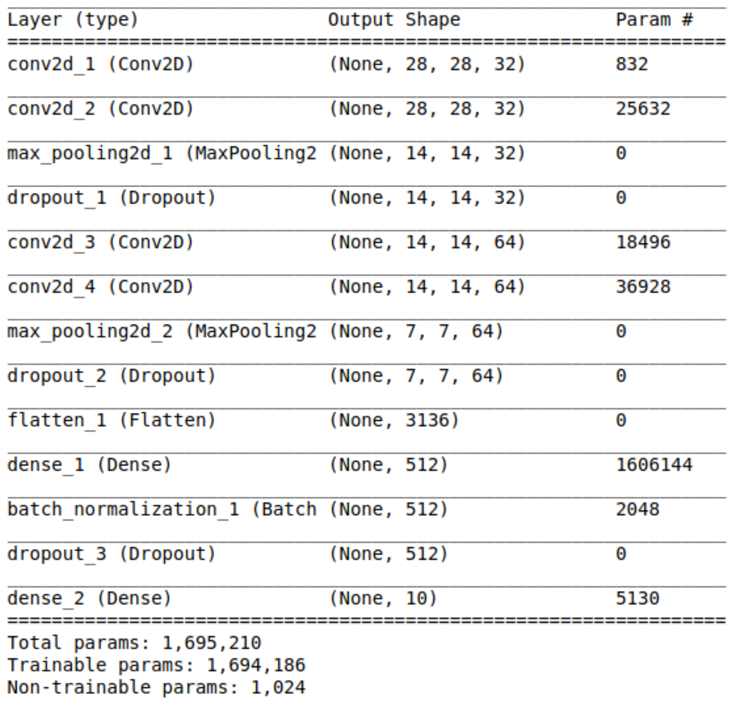
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 11. Sample images with corresponding class name from Fashion MNIST [24]

Figure 11 visualizes sub-samples along with their class names from the original Fashion MNIST dataset. There are totally 70 000 samples evenly distributed to 10 classes. The dataset is randomly segmented into training and test sets with 60 000 and 10 000 samples, respectively.

4.2 Implementation

The convolution neural network model was implemented using Keras – a framework API designed for building ANN implemented in Python and capable of using Theano, CNTK, and TensorFlow as backend.

