



Trinity College Dublin

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

School of Computer Science and Statistics

**A mobile exam proctoring system with
deep-learning-based human action recognition**

Tong Ge

Supervisor: Dr. Ciarán Mc Goldrick

A dissertation

Presented to the University of Dublin, Trinity College
in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

(Future Networked Systems)

August, 2021

Declaration

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

Signed: _____

Date: _____

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Signed: _____

Date: _____

A mobile exam proctoring system with deep-learning-based human action recognition

Tong Ge, Master of Science in Computer Science
(Future Networked Systems)
University of Dublin, Trinity College, 2021
Supervisor: Dr. Ciarán Mc Goldrick

Abstract

The COVID-19 pandemic has prevented students from congregating to take traditional in-person exams, shifting the attention of pedagogical institutions to online exam systems accessed remotely. This research surveys previous review papers on human action recognition to confirm the feasibility of obtaining action features from spatiotemporal data sets such as videos. This research reviews multiple state-of-the-art deep models and optimisation methods under the premise of achieving a balance between performance, function and resource demand constraints of mobile devices. Then a deep learning model is developed to classify the behaviour of examinees to a series of activities, approved or prohibited by the exam holder. The model evaluation carefully considers the model performance in machine learning classification metrics and computational performance. Experimental results show that the proposed model achieved over 90% accuracy over 12 exam-related action categories with 23.6M parameters, making it possible to run inference smoothly on most modern mobile phones. Finally, this research concludes with a discussion of model design and optimisation experiences with possible directions for future research in this area.

Keywords: Deep learning, pose-based features, video data, classification, mobile device, exam proctoring

Acknowledgements

First and foremost, I would like to express my profound gratitude to my supervisor Associate Professor Ciarán Mc Goldrick, who has provided me with constant support and guidance through each stage of this dissertation. With his patient guidance and gentle encouragement from time to time, I complete this study with my effort and propose directions for future research.

I am deeply grateful to Trinity College Dublin and the School of Computer Science and Statistics for teaching me the theoretical foundation, practical techniques of computer science and innovative research methods. In addition, the use of web hosting services and the support of computing power have provided great help for data collection and model training in this research.

Last but not least, I am thankful to my solicitous parents for their warm cares and financial funding for supporting me in studying abroad and upholding my chosen academic path.

TONG GE

*University of Dublin, Trinity College Dublin
August 2021*

Contents

Declaration	i
Abstract	iii
Acknowledgements	iv
List of Figures	vii
List of Tables	ix
List of Algorithms	ix
Abbreviation	x
1. Introduction	1
1.1 Research background.	1
1.2 Research aims	2
1.3 Research ethics	3
1.4 Dissertation overview	4
2. Literature review	5
2.1 Data features and classic methods.	5
2.2 Online exam security system	7
2.3 Deep models detail.	8
2.3.1 Convolutional neural network	8
2.3.2 Recurrent neural network	11
2.3.3 Attention and Transformer	13
2.4 Mobile device optimisation.	17
2.4.1 Mobile optimisation overview	17
2.4.2 Evolution from MobileNet to EfficientNet	18
2.4.3 Optimisation of Transformer Networks	20
2.5 Related works	22
2.5.1 Human action recognition overview	22
2.5.2 3D-CNN and RNN based methods	23
2.5.3 Transformer-based Neural Networks	24
3. Design	27
3.1 Data set collection	27
3.1.1 Requirements of data set	28

3.2 Data preprocessing.	29
3.2.1 Face detection with Viola–Jones algorithm.	30
3.2.2 Face landmark extraction and face concealment	31
3.3 Deep model design	32
3.3.1 Data loader and input layer.	32
3.3.2 The model architecture	33
3.3.3 Loss function and training hyperparameters	34
4. Implementation	36
4.1 Data collection web app	36
4.2 Mobile app implementation	37
4.2.1 Architecture overview	38
4.2.2 Display and image processing pipeline.	39
4.2.3 Tensorflow Lite adoption and model inference	41
5. Evaluation	43
5.1 Evaluation metrics	43
5.2 Deep model evaluation	44
5.3 Mobile app evaluation.	48
6. Conclusion	50
6.1 Summary	50
6.2 Future work.	51
Bibliography	52
APPENDIX A. Code repositories	59
APPENDIX B. Framework details	60
Appendix B.1 Web development.	60
Appendix B.2 Deep learning model development	61
Appendix B.3 Android application development.	62
APPENDIX C. Web recorder designs	63
Appendix C.1 Functionality description in user stories.	63
Appendix C.2 Participant use cases and user logics.	64
APPENDIX D. Model design details	65
Appendix D.1 Output categories	65
Appendix D.2 Model detailed layers	66
APPENDIX E. Evaluation result	68
APPENDIX F. App showcase	69
Appendix F.1 Web-based data collection app	69
Appendix F.2 Deep model equipped mobile app.	73

List of Figures

1.1	Project overview	4
2.1	Cubic volume of video data	5
2.2	Slicing the video cube parallel to the X-Y axis	5
2.3	CNN applications in vision by LeCun, Kavukcuoglu and Farabet [19]	9
2.4	Residual building blocks by He et al. [22]	11
2.5	VGG and Residual ImageNet by He et al. [22]	11
2.6	RNN, unfolded unidirectional RNN and unfolded bidirectional RNN	12
2.7	LSTM and GRU structure comparison by Phi [27]	13
2.8	Scaled Dot-Product Attention [31]	14
2.9	Multi-Head Attention [31]	14
2.10	The Transformer model architecture [31]	14
2.11	Multi-head attention matrix visualisation [32]	15
2.12	Compare the architecture of ResNet-50 and DeiT-S (ViT-S)[34]	16
2.13	Depthwise, pointwise and atrous convolution [39]	18
2.14	Compare residual block and inverted residual block [40]	19
2.15	The architecture of baseline network EfficientNet-B0 [41]	19
2.16	Lite Transformer block [43]	20
2.17	Full self-attention and attention patterns proposed in Longformer [44]	20
2.18	2D convolution and 3D convolution [47]	23
2.19	Video Action Transformer Network architecture [53]	24
2.20	Video Transformer Network architecture [54]	25
2.21	Video Vision Transformer model and factorised variants [55]	26
3.1	Haar Cascade structure [57]	30
3.2	Landmark masked face concealment and face landmark schematic	31
3.3	The data flow	34
4.1	Mobile app architecture and data flow	38
4.2	Sequence diagram of the multi-threaded app	39
5.1	Data per category distribution	45
5.2	Learning rate warm-up	46
5.3	Model training steps	47
5.4	Model validation steps	47
C1	New participant registration	64
C2	Registered participant login	64
C3	Participant recording and uploading	64

E1	Confusion matrix on validation data set	68
E2	Confusion matrix on training data set	68
F1	Welcome screen	69
F2	Consent form screen top	69
F3	Consent form screen bottom	70
F4	New participant register screen	70
F5	Task selection screen	71
F6	Recording screen	71
F7	Review task details	72
F8	Upload screen	72
F9	App welcome screen	73
F10	App register	73
F11	Filled app register	74
F12	Mock exam list	74
F13	App drawer in mock user	75
F14	App user profile	75
F15	Drawer after profile update	76
F16	Deep model loading	76
F17	Result in dark theme	77
F18	Result with mouth open	77
F19	Leave result	78
F20	Look down result	78

List of Tables

1.2	Research objectives and corresponding methods	3
2.1	Comparing tasks in NLP and video understanding	22
2.2	Video Transformer Network experiment results [54]	26
3.1	Model input tensors	33
5.1	Confusion matrix in binary classification	43
5.2	The evaluation metrics, the formulae are summarised by Foss [61]	44
5.3	Classification report of validation data set	48
5.4	Experiment results on mobile devices	49
C1	User stories for the video data collection app	63
D1	Model output for exam activity categories	65

List of Algorithms

4.1	User authentication	37
4.2	Video uploading	37
4.3	Image processing pipeline thread	40
4.4	Temporal sampling and triple buffering	42

Abbreviation

<i>AuC</i>	Area under the Curve (ROC Curve)	<i>PoS</i>	Part-of-Speech
<i>BERT</i>	Bidirectional Encoder Representations from Transformers	<i>R-CNN</i>	Region-based Convolutional Neural Networks
<i>BoT</i>	Bottleneck Transformer	<i>ReLU</i>	Rectified Linear Unit
<i>CNN</i>	Convolutional Neural Network	<i>ResNet</i>	Residual neural Network
<i>JSON</i>	JavaScript Object Notation	<i>RGB</i>	Red-Green-Blue
<i>JSP</i>	Jakarta Server Pages	<i>RNN</i>	Recurrent Neural Network
<i>JWT</i>	JSON Web Tokens	<i>ROC</i>	Receiver Operating Characteristic
<i>GELU</i>	Gaussian Error Linear Unit	<i>RPC</i>	Remote Procedure Call
<i>GRU</i>	Gated Recurrent Unit	<i>SVM</i>	Support vector machines
<i>I3D</i>	Inflated 3D ConvNet	<i>VGG</i>	Visual Geometry Group
<i>LBF</i>	Local Binary Features	<i>ViT</i>	Vision Transformer
<i>LRCN</i>	Long-term Recurrent Convolutional Networks	<i>VTN</i>	Video Transformer Network
<i>LSRA</i>	Long-Short Range Attention	<i>ViViT</i>	Video Vision Transformer
<i>LSTM</i>	Long-Short-Term Memory	<i>UI/UX</i>	User Interface / User eXperience
<i>MLP</i>	Multi-layer Perceptron	<i>YOLO</i>	An object detection model abbreviated from You Only Look Once
<i>NER</i>	Named Entity Recognition		
<i>NNAPI</i>	Neural Networks API		
<i>PHP</i>	PHP Hypertext Preprocessor		

1 Introduction

This chapter introduces the background 1.1 of the current COVID-19 pandemic, leading to the research questions and aims 1.2 to be achieved in this research. Furthermore, since human participants are involved in data collection and model training, research ethics 1.3 are discussed. Finally, this chapter also presents the overview and structure 1.4 of this dissertation.

1.1 Research background

COVID-19 has brought many challenges to higher education. Marinoni, Van' t Land and Jensen [1] reiterate the facts released by UNESCO regarding the significant impact of the pandemic on education in countries around the world. For example, there are approximately 130 million students, accounting for 89.4% of total enrolled learners, whose academic career has been vitiated because of the impact of pervading virus caused school closures.

To better respond to the crisis and allay public concerns caused by the epidemic, the International Association of Universities (IAU) launches a survey of the impact of COVID-19 on higher education. Marinoni, Van' t Land and Jensen [1] present the survey results showing that COVID-19 has inevitably affected many processes in education, including teaching & learning, researching, and assessment.

In teaching and learning, most classroom lectures have been substituted by distance teaching and learning, which may not be difficult for professors and students who are familiar with computers and the Internet. On the other hand, experiments and research that require the use of shared professional equipment are much more severely affected. The survey shows that 80% of researchers have reported that their research progress is decelerating and moving at a creep.

However, as for assessment and examination, institutions have no way to perpetuate the traditional assessment methods, which is indicated by Clark et al. [2] as one of the most important challenges for students. "Learning assessment and examination

approaches will be reviewed, and institutions may choose to invest further in technical infrastructures”, said by Marinoni, Van ’t Land and Jensen [1] points out the fact. Although some universities shift from closed-book exams to pure continuous assessments immediately, other educational institutions are craving for new technologies to help them out of trouble.

As a result, the motivation of this research is to investigate the feasibility of deep-learning-based human action recognition techniques in the distance invigilation application and provide a way for educational institutions to hold exams and reduce labour costs.

In recent years, ubiquitous influences of deep learning methods have brought tremendous advancement to many computer science fields and individual’s daily life. In many fields, such as computer vision and natural language processing, deep learning approaches achieved higher performance than traditional algorithms. McCay et al. [3] show that in the research field of automatic human action recognition, the analysis and reconstruction of human actions have copious applications, including content-based video indexing, intelligent surveillance, human-computer interaction, and virtual reality. Therefore, this research will explore deep learning techniques to solve automatic exam invigilation problems in the assessment process of distance education.

This research is also based on previous studies in deep learning and deep model mobilisation. Currently, there are some existing models for human posture analysis and detection. But many of them use static imagery as input and key points of the body as output, and they are not optimised for mobile devices with limited computing power. Beside, the personal computer hardware, the bridge between the software system and the physical space, is not trusted. However, a trusted software environment can be ensured on mobile phones with hardware attestation functions provided by operating system and device manufactures.

1.2 Research aims

After clarifying the research background, the research question of this dissertation is summarised as the following:

Is it possible to understand sequential human actions, using an on-device camera video input stream with deep learning technologies, to classify the actions of an exam attendee in respect of approved or suspicious behaviours to a high level of accuracy?

In order to answer the research question, this research aims to achieve the objectives through the corresponding methods shown in Table 1.2.

• Research objectives	Method to achieve objectives
• To do the literature review	By surveying and analysing the state-of-the-art technologies of deep-learning-based human action recognition
• To have an available data set	By obtaining a suitable image and video data set and labelling them for training the model
• To design a deep learning model	By designing a deep learning model
• To code and train the deep model	By training the model and fine-tuning hyper-parameters
• To optimise the model for mobile	By developing a working mobile app that equips the model

Table 1.2: Research objectives and corresponding methods

After completing the above objectives, this research will also evaluate the project outcomes, especially for model performance and other following aspects.

- Model accuracy, precision and other evaluation metrics on the classification task.
- Balanced performance between accuracy and efficiency of the deep learning model.
- User experiences of the overall system.

1.3 Research ethics

According to the requirements of research ethics, any research involving human participation must have reviewed and got an ethical approval by the Research Ethics Committee. Obviously, this research is about human activity, and videos also need to be captured from participants as training and validation data set, so it requires some ethical discussions before conducting research.

The ethics committee raised privacy concerns during the review process that people in the videos have facial biometric information which directly identifies participants, and inadvertent capture of sensitive information from surroundings or computer screen might menace examinees' privacy.

In terms of privacy, Coghlan, Miller and Paterson [4] philosophically describe the ethical rationality of online proctoring technologies, which highlights "academic integrity, fairness, non-maleficence, transparency, privacy, respect for autonomy, liberty, and trust". Finally, they conclude that the online proctoring function requires a precedent balancing between concerns and possible benefits. Educational institutions also have the duty to take ethical considerations should they decide to adopt the technologies.

Bozkurt and Sharma [5] mention that laws and regulations should strive for online

data protection and privacy, such as the General Data Protection Regulation (GDPR) in Europe, which gives people more control over their data and regulates enterprises in storing and transmitting user data. In this way, with the continuous advancement of the security measures, invigilation and monitoring strategies in online exams, laws and regulations will protect learners from the side-effects of invigilation and monitoring related practices.

In order to alleviate the privacy concerns, this research uses anonymisation algorithms in the data preprocessing process and the mobile app to ensure that participants will be anonymised in the videos. This process will minimise the presence of any unnecessary material that might enable the identification of an individual, for example, to cover the detected face area with extracted facial landmarks. The detailed processing steps and algorithm details will be explained in chapter 3.

1.4 Dissertation overview

This dissertation comprises six chapters, beginning with this introduction chapter 1, presenting the research background, aims and ethics, followed by the literature review chapter 2, which comprehensively reviews previous researches on deep learning models and the related state-of-the-art works. In chapter 3, this dissertation proposes the deep learning model used to recognise and classify actions, based on various designing trade-offs and mobile optimisation methods learned from previous research. Figure 1.1 shows the project overview, in which these process pipelines and the proposed deep learning model are implemented in chapter 4, including the video capture and data collection app, model training and Android app development equipped with the deep model. Further, experiments are performed in chapter 5 to evaluate the performance of the model in terms of efficiency and accuracy and to compare it with other human activity recognition solutions. Finally, chapter 6 summarises the evaluation results and the experience of model design and optimisation, and proposes possible future work in this research area.

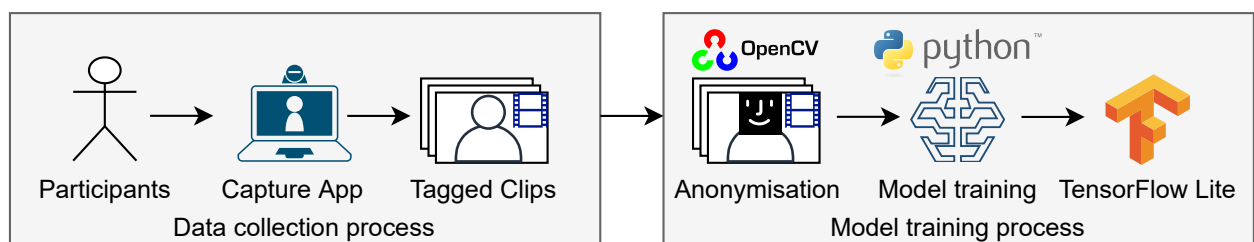


Figure 1.1: Project overview

2 Literature review

This chapter firstly reviews the spatial features of human poses and the temporal features involved in the video data set, as well as the classic methods used to processing these features in section 2.1. Then, section 2.2 surveys the application scenarios of this project, the online exam security system. Before studying the related research using deep learning, section 2.3 studies the details of several deep models comprehensively. After understanding the related concepts and basic operations and backbones of the deep models, section 2.5 reviews numerous human action recognition related works based on deep learning models. Finally, section 2.4 mentions some methods of optimising models so that it can run on devices with limited computing power such as mobile phones.

2.1 Data features and classic methods

To study the feasibility of this project, and to utilise deep learning methods to classify actions from sequential imagery features, videos containing pose-based data, comprehensively reviewing the previous research on data features and feature engineering methods is beneficial to the design of deep models.

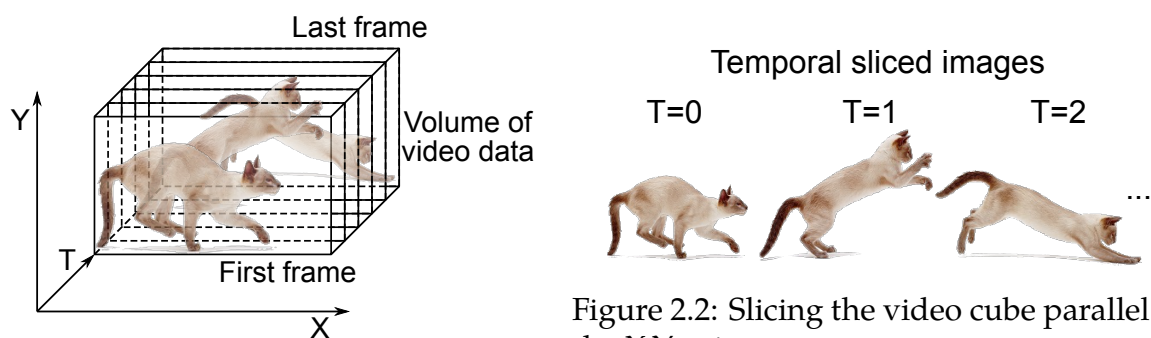


Figure 2.1: Cubic volume of video data

Video is composed of a sequence of image data and synchronised audio arranged in the timeline. If the audio data beyond the scope of this study is ignored, the 2D images can be stacked along the time axis to form a 3D cube representing the video. Figure 2.1 illustrates an intuitive representation proposed by Fels and Mase [6], which highlights the

spatial and temporal characteristics for all videos. Further, figure 2.2 shows that regular video frames produced by slicing the video cube parallel to the $X - Y$ axis so that computer vision algorithms or deep models for visual tasks can be applied. As a result, video can be simplified into a series of images to enable applying image processing technology.

In each static image containing human pose-based features, what features can be used for posture analysis to classify each image? Thrasher et al. [7] model the features worthy of attention in human upper body poses and then conduct a pose-based study on mood recognition by analysing and rating the postures of the head, shoulders, trunk, arms in video data. The data set in their research is closely analogous to the data set in this research because both data sets comprise human facial expressions and upper body poses.

From posture analysis in a single image to human action recognition in sequential images, it combines pose-based features and video-based data. For example, Yao, Liu and Huang [8] propose a paper on spatio-temporal feature extraction with a variety of application scenarios in automatic human activity recognition, such as video information retrieval, intelligent surveillance, and human-computer interaction. In future research, they point out that deep learning can be a powerful tool for processing these spatio-temporal features and thus will enable the low-level engineered features to fuse with a deep learning framework.

Before deep learning methods are prevalent in the computer vision field, some previous studies have built data sets for the human action recognition task and have conducted studies with classic machine learning methods.

