Imperial College London

BEng Individual Project Interim Report

Department of Computing

Imperial College of Science, Technology and Medicine

Machine Learning grading and feedback using Deep learning and NLP

Author:
CHAN Chun Ki

Supervisor:
Prof Bernhard Kainz

June 1, 2019

Submitted in partial fulfillment of the requirements for the Computing BEng of Imperial College London

Contents

Chapter 1

Introduction

1.1 Introduction

There are lots of challenges associated to todays learning processes due to its increasing complexity and scale. It is harder for teachers to observe, evaluate and give feedback to students piece of work. On the other hand, machine learning techniques are recently a very popular topic and there are a lot of research of machine learning in application to different industries. The capability of machine learning in solving complex problems. It is therefore interesting to investigate into how machine learning techniques can be associated to smart learning environments.

In this paper we use machine learning techniques to solve different problems and to provide feedback to students. Here, we first investigate how to give accurate and precise grading and later extend to providing accurate yet constructive feedback to students. We use the Imperial hardware coursework as a starting point and investigate the possibility to extend the feedback system to a more general context.

The aim of the project is to create a machine learning grading and feedback system, we shall discuss the ability of different machine learning approaches, also the possibility of migrating the system into a general purpose system which could evaluate other of coursework.

Chapter 2

Background

2.1 Background

2.1.1 Smart Learning Environment

SLEs (Smart Learning Environments) according to Koper **?** , are Physical environments that are improved to promote better and faster learning by enriching the environment with context-aware and adaptive digital devices that, together with the existing constituents of the physical environment, provide the situations, events, interventions and observations needed to stimulate a person to learn to now and deal with situations, to socialize with the group, to create artefacts and to practice and reflect. SLEs is a broad concept that covers a wide range of systems and devices. Compared to Kopers approach, Gros **?** sees the role of SLEs a coach or a learning guide and is based on these educational principles: Conversation, reflection, innovation, self-organization.

Sirkka Freigang **?** presented different patterns when designing SLEs, each emphasises a core element. Human centricity is a major factor of success for SLEs, a popular SLE has to gain the trust of its users to be able to sustain. This is an inceptive phase which is the basis when designing a SLE. We should perform ”Activity and context analysis” to ensure the goals of the designer are aligned with the learners. By constantly improving the SLE, ultimately we want to create a personal learning environment that maximizes user centricity and effectively satisfies learning needs.

Design Criteria

Didactical variety is built on user centricity where it enables effective learning by combining a variety of didactical models to create different methods of teaching. As knowledge is being updated constantly and new research are created, highly personalized learning has to be built to meet the demand. A large number of teaching materials and tools needs to be available for SLE development to provide diversity such that students are motivated throughout the learning process.

Hybrid learning space combines physical and digital space in to hybrid environments. The physical aspect such as room size, shapes, colours affects the mood of the learner. By taking advantage of Internet of things(IoT), we can control the physical space such as heating and cooling, noise suppression, amount of daylight etc. in a more efficient manner. We can achieve these by enhancing objects such as sensors to automatically switch lights on and off, using multi-functional room dividers that can serve as a touch screen, a writable wall or a shelf.

Learning assistance is the final and the most complicated stage in Sirkka's **?** proposed design patterns. To design such assistant in SLE, one must first identify "Current and future learning objects and learning content", it is also important to have a sound knowledge base to support the assistant before interconnecting the learning sources. By applying big data analytic and machine learning, we can build summary reports to analyse the learner's progress and effectiveness and provide feedback on the student's work.

Relation to our grading and feedback system

Both the descriptions from Koper and Gros indicate feedback and reflection as an important element in SLEs and this is the focus of this research. In terms of user centricity, the ultimate goal of designing a grading and feedback system is to enhance the effectiveness of learning. In addition to grading, the system should provide feedback on what went wrong and suggest ways to improve. When it comes to didactical variety, we should focus on how the feedback can provide theoretical knowledge to enhance the learning and motivation of the student. To combine physical space and the digital space into a hybrid learning space, although controlling the physical environment is not the goal of this research, it is worth investigating into how to integrate physical objects in the context of the system. One such direction could be to use computer vision in recognising the answers from a piece of paper as the input to the feedback system. Learning Assistance is the core to our investigation, this is the main part of the grading and feedback ststem, after identifying user needs and the context of the feedback system, we shall build it in a way that we can provide our user with the necessary support on their coursework.

2.1.2 Massive Open Online Course

With the advancement in technology, more and more traditional classroom learning is conducted in digital and online form. This gives rise to Massive Open Online Courses which provides courses to its Users from all over the world. An important element for these online courses is to provide feedback and marking. Since MOOCs have to deal with a huge crowd, it is therefore not feasible to assign human instructors to carry out most of the feedback and marking work. Automated grading and feedback has been adopted by a lot of the MOOC platforms such as Coursera, Udacity, edX.

2.1.3 Defining the context of our investigation

The context of our learning assistant can be categorized according to the types of input which determines what kind of feedback should be given and what machine learning techniques we can apply to evaluate the coursework.

Short numeric and text answers

For this type of coursework, we only need to match the answer with a model answer, when there is only one fact, it is very easy to evaluate and provide feedback, marks are given based on the number of matching results. A very important requirement is that the sample answer should be as accurate and as broad as possible to ensure the correctness of marking. Sometimes when the solution to the answer is broader than a few answers, we can take advantage of machine learning techniques to compare the students' solution with the sample using some predefined or generated metrics.

Essays

Essay questions are hard to evaluate, for a machine to evaluate a piece of essay, we must first extract a set of features to use as the basis or criteria of awarding marks, we then train a machine learning model and use it to evaluate students' essays. This involves natural language processing to transform the words and phrases of an essay to something a machine can understand. There are also numerous machine learning techniques involved in natural language processing and I will explain them in the next section. Other measures can also be used to evaluate an essay but these are more trivial techniques such as word count, and counting the number of occurrence of keywords to judge how relevant a piece of essay is to the topic.

Graphical solutions

When the graphical solution is generated using a piece of software, it is similar to the previous cases, since all data can be extracted from the software, but we still need to generate metrics or features to provide guidelines for marking. However, when we don't have access to the parameters in the case of a solution drawn on a piece of paper, computer vision techniques are useful to extract information such as the orientation, connections between nodes and many other features from the graph, we can then use similar techniques to train a machine learning model and evaluate the work.

There are generally two ways to provide feedback, first we can do it by pre-defining feedback for each classification result, for example we divide according to score, the same feedback is associated to a particular score range. The second method will be a challenge, which is to provide custom feedback according to the machine's understanding of the information in the students' solutions. In particular in essay questions we need to use natural language understanding to provide suitable feedback. We should train another machine learning model to try and generate custom feedback according to the comment of similar works marked by professors.

2.1.4 Existing automatic grading systems

Computer program evaluation

(Automatic grading systems - Automatic grading of computer programs: a machine learning approach)*

This paper from Srikant **?** analyses the problems of existing approaches to evaluating computer programs and suggests a machine learning approach to the automatic grading system. The most commonly used approach to automatic assessment in computer programs is by looking at the number of test cases they pass, the problem associated with this method is that there are a lot of programs that is inefficient and written with bad practices and passes all the test cases while there are programs that are close to the sample solution but does not pass all test cases. Another popular approach is to measure the abstract representations such as control flow diagrams. However, there are always multiple representations that are considered correct and it is difficult to derive all the possible solutions to a complex problem.

Srikant proposed a machine learning approach to solving the automatic grading problem. In this approach, features are first derived and later used to learn a model using evaluations done by experts as the ground truth. To determine the features, the author follows the human evaluation process and derived a number of features we can look at in the program. Below we use the following problem as an example "Given an integer n, print out the nth fibonacci number"

Grammar of features: There are certain occurrences of keywords and tokens that should appear in an algorithm, here we are expecting control structures such as "while", "if" and operator such as "+". These are useful to see if there even are the right construct in the program. We can also look at the expression features and these features in a control context to generate graphs such as the Control Flow Graph(CFG), Data Dependecy Graph(DDG) and DDG Annotated with Control Flow Information. Other feature such as Abstract Syntax Tree is also considered.