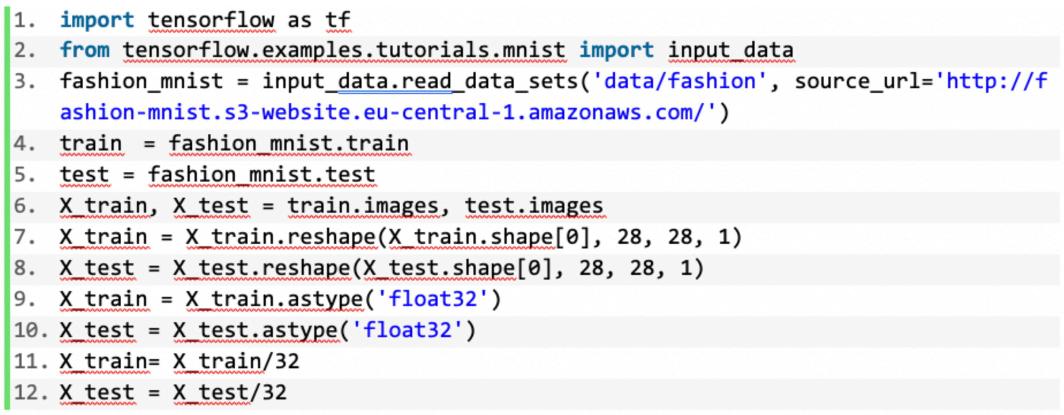


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 32)        25632
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1606144
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,695,210
Trainable params: 1,694,186
Non-trainable params: 1,024
```

Figure 12.   Summary of the neural network model

Figure 12 demonstrates the architecture of the CNN model. The model consists of four convolution layers, two max-pooling layers and two dense layers with batch normalization. Dropout layers with 25% rate were also added to counter the overfitting problem.

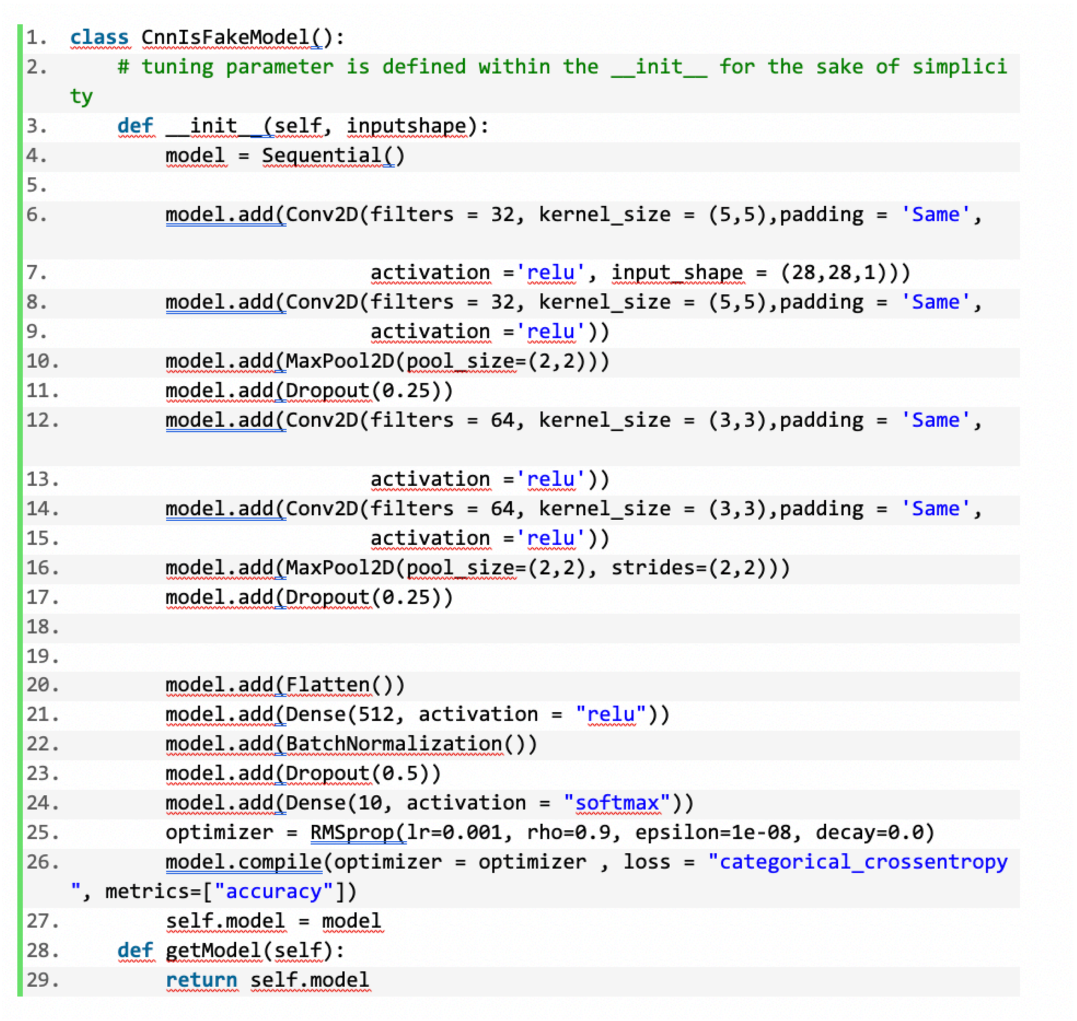
Figure 13 shows the Python code snippet that describes the process of loading and reshaping the train and test dataset:



```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
fashion_mnist = input_data.read_data_sets('data/fashion', source_url='http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/')
train  = fashion_mnist.train
test = fashion_mnist.test
X_train, X_test = train.images, test.images
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train= X_train/32
X_test = X_test/32
```

Figure 13.   Dataset loading and reshaping

The model was built using Keras Sequential class, as shown in Figure 14:



```
class CnnIsFakeModel():
    # tuning parameter is defined within the __init__ for the sake of simplicity
    def __init__(self, inputshape):
        model = Sequential()

        model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',

                         activation ='relu', input_shape = (28,28,1)))
        model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                         activation ='relu'))
        model.add(MaxPool2D(pool_size=(2,2)))
        model.add(Dropout(0.25))
        model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',

                         activation ='relu'))
        model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                         activation ='relu'))
        model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
        model.add(Dropout(0.25))


        model.add(Flatten())
        model.add(Dense(512, activation = "relu"))
        model.add(BatchNormalization())
        model.add(Dropout(0.5))
        model.add(Dense(10, activation = "softmax"))
        optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
        model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])
        self.model = model
    def getModel(self):
        return self.model
```

Figure 14. Model building

The model is built within the CnnIsFakeModel class constructor and exposed via the getModel method.

Figure 15 shows the Python script for training the model.

```python
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.optimizers import Adam
from keras.layers.normalization import BatchNormalization
from keras.layers import Convolution2D, MaxPooling2D
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D, GlobalAveragePooling2D,MaxPool2D
from keras.optimizers import RMSprop
from keras.callbacks import ReduceLROnPlateau
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
myCNN = CnnIsFakeModel((28,28,1))
history = myCNN.getModel().fit(X_train,Y_train,
                              epochs = 40, validation_data = (X_test,Y_test),