Support vector machines (SVM), a well-known classic machine learning method, has been applied in many researches related to human action recognition. For example, Schuldt, Laptev and Caputo [9] extract local events such as size, frequency and velocity of moving patterns from videos and create a new video database, KTH data set¹, to evaluate the proposed human actions classification with the SVM method. Using the SVM method, they carefully model the data representation based on local spatio-temporal features and select appropriate features, kernel tricks, and hyperparameters to obtain the desired results.

Further, Marszalek, Laptev and Schmid [10] believes that human actions are highly correlated with background scenes, which means the context of the scene can help identify human actions in videos. They creatively use movie scripts and movie videos, Hollywood-2 data set², to train “a joint scene-action SVM-based classifier” with script-to-video align-

¹Recognition of human actions: <https://www.csc.kth.se/cvap/actions/>

²Human Actions and Scenes Dataset: <https://www.di.ens.fr/~laptev/actions/hollywood2/>

ment. They explained in detail how to get aligned target actions and target scenes from movies and also used \mathcal{X}^2 kernel function in SVM classifier, similar to the research by Schuldt, Laptev and Caputo [9]. In another example, Soomro and Zamir [11] focus on action recognition in realistic sports videos. They conduct research on action localisation and recognition using UCF Sports data set ³, including many sports actions collected from television channels.

By reviewing the early research in the human action recognition field, the spatio-temporal features in video data sets and the approaches of organising and building data sets are introduced. Unlike image classification tasks with numerous public data sets available, video data sets are always for individual research questions, so there is no public data set suitable for this research. As a result, creating a data set is a part of this research, detailed in chapter 3 and chapter 4.

2.2 Online exam security system

In order to learn about the background of the security system of remote exams and propose a new system that better adapts to the status quo, this research reviews the previous research on the security system of online exams in this section.

Before the COVID-19 pandemic, the technology of paperless testing through computer systems has been widely used. For example, TOEFL iBT (Internet-based test) has gradually replaced the PBT (paper-based test) by using the Internet and computers for paperless exams from late 2005. Another example is specific exams that are impossible to be sat on paper, such as programming competitions or exams requiring running compilers. Although all these exams were in the form of using computers and the Internet, students were required to congregate in testing centers.

As a result, most of the previous research in this field assumes that the exam can still be organised in test centres, so the research direction of exam security focusing on biometric authentication. For example, Traoré et al. [12] propose a multi-modal biometric authentication framework, including face biometric and dynamic biometric from computer input devices, such as mouse and keyboard. The purpose of biometric authentication is to prevent imposters in the exam, and the invigilators of the test centre should detect other cheating in time.

Nowadays, the pandemic segregates students at home for remote exams, invalidating the previous assumption. There are still some studies proposing new solutions for biometric authentication in exams. In Japan, Yasuda and Ogeta [13] propose a new continuous biometric authentication method based on hand image features, which can prevent

³UCF Sports Action Data Set: https://www.crcv.ucf.edu/data/UCF_Sports_Action.php

cheating, especially impersonation due to lack of invigilation. They use a mirror and a wide-angle lens to capture the images of students' hands when they take the exam and obtain the contour features of the hands through image processing. However, any biometric authentication cannot prevent cheating by the students per se, such as using mobile phones to search online or seeking help from others.

In 2020, Garg et al. [14] point out that many security issues still exist in online exams, and propose a system based on Haar Cascade Classifier and Convolutional Neural Network to detect, track, tag, and identify the student's face. Although this system innovatively uses deep learning models, only using facial features and constraints in the design is not comprehensive compared to action recognition. And another disadvantage is focusing on facial features may cause contention in privacy risks mentioned in the research ethics section 1.3. For example, students can still cheat with mobile phones during online exams even with the facial-based security system enabled.

After investigating many previous studies on online exam security systems, it is concluded that most of the research is limited to biometrics authentication, and there is no readily available security system that equips a deep-learning-based human action recognition model. As a result, this research originally applied the human action recognition technique to the online exam security system.

2.3 Deep models detail

Machine learning is a branch of artificial intelligence. Deep learning is also a branch of machine learning, which utilises artificial neural networks for feature learning and hierarchical feature extraction to replace manual feature engineering in other machine learning methods. Recently, the research of deep learning is a hot topic in the field of computer science, and it is also an important method used in the computer vision task in this research. This section will introduce the details of a variety of deep learning models related to this research, including CNN, RNN, attention and Transformer structures. Although the applications or target tasks of these models are different, the design ideas between them are mutually influential. Therefore, reviewing the development and design concepts of these previous models will greatly help the design of the model in this research.

2.3.1 Convolutional neural network

With the rapid development of neurobiology and cognitive science, the concept of artificial neural networks, a computational model that imitates the structure and function of biological neural networks, has been proposed as early as the last century. Fukushima

[15] proposed a network created from the animal visual system as well as some key concepts, such as **local receptive field**, **multi-layer perception architecture** and **translation equivariance and invariance property** (not affected by the shift in position).

In detail, the local receptive field means that each neuron will not perceive the image as a whole, but will only perceive the local information, then the local perception can be integrated through a multi-layer perception architecture to obtain the global perception. On the other hand, the property of translation equivariance and invariance is emerging from the combination of the local receptive field and multi-layer perception architecture.

The study of neurobiology and cognitive science then evolved into a computational model. For example, Zhang et al. [16] propose the first two-dimensional Shift Invariant Artificial Neural Network (SIANN). Then, LeCun et al. [17] proposed a original CNN with two convolutional layers and two fully connected layers using back-propagation method to train in a supervised learning approach. They also highlighted the term convolution for the first time, and named this model type as convolutional neural network (CNN). Soon after the application of this model in handwritten zip code recognition, Zhang et al. [18] applied it to a practical case of recognising medical imagery.

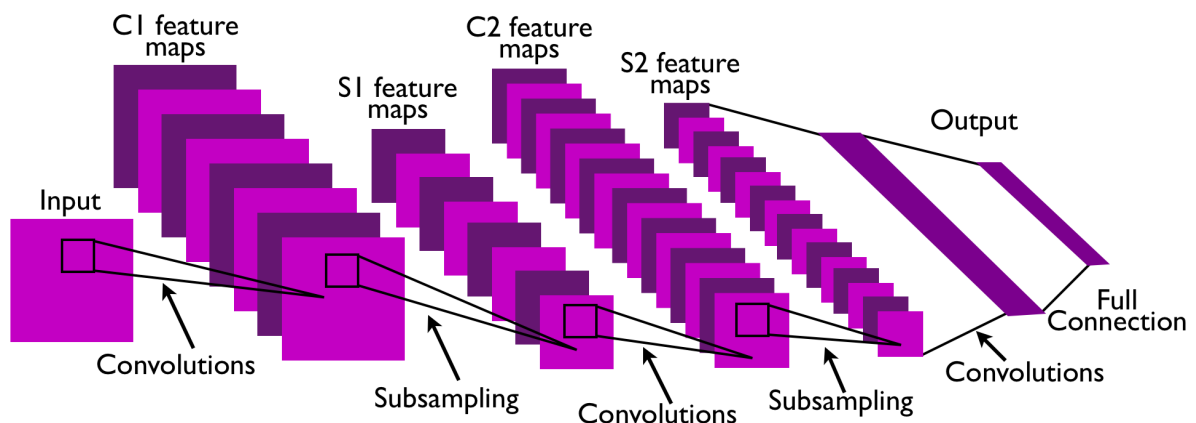


Figure 2.3: CNN applications in vision by LeCun, Kavukcuoglu and Farabet [19]

The most revolutionary progress is that LeCun, Kavukcuoglu and Farabet [19] proposed a modern architecture for this model in 2010, as illustrated in figure 2.3. This structure comprises three convolutional layers combined in between with two sub-sampling layers, and two fully connected layers. Sub-sampling, also known as pooling, reduces the size of feature maps, by retaining only important information to simplify calculation. Further, it further strengthens the translation invariance, taking maximum pooling as example, because the translation does not affect the maximum value, the pooling result remains unchanged.

In the following years of development of CNN, the number of layers in deep networks is gradually deepening to obtain greater high-level information. In 2012, Krizhevsky, Sut-

skever and Hinton [20] proposed AlexNet which use 5 convolutional layers and 3 fully connected layers. However, as the number of hidden layers increases, the model gets more complex and prone to overfitting. AlexNet uses the Dropout layer by randomly breaking neuron connection (setting random input units to zero) in training process to prevent overfitting. Two years later, Simonyan and Zisserman [21] proposed Visual Geometry Group (VGG) model, in which 3×3 size convolution kernels are fully used in a total of 5 layers, just as the title of the paper, it is a very deep convolutional network.

The experiment in the VGG model concluded that the deeper the number of network layers, the better the performance. However, as the depth of the model and the number of parameters continue to increase, the following two significant problems have been discovered.

1. The network is prone to overfit, requiring more training data, making it more difficult to train.
2. More storage resources and computing resources are required, but cannot provide adequate performance boost.

In order to solve these problems, He et al. [22] proposed a residual structure to make deep network training easier with enhanced performance but fewer parameters and lower complexity. They first identified that the root cause of these problems is that the deeper network structure leads to degradation, i.e., the training error and the verification error both increase since the deeper network does not learn anything but loses useful features. Then, figure 2.4 illustrates two types of building blocks used in ResNets with different number of layers. The first typical block just uses cross-layer connection, a linear layer from input to output connected directly. By denoting the desired underlying mapping as $H(x)$, the each layer learns the residual between input x and desired output $F(x) := H(x) - x$. As a result, such a residual structure allows the deep network to adapt to the appropriate depth, i.e., use the identity transform across unnecessary layers.

Further, figure 2.5 compares the network structure of VGG-19, 34-layer plain, and 34-layer residual and shows how to use the building block in the actual deep network. Figure 2.4 also shows a “bottleneck” building block for deeper ResNets, which first uses a 1×1 size convolution to reduce the dimensional across channels to reduce the calculation required in the 3×3 convolution.

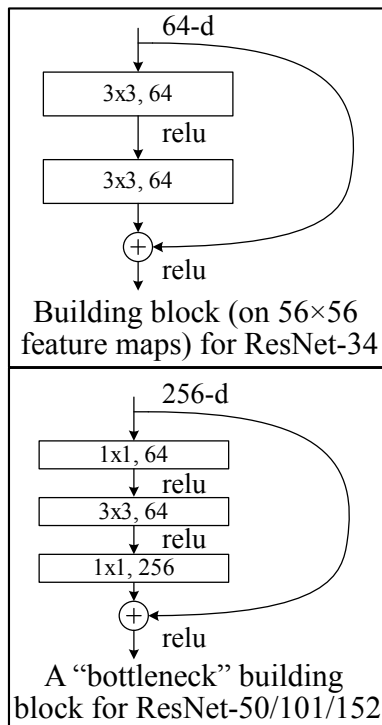


Figure 2.4: Residual building blocks by He et al. [22]

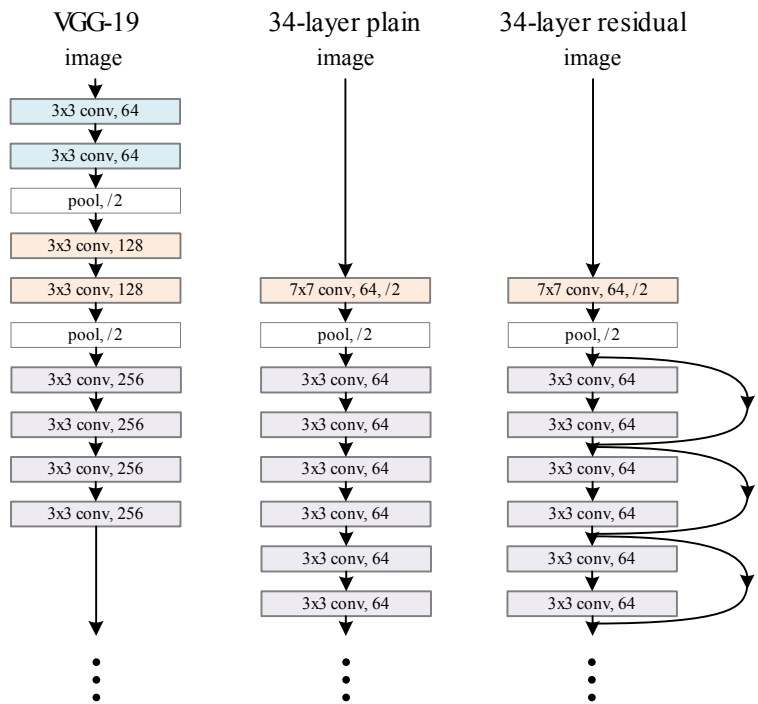


Figure 2.5: VGG and Residual ImageNet by He et al. [22]

2.3.2 Recurrent neural network

CNN solves the spatial problem of imagery and has been widely used in the field of computer vision, but it is powerless for data with time-series features. For example, it is necessary to understand the words sequentially in sentences in natural language processing tasks. And in understanding videos, it is necessary to understand the relationship between frames. In these tasks, the recurrent neural network (RNN) emerged to process the time-series features.

Jordan [23] proposed the Jordan network, by introducing a recurrent connection that feeds back the output of the whole network to the input layer after a time delay. Later in 1990, Elman [24] formally defined the recurrent neural network model, in which the output of each recurrent hidden layer goes to the subsequent layers and feeds back to the input of the layer after a time delay. The first figure on the left in Figure 2.6 shows the structure of the simplest recurrent network, with an input-output and a hidden layer, denoted as x , y and h , where the arrows indicate the flow of data.

The middle unfolded RNN figure shows the sequential structures by unfolding inputs and outputs of the network in chronological order. The input includes the initial hidden state a^0 , a series of data x^n , then the model generates sequential outputs y^n . However, the model with unidirectional memory has a limitation that the model can only know

the history data and cannot foresee the data that is about to be input. For tasks requiring two-way dependencies, such as natural language processing, each word has a strong relationship associated with other surrounding words, thus a bidirectional RNN structure is introduced as shown in the right figure in Figure 2.6.

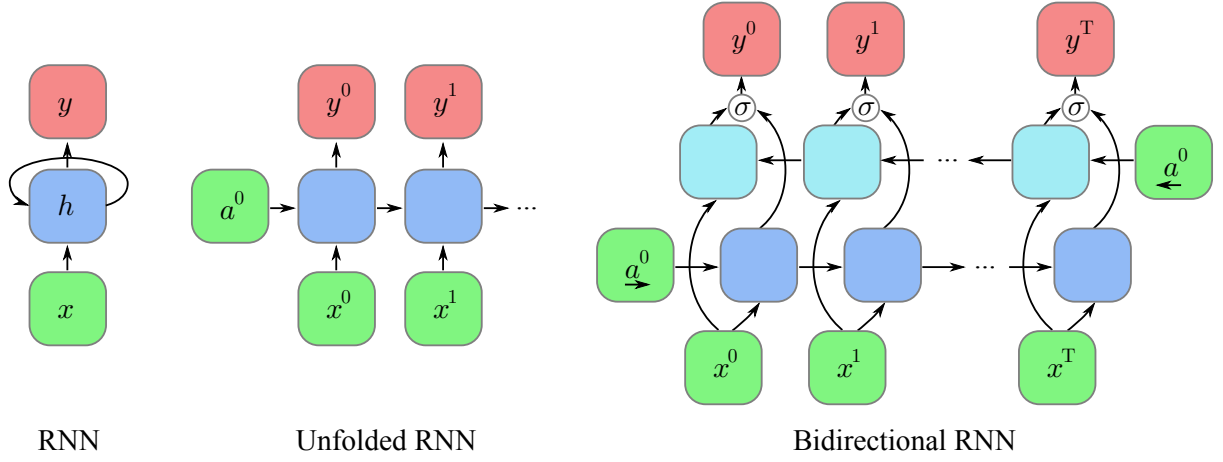


Figure 2.6: RNN, unfolded unidirectional RNN and unfolded bidirectional RNN

However, Hochreiter and Schmidhuber [25] pointed out that simple RNN tends to only keep short-term memory and is prone to lose long-term memory, because the same weight of the network is shared at all time steps, and the final gradient tends to disappear after multiple time steps, proved by Hochreiter’s analysis.

$$\frac{\partial \theta_v(t-q)}{\partial \theta_u(t)} = \sum_{l_1=1}^n \dots \sum_{l_{q-1}=1}^n \prod_{m=1}^q f'_{l_m}(\text{net}_{l_m}(t-m)) w_{l_m l_{m-1}} \quad (2.1)$$

Hochreiter and Schmidhuber [25]

Equation 2.1 shows that the back-propagation gradient in simple RNN is equal to the product of the gradients of each time step after multi-step propagation. An intuitive explanation is that the gradient is mainly dominated by the short-distance gradient, and thus it is difficult for the model to learn long-distance dependence.

Long-short-term memory (LSTM) was proposed to solve the problem that the simple RNN cannot learn long-term dependence by introducing a separated cell state C_t to keep long-term memory. And later on, Gers, Schmidhuber and Cummins [26] introduced I/O gates, a forget gate to control loss or keep memory in cell state, resulted in the modern LSTM cell structure shown in the left of Figure 2.7.

LSTM introduces lots of things into RNN, which leads to an increase in trainable parameters of each unit, making it harder to create a model with a deeper network. Gated recurrent unit (GRU) is an improved LSTM algorithm proposed by Chung et al. [28] in 2014. It merges the forget gate and the input gate into a single update gate. It also com-

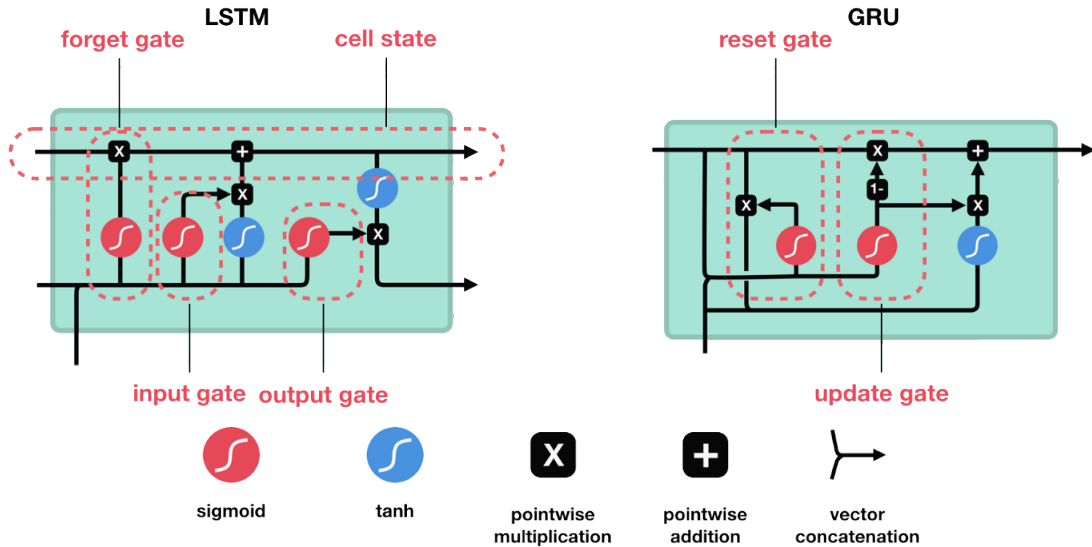


Figure 2.7: LSTM and GRU structure comparison by Phi [27]

bines the cell state and the hidden state. As shown in the right of Figure 2.7, GRU structure is much simpler than LSTM, making it possible to create a deeper network.

LSTM and GRU only remedy and alleviate the problem of gradient vanish to a certain extent in RNN, and memory loss can still occur in the case of long-range dependencies. Most importantly, this network structure has one limitation that made it not suitable for mobile applications. The enforced sequential calculation process is due to dependencies on the result from the previous time, which greatly limits the parallel capability of calculations, thus making it impossible to achieve reasonable performance on mobile devices.

2.3.3 Attention and Transformer

The attention mechanism originated from the human experience of perceiving things either visually or audibly. Intuitively speaking, when we observe something through sight, we do not pay attention to all the details but paying attention to a certain part that needs to be focused and giving low attention to the surroundings.

The attention mechanism was firstly added to recurrent neural networks as a visual attention modelling. In 2014, Mnih, Heess, Graves et al. [29] proposed a recurrent attention model that combined ideas in RNN and the attention mechanism for image classification and achieved good performance. Besides, the researchers discussed and reached forward-looking conclusions that it can be used in object recognition and video classification in future work due to the encouraging results achieved.

In 2016, Bahdanau, Cho and Bengio [30] firstly introduced the attention mechanism into

the natural language processing field. In their work, they perform translation and alignment jointly with the attention mechanism on machine translation tasks because “the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder-decoder architecture”, said by Bahdanau, Cho and Bengio [30].