After deriving the features, the author experimented two machine learning approaches to solve the regression problem: Ridge regression and Support vector machines.

Essay evaluation systems

(An Automated Scoring Approach for Arabic Short Answers Essay Questions)*

This literature by Hebah **?** studies the calculation of automatic score by comparing the similarities between the student's answer and the model answer. Preprocessing is first carried out to extract useful information from the essay. It is split into the following parts: segmentation, stop-words removal, normalization, finding synonyms and extracting roots. After preprocessing, we have a list of synonyms of all keywords to determine the similarity between the student's answer and the sample

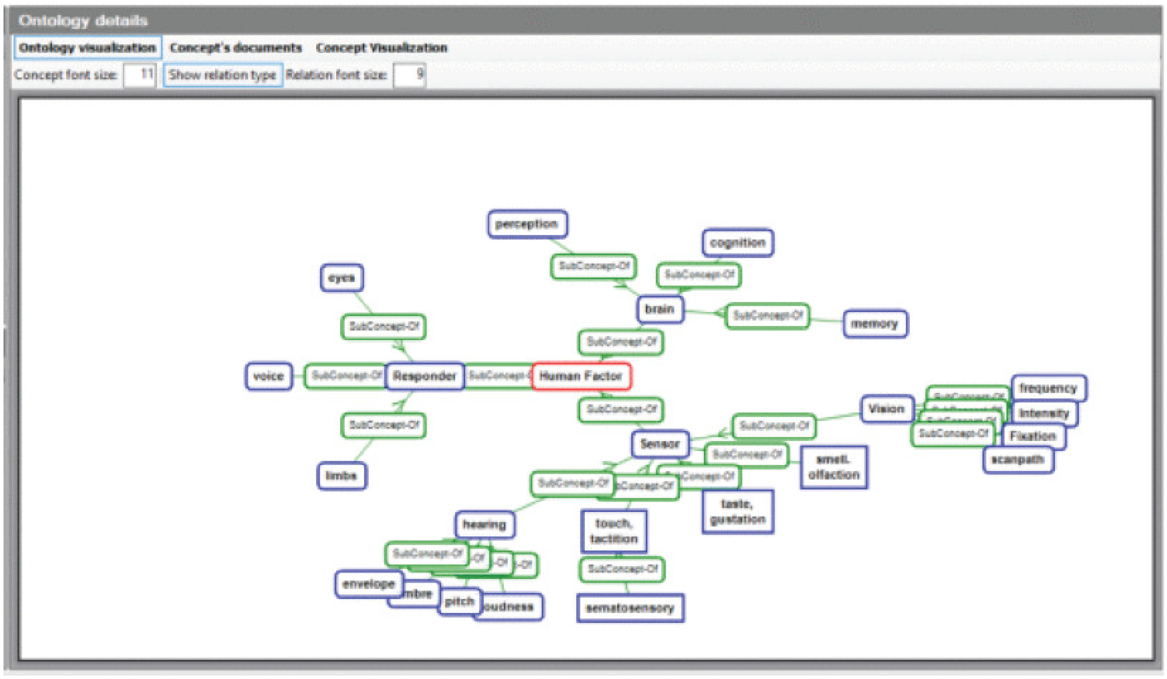
Concept font size: 11 Show relation type Relation font size: 9

Figure 2.1: An Ontology map

answer. The student answer and the sample is represented as a vector. The score is then calculated using the cosine similarity measure.

$$cosine \quad similarity(SA, MA) = \frac{SA \cdot MA}{\|SA\| * \|MA\|}$$

This method achieves 0.954 positive correlation with the score produced by human. However, this is only applicable to short answers where text similarity is the only measure to the score. When the texts become longer the scoring criteria is not only based on similarity, depending on the question, students' answer that is different to the sample may still score high if they have matching concepts. Below, we shall discuss other approaches to the automated essay grading problem.

(Automated Essay Scoring with Ontology based on Text Mining and NLTK tools)*

Below we look at an automated essay grading system proposed by Jennifer O. Contreras, Shadi Hilles and Zainab Binti Abubakar **?**. The system is divided into two phases, first it generates ontologies using a support vector machine to provide parameters for the next stage, then extract features from the essay using Natural Language Processing.

Generating Domain Ontology

Sample essays are fed into OntoGen - a data-driven ontology editor using support vector machine to generate ontologies of different important concepts that will be used as grading criteria. The root node contains the root of the concept hierachy and sub-concepts appears in the children nodes. Below is the ontology tree generated from the training corpus: **Feature Extraction**

Applying similar method, one can generate ontologies for the students' essay and compare it with the sample ontology using linear regression, hence producing a score. Other features are also taken into account on evaluating the score of the essay, these include numeric features like word count and length of sentence, part of speech count, vocabulary analysis which the broadness of vocabulary used is believed to correlate positively with the essay score.

Graphical coursework evaluation

(Auto-grading for 3D modeling assignments in MOOC)*

The technical aspect in evaluating this type of coursework involves parsing the user inputs and compare them with the sample answer using predefined parameters. The literature from Swapneel**?** suggests an auto-grading system for 3D modeling assignments which uses location and rotation, scale of the object and number of polygons and objects as parameters to grade the students' work.

2.2 Machine Learning and Deep Learning

2.2.1 Artificial Neural Network

(Reference: Lecture by Wen Jia Bai)* Artificial neural network is a computational model inspired by neurons in Biology. Although its simplest form was developed in 1957, it was not popular in the 1990s and 2000s as SVM proposed in 1992 offers better results. It has become popular again in the recent years as computation problems becomes more complex and the improvement of hardware allows us to build networks with more layers to achieve better results. The simplest form of an artificial neural network consists of a "Neuron" - a computational unit that accepts an input then compute an output using an activation function. A neural network can then be formed by putting many layers of neurons into connection, where the output of a neuron can be a weighted input of another neuron in the next layer. Fig. 1 below shows a fully connected multi-layer network which is called multi-layer perceptron (MLP)

The Multi-layer Perceptron is an example of feed forward neural network, where the connection between nodes never form a cycle such that information flows only in one direction. We will explore another type of neural network: recurrent neural network in NLP where the output of a neuron can be the input of the neuron in a previous layer.

We use back propagation to train a MLP. It is an algorithm to propagate error from the last layer to previous layers. we can evaluate the gradient of the loss function to minimise the loss (The error between the prediction and the ground truth). The goal of back propagation is to minimise the difference between the prediction and the groud truth, we use a technique called gradient descent to achieve this.

An important neural network in the field of computer vision is Convolutional Neural Network (CNN), it is similar to MLP but the neurons in CNN are not fully connected hence provides more efficient computation and requires less parameters. Unlike

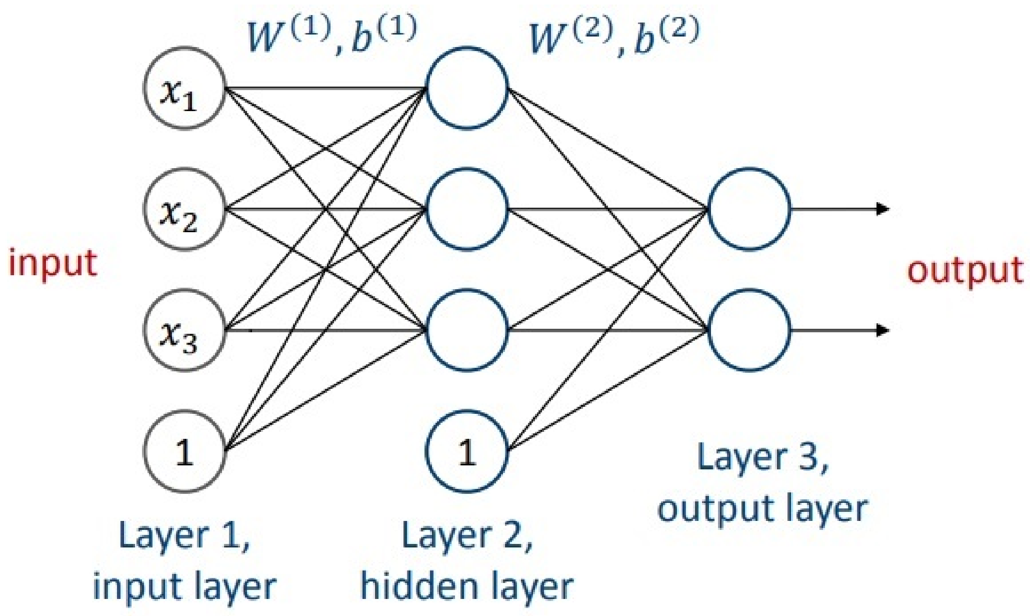

Figure 2.2: A multi-layer perceptron

MLP, a neuron in CNN only depends on a small local region in the previous layer, we can increase the amount of neurons that correspond to the same local region and this is called depth. We move across the orginal image to obtain a convolutional layer (a cube of the dimension XYD)