                              verbose = 1, batch_size=512
                              , callbacks=[learning_rate_reduction])
```

Figure 15. Mode training with 40 epochs

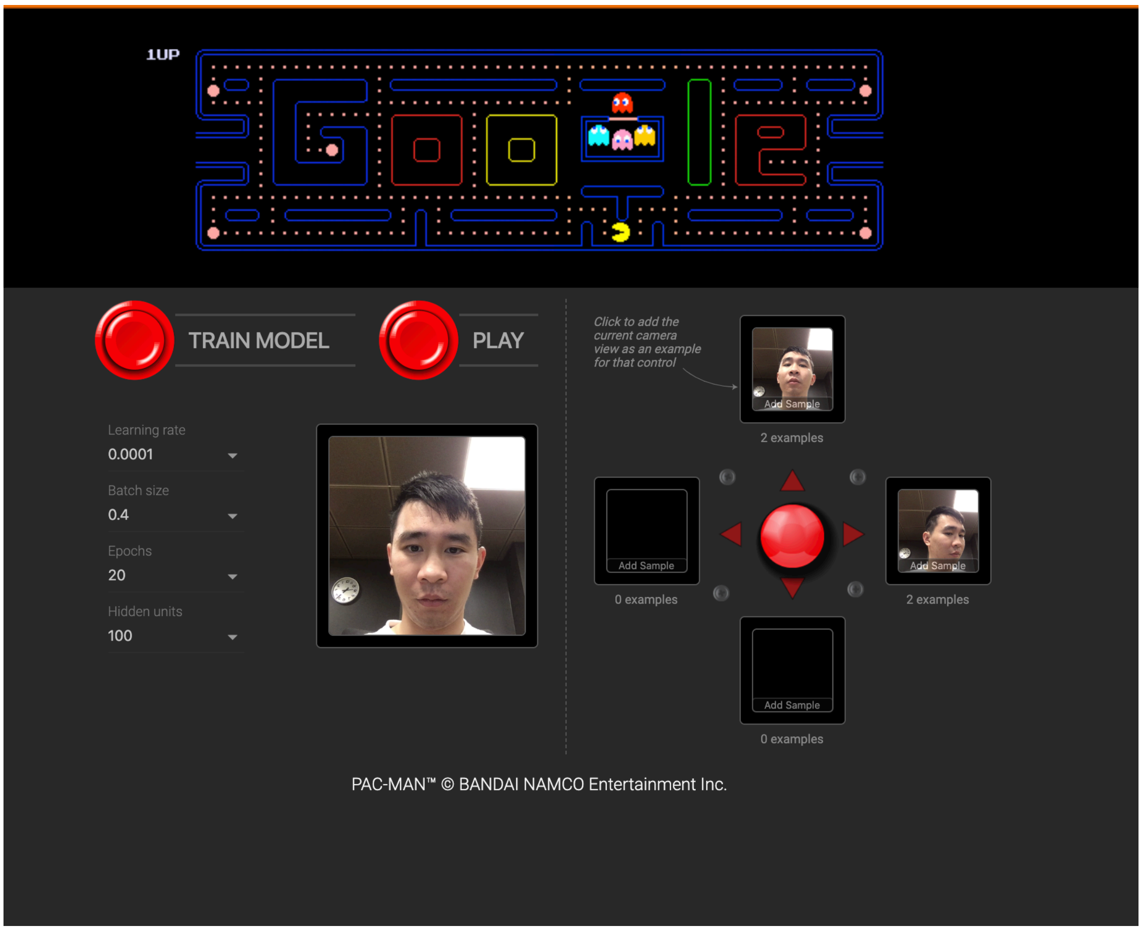
The model is trained with 40 epochs, and the batch size is 512. The training time is roughly seven seconds for each epoch. The learning rate is reduced over time based on the changes in validation accuracy to prevent the divergent problem. The model performs better as the training progresses, and eventually achieves an accuracy of 93.79% for the test set.

5 Real-time Artificial Intelligence

For years, the AI community has attempted to eliminate the dependence of AI applications on centrally hosted servers. With the development of real-time AI technologies, AI applications could finally collect data, train Deep Learning models, and make informed decisions independently of network connection. This section introduces the concepts of in-browser AI applications and how it’s implemented in this thesis.

5.1 In-browser AI applications

In-browser AI technology enables developers to build and execute Machine Learning models entirely in the client-side. This unlocks tremendously exciting new opportunities for AI researcher and practitioners, building interactive Machine Learning applications, for example.



Learning rate
0.0001

Batch size
0.4

Epochs
20

Hidden units
100

Click to add the current camera view as an example for that control

Figure 16. Interactive Pac-Man games using In-browser AI technologies

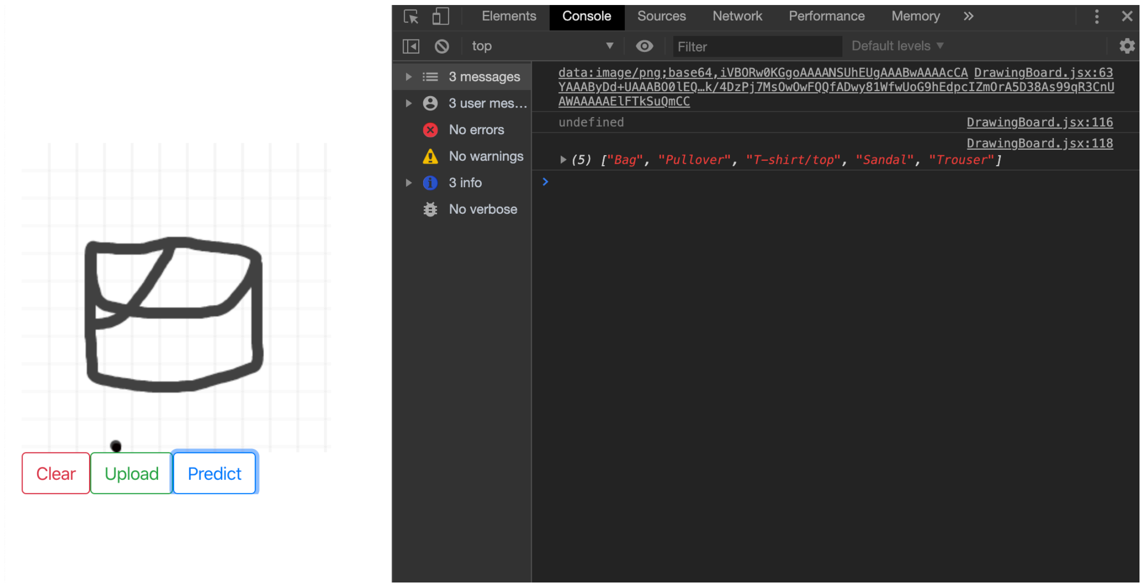
The Pac-Man game in Figure 16 uses facial gestures as the movement controlling mechanism. The neural network modes were loaded to the browser and perform classification tasks in real-time to decides the moving direction of Pac-Man.

5.2 Model conversion

Converting pre-trained neural network models into TensorFlow.js format is a critical task of building in-browser AI applications. The Keras model is first saved in h5 file format and then converted into JSON format by using TensorFlowJS Converter. The model in JSON format is loaded into the browser through TensorFlowJS Application Programming Interfaces for real-time computing.

5.3 Implementation

This thesis features a web-based application that allows users to sketch an arbitrary fashion item, and the application will make predictions based on the depicted image.