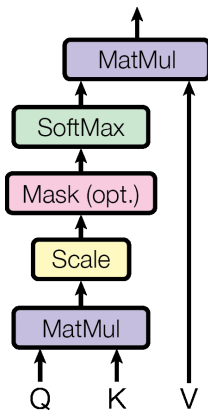


Figure 2.8: Scaled Dot-Product Attention [31]

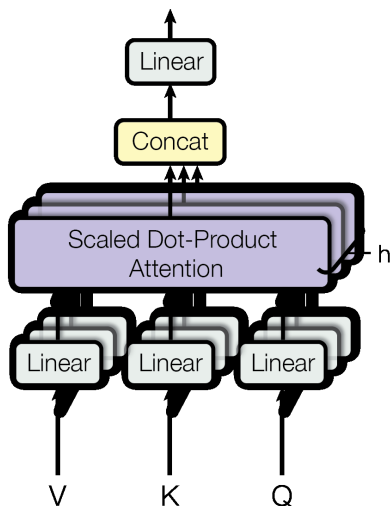


Figure 2.9: Multi-Head Attention [31]

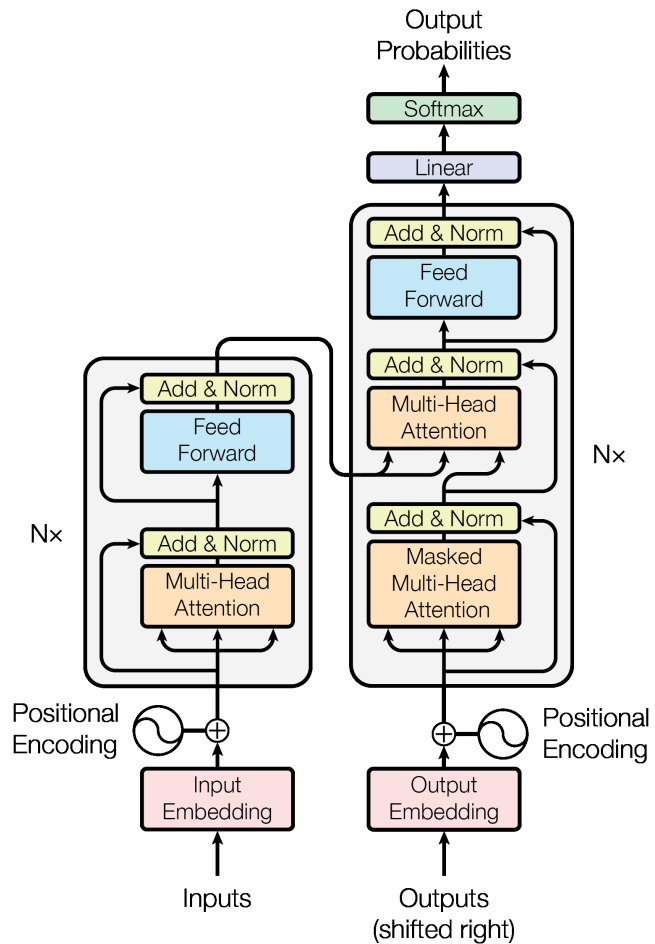


Figure 2.10: The Transformer model architecture [31]

One year after the attention mechanism was applied in the machine translation tasks, Vaswani et al. [31] proposed scaled dot-product attention (Figure 2.8), multi-head attention (Figure 2.9) and the Transformer model (Figure 2.10) that is “relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution”, said by Vaswani et al. [31]. Then, experiments were performed on the machine translation task using the proposed Transformer model and achieved better performance and results, making the attention mechanism the most preferred solution for natural language processing tasks.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.2)$$

Vaswani et al. [31]

Figure 2.8 shows the scaled dot-product attention mechanism, where Q, K, and V represent Query, Key and Value respectively. Further, formula 2.2 shows the calculation process of it mathematically. In this formula, the dot-product between Query and Key matrix represents their similarity, which is normalised by the square root of the dimension of d_k , since dot-product may grow large in magnitude, pushing the softmax activation function into regions where it has extremely small gradients. The result from softmax activation is normalised probabilities representing the attention matrix visualised in Figure 2.11.

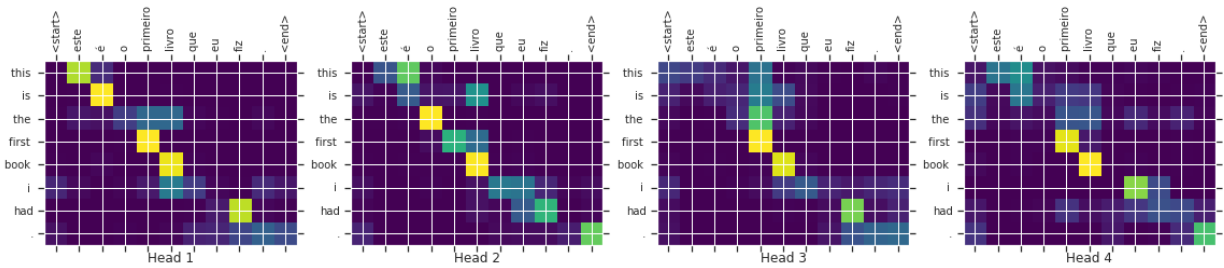


Figure 2.11: Multi-head attention matrix visualisation [32]

The idea behind multi-head attention is analogous in using multiple kernel filters in CNN to help the network capture richer feature sub-spaces on different aspects of data. Figure 2.9 illustrates the structure of the multi-head attention. It contains linear layers at inputs and the output, h times the scaled dot-product attention computation process, and a concatenation layer that combines the outputs of each scaled dot-product attention.

The building block in the Transformer consists only of multi-head attention and feed-forward network and does not contain any convolution or recurrent structure. These building blocks are stacked in multiple layers and connected in residuals to form an Encoder-Decoder structure that is the Transformer model shown in Figure 2.10.

Another detail worth mentioning in the Figure 2.10 is that the input content needs to add position encoding because this model is not sequential input, which implicitly includes position information. If there is no positional encoding added, Transformer will not be able to capture the order of input sequence, which degenerates it into a bag-of-words model in natural language processing tasks.

While convolutional neural network (CNN) is in the ascendant among the fields of computer vision, the Transformer model composed of pure attention mechanism has

achieved state-of-the-art results in many natural language processing tasks.

In 2018, Devlin et al. [33] published the Bidirectional Encoder Representations from Transformers (BERT) model, in which the Mask Language Model (MLM) and Next Sentence Prediction (NSP), two methods are used in the pre-training process to make Transformers capture the relationship between words and sentences. The amazing results in many NLP tasks from this model became the most important contribution that started a new era of NLP in 2018.

As a result, the Transformer model gradually replaced the RNN-based seq2seq model that inherently has sequential computing and long-term memory loss issues. Innovative design concepts from the Transformer model are also carried forward from the fields of natural language processing to computer vision territory.

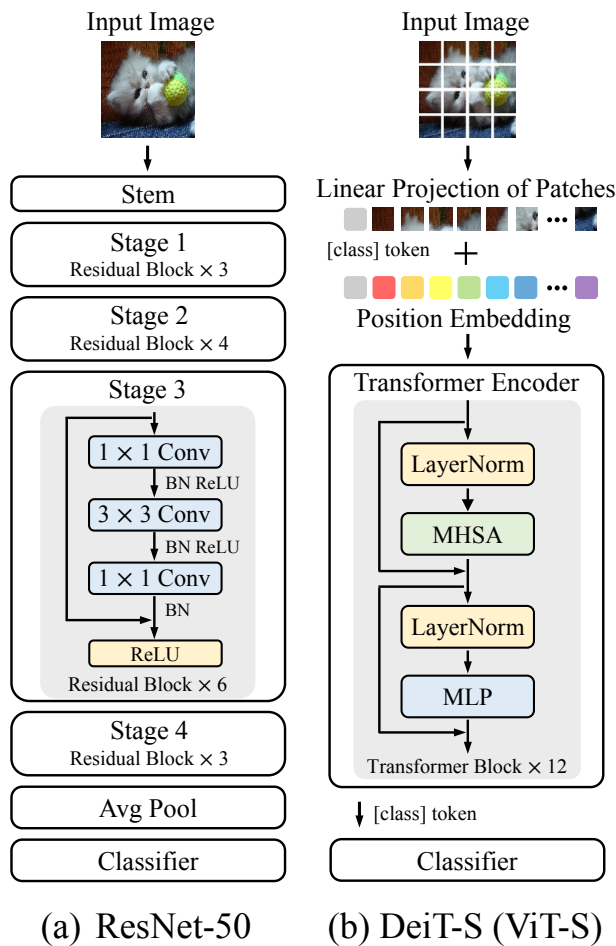


Figure 2.12: Compare the architecture of ResNet-50 and DeiT-S (ViT-S)[34]

Recent studies are exploring the application of Transformers architecture in the field of computer vision recently, such as Data-efficient image Transformers (DeiT) and Shifted window (Swin) Transformers. Many researchers have tried to introduce the attention mechanism in imagery tasks and compared model designs and results with widely-adopted CNN-only models such as ResNet, illustrated in Figure 2.12.

However, Dai et al. [35] pointed out that the performance of Transformers-based vision models still has a gap compared with state-of-the-art convolutional networks. They also showed that the generalisation is worse than convolutional networks because of the lack of the right inductive bias in the Transformers-based vision models. As a result, they proposed a hybrid model that married convolution and attention, leading the future research direction from this attempt.

2.4 Mobile device optimisation

After introducing the background and details of several deep models related to achieving the research goals, mobile device optimisation for deep models is another focus of this research. To protect user privacy by running model inference on mobile devices with limited computing power means model architecture, computational and spatial complexity need to be optimised. In this section, I will discuss the mobile optimisation research of deep models, especially the CNN and Transformers used in this research.

2.4.1 Mobile optimisation overview

In order to run deep learning models on mobile devices, it is necessary to understand the target platforms. Deng [36] investigated the support of the mobile hardware architecture and mobile operating system for deep learning, as well as the status quo of mobile deep learning frameworks, such as Tensorflow Lite, CoreML and etc. Then he explained the model compression techniques such as algorithmic optimisation and joint software-hardware co-design.

In 2020, Chen et al. [37] conducted an in-depth survey of the important compression and acceleration techniques on mobile devices, and classified them into “pruning, quantisation, model distillation, network design strategies, and low-rank factorisation.” Pruning and quantisation are the two techniques for compressing model parameters to save storage, while model distillation, network design strategies are the two techniques for improving the inference speed of model.

In detail, pruning is a compression technique to trim unimportant weights in models, then achieve model sparsity to allow compression algorithms to become more effective and reduce the storage cost. This compression technique may also bring about little speed increase, because some unnecessary residual paths may be short-circuited. Another model compression technique is quantisation that directly lower the precision of the model parameters, e.g. use 16-bit float instead of 32-bit float. The process can be carried out in the model deployment and conversion even after the training is completed. These two compression technologies may bring uncertain and insignificant acceleration, so the improvement of model calculation performance mainly depends on model distillation and network design strategies.

The idea of model distillation is derived from transfer learning, which has a two-step training process, first pre-training the model on a large dataset and then fine-tuning the pre-training model on another smaller data set. The model distillation process has similar ideas that transfer dark knowledge from the teacher network to train a smaller student network. Then, the large teacher network is converted into a small network while

retaining the performance close to the teacher network. Although model distillation is a very effective optimisation and has quite a lot of successful researches in the NLP field, the implementation is more complicated, because transferred knowledge needs to be carefully defined according to different models.

2.4.2 Evolution from MobileNet to EfficientNet

The network optimisation strategy for CNN models is covered in the evolution process of MobileNet and EfficientNet. This subsection will introduce vital ideas and innovations proposed by each version.

Howard et al. [38] proposed a lightweight convolutional neural network focused on mobile devices, namely MobileNet. The first version of the proposed model uses sequential convolutional layered architecture like VGG but replaces standard convolutions with depthwise separable convolutions to greatly reduce the calculations required in convolutions.

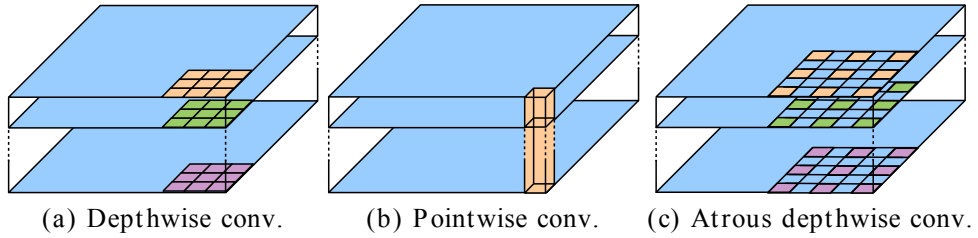


Figure 2.13: Depthwise, pointwise and atrous convolution [39]

The depthwise separable convolution splits the standard convolution into two operations, depthwise convolution (Figure 2.13 (a)) and pointwise convolution (Figure 2.13 (b)) respectively. In detail, Formula 2.3 expresses the calculation of standard convolution, and it has the computational cost of $D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$, where M and N denote the number of input and output channels; D_K and D_F denote the dimension of kernels and feature maps.

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \quad (2.3)$$

Howard et al. [38]

$$\hat{\mathbf{G}}_{k,l,m} = \sum_{i,j} \hat{\mathbf{K}}_{i,j,m} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \quad (2.4)$$

Howard et al. [38]

Formula 2.4 shows that depthwise convolution uses only one convolution kernel for each channel, and there is one to one map from each input channel to each output channel, thus the computational cost is $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$. The pointwise convolution is a standard convolution with kernel size of 1×1 that perform a combination of the inputs along the channel direction, so the computational cost is $M \cdot N \cdot D_F \cdot D_F$. By comparing

computational cost formulae, total deduction for depthwise separable convolution is $\frac{1}{N} + \frac{1}{D_K^2}$.

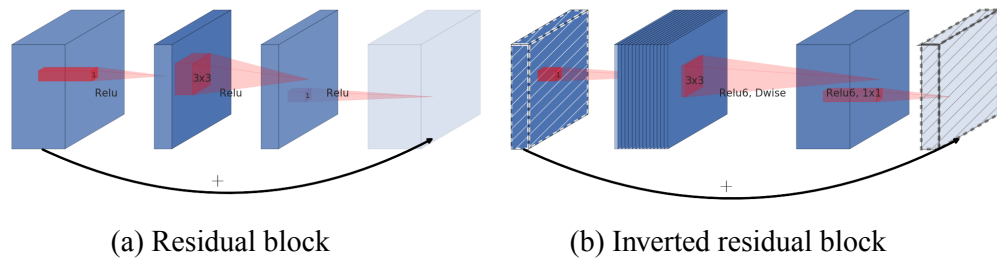


Figure 2.14: Compare residual block and inverted residual block [40]

One year after the release of the first version of MobileNet, Sandler et al. [40] contributed two new ideas in the second version to further optimise the performance of the MobileNet model. The first one is the inverted residual building block shown in Figure 2.14 (b). As compared in the figure, the original residual design in ResNet (Figure 2.14 (a)) reduce dimension first and extraction with standard convolution, but in inverted residual design, the dimension is increased first and use depthwise convolution in the middle. The second alternation is using a linear bottleneck in the last layer in each inverted residual building block to avoid feature information lost in non-linear activation, especially for Rectified Linear Unit (ReLU) that always outputs zero gradients for any neuron weights zero (dead neuron).

In 2019, Tan and Le [41] proposed EfficientNet that inherited the design ideas from MobileNetV2 with a systematical methodology to scale up the model in network depth, width and input resolution to obtain better balance between accuracy and efficiency. Figure 2.15 illustrates the architecture of the proposed baseline network EfficientNet-B0. The network is mainly composed of mobile inverted bottleneck convolution (MBConv) proposed in MobileNetV2. The result shows that it achieves a Top-1 accuracy of 77.1% on ImageNet with 5.3M parameters and 0.39B Flops.

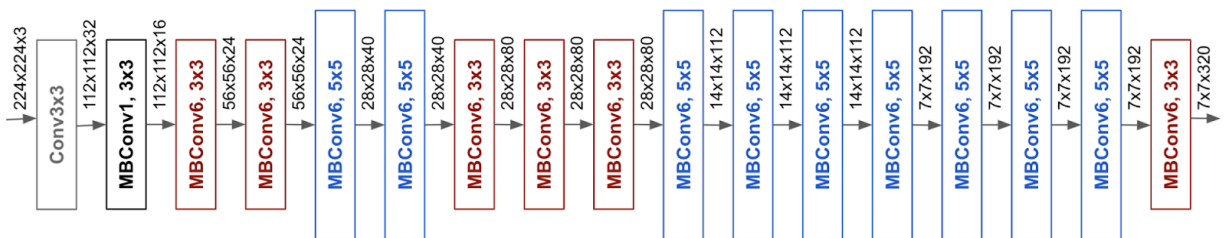


Figure 2.15: The architecture of baseline network EfficientNet-B0 [41]

Two years after the debut of EfficientNet, Tan and Le [42] proposed the second version of EfficientNet in 2021, which has a smaller model and faster training and achieves state-of-the-art results on ImageNet.

2.4.3 Optimisation of Transformer Networks

Wu et al. [43] pointed out that even though Transformer-based models achieve state-of-the-art results in natural language processing tasks ubiquitously, it is unsuitable for running on mobile devices because the self-attention mechanism puts a lot of pressure on storage and computation. To tackle the problem, Wu et al. [43] proposed a Lite Transformer model that uses Long-Short Range Attention (LSRA) to replace standard full self-attention used in Transformer.

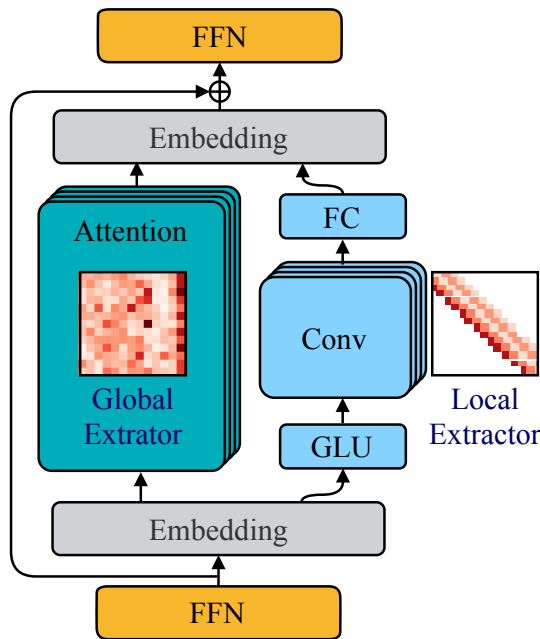
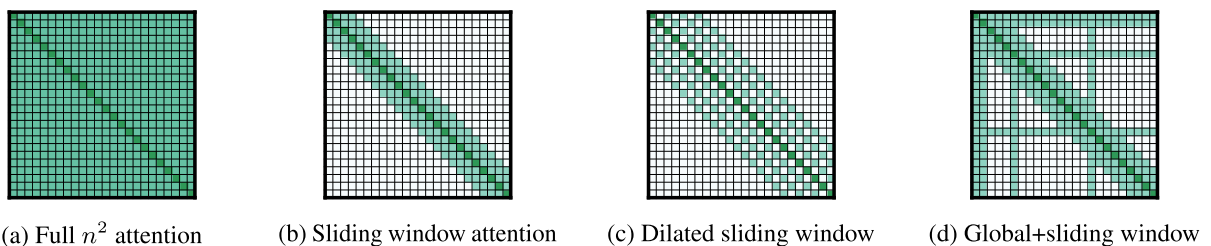


Figure 2.16: Lite Transformer block [43]

Figure 2.16 displays the structure of the proposed Long-Short Range Attention that one group of heads models the local context with convolution while another group focuses on the long-distance relationship with full attention. This method takes advantage of the convolution that is good at capturing local features with small calculation resource required while using the attention mechanism that is good at long-distance. It concludes that using self-attention only overemphasises the local features, and it is possible to use another modelling method in Transformer-based models.

Beltagy, Peters and Cohan [44] published Longformer model, another optimised Transformer-based model, almost at the same time in April 2020, which also pointed out that the self-attention mechanism has limitations when targeting long sequences, depicted in Figure 2.17 (a).



(a) Full n^2 attention

(b) Sliding window attention

(c) Dilated sliding window

(d) Global+sliding window

Figure 2.17: Full self-attention and attention patterns proposed in Longformer [44]

In full n^2 attention, the time and space complexity are all increased quadratically with the sequence length n , because the Queries at each location needs to pay attention to Keys at each location. An idea proposed in Longformer uses three different attention patterns to build a sparse attention matrix rather than a complete attention matrix.