2.2.2 Types of problems machine learning solves

There are two types of problems that machine learning solves: Regression problem and classification problem. In regression problems, we are trying to predict a quantity value, in this investigation, grading coursework is a form of regression problem. We try to predict the score of a coursework given some input. Classification problems is an instance of supervised learning, using a training set as a basis, we try to identify which set an observation belongs to. The corresponding unsupervised learning instance is called clustering which group data into categories based on measures of similarity or distance. This is particularly useful in Natural language processing when we classify words into entities. Also when we experiment providing feedback in additional to grading when evaluating a piece of coursework, we can classify the output into different categories to provide a corresponding feedback. We shall explore this in depth in our experimentation.

2.2.3 Supervised Learning

K-nearest neighbour

KNN is a non-parametric classifier, it assigns the class of the test data according to its nearest neighbour according to a distance metric. When k = 1, the test data has the

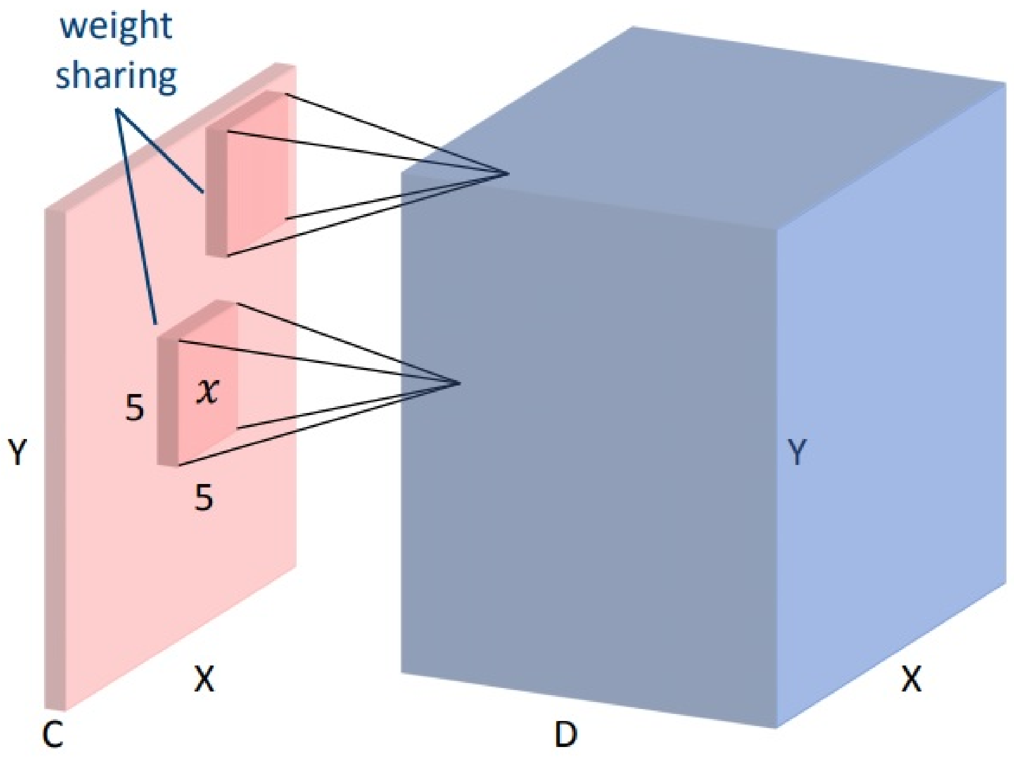

Figure 2.3: A Convolutional neural network

class of the nearest neighbour, when k > 1, the class is determined by a majority vote of the nearest trained data. The most typical distance metric we use is the Euclidean distance which is the L^2 norm of the vector between x and y, this is generalised to the L^p norm as shown:

$$D_p(x,y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$$

We can also use other distance metric such as the cosine distance:

$$D(x,y) = \frac{x \cdot y}{\|x\|\|y\|}$$

Support Vector Machine

The Support Vector Machine is an algorithm to solve the binary classification problem by finding the best line that separates the two classes. In general the linear model(the line) is given by:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

where w is the weights, x is the data and b is the bias. The rule to assigning which class c data x belongs to is given by:

$$c = \begin{cases} 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b = 0 \\ \text{-1}, & \text{otherwise} \end{cases}$$

We train the model by finding a maximum margin hyperplane, which is determined by the parameters **w** and b that best fit the training data. The most inner points that determines the maximum margin hyperplane are called support vectors, hence the name of the algorithm.

The problem can be formulated as an optimisation problem

$$min_{x,b}\|\mathbf{w}\|^2$$

To eliminate possible mistakes in the cases where the points are not linearly separated, we add slack variable to tolerate mistakes, this is later formulated as the hinge loss. It then becomes an optimisation problem with this function:

$$min_{x,b}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x_i} + b))$$

where C is a hyperparameter.

There are two approaches to solve this, first, by gradient descent and calculating the subgradient; Second, by solving the Lagrangian duality problem which we will not discuss in detail.

2.2.4 Unsupervised Learning

K-means clustering

This is an important technique in classification, it is also very useful in computer vision for instance when we perform image segmentation. The way we classify data points are summarized in the following few steps:

1. Assign each data point to a cluster centre

2. Update the cluster centres

3. Repeat step 1 and 2 until there is no more change to the class membership of all data points.

The cluster centres are defined randomly in the beginning, we then assign each data point according to the nearest cluster centre, in the optimal case, the intra-class variance is minimized. This can be formulated as the following mathematical expression:

$$min\sum_{k=1}^{K}\sum_{x\in C_k}(x-\mu_k)^2$$

where C_k refers to the cluster k, x is the data point, μ_k is the mean of all the data point in cluster k i.e. the centre.

To determine how many clusters to use, we decide by looking at the intra-class variances and plot a graph of variance against number of clusters , the point that gives the greatest change in gradient is usually a good point.

2.3 Natural Language Processing

2.3.1 Overview

Natural Language Processing is for marking and providing feedback on text responses. There are a number of NLP techniques which gives useful output. Most of them combine with machine learning techniques to solve regression or classification problems.

2.3.2 NLP techniques

(Inflectional Review of Deep Learning on natural Language Processing)*

According to Fahad and Yahya**?**, there are six main stages in NLP to ensure results are more sustainable and accurate by implementing DL.

Splitting and Tokenization: Splitting cleans the document and separates unwanted content, it determines sentence boundaries. By tokenising the document, we parse the stream of text into words, phrases, symbols using delimiters which can then be processed easily.

Part of speech tagging: We assign part of speech labels, such as noun, verb, adjective to each word in the document. Different tagging techniques are used to improve the accuracy of the NLP result.

(RBA for Arabic POST and NER)*

Below are the most famous tagging methods**?**:

1. Rule-based tagging first assigns multiple potential parts of speech to a word using a lexical dictionary, it then uses disambiguation rules to eliminate items in the list until each word only has one part of speech.

2. Stochastic tagging uses a tagged corpus to get probabilities of how likely a word belongs to a particular tag, it then choose the most likely tag.

3. Transformation-Based tagging combines Rule-based tagging and Stochastic tagging. There are other approaches such as combining neural networks with rule-based approach to remove ambiguity as proposed by Soufiane

Morphology: This is the study of the relationships between words in the same language.