```
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABwAAAAcCA DrawingBoard.jsx:63
YAAAByDd+UAAABO0lEQ…k/4DzPj7MsOwOwFQQfADwy81WfwUoG9hEdpcIZmOrA5D38As99qR3CnU
AWAAAAAElFTkSuQmCC
undefined                                              DrawingBoard.jsx:116
                                                       DrawingBoard.jsx:118
▸ (5) ["Bag", "Pullover", "T-shirt/top", "Sandal", "Trouser"]
>
```

Figure 17. Overview of the application

Figure 17 describes the usage of the application. The users can sketch the fashion item on the left side, then they can click on the "Predict" button, and the result is displayed on the right side. The application makes top-5 prediction, meaning that it generates five class names that best described the input. In Figure 15, the author intended to sketch a bag and got "Bag" as the top result.

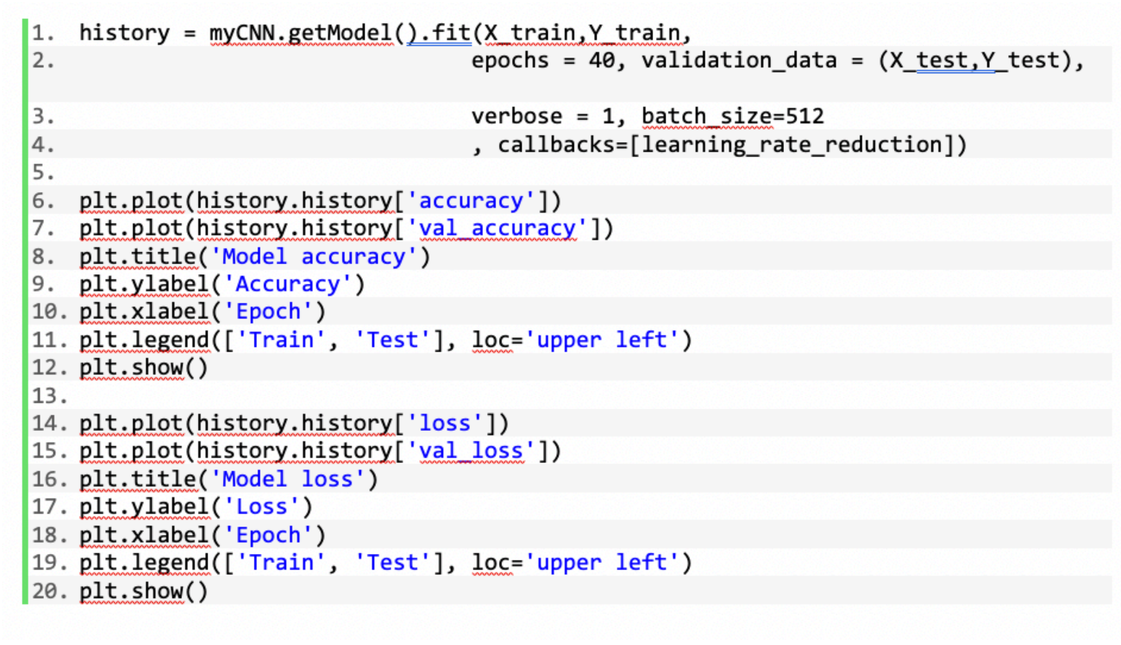
6 Evaluation

This section covers the evaluations of both the convolution neural network model and the web application to visualize the full picture of how well the web application performs against the Keras model. There’s also detailed discussion about future improvements and research objectives.

6.1 Neural Network model evaluation

There are several approaches to measure the performance of a neural network model. This thesis focus on evaluating the accuracy of the top-1 prediction of the model since it is how benchmarks are evaluated on the official FashionMNIST GitHub.

We first need to plot the accuracy and loss of the convolution neural network model over epochs to determine whether having more epochs would result in better performance. Figure 18 demonstrates the Python snippet plots the line-graph of the accuracy and loss of training and validation dataset:



```
history = myCNN.getModel().fit(X_train,Y_train,
                              epochs = 40, validation_data = (X_test,Y_test),

                              verbose = 1, batch_size=512
                              , callbacks=[learning_rate_reduction])

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Figure 18. Accuracy and loss graph plotting

The accuracy and loss of the training and validating process are plotted in the following images:

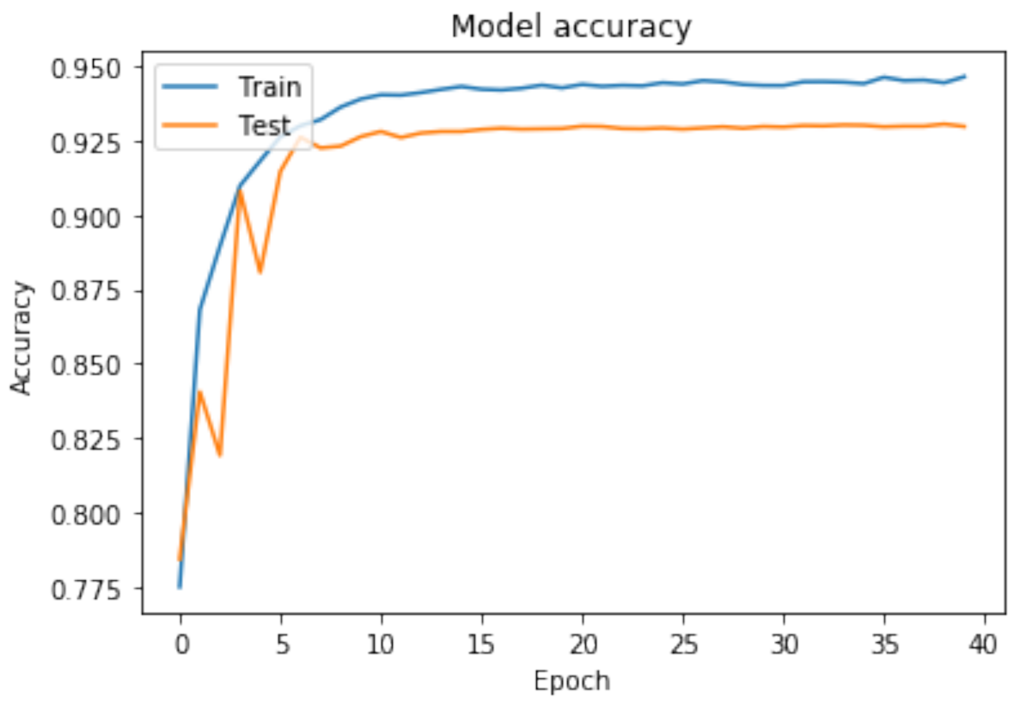


Figure 19. Accuracy over epochs

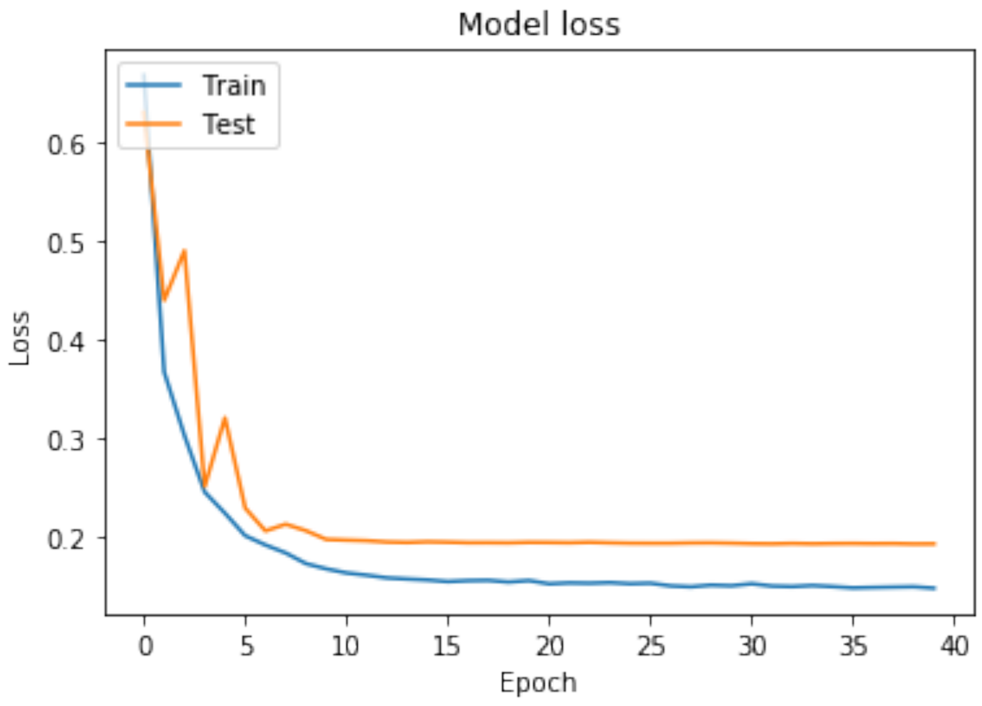


Figure 20. Loss over epochs

As can be seen from Figure 19 and Figure 20, the accuracy of both training and validating process increases significantly from the first to tenth epochs. The loss for training and validating process also decreases dramatically over the first ten epochs. After that, both accuracy and loss remain stable until the termination of the process. This phenomenon signifies that the model has converged since the tenth epochs and training the model any further adds little value to the overall performance.