The first method is to use a fixed-size sliding window that goes through each token to calculate local attention (Figure 2.17 (b)). In this way, each token only needs to pay attention to the tokens within the sliding window size w . Although it undermines some long-distance attention ability, it greatly reduces the amount of calculation required to $O(w \times n)$ in terms of computation complexity.

The second method is using a dilated sliding window (Figure 2.17 (c)) in attention to create larger receptive field, which is analogous to using dilated CNN in atrous convolution. This method also has computation complexity $O(w \times n)$, but expands the receptive field size to $l \times d \times w$, where l denotes the number of layers in Transformer, and d for dilation rate.

The third method is designed specifically to handle tasks that require global attention (Figure 2.17 (d)), such as text classification and question answering. For example, in classification tasks, a $[CLS]$ token is added to the head of all tokens with a global attention mark, indicating that the token requires global attention computation on all other tokens in the matrix. And in question answering tasks, the question and document are concatenated together and input to the model. Then mark the entire question sentence to calculate the global attention, and find the answer in the document.

In a recently published paper, Mehta et al. [45] summarised three previously extensively studied methods to optimise the efficiency (higher performance with fewer parameters) on Transformer-based models, which are:

1. Improving transformers

- (a) Improving the usage of self-attention on long sequences. (Longformer proposed by Beltagy, Peters and Cohan [44])
- (b) Explaining multi-head attention and possible redundant representations.
- (c) Learning better representations, e.g. using convolution (Lite Transformer proposed by Wu et al. [43])

2. Model scaling: a common way to improve the performance of deep models.

Increasing dimensions in width-wise scaling and stacking more blocks in depth-wise scaling, which is analogous to the model scaling ideas in EfficientNet.

3. Improving sequence models

- (a) Using better token-level representation.
- (b) Using model compression, pruning and distillation.

2.5 Related works

The previous two sections introduce the model backbone related to the project and a series of optimisation methods that can enable models running on mobile devices. This section will mainly explain the background of video-based human action recognition tasks and two deep-learning models that are mainly targeted to such tasks, namely 3D Convolutional Neural Networks (3D-CNN) and Transformer.

2.5.1 Human action recognition overview

Video understanding, like natural language processing, is a class of many subtasks and has many similar types of tasks as shown in Table 2.1. Compared with text classification tasks in natural language processing, one type of classification task in video understanding is human action recognition based on sequential pose features in a video.

Task type	NLP task	Video understanding task
Sequence labelling	Part-of-speech (PoS) tagging, Named Entity Recognition (NER)	Action labelling with start and end frame position
Classification	Based on sentiment, language or topic	Based on action, environment or topic
Generative tasks	Translation, summarisation, Question-Answering	Video caption, video summarisation, Question-Answering based on video context

Table 2.1: Comparing tasks in NLP and video understanding

Both video-based data set and text-based data set have characteristics of sequence, which brings the similarity of these tasks described in Table 2.1. However, video data sets are also different from the text and have their complexity. Wu, Sharma and Blumenstein [46] reviewed the wide range of applications and challenges involved in human action recognition in videos, as well as the deep learning-based techniques proposed to address the challenges. One challenge is that videos are divided into different classes, for example, single or multiple viewpoints and the modalities include monochrome (infrared) sequence, RGB sequence, depth image sequence, and skeleton point sequence. As for deep models, 3D-CNN, combinations of CNN and RNN was widely used before the Transformer architecture emerged.

Considering that most mobile phones capture monocular RGB sequences, therefore, my research only investigate this video modality. Besides, the subsection 2.5.2 will detail the 3D-CNN and models combined from CNN and RNN.

2.5.2 3D-CNN and RNN based methods

The standard two-dimensional convolution uses a two-dimensional kernel filter on each channel, and only two-dimensional features can be extracted. Tran et al. [47] reviewed many previous 3-dimensional convolutional networks (3D-CNN) and highlighted relevant applications on spatiotemporal features in video dataset. As shown in Figure 2.18, 2D convolutional kernel filters are expanded to 3D and allow them running on multiple channels simultaneously. 3D-CNN is widely used in tasks that process multiple images at the same time, such as medical computed tomography (CT) scan data and video data.

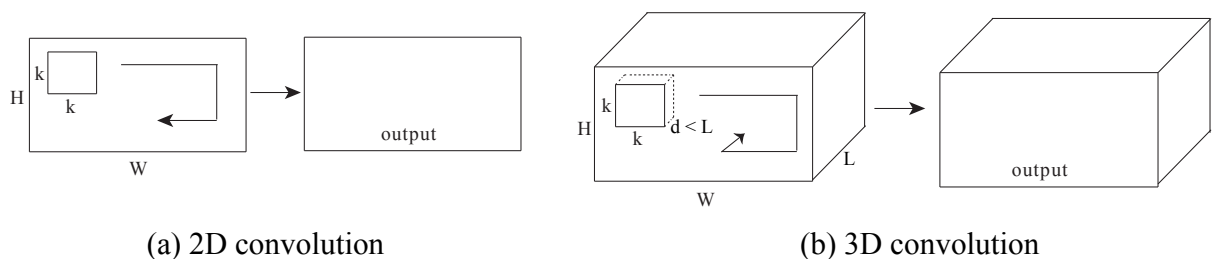


Figure 2.18: 2D convolution and 3D convolution [47]

The difference between the proposed 3D convolution and the multi-channel 2D convolution is that the 3D convolution uses weight-sharing kernel filters, while the multi-channel 2D convolutional kernel filters have different weights for each channel and ignore adjacent channels in computation.

Back to 2011, Baccouche et al. [48] firstly applied a composed architecture of 3D convolution and RNN to the human action recognition task. In this work, 3D convolution is firstly used for learning spatiotemporal features automatically, followed by an RNN “to classify each sequence considering the temporal evolution of the learned features for each timestep”, said by Baccouche et al. [48]. The researchers also spotted the performance drawback of the RNN and will investigate the possibility of using a single-step model in future work.

One year after, Ji et al. [49] proposed a human action recognition model composed purely of convolutional operations. The premature network design is shallow and does not use residual connections, which are very similar in architecture to LeNet, consisting of multiple convolutional layers and pooling layers. In addition, they add some designed features (optical flow fields in many directions) to input, which is not an end-to-end trainable network.

After several years of development, Tran et al. [47] proposed model Convolutional 3D (C3D) that achieved the state-of-the-art in video tasks in 2015. This model does not involve the input of additional handcrafted features at all, but it also exposes some

weaknesses of the 3D convolution. For example, 3D-CNN cannot use pre-trained results from 2D images, such as ImageNet. Also, the network has too many parameters, which is prone to over-fitting in the case of using an insufficient amount of video training data.

To overcome the weaknesses of using CNN or RNN individually, Simonyan and Zisserman [50] proposed a Two-Stream network. The network consists of a spatial flow focused on each image frame and a temporal flow focused on the calculated optical flow. Based on the idea of using the Two-Stream network, Feichtenhofer, Pinz and Zisserman [51] pointed out that the fusion process in the last Softmax layer caused losses and 3D convolution can perfectly fuse the spatiotemporal information in the convolutional layers.

In 2018, Carreira and Zisserman [52] summarised the previous research on different methods used in the field of action recognition, which is also discussed in this subsection, namely ConvNet+LSTM (CNN+RNN), 3D ConvNets (3D-CNN) and Two-Stream network. Based on these previous ideas, Inflated Two-Stream 3D ConvNets (I3D) is proposed and achieved good results on both HMDB-51 and UCF-101 data sets.

2.5.3 Transformer-based Neural Networks

Girdhar et al. [53] first incorporated the Transformer architecture into deep-learning-based action recognition tasks and proposed the Video Action Transformer network to identify and locate human actions in video data sets. The architecture of the proposed Video Action Transformer is shown in Figure 2.19 that the state-of-the-art model of 2019, Inflated Two-Stream 3D ConvNets (I3D) is used to extract features from video clips. The target task of this model is to recognise and locate human actions. It not only needs to detect the position of all people in the video but also classify the actions of each people, so the network structure is very complicated and cannot be applied on mobile devices.

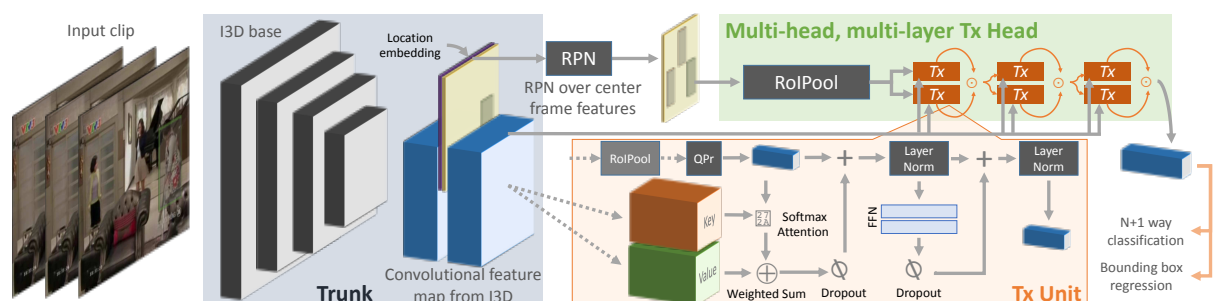


Figure 2.19: Video Action Transformer Network architecture [53]

If the target task is not to mix recognising and localising like the famous object detection model YOLO, the region proposal network (RPN) network used to obtain the position

can be removed. In this way, the paradigmatic architecture of using spatial convolution firstly and then temporal Transformer inspire research on classification tasks in the video understanding field.

In February 2021, Neimark et al. [54] proposed the Video Transformer Network (VTN), which intuitively uses Transformer to model long-range context relationships in videos for classification tasks. The architecture of this network is clearly shown in Figure 2.20. It first uses the spatial network backbone $f(x)$ to extract features from frames, adds position embedding frame index number PE_n to extracted feature vectors, and finally uses a fully connected multi-layer perceptron (MLP Head) to generate classification results.

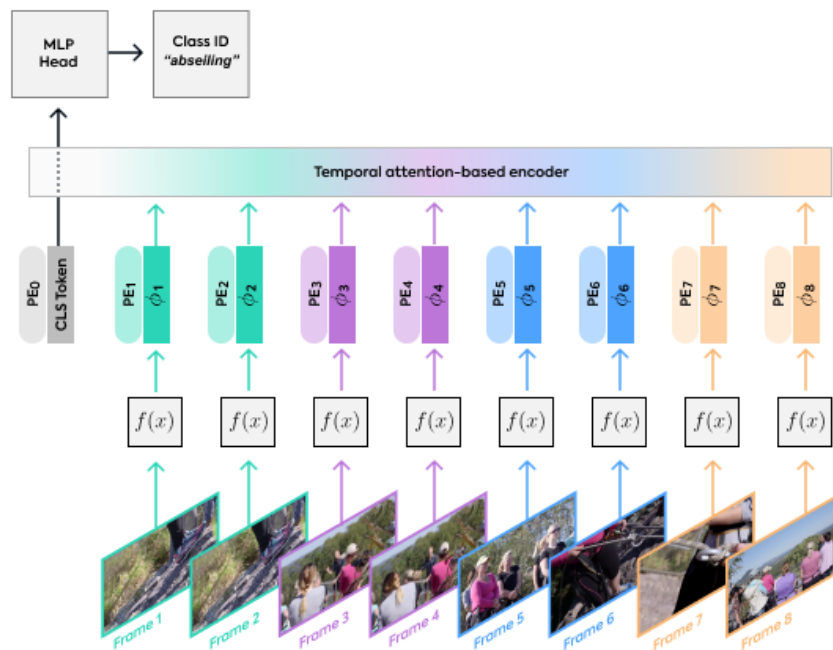


Figure 2.20: Video Transformer Network architecture [54]

This research used Kinetics-400 as training and evaluation data set, and implemented the proposed model in PyTorch framework based on Facebook SlowFast⁴ code base. They used a powerful 8-V100-GPU machine for experiment, in which each GPU has 16GB memory, enabling them to conduct experiment on the large-scale video data sets.

As for contribution, they proposed a novel modular transformer-based model for the video classification task that is much more performant and efficient compared to previous methods. Under this modular model, they also experimented and evaluated two spatial backbones, ResNet and Vision Transformer (ViT), as shown in Table 2.2. The table shows that the ResNet (CNN-based) scheme has lower computation complicity

⁴PySlowFast: video understanding codebase: <https://github.com/facebookresearch/SlowFast>

than ViT-based scheme, which is denoted in inference GFLOP, but the ViT-based backbone brings a higher accuracy to the model.

Model	training runtime (mins)	training epochs	validation runtime (mins)	params (M)	inference GFLOPs	top-1
R50-VTN	62	40	32	168	1,059	71.2
R101-VTN	110	40	32	187	1,989	72.1
ViT-B-VTN(1 layer)	107	25	48	96	4,214	78.6
ViT-B-VTN(3 layers)	130	25	52	114	4,218	78.6

Table 2.2: Video Transformer Network experiment results [54]

Almost at the same time in March 2021, Google researchers Arnab et al. [55] published Video Vision Transformer (ViViT) that is a pure-transformer based model for video classification tasks. The first Figure 2.21 shows the architecture of the proposed model without factorisation with three model variants that factorise the different components of the transformer encoder in spatial and temporal dimensions to enhance efficiency.

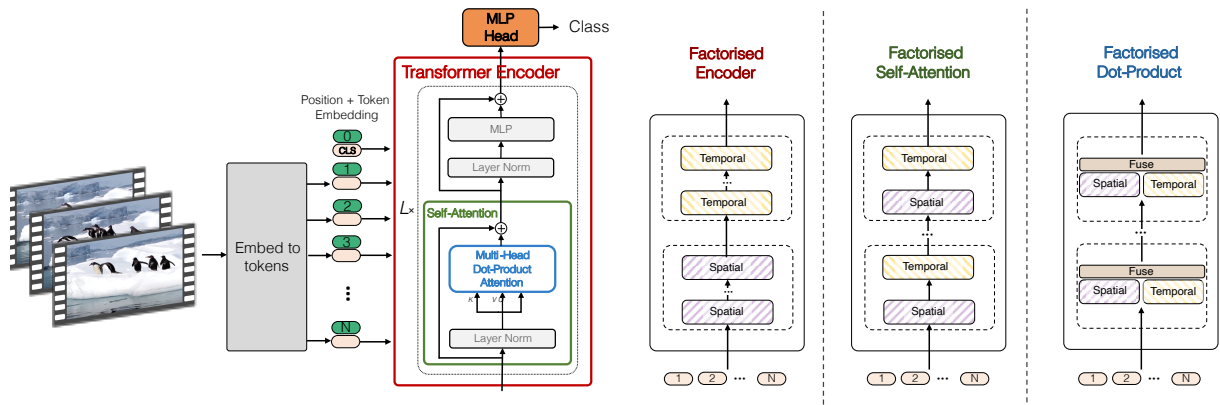


Figure 2.21: Video Vision Transformer model and factorised variants [55]

These researches on Transformer-based networks on video classification are important references to my research. From here on, reviewing a variety of related studies has taught me a lot and give me a clear direction for designing optimised model for the application of exam invigilation. I need to complete the following tasks for model design in the next design stage: To design the model with appropriate spatial backbone and temporal Transformer backbone; and to optimise the model so that it can run on mobile devices with limited computing power.

3 Design

After reviewing related previous studies, this chapter will discuss various possible designs and rationales for the system design before actual implementation according to project requirements.

Firstly, the design of the data collection process is introduced in section 3.1, including the data type to be collected, the labelling approach and the functionalities that need to be implemented in the data collection app.

Secondly, it is necessary to design the data preprocessing pipeline in section 3.2 according to the anonymity requirement detailed in research ethics (section 1.3), and transform the data to make it suitable for model input.

Thirdly, the design of a deep model suitable for solving the problem of this research is detailed in section 3.3, based on the related previous research and the background of deep models.

3.1 Data set collection

To build a video data set suitable for this research, in section 2.1, I reviewed the feature engineering and classic machine learning models before the deep-learning. The feature engineering process is directly related to the understanding of the characteristics of the data set. It helps to design the data set collection scheme and the depth model architecture.

The main difficulty in human action recognition tasks lies in the acquisition of video data sets. In detail, it is difficult to obtain enough data in both number and quality to training deep-learning models, especially for a video data set. Although many public data sets for video understanding exist, unfortunately, they cannot be used in this research. For example, the Kinetics data set contains many labelled human actions, but none of these actions is a student taking the exam. Therefore, this research cannot use the public data set, which leads to the first step in my research: to collect a labelled video data set containing different actions of students taking the exam.

The design of the data collection process has two aspects: one is to clarify the requirements for the video data set, and the other is to design the data collection process and the functionalities needed in the collection app according to the requirements.

3.1.1 Requirements of data set

The purpose of constructing a data set is to train a deep model to solve my research problem. In the research problem, the on-device camera video input stream is used as the input, so the format and modality requirements for the video data set are monocular RGB video. Besides, considering that the actual use case is videoing from different angles, so the data set should contain multiple angles. In this research, to simplify the data collection process, multiple angles are the front and side.

Another aspect is the appropriate video length in the data set. Too long videos may contain multiple actions that should be classified into different categories, while a too short video may result in too few data features to classify. Based on the experience in public video data sets, this study designs the length of every video clip in the data set to be 10 seconds.

From my experience of doing classification tasks in machine learning, the amount of data in each category should be roughly balanced. An unknown category whose data volume does not exceed the others can be added to the multi-classification task to represent an activity beyond examination.

Based on the above discussion, the video data collection requirements are finally summarised as the followings:

- The data set contains monocular RGB videos taken from the front and side of the target examinee.
- The length of each video clip in the data set should be equal to 10 seconds.
- Each video should be labelled with a target category when captured.
- An 'unknown' category should be added. The total amount of the unknown labelled data should be less than or equal to the total number of the other.
- The effective categories should cover all the activities that examinees may perform during the examination.
- The total amount of videos in each effective category (excluding Unknown) should be roughly balanced.

After the data set requirements are determined, the functional requirements of the data collection APP are also clarified. The design before development and implementation

adopts the user story method commonly used in the agile software development process.

User stories are informal descriptions of features from the user's perspective, which intuitively explains how various users interact with the system, so as to define the system functions. Usually, user stories are formed in natural language description, for example:

As a [*role*], I can [*capability*], because [*reason*].

Appendix C.1 Table C1 lists the key contents of user stories in the data collection app.

3.2 Data preprocessing

After collecting videos uploaded by participants to create a video data set, data preprocessing is the next process of adjusting the data to meet the research ethics requirements processing it into an input format suitable for the model data loader. The video encoding format defined in WebRTC specification is *video/webm*, with three types of video codecs available in different browsers – VP8, H.264, and the latest VP9 that supports lossless compression. Most of these video codecs use lossy compression to reduce the size of transmitted data, especially for online use cases. Although the lossy compression may change the video slightly, the alteration is too small to be distinguished, and the compressed video is visually original. Therefore, there is no impactful data loss in the lossy compression process.

The compression reduces the size of transmitted data, but the side-effect is computation overhead in decompression. If the model data loader decodes the video data during the training process, it will consume lots of CPU during the decompression and anonymisation process. An optimisation method is decoding the video into a sequence of pictures in the data preprocessing process to reduce the computational overhead in the training process. An anonymous algorithm will be applied to each frame and save the results in picture format. In this way, the data loader only needs to load the processed pictures during the training process.

According to research ethics requirements, the data preprocessing process will use an anonymisation algorithm to minimise the presence of facial details that might enable the identification of an individual. The anonymisation algorithm cannot directly mask the face completely. Because the facial information contains many features that help the model to classify activities, such as the direction of sight, the lip movement during speaking, etc.

The anonymisation algorithm should strike a balance between the complexity of implementation, performance, and the ability to keep research related facial features. In the research, a simple two-step algorithm, namely feature masked face concealment, is used to extract the key facial features while covering the other facial areas.

The following two subsections will introduce the details of the two-step process in the anonymisation algorithm. The first step is to find and locate faces in the video, and the second step is to extract facial feature points and perform masking on each frame.

3.2.1 Face detection with Viola–Jones algorithm

Viola and Jones [56] proposed a fast object detection approach using Haar-like features, namely Viola–Jones algorithm, which extracts features from input images using a series of adjacent rectangular regions. This operation is very similar to the convolution operation in convolutional neural networks. However, the difference is that the proposed Haar Cascade uses predefined feature extraction kernels, such as edge features, line features, etc. The classifier part uses AdaBoost algorithm, a famous Boosting method in ensemble learning, in which it trains many weak classifiers to form a more accurate strong classifier.