Named Entity Recognition: An important task to identify unstructured texts and categorize them into named entities such as names, locations, organizations, time expressions etc. Common approaches includes rule-based, machine learning based and hybrid methods that combines the two. A new approach proposed by Liang Zhang uses a convolutional neural network to remove the need to man-made name entity features, instead, the system learn the word features by itself.

Syntactic Parsing: The stage where we assign a syntactic structure to a sentence. For example, a geographical name can refer to a place where some event occurs (”This company is investing 1 billion USD in China”) or a geographical object of no particular interest (”The great wall is in China”)

Coreference Resolution: According to the Stanford Natural Language Processing Group **?**, “Coreference Resolution is the task of finding all expressions that refer to the same entity in a text.” This means identifying all antecedent and anaphor of an entity. For example, in the sentence ”I like Nick because he gets along well with a friend of mine,” she said. Here “Nick”, “he” refers to the same entity and “I”, “mine”, “she” refers to another entity.

2.3.3 Sentiment Analysis

It is a useful technique to analyze the sentiment of a word or a sentence. By using NLP techniques, we can split and tokenize the sentence and apply sentiment analysis so that we know whether a sentence is positive or negative. This is particularly useful when we need to analyse the subjectivity of the author of the text.

Chapter 3

Method

3.1 Overview

To investigate the possibility of integrating machine learning techniques into smart learning environments, we look at a piece of coursework from the third year computing graphics course. In this piece of exercise, the students are required to write GLSL code to create the correct Phong Shading on the Utah teapot model. In our experiments, we try to use machine learning techniques to provide feedback as the student is writing the GLSL code. This is achieved by feeding pictures of the graphics scene and use computer vision to identify the error the student is making. We pre-train the ML models with our dataset which represents possible errors that the student could make and use the trained model to predict the error the student is making.

Originally the coursework is carried out using the ShaderLabframework - an OPen GL framework written by prof kainz **ref** to provide an interface for writting GLSL shader code. In our investigation, instead of reusing the Shaderlabframework, we reimplement the framework on the web, combining javascript machine learning technologies, the final product an inclusive webapp that allows easier set up and more flexibility in terms of deployment and extension.

3.2 Graphics

3.2.1 The coordinate system

In computer graphics, objects are defined in world coordinate system with an x,y and z coordinate. In order to transform the 3D object into 2D screen space, we require a graphics techniques called projection. Projection transform a higher dimensional space to a lower dimensional space, in our case, a 3D space to a 2D space. To project an object to a surface, we select a viewpoint then define projectors(lines) that join each vertex of the object to the viewpoint **(Graphics notes 1 figure 3)**. The point where the projection plane intersects the projector is the definition of our projection of that particular vertex.

Perspective Projection

There are two common types of projections - parallel projection and perspective projection. In parallel projection, the viewpoint is considered to be at infinity, hence all projectors have the same direction **d**. If we look at a vertex $\mathbf{V} = (V_x, V_y, V_z)^T$, its projector can be represented by the following parametric equation

$$\mathbf{P} = \mathbf{V} + \mu\mathbf{d}$$

The type of projection we will be referring in the rest of the paper is perspective projection. In this type of projection, all projectors are passed from the viewpoint (centre of projection). To simplify the problem, we usually make two assumptions when calculating the projections.

1. The viewpoint is at the origin
2. The projection plane is set at a constant z value, i.e z = f

Making these assumptions allows us to use the canonical form for calculations. The projector through the origin has the equation

$$\mathbf{P} = \mu\mathbf{V}$$

At the point of intersection, since z = f for the projection plane, we have:

$$f = \mu V_z$$

And if we write the parameter μ for the intersection point as $\mu_p = \frac{f}{V_z}$ we get:

$$P_x = \frac{fV_x}{V_z}$$

$$P_y = \frac{fV_y}{V_z}$$

include graphics fig6 from lecture notes 1

Space Transformations

In order to match our assumptions above on perspective projection, we need to transform the coordinates of the scene such that the viewing direction is along the z-axis and the viewpoint is at the origin. We can achieve this by multiplying every point in the scene by a transformation matrix. We will not go into details of how the matrix is generated since the viewpoint is fixed in my experiments and the coordinate system transformation is implicit. **Notes 1, figure 8**

3.2.2 The scene

The Utah Teapot

Utah teapot is one of the world's first complex object that is generated by the computer graphics, it has blah blah blah and therefore is used for the graphics coursework. The Utah teapot model I used in the experiments has less vertex than the Utah teapot model in the original graphics coursework. The resulting scene is therefore less smooth, however, all the useful features are preserved so this is not a problem. The reason behind is mainly because of computational complexity. Since I have to generate thousands of teapot images, reducing the complexity in graphics rendering lightens the load of the GPU and allows future extensions such as texture rendering to render quicker. The bottom of the center of the teapot is positioned at the origin in the world coordinate system, facing the positive z-axis.

Camera

The camera (viewpoint) is fixed at x = 0, y = 1 and z = 3 in the world coordinate system, enough to view the whole teapot showing all necessary features. The original framework allows the camera to move when dragged by the user, however, geometry is not an assessed area for this coursework, therefore we fix the camera for simplicity.

3.2.3 Phong Shading model

Add picture of complete Phong, ambient, diffuse, specular alone

The focus of the coursework is the Phong Shading model, it is a lightning model in 3D graphics to calculate the illumination of objects. The model is empirical and has no basis on physics, it contradicts the physics law of conservation of energy. However, the model is simple compared to other graphics illumination techniques such as ray-tracing. It is based on four assumptions:

1. Point light source meaning lights emit from a single point
2. Surface geometry is not taken into consideration
3. Ambient reflection is modelled globally
4. Diffuse and mirror reflection is modelled locally

The student is required to implement Phong shading using GLSL code. The Phong shading equation can be divided into three parts and has the general equation

$$I_{out} = I_{ambient} + I_{diffuse} + I_{specular}$$

Ambient Term

The ambient term is independent of any light source, it is empirically defined by a predetermined intensity I_a and a material constant k_a

$$I_{ambient} = k_a I_a$$

Diffuse Term

The diffuse component is dependent on the incident angle of the point light source but independent of the viewing position, meaning light is reflected in all direction.

We achieve this by finding the component of the incident light ray along the direction of the surface normal. FigureX(Lecture6 slide19)

$$I_{diffuse} = k_d(cos\theta)I_{in} = k_d(\mathbf{n} \cdot \mathbf{l})I_{in}$$

where k_d is the reflection factor empirically defined for the diffuse component, I_{in} is the intensity of the illumination from the point light source, the angle θ being the angle between the normal vector **n** and the unit vector for the direction of the incident ray **l**.

Specular Term

The specular component calculates the ideal mirror reflection of the incident light ray, it is dependent on the angle of reflection of the light source according to snells's law and the angle of the viewing position. The diagram below illustrate the situation. **Diagram lecture 6 slide 27**

The snell's law states that the angle between the surface normal and the ideal reflection direction is the same as the angle between the surface normal and the incident ray direction: $n_i sin\theta_i = n_r sin\theta_r$; $n_i = n_r$; $\theta_i = \theta_r$. In order to find the amount of light reflected in the direction of the viewer, we project the reflected light ray onto the direction of the viewer, which gives the following equation:

$$I_{specular} = k_s(cos^n\theta)I_{in} = k_s(\mathbf{r} \cdot \mathbf{v})^n I_{in}$$

where k_s is again some material constant determined empirically, I_{in} is the intensity of the lumination of the incident ray, θ is the angle between the ideal reflection of the incident light ray **r** and the viewing direction **v**, n is a constant exponent to decribe how reflective the surface texture is - 1 being a very rough surface and ∞ being a perfect mirror.

Therefore, the entire phong shading equation can be written as

$$I_{out} = k_a I_a + I_{in}(k_d(\mathbf{n} \cdot \mathbf{l}) + k_s(\mathbf{r} \cdot \mathbf{v})^n)$$

where k_d + $k_s \leq 1$ and $k_a \leq 1$

3.3 Machine Learning

Machine learning is core of our research, we use it to identify the error the student is making and provide suitable feedback to guide them through the exercise. We will explore different neural networks and different types of classification to predict the error.