One noteworthy point is that accuracy and loss sometimes become worse than they're at the previous epochs. There are several problems that might cause this issue like a large learning rate, overfitting and small batch size. To resolve this issue, the author applied the learning rate reduction call-back to gradually decrease the learning rate over time.

6.2 Web application evaluation

Measuring the web application requires human interaction, and therefore, massive test batch size is not a practical solution. The most appropriate approach, in this case, would be sketching several images and observing the results.

One noteworthy point in evaluating the web application is that the input images for the neural network training process are captured using digital cameras, whereas the input images for the web app are sketched ones. This significantly aggravates the performance of the web app as sketched images do not convey as much information as the digital ones.

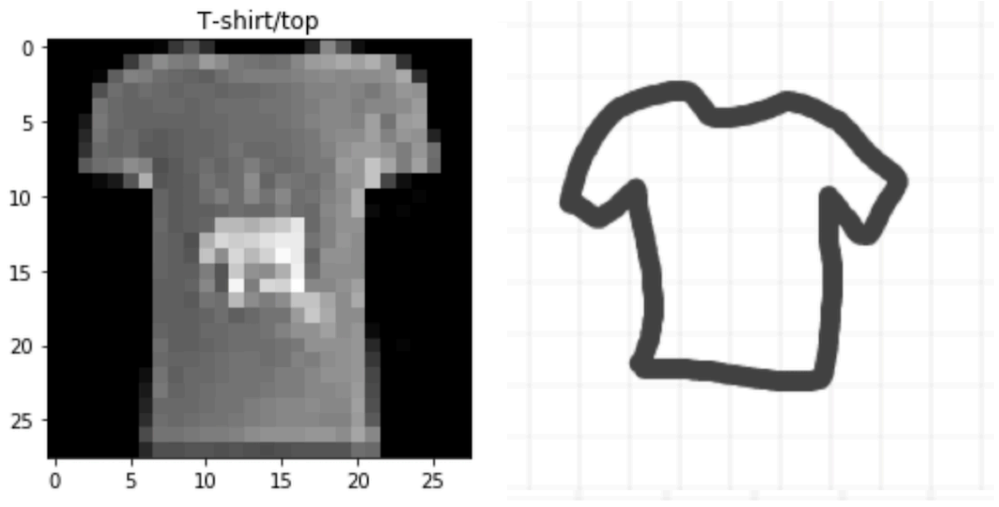
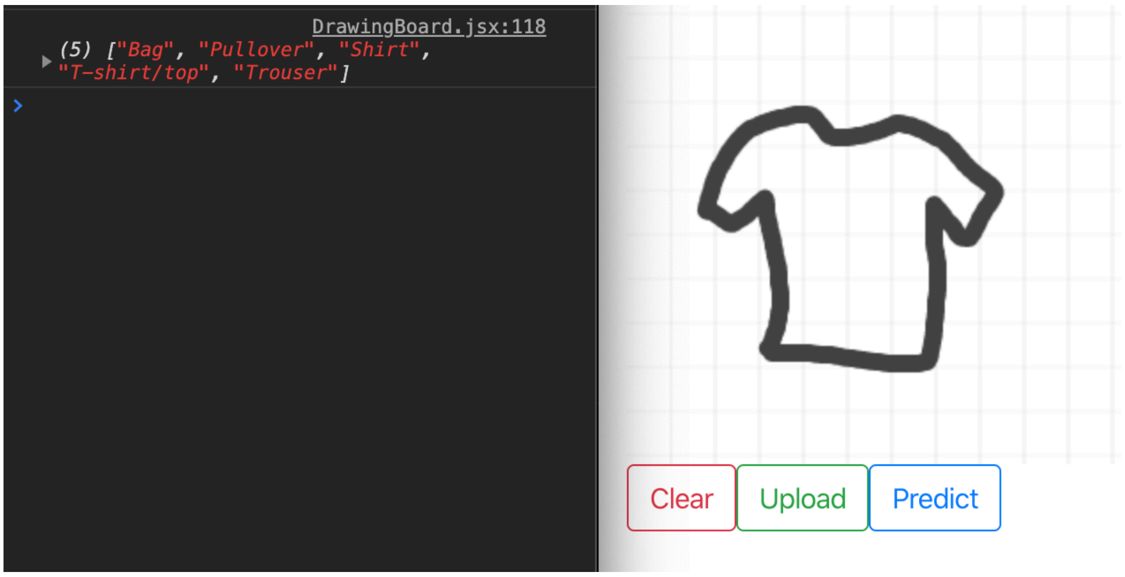