Currently, this algorithm is very mature and has been widely used in many studies. For example, a previous study of virtual exam controller proposed by Garg et al. [14] was introduced early in the literature review section 2.2. In this case, they use Haar Cascade Classifier to detect, tag, and identify the students face and using deep learning to apply exam constraints.

Technically, Haar Cascade Classifier use cascade structure to form multiple stages to classify face regions as shown in Figure 3.1. This classifier has four vital features that enable it to be widely used in face detection tasks quickly, summarised by Garg et al. [14] as follows:

- Using Haar-like features.
- Using the integral picture.
- Using AdaBoost learning.
- Using cascading classifiers.

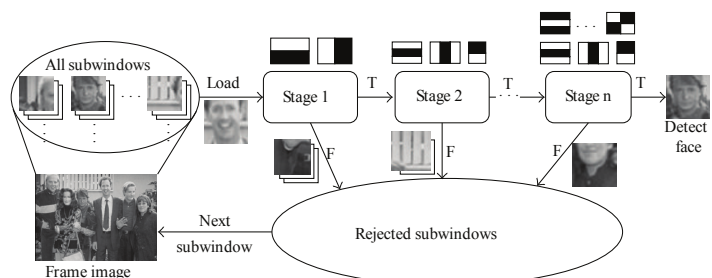


Figure 3.1: Haar Cascade structure [57]

OpenCV has provided a good implementation, `cv::CascadeClassifier`¹ and a variety of

¹cascadedetect.cpp: <https://github.com/opencv/opencv/tree/4.5.3/modules/objdetect/src>

trained models² for selection. In addition, the API design for this module is clear and easy to use. The trained model can be loaded through `cv::CascadeClassifier::load` method, and then `cv::CascadeClassifier::detectMultiScale` can be used to locate the face positions in the input image and return the result of faces in the format of rectangular areas.

To summary, this Haar Cascade object detection method is suitable for real-time face detection on mobile devices with the advantage of requiring a small amount of calculation. Therefore, this research will use this method to perform the face detection procedure in the anonymisation algorithm.

3.2.2 Face landmark extraction and face concealment

As mentioned above, after obtaining the position of faces from the input video, it also needs to extract face landmark points and overlay them on the top after masking face details.

A highly efficient and accurate face alignment algorithm was proposed by Ren et al. [58] in 2014. This research still uses classic machine learning methods for low computation complexity by using a combination of random forest and global linear regression to regress key points of facial features. It learns the local binary features (LBF) of each key point through random forest, then combines the features and uses linear regression to obtain the key points. A facial landmark detection survey study published by Wu and Ji [59] concluded that this regression-based method has both good performance and fast speed.

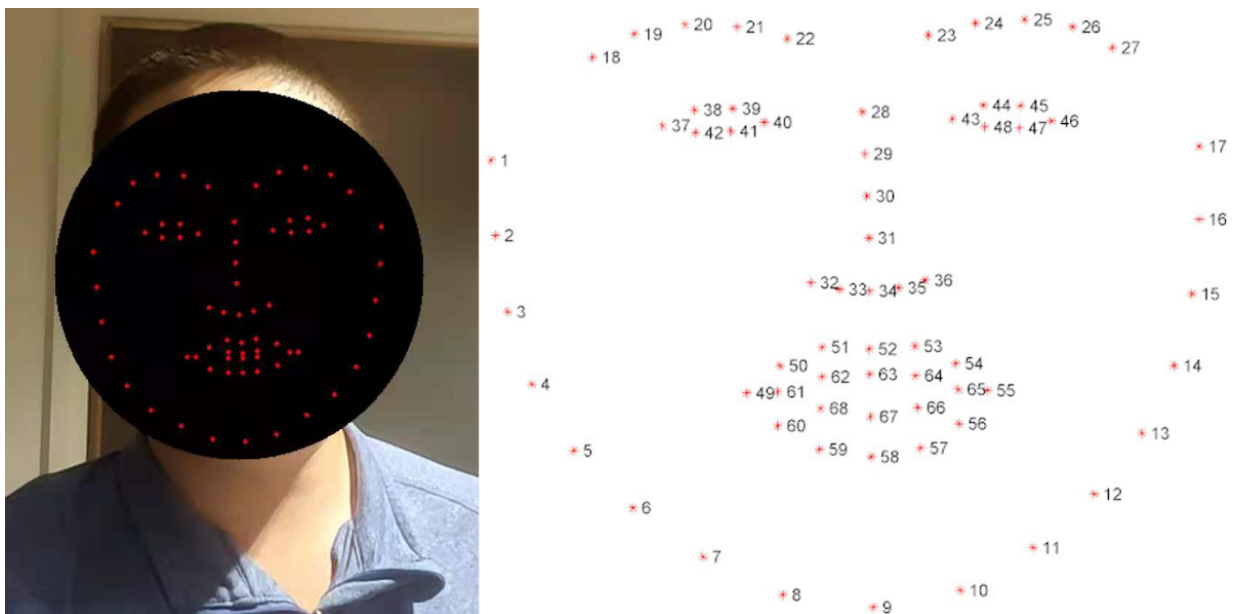


Figure 3.2: Landmark masked face concealment and face landmark schematic

²Available models: <https://github.com/opencv/opencv/tree/4.5.3/data/haarcascades>

The left picture in Figure 3.2 illustrates an exemplar face image (anonymised result) with landmark masked face concealment applied. The right side in the figure displays the schematic of the face landmarks used in this research. The schematic has 68 key points to locate face landmarks in the image. These key points can represent many facial features, such as face orientation, lips movement and other details without exposing unrelated details in classifying exam activity.

In OpenCV, this algorithm is also implemented in extra contrib modules. The class of detecting face landmarks with LBF algorithm is defined in `cv::face::FacemarkLBF`³, and the corresponding trained model⁴ is provided by the algorithm contributors. In addition, the algorithm implementation inherits the simple and easy-to-use API of the object detection framework in OpenCV. Generally, using `FacemarkLBF::loadModel` method to load trained model and `FacemarkLBF::setFaceDetector` to set a custom face detector function where Haar Cascade will firstly detect face regions in the input image.

3.3 Deep model design

Since deep networks have the duties of feature extraction, deep learning methods no longer require manual feature engineering. But it is still necessary to consider that different deep networks have inductive biases (preferences) for different feature types. The literature review section 2.3 has reviewed models detail and instinct properties. For example, CNN has properties of translation invariance and is suitable for imagery features; Transformers has properties of long-range dependency modelling and adapts to time-series features.

The model design procedure includes not only the network architecture but also the data loader. The latter is responsible for reading the result from the previous data pre-processing procedure and transforming them into the tensor shapes required by the model input layer. Because the data loader is the foremost part of the model design, this section will begin with the detail of loading data. Then, the deep network architecture for the research goal will be detailed, as well as the output layer for generating probability results.

3.3.1 Data loader and input layer

The data loader is a binding bridge between the processed data set and the model input layer, which converts the data format from the former into information that the model's input layer can accept. As discussed in pre-processing section 3.2, the results from the

³facemarkLBF.cpp: https://github.com/opencv/opencv_contrib/blob/4.5.3/modules/face/src

⁴LBF model: <https://github.com/kurnianggoro/GSOC2017/raw/master/data/lbfmodel.yaml>

previous procedure are individual video frames saved in picture format. Thus, any picture library in Python can read the inputs into memory, such as OpenCV, which was also previously used in data pre-processing.

Table 3.1 shows the design of tensors of the model’s input layer. The model needs to input these two tensors in each step in training or inference. The first tensor in the table represents the image sequence, and the second tensor describes the frame position in the image sequence corresponding to the original video clip.

Tensor ID	Name	Shape	Description
0	Images	[1, 3, 16, 224, 224]	[Batch size, RGB channels, Frames, Height, Weight]
1	Positions	[1, 16]	[Batch size, Frame position]

Table 3.1: Model input tensors

In detail, the first tensor represents the image sequence, which is a five-dimensional array defined in tensor shape. The first number of this tensor shape is batch size, which represents the number of samples utilised in one iteration. Usually, in the model training process, the batch size should be increased as much as possible under the resource allowance of the hardware so that more samples can be taken into account when calculating the back-prorogation gradient and avoid over-fitting. In the inference process after model training and deployment, the batch size is always one. The second part of the tensor shape is channels, which represents the three channels of red, green and blue in the case of an RGB image. The third number in the tensor shape is the number of frames uniformly sampled from the video. The last two numbers in tensor shape are the height and width of the image.

The second tensor represents the frame position number in the video clip. This position number will be used as position embedding in the temporal backbone network to capture the temporal features. A larger number of frames in one iteration can give more data to the model, improve the accuracy and the ability to predict long-term activities, but it also requires longer capture time and computing resources. To strike a balance between the model performance and resources required, I define the number of frames and frame positions in each iteration to be 16 in this study.

3.3.2 The model architecture

After the data loader converts the input data into a suitable format for the input layer, the next step is to select a spatial backbone and a temporal backbone, then design a data flow to connect them in the model architecture. The block flowchart in Figure 3.3 shows

the data flow from the input layer through spatial and temporal deep networks to the final category possibility output.

As for the spatial backbone, this research uses a minimal EfficientNet-B0 backbone pre-trained on the ImageNet data set to do spatial feature extraction from frames. The detail and evolution history of EfficientNet is reviewed in subsection 2.4.2. To summary, it is a further optimisation from MobileNet, also a CNN deep network designed for mobile devices.

As for the temporal backbone, Longformer is used in this research as a temporal encoder with global attention in $[CLS]$ token and local sliding window attention pattern. In subsection 2.4.3, many previous studies have pointed out the high computational complexity of the self-attention mechanism in Transformer, and proposed corresponding improvements to facilitate running on mobile devices.

The output layer receives hidden states of the $[CLS]$ token from the Transformer-based temporal encoder and outputs the normalised category probability. This research uses two fully connected dense layers as the multiple layer perceptron classification header reviewed in Figure 2.20, Video Transformer Network architecture.

The output dimension of the first dense layer is equal to the output shape of the temporal encoder, and a Gaussian Error Linear Unit (GELU) is used for nonlinear activation mapping. The final output layer dimension of the model is the number of classification categories.

In this study, considering the difficulty of data set collection, 12 activities were designed in Appendix D.1 Table D1, covering the common types of remote exams. This table also contains related descriptions and predefined risk levels for each activity, expressed in three alert levels.

3.3.3 Loss function and training hyperparameters

The categorical crossentropy loss is a commonly used loss function in classification tasks. In TensorFlow, if the labels is in one-hot representation, *CategoricalCrossentropy* loss should be used. In the case of labels provided as integers, *SparseCategoricalCrossentropy* should

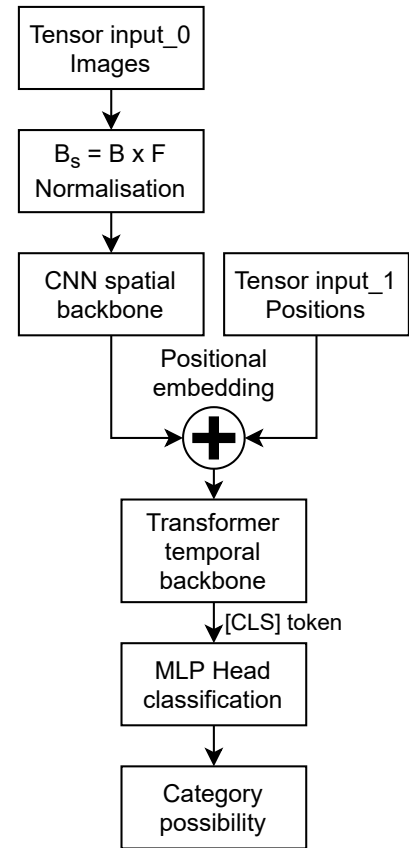


Figure 3.3: The data flow in the Video Transformer Network architecture.

be used. Since this study uses a deep learning model to solve the multi-classification problem where the label of each data sample is a integer category index of the matching category, the appropriate loss function is `tf.keras.losses.SparseCategoricalCrossentropy`.

Many hyperparameters are involved in the deep model training process, where the model performance is closely related to fine-tuning. The process of training any deep model involves two important hyperparameters, batch size and learning rate. The batch size represents the number of samples input to the model in each iteration. The learning rate represents the update factor of the model weight moving toward a minimum of the loss function. A larger batch size allows the trainer to better calculate the back-propagation gradient with more samples, where appropriately increasing the learning rate can make the model converge faster. A too-large learning rate will cause an oscillating loss value making the model difficult to converge. Conversely, a too-small learning rate will slow down the training process and make the optimiser easier to get stuck at saddle points.

Because the Transformer has high versatility and low inductive bias than CNN, many previous studies have shown that learning rate warm-up is an effective and necessary design to prevent over-fitting when training Transformer-based deep networks. An intuitive explanation for the learning rate warm-up is that the model knows nothing about the data attributes and distribution at the beginning of training. Using a large learning rate at the beginning will cause the model to over-fit the first few samples immediately, making it hard to remedy it in the subsequent training steps.

Equation 3.1 shows the learning rate scheduler function, where d_{model} defines the maximum learning rate applicable to the model complexity, and $step_num$ defines the number of iterations to reach the peak learning rate.

$$lrate = d_{model}^{-0.5} * \min \left(step_num^{-0.5}, step_num \cdot warmup_steps^{-1.5} \right) \quad (3.1)$$

Vaswani et al. [31]

There are many more hyperparameters other than the learning rate for the training process, but they are less important than hyperparameters that directly controlling the model complexity and the training process. For example, some hyperparameters used in data augmentation – scaling factors, cropping ranges and possibilities for each augmentation operation. Stronger data augmentation can better prevent overfitting but may introduce more noises in the training process. To reduce the search space for hyperparameter fine-tuning, this study reviews the implementation of related researches to set those less impactful hyperparameters.

4 Implementation

After designing the overall system, the next step is to program and implement the system. This chapter introduces three system modules, the data collection app is developed with React in section 4.1; the deep learning model is programmed in Python 3 with TensorFlow 2.5; adapting the model to the mobile app on Android with TensorFlow Lite and React Native framework in section 4.2. The selection rationales of these frameworks and libraries are detailed in Appendix B.

4.1 Data collection web app

This section introduces several key user logics and algorithms in the data collection web application. From the user's perspective, the data collection system mainly provides three user processes: new participant registration, participant login and the core function of participant recording and uploading, which are illustrated in Figure C1, C2 and C3 respectively.

Figure C1 shows the registration process for new participants. According to the research ethics, each participant needs to clearly understand the research content and provide consent before registering an account. Another worth mentioning is that the system will generate a random password in the registration process and send it to the email provided by the registrant. This registration process can prevent the system from collecting users' private passwords.

After the registration process, Figure C2 depicts the authentication process for registered participant. The participant will input the email address and the random password received during the registration process. If a participant forgot or lost the random password, he or she can request a new random password and restart the login process.

Figure C3 shows the main workflow of the data collection app after the participant logs in. In the task selection stage, the web front-end first queries the currently available tasks from the back-end and display them in a list that participant can select. Each task has detailed text and picture descriptions, which is easy for participants to understand.

After the participant selects desired tasks, the next step is video recording. The system uses the web real-time communication (WebRTC¹) API provided in most modern browsers to record videos for the participant. Finally, in the upload step, participants

¹Real-time communication for the web: <https://webrtc.org/>

can review each recorded videos and decide whether to upload or try rerecording.

The user registration and login process in the web app adopts a common secure authentication paradigm for online applications. Algorithm 4.1 displays the process of user authentication, where the password is double hashed and salted in back-end database storage. In addition, the system uses short-term-valid JSON Web Tokens (JWT) as the access credentials for RPC calls, also incorporating the mechanism of a periodic token refresh, which further enhances system security and protects users' privacy.

Algorithm 4.1: User authentication	Algorithm 4.2: Video uploading
<p>Data: Email e, Password p Result: Auth T_a, Refresh T_r</p> <ol style="list-style-type: none"> 1 Web browser front-end: $EP \leftarrow \text{concat}(e, p)$ $h \leftarrow \text{sha256}(EP)$ $\text{send}(e, h)$; 2 Back-end: $\text{receive}(e, h)$; 3 if user e does not exist then 4 return error; 5 if bcrypt hash compare h failed then 6 return error; // Authenticated 7 $T_a \leftarrow \text{JWT}(\{\text{userDetail}, \text{authUUID}\})$; 8 $T_r \leftarrow \text{JWT}(\text{refreshUUID})$; 9 $\text{return } T_a, T_r$; 	<p>Data: Blob video data v Result: Data in chunks $c[x]$</p> <ol style="list-style-type: none"> 1 $s \leftarrow 3 \times 10^4$; // gRPC 32768 bytes read limit 2 for $i \leftarrow 0$ to $\lceil \frac{\text{sizeof}(v)}{s} \rceil$ do 3 $b_{\text{left}} \leftarrow i \times s$; 4 $b_{\text{right}} \leftarrow$ $\max((i + 1) \times s, \text{sizeof}(v))$; 5 $c[i] \leftarrow v[b_{\text{left}} : b_{\text{right}}]$; 6 $\text{send}(i, c[i])$; 7 $\text{progress} \leftarrow \text{receive}()$; 8 if receive progress failed then 9 return error; 10 $\text{updateUI}(\text{progress})$; 11 $i \leftarrow i + 1$;

Algorithm 4.2 shows the process for segmenting large video data before uploading it to the backend server. This segmentation algorithm runs on the browser front end, using *slice* in binary large object (Blob) API for slicing the video. The communication between the front-end and back-end uses the gRPC bidirectional streaming. When the front-end uploads data, it also receive acknowledgement and receiving progress from the server.

4.2 Mobile app implementation

After the data collection app is implemented, the data collection process begins. While waiting for the data set to be collected, I designed the deep model as detailed in section 3.3. After the design of the model is completed, the data loader and the deep model are implemented using Python programming language and TensorFlow framework. The im-

plementation of EfficientNet (*tf.keras.applications.efficientnet.EfficientNetB0*) and the MLP classification head (*tf.keras.layers.Dense* and *tf.keras.layers.Dropout*) is composed directly from the TensorFlow Keras-style high-level APIs. The Longformer implementation uses the Hugging Face Transformer² library. The detail of each layer in the implemented model is shown in Appendix D.2. After finish implementing and training the deep model, the remaining research goal is to port the deep model to Android platform.

Although the most widely used programming language for Android development is Java, this project does not use Java except for the automatically generated initialisation codes and native function bindings, Instead, this study uses React Native to develop UI and application logic. Even though JavaScript development is more convenient, it is single-threaded and is not suitable for high performance parts in the application. For the parts that require high performance, such as the image pipeline, the anonymisation process, and the deep model inference process, the C++ programming language is used for multi-threaded Android native development.

4.2.1 Architecture overview

The architecture of the mobile application developed in this research is shown in Figure 4.1. The arrows in the figure indicate the direction of data flow and clearly show dependencies across different functional blocks. The left half of the figure describes the user interface and user interaction logic developed using React Native, and the right half describes the core functions of the application developed using C++. These two parts use React Native Javascript Interface (JSI) for communication.

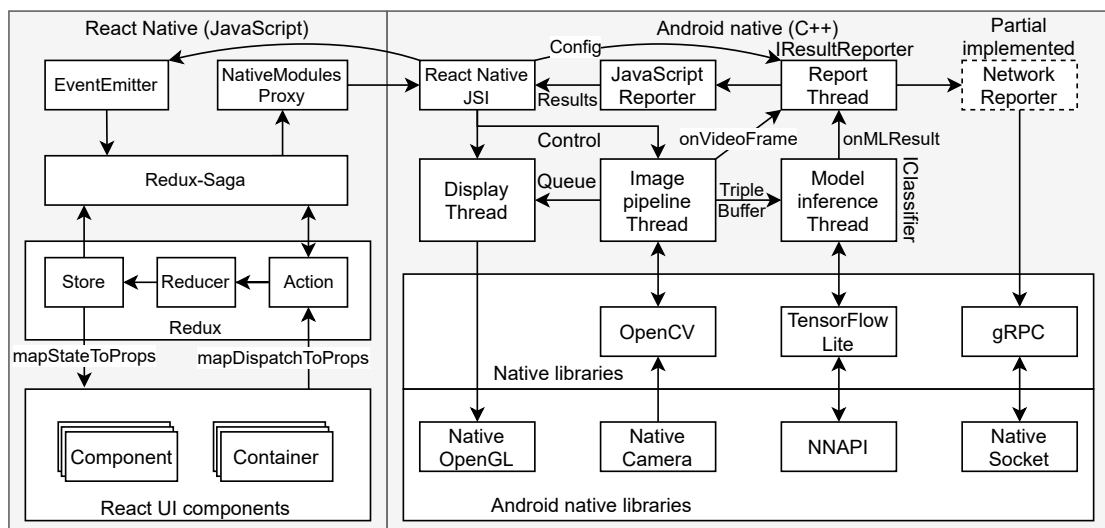


Figure 4.1: Mobile app architecture and data flow

The core functions of this project in the native part shown on the right side in Figure

²Hugging Face Transformers: <https://huggingface.co/transformers/>

4.1 is much worth discussing than UI implementation on the left side. These core functions include displaying images without lag and running model inference at the same time. To achieve the goal, this project implements these functions in C++ by using multi-threading to enable asynchronous processing for display (a real-time task), model inference and network transmission (time-consuming tasks). It can better take advantage of the multi-core of the mobile phone processor, and time-consuming tasks will not affect the real-time task, which ensures a smooth user experience.