3.3.1 Training Data

Since the coursework is on the phong shading model, we try to replicate some of the possible errors made by previous students in the training data. Since the colour and the position of the light source is chosen by the student, we generated training data with random light positions and colour combinations. The specular component is always going to be white. **Figure of a bunch of correct phong shading models**

Below are the images of the Utah teapot separated into different components, the background shading is controlled by the ambient term $I_a = k_a I_a$, the gradual change in illumination across the teapot is controlled by the diffuse term $I_d = k_d(\mathbf{n}\cdot\mathbf{l})I_{in}$ and the white dot representing the mirror reflection is controlled by the specular term $I_s = k_s(\mathbf{r}\cdot\mathbf{v})^n I_{in}$, in which **r** is calculated **n** and **l**

This is shown in the figureKK

The training data is changing over the course of our experiment, we first start with a 4 class classification, then, we extend the number of classes to 10. After all, we create more training data to perform multi-label classification given the fact that the student may make more than one of the errors. **See Appendix for all the figures: A list of all types of training data and its corresponding error: Full explanation of what the errors do and how it is done**

multi-class classification

In the very beginning of the experiment, we separate the data into the following classes with three most obvious possible errors:

class 0 ground truth: Correct phong Shading model

class 1 error: **n** is not normalized before calculating $\mathbf{r}\cdot\mathbf{v}$ which is affected by **n** in the specular component. This results in an ”edgy” specular reflection shown in FigX

class 2 error: No specular term results in no reflection spot on the image, see FigX2

class 3 error: No diffuse component, this makes the gradient of the change in colour zero across the teapot in places other than the specular reflection spot.

See figures

Extending the 4 classes to 11 classes

After successfully training the 4 classes on a MLP network, we extend the classes to 11, this includes errors in missing important features due to misunderstanding of the concepts in the phong shading model and mixing up important terms such as swapping $\mathbf{n}\cdot\mathbf{l}$ and $\mathbf{r}\cdot\mathbf{v}$ in the calculations of the diffuse component and the specular component. Below is some examples of the extended classes. **See full list in appendix Figure**

Multi-label classification

In order to better replicate the errors the student make, it is important to use multi-label classification as the student can make more than one mistake at a time. We extend the original dataset into dataset with multiple errors, a notable example would be no diffuse term, no ambient term, light is not normalized before calculating the reflection in the specular term and mistakenly reversing the direction of the light vector. This results in a black and white image with the illumination of specular lighting and the background inverted.

Add Figure for 3-4-5-11

3.3.2 Data Preprocessing

Before feeding the training data into the ML network, data is preprocessed to amplify certain features and reduce the size of the input data to reduce network load. There are two phase of data preprocessing decision with the latter an improvement to the original data.

The original preprocessed data for 4 class classification

We preprocess the data according to the following measures:

1. Crop the image to the area of interest - the area of the specular lighting, which is enough to classify between the 4 classes

2. Resize the image to 28 * 28 to speed up the network

3. Normalize the image to reduce sensitivity to extreme value

4. Make the image greyscale to compress the RGB channels into one channel.

figure for greyscale images

Improved preprocessed images for the extended datasets

After an initial experiment we found that the training data does not provide consistent network training results, we therefore made a few amendments to the data.

1. Since the data is extended, features are not limited to the specular lighting area, we crop the images such that the entire teapot is just visible.

2. Although colour is not one of our major concerns, it is important to preserve colour feature to allow better identification of features, putting the images to greyscale makes it harder the identify the gradient of illumination across the teapot.

3. A major drawback normalizing the images is that we compress and scale down the pixel intensities, this reduces the distinctive feature of the specular lighting since the difference between the specular lighting and other parts of the images is reduced, in the case when the ambient colour and the diffuse colour is very similar to that of the specular colour, there will be difficulties for the network to identify these important features.

Figure for after data

3.3.3 Machine Learning methods - MLP

There are two neural networks used throughout the experiments - multilayer perceptron and convolutional neural network. Because of the fact that all neurons in the MLP is fully connected, we have to reduce the size of the input hence all the data preprocessing as described in **sectionXX**, however, using CNN avoids this problem by not having a fully connected network, we can also use techniques such as pooling, stride etc. to quickly downsample the image. As a result, CNN produce more consistent results since we loss too much details on the original image when preprocessing training data for the MLP. **Unreliable conclusion**

A MLP is used for the initial 4 class classification network, it consists of three layers: a relu layer as the input layer, a linear hidden layer and a softmax output layer, the network diagram is shown below.

Diagram for MLP network

Relu Layer

The Relu (rectified linear activation) layer has the following equation

$$relu(x) = \begin{cases} x, & if \quad x > 0 \\ 0, & otherwise \end{cases}$$

Softmax Layer

The softmax layer is essential for multi-class classification as it provides us the probability of the input belonging to each class, it compresses the sum of the output vector to 1 so we can choose the class with the highest predicted probability to be our prediction, it has the following equation:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_k e^{z_k}}; z \in \mathbb{R}$$

3.3.4 Convolutional Neural Network (CNN)

Convolutional neural networks are heavily used in computer vision since it reduces the number of parameters needed in the network without compromising the original features in the input data.

Multi-class classification

The CNN for multi-class classification has the following architecture:

Figure for architecture

The first layer is a 2d convolutional layer with some filter and some strides, we use relu as our activation, and the kernel is initially scaled using the variance scaling method -**elaborate**. We then start a maxPooling layer with **stride x poolsize x** to downsample the image. We then use a second convolutional layer followed by a max pooling layer to further extract the features. The output from the 2D filters is the then flattened into a 1D vector followed by a softmax layer to output the final classification results.

Categorical cross entropy

The loss function used for the multi-class classification problem is categorical cross entropy, it has the following equation:

$$L(y^{(i)}, a^{(i)}) = -\sum_k y_k^{(i)} \log(a_k^{(i)})$$

where k is the number of classes, i is the number of layers, y is the desired output and a is the predicted output. In the final layer, we use the softmax activation function to get a probability distribution and we try to minimise it. For example, our desired output for the class 1 error (not normalizing **n** before calculating $\mathbf{r} \cdot \mathbf{v}$) is [0,1,0,0,0,0,0,0,0,0], suppose currently we have [0.05,0.5,0.05, 0.05,0.05,0.05,0.05,0.05,0.05,0.1], minimizing the categorical cross entropy would push the output to [0.001,0.991,0.001,0.001,0.001,0.001,0.001,0.001,0.001,0.001].

Multi-label classification

The architecture of our network in Multi-label classifiation is similar to that of multi-class classification, a major difference is that the we need to output 0s and 1s in the output vector rather than 0s and only a 1. To achieve this, the activation function in the final layer is changed to sigmoid so that we predict the probability of the image belonging to each class separately, and we define it being in a certain class by defining a threshold

Elaborate how we get the threshold

Binary cross entropy

The binary cross entropy is a negative log likelihood function and has the following equation:

$$L(y^{(i)}, a^{(i)}) = -(y^{(i)} \log(a^{(i)})) + (1 - y^{(i)}) \log(1 - a^{(i)})$$

Maybe some elaboration

3.4 text-feedback

Generalize NLP/ Term for easy feedback

The feedback based on the classification result is the key to our investigation, in our first approach, we generate preset text as the feedback base on our classification result. Below is the text for each class

Neural Talk NLP

3.5 AUTOTEACHER Architecture

AUTOTEACHER is a web application that integrates the graphics coursework framework and machine learning in order to output real time feedback to guide the student through their coursework. We can separate the app into two parts, the front end, responsible for the UI and generating teapot graphics and training data; The back end server, where all the machine learning, training and feedback generation is done. **Architecture diagram: figure**

3.5.1 UI

Not created yet

3.5.2 Graphics Rendering

All graphics data are generated in the front-end, this is mainly written in JavaScript. We will cover more about how we generate the graphics scene later in the implementation **(Section XX)**.