Figure 21. Data for training versus data for predicting

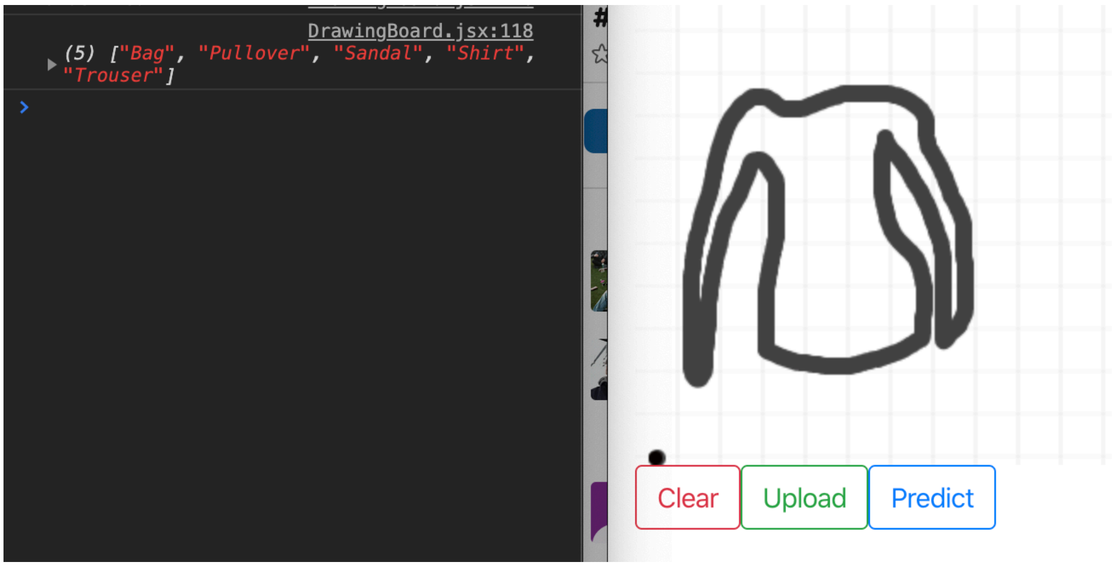
Figures 21 shows differences between two samples of T-shirt class. As can be seen from the image, the T-shirt on the left-hand side clearly has more useful features for classifying task than the T-shirt on the right-hand side does.



```
DrawingBoard.jsx:118
▸ (5) ["Bag", "Pullover", "Shirt",
"T-shirt/top", "Trouser"]
>
```

Figure 22. Prediction for T-shirt

Pullover, Shirt and T-shirt are sensible predictions for the image in Figure 22. Bag was the prediction with the highest confidence, so the model does not work well on this image.



```
DrawingBoard.jsx:118
▸ (5) ["Bag", "Pullover", "Sandal", "Shirt",
"Trouser"]
>
```

Figure 23. Prediction for Pullover

The author intended to sketch a pullover in Figure 23. Bag was the prediction with the highest confidence again. The author hypothesized that since images in class Bag and images in class Shirt, T-shirt and Pullover have visually similar bottom the model confuses images in these classes.

6.3 Further improvement and research

It can be seen that the model achieved only trivial improvements in accuracy after the tenth epochs. The final accuracy was 93.4, and the accuracy at tenth epochs was 91.33. The author concluded that it’s not sensible to increases the number of epochs any further since it might lead to the overfitting problem. The right approach in this situation would be modifying the architecture of the neural network and image augmentation.

Concerning the web application, the main improvement would be seeking a better dataset for training the neural network. There are significant differences between the input for training and the one for testing as discussed in the “Web application evaluation” section, which causes the performance of the web application to drop dramatically.

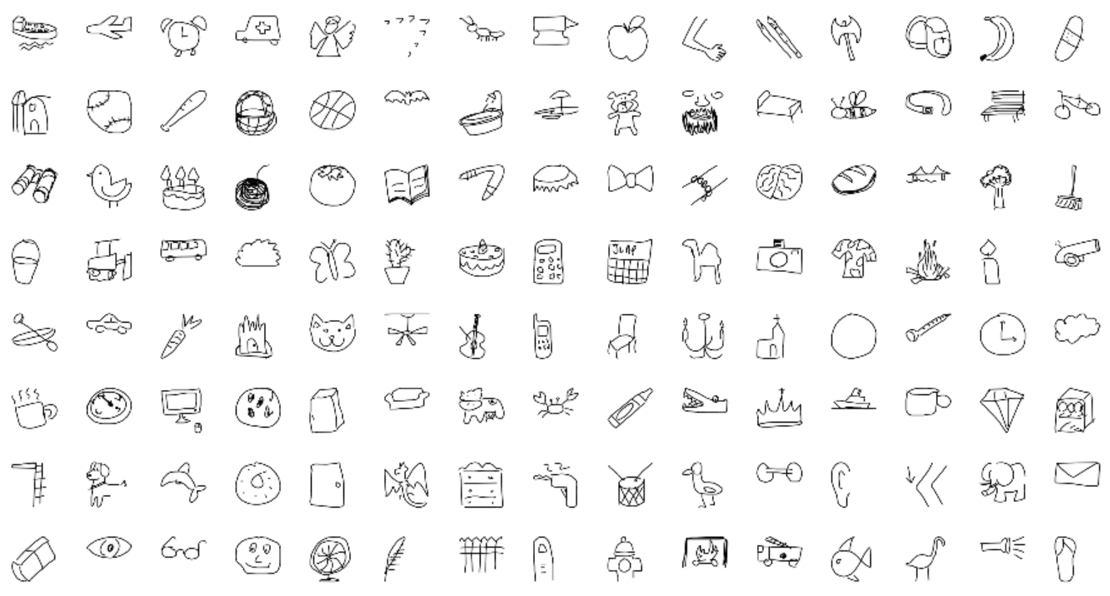


Figure 24. Samples from the Google Quick Draw dataset

Figure 24 illustrate some samples from the Google Quick Draw dataset. The dataset features a collection of 50 million hand-drawings images spanning across 345 categories. Since the web application makes predictions based on sketched images, the

Google Quick Draw data set is perceived as a more suitable and robust dataset than the Fashion MNIST.