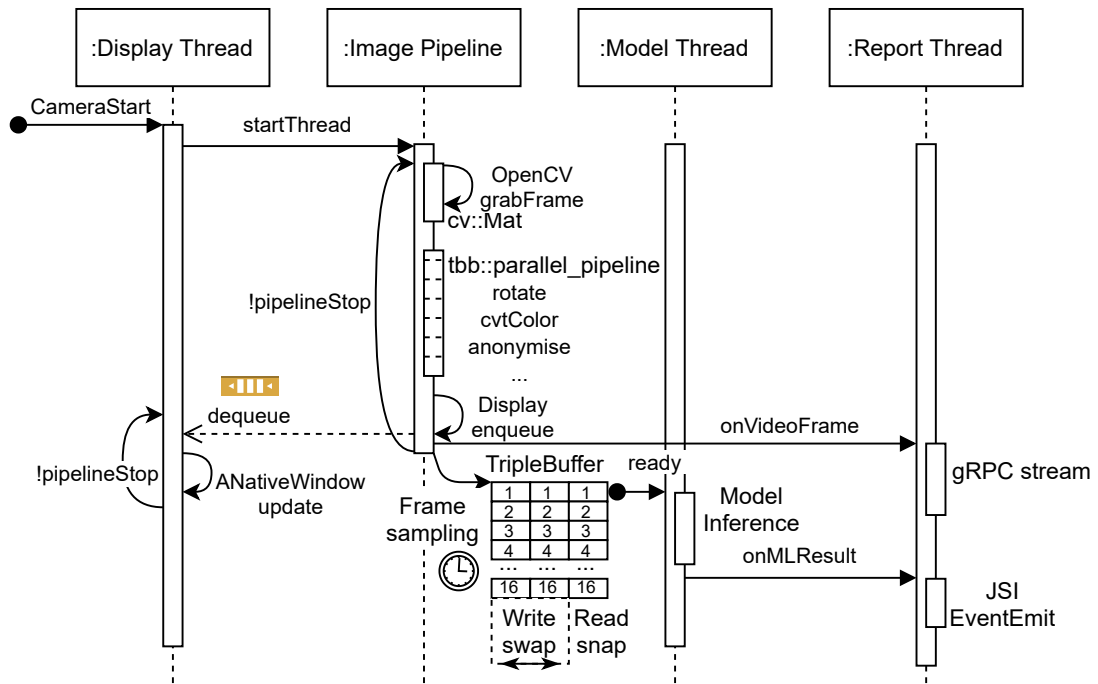


Figure 4.2: Sequence diagram of the multi-threaded app

The sequence diagram in Figure 4.2 describes the interactions between four threads in the Android application. After the user enters the exam screen, the display thread will receive the processed image data through a queue and update the display. After the exam begins, the model inference thread will be started to receive processed image data from a triple buffer, and performs model inference, then informs the result report thread after getting the result.

The rest of this section will introduce in detail the multi-threaded design for the core function, including display thread, image processing pipeline thread, and model inference thread.

4.2.2 Display and image processing pipeline

This subsection introduces a real-time task involving the display and image processing pipeline in the app. The image processing pipeline firstly receives an image from a cam-

era on the mobile phone then transforms and anonymises it with OpenCV. The display thread is responsible for receiving the image processed by the image processing pipeline and showing it through OpenGL. This process is a real-time task, where the display process takes less time than the image processing process. The display thread needs to wait for the processed image data from the processing pipeline thread. Therefore, transferring data across two threads should use the producer-consumer programming paradigm with the blocking queue data structure.

A queue is a commonly used first-in-first-out (FIFO) data structure of a sequential organised collection of entities which can be modified by *enqueue* (adding entities at one end) and *dequeue* (removing entities from the other end). A thread-safe blocking queue is a queue that may block in *dequeue* operation or *enqueue* operation. If the blocking queue has already been empty, a *dequeue* operation will block the calling thread until new data are available in the queue. Conversely, a *enqueue* operation will block the calling thread if the blocking queue has already full.

Algorithm 4.3: Image processing pipeline thread

Data: Readable image stream v of `cv::VideoCapture`
Result: Processed image array $p[]$ of `cv::Mat`

```

1 initialisation; // OpenCV VideoCapture, CascadeClassifier, FacemarkLBF
2 STOP ← false;
3 while not STOP do
4    $p_{bgr} \leftarrow read(v)$ ;
5    $p_{grey} \leftarrow cvtColor(p_{bgr}, BGR2GRAY)$ ; // Grey is used for detecting face
6    $p_{half} \leftarrow resize(p_{grey}, 0.5f, 0.5f)$ ; // Scale down for better performance
7    $p_{half} \leftarrow equalizeHist(p_{half})$ ; // Histogram equalisation for better
   adaptation to different environmental lighting
8    $f[] \leftarrow CascadeClassifier :: detectMultiScale(p_{half}, parameters)$ ;
9    $l[][] \leftarrow FacemarkLBF :: fit(p_{half}, f[])$ ;
10  for  $i \leftarrow 0$  to sizeof  $f[]$  do
11     $p_{anonymous} \leftarrow drawCover(p_{bgr}, f[i])$ ;
12     $p_{anonymous} \leftarrow drawLandmarks(p_{anonymous}, l[i][[]])$ ;
13   $p_{rgba} \leftarrow cvtColor(p_{anonymous}, BGR2RGBA)$ ; // ANativeWindow requires
   image in RGBA format
14  enqueue(displayQueue,  $p_{rgba}$ ); // Enqueue to display thread
15  addImage( $p_{anonymous}$ ); // Perform temporal sampling and add to the
   current writing queue in the triple buffer

```

Algorithm 4.3 shows the simplified processing procedure in the image processing pipeline thread, which includes reading camera data, performing anonymisation, and sending the processed data to other threads. For the display thread, it only needs to dequeue the RGBA data processed and converted by Algorithm 4.3 from the *displayQueue*, and invoke the Android API to set the OpenGL texture. In this way, the new frame will be displayed

in real-time from the OpenGL rendering loop.

4.2.3 Tensorflow Lite adoption and model inference

Although Google provides some ready-made models and easy-to-call Java bindings, it is necessary to use C++ for custom model workflow. Besides, not all deep models implemented in TensorFlow can be converted to the Lite version due to limited operators compatibility. In the deep model conversion stage, this project firstly loads the trained model, then uses the *from_keras_model* mode³ of the TensorFlow Lite converter by specifying *SELECT_TF_OPS*⁴ parameter to allow the usage of certain TensorFlow operators in the TensorFlow Lite version.

To reduce the app size and make the converted model generated in the previous step better adapt to mobile devices, this project strips unnecessary operators by compiling the executable binary files from TensorFlow Lite source code according to the document⁵. Besides, another important reason why this project chooses to compile libraries from source code is to use TFLite C++ API as documented in the official guide⁶.

One failed attempt is trying to use the built-in optimisation of the TensorFlow Lite converter, such as pruning and quantisation introduced in mobile device optimisation section 2.4. However, due to an unresolved issue⁷ in the TensorFlow v2.5.0 library, further optimisation attempts are failed. After this issue of library is solved in the future, the model should be able to be further optimised.

After converting the model into the TFLite version and building the runtime binary, then it is time to implement the model inference thread. As mentioned earlier, the key task in the model inference thread is to use the frames received from the image processing pipeline for temporal sampling to write sampled frames to the triple buffer and invoke the TensorFlow Lite library. The temporal sampling algorithm as shown in Algorithm 4.4 is to sample 16 images from the video stream at an equal time interval same as one iteration in the model training process (25fps).

As for triple buffering, it is a bridge that connects the image processing thread and model inference thread. This data structure type has been widely used in the producer-consumer paradigm to deal with the inconsistency of the consumer has a slower speed than the producer. When the producer is much faster than the consumer, some data loss is inevitable, so it is only necessary to ensure that the consumer gets the latest data in this scenario.

³TensorFlow Lite converter: https://www.tensorflow.org/lite/convert#convert_a_keras_model_

⁴Select TensorFlow operators: https://www.tensorflow.org/lite/guide/ops_select

⁵Reduce TensorFlow Lite binary size: https://www.tensorflow.org/lite/guide/reduce_binary_size

⁶Use TFLite C++ API: https://www.tensorflow.org/lite/guide/android#use_tflite_c_api

⁷Did not get operators or tensors in subgraph: <https://github.com/tensorflow/tensorflow/issues/45313>

Algorithm 4.4: Temporal sampling and triple buffering

Data: Processed images from pipeline $x[]$; Current frame index C_{frame}
Result: Temporal sampled images queue $s[16]$; Image index array $p[16]$

```
1  $T_{now} \leftarrow getTime();$  // Assume time in second for simplex
2  $T_{diff} \leftarrow T_{now} - T_{last};$ 
3  $f \leftarrow \lfloor \frac{T_{diff} \times TARGET\_FPS}{10} \rfloor;$  // Sampling frames at 0s(frame 0), 0.4s, 0.8s,
   1.2s, 1.6s...6s(frame 15)
4 if  $f \geq MAX\_POSITION\_EMBEDDINGS - 2$  then
   // Position embedding overflown, restart frame sampling
5    $T_{last} \leftarrow T_{now};$ 
6    $C_{frame} \leftarrow 0;$ 
7   empty  $s$  and  $p$ ;
8 if  $f \geq sizeof\ s$  then
   // Sampling a frame to the queue
9    $push(s, x);$ 
10   $push(p, C_{frame});$ 
11  if  $sizeof\ s == BATCH\_FRAME\_NUM$  then
   // Finish sampling 16 frames
12   $T_{last} \leftarrow T_{now};$ 
13   $C_{frame} \leftarrow 0;$ 
14   $flipWriter();$  // flip the triple buffer
15  empty  $s$  and  $p$ ; // clear old data after flipped to new buffer
```

The implementation of triple buffering is an strengthened special case of ring buffering. In the triple buffering, there are three buffers in the memory, one is snap for reading from the consumer, and the other two are for flip writing from the producer. When the consumer reads, the newly written buffer will become a reading snap, and then the producer will write the released buffer and the remaining buffer. Thanks to the atomic operation support provided by C++, Neves [60] proposed a lock-free triple buffer implementation for the processes of creating snap and flip writing.

One problem with the initial version of this project is that the model inference thread may read empty data from the triple buffer because the image processing pipeline takes time to produce. To tinkle this problem, a *readLastBlock* is implemented using mutex and a conditional variable to block the model inference thread until the first set of image data is valid. After completing the implementation of temporal sampling and triple buffering, this project only need to copy the reading snap into tensor and invoke the TensorFlow Lite inference API to get the result.

5 Evaluation

After implementing system functions and confirming that the system is operating normally, it is necessary to evaluate the collected data quality, deep model performance, and mobile app performance.

This chapter first introduces each module that may need to be evaluated and the commonly used evaluation metrics in section 5.1. Then, the deep model and the mobile app will be evaluated in section 5.2 and section 5.3 respectively.

5.1 Evaluation metrics

In this study, the application of the deep model is to solve a multi-classification problem that classifies the video input from a mobile device into different exam activities. To introduce the evaluation metrics of multi-classification problems, it is necessary to start with the simplest binary classification problem.

		Prediction	
		Positive	Negative
True	Positive	TP: a correct result where the model correctly predicts the positive class	FN: a wrong result where the model predicts the positive class to negative class
	Negative	FP: also a wrong result where the model predicts the negative class to positive class	TN: also a correct result where the model correctly predicts the negative class

Table 5.1: Confusion matrix in binary classification

Table 5.1 shows the confusion matrix, a matrix commonly used to visually express the performance evaluation of a classifier. In the binary classification problem, there are four values in this matrix, namely True Positive(TP), False Negative(FN), False Positive(FP) and True Negative(TN), as explained in the table. It is worth noting that the diagonal

elements starting from the upper left corner represent the samples whose predicted label is equal to the true label, while the off-diagonal elements are samples incorrectly predicted by the classifier.

The binary classification problem has many evaluation metrics calculated using the four values in the confusion matrix, such as accuracy, accuracy, and recall. In order to adopt these metrics in multi-classification problems, a common solution is to convert it into n numbers of binary classification problems, that is, for a certain category, samples that are correctly classified as that category are True Positive, and samples that are correctly classified as not that category are True Negative. In multi-classification problems, the definition of some metrics have expanded to allow different ways of averaging. Table 5.2 illustrates widely used evaluation metrics summarised by Foss [61] under multi-classification model evaluation scenario, where M denotes macro-averaging.

Metric	Formula	Meaning
Average accuracy	$\frac{1}{k} \sum_{i=1}^k \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}$	A average per-class ratio between the number of correctly classified samples and the total number of samples.
Precision $_M$	$\frac{1}{k} \sum_{i=1}^k \frac{TP_i}{TP_i + FP_i}$	For each category, calculate the precision separately, and then take the average.
Recall $_M$	$\frac{1}{k} \sum_{i=1}^k \frac{TP_i}{TP_i + FN_i}$	For each category, calculate the recall separately, and then take the average, showing the average per-class effectiveness of a classifier to identify labels.
F1-score $_M$	$\frac{2 \times \text{Precision}_M \times \text{Recall}_M}{\text{Precision}_M + \text{Recall}_M}$	The harmonic mean of the macro-average precision and recall.

Table 5.2: The evaluation metrics, the formulae are summarised by Foss [61]

This study will then evaluate performances of the deep model using confusion matrix, average accuracy, as well as precision, recall and F1-score all in macro-average.

5.2 Deep model evaluation

Due to the relatively short period for data collection in this study, from the beginning of 2021 June to the end of July, the total size of validly labelled videos collected was approximately 440MB. By decoding these videos and videos randomly selected from the public data set Kinetics as the Unknown category into images, the total size of these images is approximately 2.3GB.

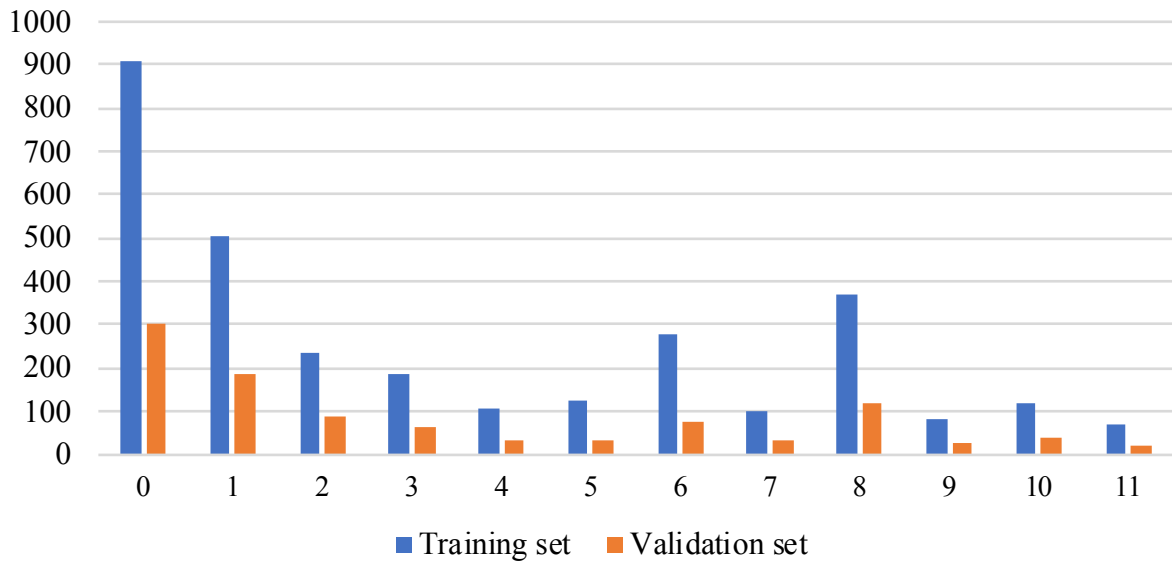


Figure 5.1: Data per category distribution

Figure 5.1 shows the per-category data distribution after dividing the anonymised data into training and validation sets. The horizontal axis of the histogram shows a total of 12 categories from 0 to 11. Table D1 in the model design chapter introduced the definition of each category. As for the vertical axis, it represents the number of labelled samples. A labelled sample has one label and 16 frames from temporal sampling in a video clip. To ensure that the validation data set has enough samples to illustrate the model performance, one-third of the total data set is divided into the validation data set, and the remaining two-thirds are used for training the deep model.

Overfitting is an unavoidable problem in deep learning, especially when using a complex model on small data sets. It is a phenomenon that the model loses its generalisation application ability, which manifests as the model has extremely small loss and very high accuracy on the training set, however, a overfitted model will perform badly on the validation set. The main reasons for overfitting and the corresponding solutions are as follows:

1. The size of the data set does not match the complexity of the model.
Solution: To use data augmentation or reduce the model complexity.
2. There are errors in the data sets, e.g. data have wrong labels, unshuffled data resulting in distribution difference between the training set and the validation set.
Solution: To visually inspect the data sets and ensure reliable data set labelling and distribution.
3. Over-training, the model has been trained too much.
Solution: To reduce the number of training epochs or use the early-stop mechanism.

ism to detect increasing loss in the validation set.

In this study, the above three solutions are all applied in the training process to avoid overfitting. In detail, the early-stop mechanism is implemented the model trainer, which monitors the loss value on the verification set after each training epoch. Once the verification loss is found to increase, it will stop training. The data loader uses common data augmentation methods in computer vision to perform the same transformation on the same batch of input data, such as scaling, randomly cropping, and flipping horizontally and vertically. Besides, the data sets are carefully validated and visually checked using Jupyter notebook with *matplotlib.animation* library.

Figure 5.2 visually illustrates the result from the warm-up learning rate scheduler function introduced in the training hyperparameters design subsection 3.3.3. The figure covers the training iterations to 6000 as the training process is expected to be completed within 10 epochs. Because this study uses a relatively small self-built data set and the deep model has a reduced complexity for mobile devices. The figure shows that the learning rate increases linearly and reaches the peak at the 2000th iteration, which is defined as a hyperparameter in the model trainer. After the learning rate reaches its peak, it gradually decreases with a negative exponent, making the training process enter the final fine-tuning stage.

Figure 5.3 and Figure 5.4 display the model training and validation iterations. At the beginning of the first epoch training, although the learning rate is low, the loss decreases steadily during this training epoch, indicating that the learning rate warm-up is effective to alleviate the overfitting. If there was no learning rate warm-up mechanism, the training loss would drop dramatically at the beginning, and the loss might increase in the subsequent training process, indicating that the model was overfitted. At the end of the first training epoch, the learning rate is about 0.00084, the loss drops to 1.92, and the accuracy on the training set achieves 0.4. As for the validation after the first training epoch, both the loss and the accuracy is about 0.8 because no data augmentation applied to the validation data set, making this data set is easier to classify.

Starting from the second epoch, the training loss instantly drops to about 1, and the

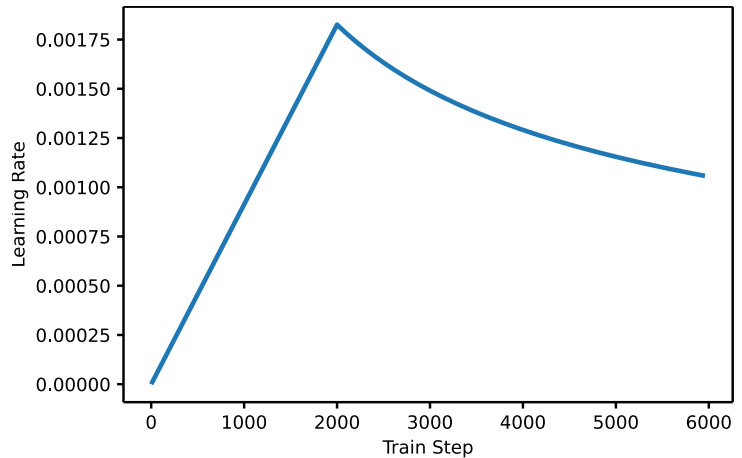


Figure 5.2: Learning rate warm-up

accuracy increases to 0.6 because all the training data has been completely input into the model once, thus, the model has already considerably understood the data form and distribution. During the training process, the loss of the training data set is always dropping, but the loss in the validation data set starts to rise from epoch 4, which means that over-train the model leads to overfitting.