Can talk a bit about graphics pipeline and its relation to the browser windows

All graphics are rendered on a HTML canvas element then captured as a PNG image, this is later sent to the back end as training data and user data for real time prediction

3.5.3 Connecting the front-end and the back-end

Two ways of communication is used in the web application, HTTP requests and Web Socket

HTTP requests

HTTP requests are mainly used to send the generated training images from the front end to the back end, POST request is used. In order to receive the images correctly, a custom parser is built to restore the images and the images are saved locally on the computer.

Web sockets

In order to provide real time feedback to the student, we need to maintain an ongoing connection between the client and the server such that the graphics generated by the student is immediately passed into the pre trained model to generate feedback.

3.5.4 Server

There are three major tasks carried out in the server:

1. Communicate with the front end for data transfer
2. ML
3. NLP

For data transfer, we create connections in HTTP/Web socket. Mainly parsing incoming photos and save them locally and sending feedback back to the front end **For more See section XX**

For Machine learning, we preprocess the image, convert it to tensors and feed into the machine learning model to train and to predict the corresponding class(es) that the image belongs to. **See section XX for more**

For NLP, we pass the prediction result to a text generator, and it will generate the corresponding feedback according to the predicted class for the user image. **See section XX for more**

Chapter 4

Graphics System

4.1 Reimplementation of Shaderlab framework

The Shaderlab framework is originally developed in C++ and OpenGL by PROF Kainz **ref** and his team as the coursework platform for the Graphics course at Imperial College London for 3rd year students, it contains a series of shaders including vertex,fragment, R2T fragment shaders, geometry etc. each serving a role in graphics rendering such as applying shading to the model, alternating the geometry of the shading. Using this framework, students are able to implement numerous tasks by writing GLSL code. Using this framework requires some knowledge in C++ compilation and might take some time to set up, in fact the first coursework for the graphics course is on getting familiar with the ShaderLab Framework, which in some degree deviates from the sole purpose of learning pure graphics concepts. Moreover, most of the support for the set up of the Shaderlab framework in on Linux machines, meaning that it could be a challenge to setup the framework if the student wants to work on a home laptop that runs on Windows/MAC OS.

Figure X shows the user interface for the shaderlab framework**Figure**

In our experiment, we focus on one of the most important coursework in the course: The Phong Shading exercise, where students are required to complete the GLSL code for Phong Shading on a Utah teapot model, **see section XX for the theory**. To provide a more comprehensive experience to the students, instead of extending the ShaderLab Framework and linking it to some machine learning backend, which requires a lot of installation knowledge from the student, our final decision is to migrate the system to the browser and use a javascript backend for machine learning purposes to create a complete and functional online platform such that students can implement the required GLSL code without the need to handle compilation problems and is cross platform compatible.

To achieve this, we use the web version of OpenGL - WebGL, in which a graphics scene can be implemented using javaScript and GLSL in the browser.

4.1.1 WebGL

WebGL, a DOM API for 3D graphics, is created based on OPENGL ES 2.0.. It uses the OPENGL GLSL Shading Language and therefore inherits most of the features from the standard openGL API. The major difference between WebGL and OpenGL is that WebGL is fully integrated into the browser, it renders on the <canvas> element and can be combined with HTML and other web frameworks. **ref WEBGL kronos group**

4.1.2 Three.js Library

The Three JS library is a high level javasrcipt 3d library built on top of WEBGL. It provides sufficient functionality for us to implement the Phong Shading model while abstracting away the complicated low level geometry in the graphics rendering pipeline. It works well with html, to use the library, we simply import the three.js script in a script tag and we can start using the library.

4.2 My Implementation

The implementation of the Graphics system consists of mainly the use of threeJS library and some pure JavaScript to capture the pictures and send it to the backend. **Figure shows a general pipeline of our graphics system**

4.2.1 Defining the Scene

In all three js graphics, we need to first define a scene by calling the function THREE.Scene(); Then we need to define a camera with input fields setting the camera frustum, here we use a perspective camera instead of orthographic camera **see section research method**, below is a visualization of the camera's viewing frustum **Figure: Picture of a camera viewing frustum**; After that, we need a WebGL renderer to render the scene. Have a look at the following code:

```
var scene = new THREE.Scene();

var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.
    innerHeight, 0.1, 1000 );

var renderer = new THREE.WebGLRenderer({preserveDrawingBuffer: true,
    autoClear: true});

camera.position.y = 1;
camera.position.z = 3;

renderer.setSize( window.innerWidth, window.innerHeight );

document.body.appendChild( renderer.domElement );
```

The camera position is set to world coordinate (0,1,3) since the bottom center of the teapot model we will be using is at (0,0,0)

The size of the renderer is set to be the width and height of the browser window, this is the case for early experiments when generating the training data. However, changes are made later mainly for the purpose of the UI. In line 12, we add the renderer element to the HTML document body, the scene is then rendered on the <canvas> element.

4.2.2 Loading model

To load the teapot model, we use a three js object loader together with a loading manager such that we ensure a synchronous operation of loading model, loading GLSL Shaders code and rendering the scene. The original Utah teapot model is generated using Blender 3D **Ref: A graphics library**, the resulting model is stored in a json file with the vertex position array stored in the geometry->data->attributes->position attribute. The model is later changed into teapot-claraio.json which is from the original three js demo library, the reason behind is that it has more vertices and greater precision, hence, a more refined teapot can be rendered. The teapot model is then loaded as an Object3D object in three js, and we move on to adding shaders written in GLSL code.

4.2.3 Writing GLSL Code

The GLSL code we use consists of two parts, the vertex shader and the fragment shader. In the vertex shader we take the model view matrices, vertex position and calculated the normal, normalized position in camera space and pass it through to our fragment shader. In generating training data, we also pass a seed value to generate random numbers in the fragment shader.

In the fragment shader, we have two functions and one operation in the main function. The functions **float rnd(float seed)** is to generate a random number according to the normal distribution **TO BE VERIFIED** given a seed value, which ranges from 0 to 2000 in my training data. The function **change_sign(float seed)** changes the sign of the random number if the number is greater than 0.5, this is useful when creating random lighting positions since the centre of the bottom of the teapot is at world coordinate (0,0,0), hence negative numbers for the lighting is needed **Figure, random lighting position effect: positive lighting, negative lighting**

Ultimately, what we want to achieve in the fragment shader is that we would like to use the values passed in from the vertex shader and other pre defined values to put into the phong equation and overwrite the original output colour which gives a flat shading. Recall the Phong Shading equation in **Refer to Section XX**, in practical terms, the Phong Shading equation depends on numerous values and we will go through them below. If we separate the input lighting of the diffuse and specular terms, we can rewrite the phong equation as follows:

$$I_{out} = k_a I_a + k_d(\mathbf{n} \cdot \mathbf{l}) I_d + k_s(\mathbf{r} \cdot \mathbf{v})^n I_s$$

In generating the training data, the constants k_a, k_d, k_s, n, I_s are preset to 0.3, 0.6, 0.9, 30.0 and vec3(1,1,1)(The colour white) where the ambient colour I_a and the diffuse colour I_d are generated randomly, the reason for choosing these constants is that they generate a clear picture and give distinct differences between different terms which helps in the classification problem we are trying to solve later.

The lighting is generated according to the random number generator function described above, it has the following xyz range for its world coordinates:

x value: -10.0 to 10.0

y value: 0.0 to 5.0

z value: 10.0 to 15.0

Make sure you know whats happening in the Phong shading code + elaborate
After finding the values of **n**, **l**, **r**, and **v**, we need to calculate their dot products, note that we are not interested in the negative numbers hence we put the values of the dot products to 0 if we encounter a negative number. This can be achieved by using the **max** function. The final output after applying phong shading is shown below in the code.

```
gl_FragColor =
vec4(
ka * ambientColour +
kd * rDotV * diffuseColour +
ks * pow(nDotL, shininess) * specularColour
, 1.0);
```

4.2.4 Linking GLSL Code to the model

Adding the GLSL shader to the model is done in the user defined function **applyMaterials()** with numerous helper functions each handling part of the operation.