7 Conclusion

Returning to the research objectives mentioned in the “Introduction” section of the thesis: how to build a real-time Artificial Intelligence system and demonstrate its usability using the Fashion MNIST dataset. It can be concluded that the thesis has successfully introduced an image classifier with outstanding performance compared to other submissions on Fashion MNIST GitHub. The thesis also demonstrated the core features of in-browser Artificial Intelligence system by implementing the web application. The background theory of Artificial Intelligence, Machine Learning, and Deep Learning were explained meticulously along with the concept of neural network and its applications.

There are, however, several improvements left for future work. Many popular neural network architectures such as AlexNet, MobileNet, and VGG were not used due to computing resources limitation. The author firmly believes that by employing highly ranked neural network in the ImageNet competition would considerably improve the overall accuracy of the classifier. Also, the performance of the web application would significantly improve should the classifier were trained with sketched images instead of digital photos.

In summary, the thesis emphasizes the importance of real-time Deep Learning system in solving real-world problems. Real-time technology has been playing an indispensable role in human life and now, with artificial intelligence embedded to it, thousands of opportunities have just arisen.

Bibliography

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza , Bing Xu, David Warde-Farley, Sherjil Ozair, “Generative Adversarial Nets,” in *Advances in Neural Information Processing Systems 27*, 2014.

[2] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[3] G. Ognjanovski, “Towards Data Science,” 14 January 2019. [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a. [Accessed 30 8 2019].

[4] C. M. Bishop, Pattern recognition and machine learning, 5th Edition, Springer, 2007.

[5] Yann LeCun et al, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computation,* vol. 1, pp. 541-551, 1989.

[6] Kevin Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.

[7] Yann LeCun et al, “Gradient-based learning applied to document recognition,” *Proceeding of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[8] Yi-Tong Zhou, Rama Chellappa, “Computation of optical flow using a neural network,” in *IEEE 1988 International Conference on Neural Networks*, 1988.

[9] Dominik Scherer, Andreas Müller, Sven Behnke, “Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition,” in *20th International Conference on Artificial Neural Networks (ICANN)*, Thessaloniki, Springer, 2010, pp. 99-101.

[10] Kurt Hornik, Maxwell Stinchcombe, Halbert White, “Multilayer feedforward networks are universal approximators,” *Neural networks,* vol. 2, no. 5, pp. 359-366, 1989.

[11] Richard Hahnloser et al, “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,” *Nature,* vol. 405, pp. 947-951, 2000.

[12] A. Mass, A. Ng and A. Hannun, “Rectifier Nonlinearities Improve Neural Network Acoustic Models,” in *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[13] Geoffrey Hinton, David Rumelhart, Ronald William, “Learning representations by back-propagating errors",” *Nature,* vol. 323, pp. 533-536, 1986.

[14] Yann LeCun, Yoshua Begio, Geoffrey Hinton, “Deep Learning,” *Nature,* vol. 521, pp. 436-444, 2015.

[15] G. Hinton, “Learning translation invariant recognition in a massively parallel networks,” in *PARLE Parallel Architectures and Languages Europe*, Springer, 1987, pp. 1-13.

[16] Yann LeCun , Léon Bottou, Bottou, Genevieve Orr, Klaus-Robert Müller , “Efficient BackProp,” in *Neural Networks: Tricks of the trade*, Springer, 1998, pp. 9-48.

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision,* vol. 115, pp. 211-252, 2015.

[18] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Neural Information Processing Systems,* vol. 25, 2012.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, Pattrick Haffner, “Gradient-based learning applied to document recognition,” in *IEEE*, 1998.

[20] Karen Simonyan, Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *ICLR*, 2015.

[21] J. Dean, “TensorFlow,” Google, [Online]. Available: https://www.tensorflow.org/about. [Accessed 5 1 2020].

[22] F. Chollet, “Keras IO,” [Online]. Available: https://keras.io/. [Accessed 5 1 2020].

[23] D. Smilkov, “Medium,” Google, 30 March 2018. [Online]. Available: https://medium.com/tensorflow/introducing-tensorflow-js-machine-learning-in-javascript-bf3eab376db. [Accessed 5 1 2020].

[24] Han Xiao, Kashif Rasul, Roland Vollgraf, “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms,” 2017.

[25] Sáez Yago, Baldominos Alejandro, Isasi Pedro, “A Comparison Study of Classifier Algorithms for Cross-Person Physical Activity Recognition,” *Sensors,* vol. 17, p. 66, 2016.