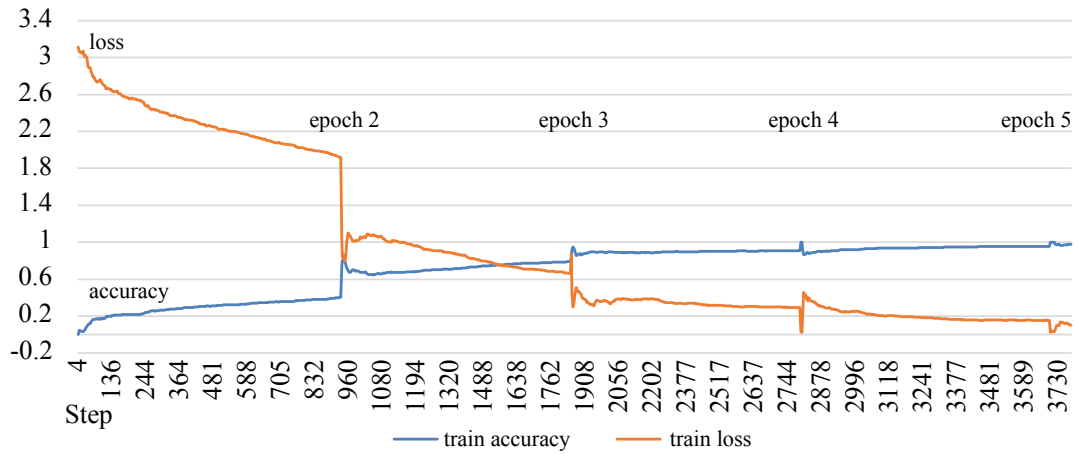


Figure 5.3: Model training steps

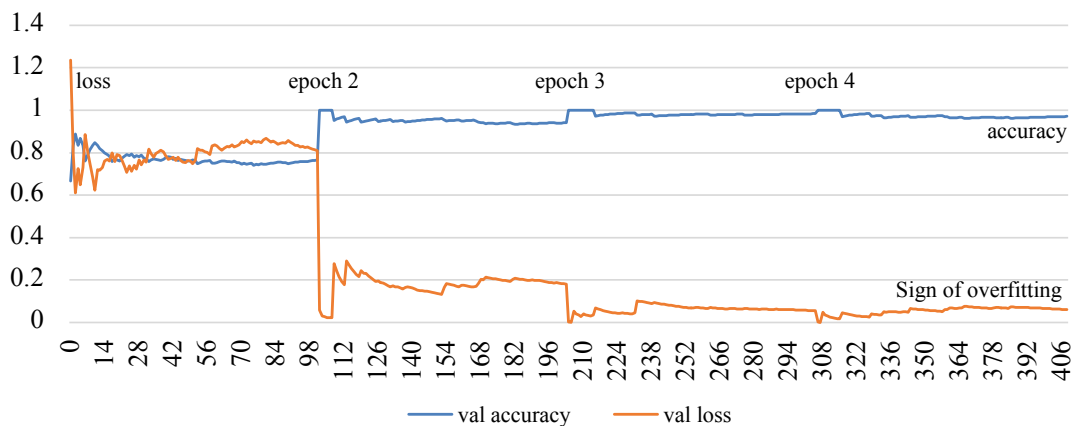


Figure 5.4: Model validation steps

Appendix E Figure E1 and Figure E2 shows two confusion matrices on validation and training data set, respectively. It is clear that most of samples are on the diagonal of the matrices, which means that they are classified correctly in both training and validation data set. The validation result vitally reflects the model performance on samples that have not been used in the training process.

According to the formulae introduced in the evaluation metrics section 5.1 above, Table 5.3 shows a classification report of the validation data on Average Accuracy; Precision, Recall and F1-Score on macro average. For those categories where precision and recall are 1.00, all samples in those categories are correctly classified. This classification report

highlights two data insights, first is that some samples from the Unknown category are classified to the other categories of exam activity, especially for the Leave category 5; second is that some samples labelled with 10 Drinking are classified to the 9 Scratching category.

	Precision	Recall	F1-Score	Support
0 Unknown	1.00	0.97	0.98	301
1 Look screen	0.99	1.00	0.99	189
2 Look down	1.00	1.00	1.00	89
3 Look side	1.00	1.00	1.00	64
4 Look back	1.00	1.00	1.00	31
5 Leave	0.87	0.94	0.90	35
6 Speaking	1.00	1.00	1.00	75
7 Look up	1.00	1.00	1.00	32
8 Use phone	0.98	1.00	0.99	122
9 Scratching	1.00	0.75	0.86	28
10 Drinking	0.83	1.00	0.91	39
11 Typing	1.00	1.00	1.00	22
Accuracy	0.98			1027
macro avg	0.97	0.97	0.97	1027
weighted avg	0.98	0.98	0.98	1027

Table 5.3: Classification report of validation data set

As for the first data insight, the Unknown category has more false-negative results than the others, which means that some videos from the Unknown category were classified into the categories of exam activity. Different proportions of the Unknown category to the exam activity categories will change the willingness of predicting the Unknown category. In this study, the ratio of Unknown category size and the size of the total exam activity categories is $\frac{2940}{1210} = 0.42$. In future works, more samples can be added into the Unknown category in the training data set to reduce false negatives in this category.

The second data insight is the misclassification between category 9 and category 10. This problem is mainly due to insufficient training data and a certain similarity in these two categories. For example, in both scratching and drinking activities, the participant needs to raise their hand first, do an action covering the face, and finally put their hand down. If more training data is available in future work, the performance of these two categories can be easily improved.

5.3 Mobile app evaluation

Since this study has the research scope of mobile platforms, experimenting with computation performance evaluation is necessary to show the deep model is optimised to

run inference on mobile devices. The experiment in this research uses two mobile phone models from OnePlus. The first is the OnePlus 5, a bit old phone released in June 2017, and the second is the OnePlus 8, one of the top phones from last year, unveiled on April 14, 2020. Table 5.4 shows the hardware details of these phones with the experiment results of average frame rate expressed in frames per second (FPS) metric.

Device	SoC	CPU	Process	Accelerator	RAM	FPS
OnePlus 5	Snapdragon 835	Kyro 280 2.46 GHz	10nm	Adreno 540 GPU Hexagon 682 DSP	8GB	18
OnePlus 8	Snapdragon 865	Kyro 585 2.84 GHz	7nm	Adreno 650 GPU Hexagon 698 DSP	12GB	28

Table 5.4: Experiment results on mobile devices

The result from the OnePlus 8 shows that the mobile app runs fast and smoothly on the high-end phone manufactured last year. Therefore, from 2021 and in the future, there will be no hardware performance barriers for mobile phones to running the deep model implemented in this study.

As for the result from the OnePlus 5, an old mobile phone released four years ago, the frame rate result is lower than the result from the OnePlus 8. Although users of old mobile phones may feel that the camera display in the app is a little lag and delayed, it does not crash or affect the user operation, UI response effect and the deep model inference because of the well-designed multi-threading architecture in the mobile app. Besides, exam attendees should not stare at or operate the invigilating mobile phone during the exam, slight lag and delay in the camera display affect nothing in the practical use case. As a result, a little old phones should be able to run the app as well.

This study originally planned to evaluate more devices and more performance metrics. However, due to the limited device resources, short research period and foreseeable cumbersome tasks, this part of the work has become one possible direction of future research. For example, future research may conduct a user study to evaluate the user interface and user logic in the proposed mobile app after the integration with an online exam platform. Besides, future research may evaluate more metrics on the computational efficiency, such as utilisation rate of CPU, GPU and neural network accelerator. More in-depth evaluation can further unveil the performance bottleneck in the deep model or program tasks so that future researches can optimise the system much specifically.

6 Conclusion

The previous chapters have outlined each part of the system from the design and implementation aspects. The evaluation chapter has shown and critically analysed the experiment results of the deep model evaluation and the mobile application performance in multiple evaluation metrics. This chapter will summarise the evaluation results and draw research conclusions regarding the research question and propose possible future work in this field.

6.1 Summary

The evaluation results have verified the research question that it is possible to classify the exam attendee's activities into approved or prohibited behaviours using an on-device camera video input stream. Through my enormous efforts contributed to this research project and dissertation, I have completed this study and summarised my contributions to the following points.

- This study spots an urgent need during the COVID-19 epidemic on a human action recognition task – the remote exam invigilation.
- This research surveys and reviews previous research on the design details and mobile optimisation insights of many related deep models.
- This study designs and implements a web-based app to collect labelled video data from research participants.
- This study designs and implements a deep model to accomplish the research goal by applying the deep-learning-based video understanding technique.
- This study ports and optimises the proposed deep model to Android mobile devices, which validates the feasibility of running the app equipped with the deep model.
- This study evaluates the proposed deep model in multiple metrics and conducts performance experiments on two Android phone models, OnePlus 5 and OnePlus 8.

The solution proposed in this research only solves remote exam invigilation in physical space, ensured exam attendees only interact with the device used to sit the exam. The limitation of the solution lies in the inability of controlling software on the personal device. For example, a dishonest student can use remote desktop software to enable impersonation, letting others cheat for the exam through the Internet. Because the dishonest student does not do any prohibited activities in the physical space, the solution does not apply to this situation.

6.2 Future work

As for future work, the possible future research directions in the field of remote examination invigilation are considered based on the project outcomes shown above, which is mainly four directions – data set volume, deep model optimisation for both model architecture and mobile devices, integration with online exam platforms and areas of security and privacy. The first two of these research directions are pushing forward the deep model performance. And the last two directions are considering more user experience engineering or related to security and privacy, which is beyond the research scope of deep learning.

In detail, the first enhancement direction is generally effective to any deep models, that is, to increase the total data set volume. The evaluation result unveils that the deep model still has the potential and capacity to train on larger data sets. Future research can integrate data collection into the mobile app and continuously improve the model to adapt to various video angles.

The second way of improving the system performance is to further optimise the deep model on the accuracy and computational complexity. Factorised model variants used in Video Vision Transformer network proposed by Arnab et al. [55] pointed out a method to further enhance the performance of deep models designed for video understanding tasks. The idea behind the factorisation is to decompose the monolithic structure of the spatial encoder first and then the temporal encoder into more and smaller encoders distributed in layers. More communications and fusions between spatial and temporal data facilitates the model to capture richer features across spatiotemporal space.

In 2021 August, as this dissertation is about to be completed, Chen et al. [62] proposed a Mobile-Former that bridges MobileNet and Transformer. They creatively proposed a parallel design of MobileNet and Transformer, interspersed with the bidirectional fusion of “MobileNet at local processing and Transformer at global interaction”, which takes the advantage *de mutuo auxilio* (of mutual aid) from both convolution and Transformer. Future works may use the Mobile-Former to achieve better computational efficiency and more representation power. Besides, more deep learning frameworks are worth trying out in the model implementation. For example, Reiss [63] presented that PyTorch used TorchScript to enable running on mobile devices and supported NNAPI by the end of 2020. On this basis, A deep learning library for video understanding research¹ is published during this research period and is worth trying in future work.

¹PyTorchVideo: <https://github.com/facebookresearch/pytorchvideo>

Bibliography

- [1] Giorgio Marinoni, Hilligje Van't Land and Trine Jensen. 'The impact of Covid-19 on higher education around the world'. In: *IAU Global Survey Report* (2020).
- [2] Ted M. Clark et al. 'Testing in the Time of COVID-19: A Sudden Transition to Unproctored Online Exams'. In: *Journal of Chemical Education* 97.9 (2020), pp. 3413–3417. DOI: 10.1021/acs.jchemed.0c00546.
- [3] Kevin D. McCay et al. 'Abnormal Infant Movements Classification With Deep Learning on Pose-Based Features'. In: *IEEE Access* 8 (2020), pp. 51582–51592. DOI: 10.1109/ACCESS.2020.2980269.
- [4] Simon Coghlan, Tim Miller and Jeannie Paterson. 'Good proctor or "Big Brother"? AI Ethics and Online Exam Supervision Technologies'. In: *arXiv:2011.07647* (2020). arXiv: 2011.07647 [cs.CY].
- [5] Aras Bozkurt and Ramesh C Sharma. 'Education in normal, new normal, and next normal: Observations from the past, insights from the present and projections for the future'. In: *Asian Journal of Distance Education* 15.2 (2020), pp. i–x.
- [6] Sidney Fels and Kenji Mase. 'Interactive video cubism'. In: *Proceedings of the 1999 workshop on new paradigms in information visualization and manipulation in conjunction with the eighth ACM international conference on Information and knowledge management*. 1999, pp. 78–82.
- [7] Michelle Thrasher et al. 'Mood Recognition Based on Upper Body Posture and Movement Features'. In: *Affective Computing and Intelligent Interaction*. Springer Berlin Heidelberg, 2011, pp. 377–386. DOI: 10.1007/978-3-642-24600-5_41.
- [8] Li Yao, Yunjian Liu and Shihui Huang. 'Spatio-temporal information for human action recognition'. In: *EURASIP Journal on Image and Video Processing* 2016.1 (2016), pp. 1–9. DOI: 10.1186/s13640-016-0145-2.
- [9] Christian Schuldt, Ivan Laptev and Barbara Caputo. 'Recognizing human actions: a local SVM approach'. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004*. Vol. 3. 2004, 32–36 Vol.3. DOI: 10.1109/ICPR.2004.1334462.

- [10] Marcin Marszalek, Ivan Laptev and Cordelia Schmid. 'Actions in context'. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 2929–2936. DOI: 10.1109/CVPR.2009.5206557.
- [11] Khurram Soomro and Amir R. Zamir. 'Action Recognition in Realistic Sports Videos'. In: *Computer Vision in Sports*. Cham: Springer International Publishing, 2014, pp. 181–208. ISBN: 978-3-319-09396-3. DOI: 10.1007/978-3-319-09396-3_9.
- [12] Issa Traoré et al. 'Ensuring online exam integrity through continuous biometric authentication'. In: *Information Security Practices*. Springer, 2017, pp. 73–81.
- [13] Shoko Yasuda and Hiroyuki Ogeta. 'Examinee Authentication from a Hand Image on a Tablet Computer for Preventing Impersonation(タブレット型端末を用いた試験における替え玉検出を目的とした手形状による本人認証)'. In: *Japan Journal of Educational Technology(日本教育工学会論文誌)* 44.4 (2021), pp. 419–429. DOI: 10.15077/jjet.44107.
- [14] Kavish Garg et al. 'Convolutional Neural Network based Virtual Exam Controller'. In: *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE. 2020, pp. 895–899.
- [15] Kunihiko Fukushima. 'Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position'. In: *Biological Cybernetics* 36.4 (1980), pp. 193–202.
- [16] Wei Zhang et al. 'Shift-invariant pattern recognition neural network and its optical architecture'. In: *Proceedings of annual conference of the Japan Society of Applied Physics*. 1988.
- [17] Yann LeCun et al. 'Backpropagation applied to handwritten zip code recognition'. In: *Neural computation* 1.4 (1989), pp. 541–551.
- [18] Wei Zhang et al. 'Computerized detection of clustered microcalcifications in digital mammograms using a shift-invariant artificial neural network'. In: *Medical physics* 21.4 (1994), pp. 517–524.
- [19] Yann LeCun, Koray Kavukcuoglu and Clément Farabet. 'Convolutional networks and applications in vision'. In: *Proceedings of 2010 IEEE international symposium on circuits and systems*. IEEE. 2010, pp. 253–256.
- [20] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. 'Imagenet classification with deep convolutional neural networks'. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.
- [21] Karen Simonyan and Andrew Zisserman. 'Very deep convolutional networks for large-scale image recognition'. In: *arXiv preprint arXiv:1409.1556* (2014).

- [22] Kaiming He et al. ‘Deep Residual Learning for Image Recognition’. In: 2015. arXiv: 1512.03385 [cs.CV].
- [23] MI Jordan. *Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986*. Tech. rep. California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986.
- [24] Jeffrey L Elman. ‘Finding structure in time’. In: *Cognitive science* 14.2 (1990), pp. 179–211.
- [25] Sepp Hochreiter and Jürgen Schmidhuber. ‘Long short-term memory’. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [26] F. Gers, J. Schmidhuber and F. Cummins. ‘Learning to Forget: Continual Prediction with LSTM’. In: *Neural Computation* 12 (2000), pp. 2451–2471.
- [27] Michael Phi. *Illustrated Guide to LSTM’s and GRU’s: A step by step explanation*. Sept. 2018. URL: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>.
- [28] Junyoung Chung et al. ‘Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling’. In: (2014). arXiv: 1412.3555 [cs.NE].
- [29] Volodymyr Mnih, Nicolas Heess, Alex Graves et al. ‘Recurrent models of visual attention’. In: *Advances in neural information processing systems*. 2014, pp. 2204–2212.
- [30] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. ‘Neural Machine Translation by Jointly Learning to Align and Translate’. In: (2016). arXiv: 1409.0473 [cs.CL].
- [31] Ashish Vaswani et al. ‘Attention Is All You Need’. In: (2017). arXiv: 1706.03762 [cs.CL].
- [32] TensorFlow Authors. *Transformer model for language understanding*. Aug. 2021. URL: <https://www.tensorflow.org/text/tutorials/transformer>.
- [33] Jacob Devlin et al. ‘BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding’. In: (2019). arXiv: 1810.04805 [cs.CL].
- [34] Jianyuan Guo et al. ‘CMT: Convolutional Neural Networks Meet Vision Transformers’. In: (2021). arXiv: 2107.06263 [cs.CV].
- [35] Zihang Dai et al. ‘CoAtNet: Marrying Convolution and Attention for All Data Sizes’. In: (2021). arXiv: 2106.04803 [cs.CV].
- [36] Yunbin Deng. ‘Deep learning on mobile devices: a review’. In: *Mobile Multimedia/Image Processing, Security, and Applications 2019*. Ed. by Sos S. Agaian, Vijayan K. Asari and Stephen P. DelMarco. Vol. 10993. International Society for Optics and Photonics. SPIE, 2019, pp. 52–66. doi: 10.1117/12.2518469.

- [37] Yanjiao Chen et al. ‘Deep Learning on Mobile and Embedded Devices: State-of-the-art, Challenges, and Future Directions’. In: *ACM Computing Surveys (CSUR)* 53.4 (2020), pp. 1–37.
- [38] Andrew G. Howard et al. ‘MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications’. In: (2017). arXiv: 1704.04861 [cs.CV].
- [39] Liang-Chieh Chen et al. ‘Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation’. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.
- [40] Mark Sandler et al. ‘MobileNetV2: Inverted Residuals and Linear Bottlenecks’. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018, pp. 4510–4520.
- [41] Mingxing Tan and Quoc V. Le. ‘EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks’. In: (2020). arXiv: 1905.11946 [cs.LG].
- [42] Mingxing Tan and Quoc V. Le. ‘EfficientNetV2: Smaller Models and Faster Training’. In: (2021). arXiv: 2104.00298 [cs.CV].
- [43] Zhanghao Wu et al. ‘Lite Transformer with Long-Short Range Attention’. In: (2020). arXiv: 2004.11886 [cs.CL].
- [44] Iz Beltagy, Matthew E. Peters and Arman Cohan. ‘Longformer: The Long-Document Transformer’. In: (2020). arXiv: 2004.05150 [cs.CL].
- [45] Sachin Mehta et al. ‘DeLighT: Deep and Light-weight Transformer’. In: (2021). arXiv: 2008.00623 [cs.LG].
- [46] Di Wu, Nabin Sharma and Michael Blumenstein. ‘Recent advances in video-based human action recognition using deep learning: A review’. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2865–2872. doi: 10.1109/IJCNN.2017.7966210.
- [47] Du Tran et al. ‘Learning spatiotemporal features with 3d convolutional networks’. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4489–4497.
- [48] Moez Baccouche et al. ‘Sequential Deep Learning for Human Action Recognition’. In: *Human Behavior Understanding*. Springer Berlin Heidelberg, 2011, pp. 29–39. ISBN: 978-3-642-25446-8.
- [49] Shuiwang Ji et al. ‘3D Convolutional Neural Networks for Human Action Recognition’. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1 (2013), pp. 221–231. doi: 10.1109/TPAMI.2012.59.
- [50] Karen Simonyan and Andrew Zisserman. ‘Two-Stream Convolutional Networks for Action Recognition in Videos’. In: (2014). arXiv: 1406.2199 [cs.CV].