To link the GLSL Shaders code, we can either embed it in our JavaScript inside a <script> tag or import it from an external file. Embedding the Shaders code in Javascript helps with performance since the loading time is greatly reduced, on the other hand, importing from an external file gives more flexibility on which shader to choose from and lots of shaders can be written in advance, this is helpful but not necessary in generating training data and would automate the process. To import from external files, we also need to auto version the files since the chrome browser automatically caches the name of the input files which causes a problem when we are generating the training data as we quickly change the shader code by passing a different seed value to the random number generator.

After loading the teapot instance as an Object3D we call the function **applyMaterials()**, it traverses the teapot object itself and all its children (If there is a group of objects) and invoke the function **addShaders()** for each child. In the function **addShaders()**, we return a new THREE Shader material which accepts a seed value to generate the random values in the shaders and link the vertex Shader and fragment Shader by referring to apropriate objects, we shall discuss the two ways of linking the shaders below.

Embed GLSL code in JavaScript

To embed the GLSL code, we put it in a JavaScipt tag and give it a type ”x-shader/x-fragment” so the program recognises it as shaders code and we can link it by giving it an approprate id.

```
<script type=”x–shader/x–fragment” id=”vertexShader”>
// Shaders code goes here
</script>
```

Listing 4.1: Example for linking the Vertex Shader

We can then add the shaders in the **addShaders()** function.

```
vertexShader: document.getElementById('vertexShader').textContent
```

where vertexShader is an attribute of THREE.ShaderMaterial

Import GLSL code from external file

To import GLSL code externally, we need to load the files before applying it to the materials. We accomplish this by using the file loader from THREE js and pass the loaded files as an input to our model loader. The loaded files will then be applied as materials in the loaded model.

Both methods are implemented, embedding the shaders code is used in the final version because it gives better performance.

In the end, the loaded model with the required Phong Shading will be added to our scene for rendering.

4.2.5 Rendering in the browser

Now we have scene ready to be rendered, we define an animate function which calls the function **requestAnimationFrame()** which takes the function itself as the callback and we call loadModels() to load the model, apply phong shading and add the completed model to the scene. After loading the model, we call renderAndPost() which renders the scene using our defined scene and camera.

```
renderer.render(scene,camera)
```

By calling **requestAnimationFrame()**, we ask the browser to repeat the same operation in the next frame, this results in refreshing and re-rendering the scene 60 times every second.

4.2.6 Generating the training data

To generate the training data, we need to change the seed value for every picture we want to capture, we also need to ensure the system is synchronous such that we do not send multiple copies of the same scene to the server.

Varying the seeds

To vary the seed value, we simply set the seed to 0 and increment it by 1 each time the function animate is called, we pass the seed value into **loadModels()** which is later passed into the shaders.

Capturing and sending the training pictures to the server

After loading each model and rendering it, we write a function **generateTraining-Data()** which first captures the rendered image on the canvas element and change it into an octet-stream ready to send to the server then send the octet-stream to the server using fetch with a POST HTTP request.

4.2.7 User GLSL system

To be implemented

4.2.8 Handling training data from front-end

The server accepts the POST request from the browser and parse the data to save it as a PNG photo. After the data is sent to the server, there are many unwanted white spaces and headers in the body that needs to be taken away, we wrote a custom parser to process the data and restore its original form, the recovered image is then saved locally with unique names assigned.

Chapter 5

Machine Learning

5.1 Overview

Using the data generated from the browser, we are able to save and use the training data to and from the server. In this chapter, we will explore different machine learning techniques to solve the classification problem. The original idea is to use multi-class classification to identify the type of error made, this is later extended to multi-label classification since in most of the cases, the students are likely to make more than one error.

This problem can be seen as a computer vision problem as we are only looking at the pictures of the teapot to determine its error class, a useful and modern approach is to build a neural network as the complexity in the input data is very high, there is a lot of parameters, methods such as MLP will be less effective when we increase the amount of training data. However, having said that, it is useful to investigate other possible ways of solving the problem as there could be a better solution than building a neural network.

5.2 Data Preprocessing with JIMP

Since our backend is written using the Node JS framework, it is good to use JIMP for image processing. JIMP stands for JavaScript Image Manipulation Program, we considered using JIMP because it is written entirely on javascipt which requires no other dependencies and is well integrated into Node JS and the npm web package.

JIMP provides support with files in JPEG, PNG, GIF and some other image formats, it also provide trivial image manipulation methods such as blur, normalize, resize, crop etc., this is more than sufficient for our image preprocessing procedure.

5.2.1 Processing images

There are three directories used to process each image in each class - raw, active, exported, as suggested by the names, the original training data generated from the browser is saved in raw, the image we are currently changing is stored in active and the exported directory stores all the image after processing. The files are stored according to the experiment then into these three categories, it would be relatively complicated if I store the three directories for each image, therefore, to access a certain image, for example,the 3rd active image in class 2 under experiment 3, we would use the following path

training_data/exp3/active/class_2/training_3.png

The reason we do this is to avoid data corruption and overwriting the original data. Because of the fact that the operation is asynchrous, we use **.then** in javascript to make sure the steps are synchronous to avoid situations when exported images are not fully processed.

In the function **preprocessPhoto()** we first create three variables: imgRaw, imgActive and imgExported to store the paths to the directories of the three forms of the current image under operation. We then read the raw image and clone it into imgActive and store it. After that, we read the active image and carry out the necessary operations and export the image to imgExported. The function **PreprocessPhotos()** then processes all photos in one call.

Just elaborate the grayscale stuff in ore detail in the methods section

5.3 Working with Tensorflow js

5.3.1 Comparing different Machine Learning Libraries

There are numerous machine learning framework projects out there, below we will focus on the most popular ones and see how compatible they are with our exeperiment.

scikit-learn

It is a very useful library and is popular, it provides great support on traditional machine learning algorithms and analysing tools for these algorithms. However, the support on neural networks is limited and therefore is not suitable for our project, which mainly focuses on computer vision.

Keras

Keras is a high level machine learning framework built on top of tensorflow and is very convenient to use. It would be great for our project since we can abstract away a lot of complicated details in tensorflow. It requires less code to build similar networks. This comes with a price though, there are less flexibility on both how we treat our data and building the networks.

Although Keras seems like a great choice, its JavaScript developement has stopped in August 2018 and all future developments are explored in Tensorflow js. It is outdated and not a great choice for our project.

Tensor Flow

Tensor flow is a Machine Learning framework developed by Google and is widely used for building neural networks and developing deep learning algorithms. It focuses on computational efficiency, this is very useful for our project since we have images as our input.

5.3.2 Current development of Tensorflow and Tensorflow js

Tensorflow is originally written in python, and the latest developments are supported in python only. It has great support in building neural networks and accept data in its self defined data structure: Tensor. Due to the popularity of web applications and machine learning, the JavaScript version of Tensorflow has started its development. The current latest version of tfjs is version 1.1.2 which supports most of the core functionality of tensorflow and is sufficient for our experiment, although there are a few things missing such as some functions in metrics and performance measure. Also, a very inconvenient thing about tfjs is that it does not provide the necessary bindings for tensorboard, a dash board application to visualise the neural network's performance, which is available in tensorflow written in python.

5.3.3 A close look at Tensorflow js

Tensorflow js is built in JavaScript and there is a node js binding called tfjs-node module that is available from npm, this exactly suits our needs since the server is built using node js.

The core to all versions of Tensorflow are Tensors. This is the main object that we will using and manipulating in our tfjs program. A tensor represents a computation that is partially computed and it will produce a result eventually. Tensorflow works by building a graph of Tensors in the beginning to store how each tensor is computed based on other tensors, then it runs part of the tensor graph to get results.**Reference, tensorflow guide tensors: tf-core**

Tensor

A Tensor has a dataType(float32, string for example) and a shape (The number of dimensionse and their sizes). For most of the time the value of a tensor is immutable, however, evaluating the same tensor twice might result in different return values.

Rank

The Rank of a Tensor is the number of dimensions it has, this is different from mathematical terms in linear algebra, for example, to create a Tensor with rank 0, we only need to pass in a value. This is shown in the following code snippet

```
code = tf.scalar(1)
```

In our experiment, we mainly work with 1-d and 2-d tensors which can be intuitively be thought as lists and arrays. The code below how we can create a 2-d tensor and print it out. We can pass in a nested array, but in our case, we would pass in a flat array and specify its shape since it is easier to process our input image as a single large array than giving it multiple dimensions.

```
tf.tensor([3,4,5,6],[2,2]).print()
```

Printing tensors to debug

TO BE WRITTEN: NOT SUPER IMPORTANT THO

5.4 Image to Tensors

Before we start our training, we need to regroup, assign ground truth value and convert the image data into tensors. In all the experiments, the most complicated situation in data conversion is in multi-label classification, since it covers all other data conversion steps, we will use the multi-label classification to explain this section.

5.4.1 Loading data and associating the target classes

The first step to data conversion is to load the preprocessed data from our file system. We do this by first creating an empty array **dataArray** to store our loaded data and their corresponding ground truth class. Then we create a canvas by using the 'canvas' module from npm and call its **createCanvas()** function. After creating an empty canvas, for each class and each preprocessed image in that class, we draw the image on the canvas context. By calling **tf.browser.fromPixels()**, we are able to extract the image on the canvas as a tensor. At this point, because we need to change the data so that we can store the target class associated to each image, we change the tensor into an array and append the corresponding target class and append the entire array into dataArray which store all the information of all images in all classes. Below is a simplified version in pseudo code of how it is done.

```
var dataArray = [];
const canvas = createCanvas(28,28);
const canvasContext = canvas.getContext('2d');
for all images:
    const image = await loadImage('IMAGEPATH');
    canvasContext.drawImage(image, 0, 0);
    const tensorObj = tf.browser.fromPixels(canvas, 3);
    const values = tensorObj.dataSync();
    const arr = Array.from(values);
    arr.push(c);
    dataArray.push(arr);
```

5.4.2 Generating training, validation and testing tensors

In this function, we take in the dataArray, which we created previously with information of all images and their target classes appended at the end of each image data array, and separate them into two arrays, the first one stores the image data separated by class and the second one stores the ground truth. They are then passed into the function convertToTensors() which takes the two arrays from a class and a test split to create the required tensors for the training, validation and testing data. This process is repeated for all classes and in the end we will get 6 arrays namely xTrains,yTrains,xValids,yValids,xTests,yTests, which stores all the required tensors as suggested by the name. They have equal samples from each class to ensure all data is unbiased towards any particular classes. Below we shall discuss how we actually convert the data from a class into tensors.

Convert to tensors

In **convertToTensors()** we take in two arrays - data, which stores the image data and targets, which stores the target class for the corresponding data. Note that the sequence is very important here. We also take in the test split, a float that ranges from 0 to 1. The number of test and validation samples are calculated by the total number of samples $*$ split $*$ 0.5 and the rest of the samples are made into training data.

There are two operations carries out here. The first step is to convert the image data into tensors and the targets into 2d tensors with a one hot encoding. The following code snippet shows how it is done.

```
const xs = tf.tensor2d(data, [numExamples, xDims]);
// For multiclass classification
const ys = tf.oneHot(tf.tensor1d(targets).toInt(), NUM_CLASSES);
// For multilabel classification
const ys = tf.tensor2d(label_to_one_hot(targets));
```

For simple cases we can convert it simply by calling tf.oneHot on the targets specifying the number of classes we have. In multilabel classification there are multiple labels for each class and therefore we wrote a custom target decoder and one-hot encoder to turn the labels to one-hot encoding tensors.

can write a sesh on decoder and the encoder

The second step is to slice them suitably to get the 6 different array for data and targets, this is then returned and pushed to the xTrains, yTrains... arrays in the caller.

5.5 Supervised Learning

With the training, validation and test tensors, we are now ready to pass them into our machine learning models for training. Different attempts are made and we try to improve the network over time.

5.5.1 SVM

Maybe Depends

5.5.2 MLP

The very first experiments is carried out with 4 classes, **refer to class 0 - 3**, and we attempt to classify them with using a multi-layer perceptron network in which all nodes in each layer is connected to all the nodes in the adjacent layers.

Since it is a fully connected network, we try to reduce the size of input images, therefore in this experiment the images are of single channel (grayscale) and are cropped and resized such that we only look at the specular reflection region. We can identify all errors by solely looking at the region, the features to be learnt are summarized below. For class 0 (truth), there are all the features: a smooth specular shading that is shaped similar to an eclipse; a slight change in the spectrum of the colour created by the diffuse lighting. To identify other classes, class 1 (no normalization) shows a rough and pointy specular lighting, class 2 (no specular) is missing the specular lighting, hence there is no sudden change in the colour spectrum, class 3 (no diffuse) shows a flat shading in the background and the correct specular lighting as usual, the resulting data shows large region of the same shading value with a sudden change in pixel intensity in the specular lighting area.

Network Construction

We build a simple MLP with several layers The first layer is a relu layer with something...

Parameter tuning

TBC

5.5.3 Moving over to convolutional neural network

5.5.4 CNN

5.5.5 Improved CNN

5.5.6 MultiLabel CNN

5.6 Other Learning methods

5.7 Predicting user Shading

5.7.1 Saving and Loading Trained Model

5.7.2 Prediction and generating feedback

5.8 Comparison, results visualization

Chapter 6

Implementation

6.1 Implementation

6.2 Research Tools

6.3 The ShaderLab framework and its reimplementation

Theoretical description of an overall approach ML, system architecture - chosen scheme etc. ML formalise it to tensorflow etc, training scheme (5-10) overview figure, overview of framework separate into npm part and ml network Implementation refer back to framework - how to do it practically

6.3.1 The shaderlab framework

The shader lab framework is orginally developed by ... it does this and that

6.3.2 Migration to Webgl and three js

The shader lab framework is reimplemented so that student can write GLSL code in the browser which controls the shader and generate the teapot on the canvas. The reason behind this is that using webgl we can eventually build a web application that does all the things and provide immediate feedback

Three js

More on THREEJS Three.js is a web-gl library that provides some functionality where we can abstract the need to write out everything in plain web gl. Its initial intention is to build quick graphics in the browser. Allowing APIs to import models and rendering. **How we use three js** However, instead of using the built-in part of the phong-shading equations, we create bindings to external files where we edit GLSL shaders code and apply it onto a three js model as a material, defining all the illumination of the scene. This part is also controlled by the student which is part of the graphics coursework.

6.4 Generating training data

How we render the scene with animation and catch what is on the screen and serialize the data to the backend

6.4.1 Data preprocessing

All training data passed from the front end is preprocessed using JIMP

Techniques 1. resize 2. crop 3. normalize 4. greyscale

6.5 General architecture of the app

6.5.1 front end

The front end is written in javascript with embedded react for UI

6.5.2 back end

The backend is written in node js

6.5.3 Communication

The front end and the backend is communicated using http requests - get and post requests mainly; Also possibly websockets
There are lots of challenges associated to todays learning processes due to its increasing complexity and scale. It is harder for teachers to observe, evaluate and give feedback to students piece of work. On the other hand, machine learning techniques are recently a very popular topic and there are a lot of research of machine learning in application to different industries. The capability of machine learning in solving complex problems. It is therefore interesting to investigate into how machine learning techniques can be associated to smart learning environments.

In this paper we use machine learning techniques to solve different problems and to provide feedback to students. Here, we first investigate how to give accurate and precise grading and later extend to providing accurate yet constructive feedback to students. We use the Imperial hardware coursework as a starting point and investigate the possibility to extend the feedback system to a more general context.

The aim of the project is to create a machine learning grading and feedback system, we shall discuss the ability of different machine learning approaches, also the possibility of migrating the system into a general purpose system which could evaluate other of coursework.

Chapter 7

1st Experiment

7.1 Overview

7.1.1 training data

7.1.2 Working with Tensorflow js

7.1.3 MLP layers network

7.1.4 Performance and Analysis

Chapter 8

Improvements

8.1 Changing the network architecture

8.1.1 Extending to use CNN

8.1.2 Extending to perform multi-label classification