- [51] Christoph Feichtenhofer, Axel Pinz and Andrew Zisserman. ‘Convolutional Two-Stream Network Fusion for Video Action Recognition’. In: (2016). arXiv: 1604.06573 [cs.CV].
- [52] Joao Carreira and Andrew Zisserman. ‘Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset’. In: (2018). arXiv: 1705.07750 [cs.CV].
- [53] Rohit Girdhar et al. ‘Video action transformer network’. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 244–253.
- [54] Daniel Neimark et al. ‘Video Transformer Network’. In: (2021). arXiv: 2102.00719 [cs.CV].
- [55] Anurag Arnab et al. ‘ViViT: A Video Vision Transformer’. In: (2021). arXiv: 2103.15691 [cs.CV].
- [56] P. Viola and M. Jones. ‘Rapid object detection using a boosted cascade of simple features’. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. 2001, pp. I–I. doi: 10.1109/CVPR.2001.990517.
- [57] Mooseop Kim, Deokgyu Lee and Ki-Young Kim. ‘System architecture for real-time face detection on analog video camera’. In: *International Journal of Distributed Sensor Networks* 11.5 (2015), p. 251386. doi: 10.1155/2015/251386.
- [58] Shaoqing Ren et al. ‘Face Alignment at 3000 FPS via Regressing Local Binary Features’. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.
- [59] Yue Wu and Qiang Ji. ‘Facial landmark detection: A literature survey’. In: *International Journal of Computer Vision* 127.2 (2019), pp. 115–142.
- [60] André Pacheco Neves. *Triple Buffering as a Concurrency Mechanism*. Jan. 2012. URL: http://remis-thoughts.blogspot.pt/2012/01/triple-buffering-as-concurrency_30.html.
- [61] Vicki Foss. *Multiclass Classification Model Evaluation*. Dec. 2018. URL: <https://parasite.id/blog/2018-12-13-model-evaluation/>.
- [62] Yinpeng Chen et al. ‘Mobile-Former: Bridging MobileNet and Transformer’. In: (2021). arXiv: 2108.05895 [cs.CV].
- [63] David Reiss. *PyTorch Mobile Now Supports Android NNAPI*. Dec. 2020. URL: <https://medium.com/pytorch/pytorch-mobile-now-supports-android-nnapi-e2a2aeb74534>.
- [64] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O’Reilly Media, Inc.", 2008.

- [65] Joseph Howse and Joe Minichino. *Learning OpenCV 4 Computer Vision with Python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning*. Packt Publishing Ltd, 2020.
- [66] Nishant Ghanate, Kartik Bhagat and Sandeep Gamot. 'Smart Security System: A Surveillance System Based on OpenCV and Android Platform'. In: *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology* 12.SUP 1 (2020), pp. 32–35.
- [67] Martin Abadi et al. 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems'. In: *arXiv: Distributed, Parallel, and Cluster Computing* (2015).
- [68] Pramod Singh and Avinash Manure. 'Introduction to TensorFlow 2.0'. In: *Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python*. Berkeley, CA: Apress, 2020, pp. 1–24. ISBN: 978-1-4842-5558-2. DOI: 10.1007/978-1-4842-5558-2_1. URL: https://doi.org/10.1007/978-1-4842-5558-2_1.
- [69] Benoit Steiner et al. 'PyTorch: An imperative style, high-performance deep learning library'. In: *Advances in Neural Information Processing Systems*. 2019, pp. 8026–8037.
- [70] Felipe De Almeida Florencio et al. 'Performance Analysis of Deep Learning Libraries: TensorFlow and PyTorch'. In: *Journal of Computer Science* 15.6 (2019), pp. 785–799.
- [71] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.
- [72] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.
- [73] Chunjie Luo et al. 'Comparison and Benchmarking of AI Models and Frameworks on Mobile Devices'. In: (2020). arXiv: 2005.05085 [cs.LG].
- [74] Anubhav Singh and Rimjhim Bhadani. *Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter: Build scalable real-world projects to implement end-to-end neural networks on Android and iOS*. Packt Publishing Ltd, 2020.
- [75] Umi Fadlilah, Bana Handaga et al. 'The Development of Android for Indonesian Sign Language Using Tensorflow Lite and CNN: An Initial Study'. In: *Journal of Physics: Conference Series*. Vol. 1858. 1. IOP Publishing. 2021, p. 012085.
- [76] Weijian Chen, Shujing Wang and Jingeng Wang. 'Realtime Multi-Person Pose Estimation Based on Android System'. In: *2020 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*. IEEE. 2020, pp. 286–289.
- [77] Ryan Dahl. *Nodejs - A JavaScript runtime built on Chrome's V8 JavaScript engine*. June 2021. URL: <https://web.archive.org/web/20210607053216/https://nodejs.org/en/>.

- [78] Facebook. *React - A JavaScript library for building user interfaces*. June 2021. URL: <https://web.archive.org/web/20210607044826/https://reactjs.org/>.
- [79] Dan Abramov and Andrew Clark. *Redux - A Predictable State Container for JS Apps*. June 2021. URL: <https://web.archive.org/web/20210601113657/https://redux.js.org/>.
- [80] Alex Banks and Eve Porcello. *Learning React: functional web development with React and Redux*. " O'Reilly Media, Inc.", 2017.
- [81] Bonnie Eisenman. *Learning react native: Building native mobile apps with JavaScript*. " O'Reilly Media, Inc.", 2015.

Appendix A Code repositories

The source code is submitted to college along with this dissertation, and the GitHub source code repository will be enabled for public access after graduation.

- Dissertation source code in \LaTeX
<https://github.com/comword/TCD20-DP-Dissertation>
- Data collection web recorder
Front-end: <https://github.com/comword/TCD20-DP-WebRecorder>
Back-end: <https://github.com/comword/TCD20-DP-WebRecorderServer>
- Data processing and deep learning model
<https://github.com/comword/TCD20-DP-DeepModel>
- Student mobile application
<https://github.com/comword/TCD20-DP-MobileApp>
Submodule libraries:
 - Compiled OpenCV binary for Android
<https://github.com/comword/opencv-android-bin>
 - Compiled Tensorflow Lite binary for Android
<https://github.com/comword/tflite-bin>
 - Compiled gRPC binary for Android
<https://github.com/comword/grpc-android-bin>

Appendix B Framework details

Because the scope of this research is relatively wide, covering data collection, model implementation, and mobile app implementation, many frameworks and libraries are used in implementing the system. Obviously, depending on target platforms, the programming language used in each part is also different. The next subsections will introduce both programming languages and frameworks used to develop the corresponding subsystem.

Appendix B.1 Web development

Web technology is developing and advancing rapidly, which is the reason for the extensive use of web technology in this study. From the earliest era of directly programming HTML, CSS, and JavaScript, the web technology developed through templated server-side-rendering, such as PHP Hypertext Preprocessor or Jakarta Server Pages (JSP). Recently, web development has become more structured and engineered by using client-side rendering framework for front-end, such as React, Vue or Angular and microservice framework for back-end. Cross-platform UI frameworks such as Electron, React-Native, and Weex make web technology the preferred solution for developing cross-platform applications on the client side.

Web technologies have been majorly used both in the web-based data collection app and the mobile app user interface/user experience(UI/UX) design. TypeScript, a strongly typed programming language building upon JavaScript, is used for the web front-end development of the data collection app. As for the back-end development, this study uses Golang for a better gRPC experience.

Node.js is used as the front-end development environment in this study. It was initiated by Dahl [77] in mid-2009, aimed to create a high-efficient JavaScript runtime. Node.js is built on Google Chrome's V8 JavaScript engine, one of the most efficient JavaScript engines and is widely adopted in front-end development because of its powerful package manager and excellent ecosystem for rich frameworks and libraries.

The data collection app requires a user interface, implement data flows, thus, using a mature front-end framework is the best choice. Although many famous frameworks to choose from, such as React, Vue and Angular, I finally decided to use React, a declarative, efficient, and flexible JavaScript library for building user interfaces. Because it is less complex compared to the other alternatives and it is being supported and maintained

by Facebook [78]. It also has a good ecosystem and many successful apps are developed based on it, which is beneficial to explore in the project.

As for Redux, it is a predictable state container for JavaScript applications also a good companion with React, developed by Abramov and Clark [79]. It helps create applications that behave consistently and predictably with complicated states, and keeps the states and connects each state with views in React. Any action that causes change to the state is reversible with a time-travelling debugger, which provides a great developing experience.

Appendix B.2 Deep learning model development

In the field of deep learning, TensorFlow and PyTorch are two famous and widely used frameworks. They both support the use of GPU for acceleration in the model training process, but there are still many differences in programming paradigm and application areas.

PyTorch is an open source Python library widely used in the academic. It was published in early-2017 by the Facebook's AI Research Lab. In 2019, Steiner et al. [69] pointed out that Caffe, TensorFlow (version 1.x at that time), and Theano all construct a static computation dataflow graph to achieve high performance at the cost of usability, debugging, and flexibility, but PyTorch is a framework with dynamic eager execution, which enables high usability without sacrificing performance. Another study by Florencio et al. [70] in the same year conducted a detailed comparative study on the performance of TensorFlow and PyTorch. After data analysis, they concluded that TensorFlow has higher GPU utilisation, but the PyTorch shows better overall performance.

As for TensorFlow, Abadi et al. [67] proposed that it is a deep learning framework developed by the Google Brain team in 2015. It has a relatively long history in the deep learning field and is widely adopted in enterprises and industries. In TensorFlow 1.x version, the computation dataflow graph is statically defined through *tf.session* and *tf.placeholder* before running the model. This old way sacrifices usability for performance and makes it impossible to set breakpoints to view data during the computation process, which causes great inconvenience to the debugging progress.

However, after the release of TensorFlow version 2.0, the previous deficiencies have been well addressed. Singh and Manure [68] compared the major changes of TensorFlow from 1.x to 2.0 in three aspects, including usability, performance, and deployment. In terms of usability, the framework provides easier APIs, which are divided into Keras-based high-level APIs and low-level APIs. Through the high-level API, a deep layer can be directly created with one line of code in high-level APIs, such as *tf.keras.layers.Dense*

to create a fully connected layer. Low-level APIs allow direct data manipulation, such as *tf.matmul* for matrix multiplication and *tf.transpose* for matrix transpose.

In terms of performance on mobile devices, Luo et al. [73] highlighted that only TensorFlow Lite supports NNAPI delegate, an Android Neural Networks API available after Android 8.1, “aiming to accelerate for AI models on Android devices with supported hardware accelerators”, said by Luo et al. [73]. The experiment result also shows that TensorFlow Lite with NNAPI delegate achieves the fastest inference speed in the performance comparison across multiple models.

In order to maximise the performance and computational efficiency, this study implements the deep learning model using the TensorFlow framework because the target deployment platform is the Android mobile devices, and TensorFlow Lite has the best compatibility and support for Android.

Appendix B.3 Android application development

As an early and widely used deep learning framework, TensorFlow has a mature deployment process and application example on mobile devices. For example, Fadlilah, Handaga et al. [75] developed an Android app for sign language by using TensorFlow Lite to run a convolutional neural network. This framework not only has a large number of successful application precedents but also has detailed documents, books, and tutorials. Singh and Bhadani [74] wrote a book that shows multiple examples of TensorFlow Lite deployment and mobile application development with Flutter framework, a UI toolkit in Dart programming language released by Google.

The Flutter framework is more efficient and developed by Google, in which all Dart code can compile by ahead-of-time (AOT) into native code during deployment. However, the usage of Dart programming language is rare, which means a high learning cost for developers and the available libraries are far less than mainstream programming languages.

A more famous framework is React Native maintained by Facebook, which fully uses the existing web technology to develop iOS and Android mobile applications. By adopting this framework, front-end developers familiar with React framework can develop mobile applications without difficulties and learning costs. Since its release in 2015, Eisenman [81] pointed out that it has attracted a large number of front-end developers and can use a majority of existing JavaScript libraries.

As a result, this study chose to use the React Native framework to implement the mobile app with deep model deployed in TensorFlow Lite library.

Appendix C Web recorder designs

Appendix C.1 Functionality description in user stories

User role	Capability	Reason
Researcher	Publish new tasks, delete unnecessary tasks and change description of tasks	The task list needs to be updated in real time according to required activity categories
Researcher	Query videos contributed and uploaded by participant	Obtain videos to train the model
Researcher	Delete opt-out user's information and uploaded videos as required	According to research ethics requirements
Participant	Read the informed consent form and register an account if agree	According to research ethics requirements
Participant	Login and authenticate to the app, reset forgotten password through email	Basic requirements for any access-controlled system
Participant	Select the currently available tasks published by the researcher and start the recording process	The videos uploaded by the user should be automatically labelled by the system
Participant	Record the videos through the on-device camera	Main function of the system
Participant	Review and record recorded videos to server	Main function of the system
Participant	Decide to opt-out the study and remove all contributions	According to research ethics requirements

Table C1: User stories for the video data collection app

Appendix C.2 Participant use cases and user logics

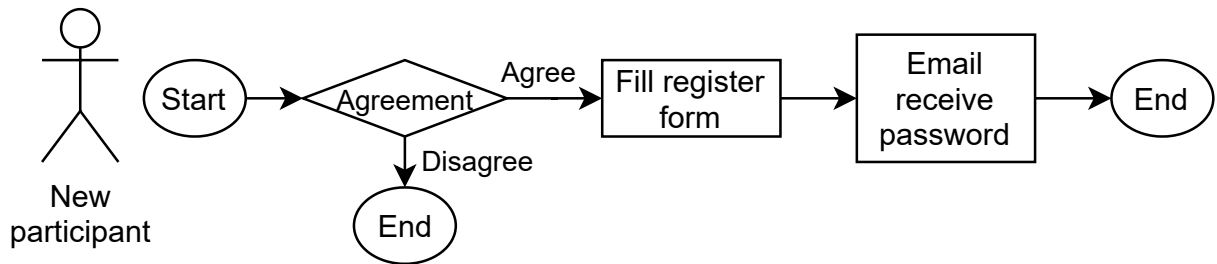


Figure C1: New participant registration

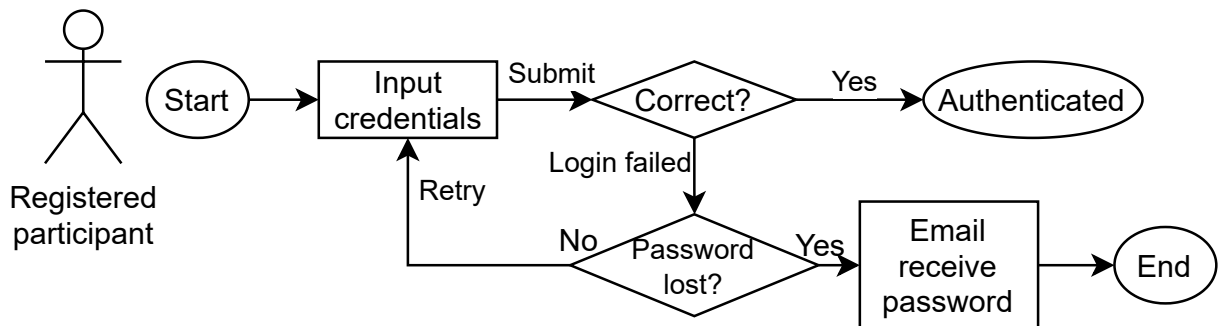


Figure C2: Registered participant login

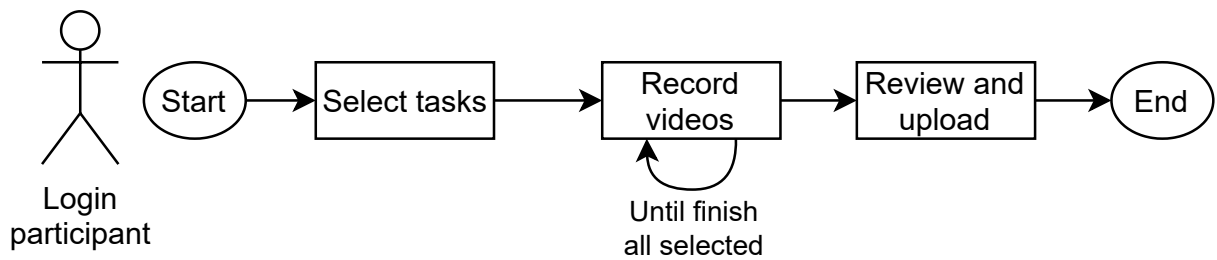


Figure C3: Participant recording and uploading

Appendix D Model design details

Appendix D.1 Output categories

ID	Activity	Alert level	Description
0	Unknown	Amber	The model cannot predict the current activity that may be a non-exam activity
1	Look screen	Green	The student is looking at the screen
2	Look down	Alert	The student is looking under the table, which maybe using the phone
3	Look side	Amber	The student is looking outside the computer screen, maybe reading a book on the table
4	Look back	Amber	The student turned and is looking behind, maybe hiding something
5	Leave	Amber	The student is out of the camera range
6	Speaking	Alert	The student has obvious lip movement and may be talking to others
7	Look up	Green	The student looks up at the ceiling, may be relaxing or thinking a difficult question
8	Use phone	Alert	The student are obviously using a mobile phone
9	Scratching	Green	The student is scratching the head or face and may be thinking a difficult question
10	Drinking	Green	The student is drinking water
11	Typing	Green	The student is typing on the keyboard and answering exam questions
12 – 15	Unused	–	Allow for adding more activities

Table D1: Model output for exam activity categories

Appendix D.2 Model detailed layers

Name	Type	Output Shape	Param #
Images input as the shape (<i>Batch, Channel, Frame, Height, Width</i>)			
input_1	InputLayer	?, 3, 16, 224, 224	0
tf.compat.v1.shape	TFOpLambda	5,	0
tf.__operators__.getitem	SlicingOpLambda		0
tf.__operators__.getitem_2	SlicingOpLambda		0
tf.compat.v1.transpose	TFOpLambda	?, 16, 224, 224, 3	0
tf.math.multiply	TFOpLambda		0
tf.__operators__.getitem_3	SlicingOpLambda		0
tf.__operators__.getitem_4	SlicingOpLambda		0
tf.__operators__.getitem_1	SlicingOpLambda		0
tf.reshape	TFOpLambda	?, ?, ?, ?	0
Images process to (<i>Batch × Frame, Height, Width, Channel</i>)			
efficientnetb0	Functional	?, 7, 7, 384	3759267
layer_normalization	LayerNormalization	?, 7, 7, 384	768
global_max_pooling2d	GlobalMaxPooling2D	?, 384	0
The CNN-based backbone (EfficientNet-B0) output extracted features The frame indices input as the shape (<i>Batch, Index</i>), add position embedding			
tf.reshape_1	TFOpLambda	?, ?, ?	0
tf.compat.v1.shape_1	TFOpLambda	3,	0
tf.__operators__.getitem_5	SlicingOpLambda		0
tf.__operators__.getitem_7	SlicingOpLambda		0
tf.broadcast_to	TFOpLambda	?, 1, 384	0
tf.concat	TFOpLambda	?, ?, 384	0
tf.compat.v1.shape_2	TFOpLambda	3,	0
tf.__operators__.getitem_8	SlicingOpLambda		0
tf.math.floormod	TFOpLambda		0
tf.math.subtract	TFOpLambda		0
tf.math.floormod_1	TFOpLambda		0
input_2	InputLayer	?, 16	0
tf.convert_to_tensor_2	TFOpLambda	2, 2	0
tf.compat.v1.pad_3	TFOpLambda	?, ?	0
tf.__operators__.getitem_6	SlicingOpLambda		0
tf.cast_2	TFOpLambda	?, ?	0
tf.zeros	TFOpLambda	?, ?	0
tf.math.floormod_2	TFOpLambda	?, ?	0
tf.compat.v1.pad	TFOpLambda	?, ?	0
tf.convert_to_tensor	TFOpLambda	3, 2	0

tf.compat.v1.shape_4	TFOpLambda	2,	0
tf.compat.v1.transpose_1	TFOpLambda	?, ?	0
tf.ones	TFOpLambda	?, ?	0
tf.broadcast_to_1	TFOpLambda	?, 1	0
tf.compat.v1.pad_1	TFOpLambda	?, ?, ?	0
tf.__operators__.getitem_10	SlicingOpLambda		0
tf.cast	TFOpLambda	?, ?	0
tf.concat_1	TFOpLambda	?, ?	0
tf.math.subtract_2	TFOpLambda		0
tf.compat.v1.shape_3	TFOpLambda	3,	0
tf.where	TFOpLambda	?, ?	0
tf.convert_to_tensor_1	TFOpLambda	2, 2	0
tf.zeros_2	TFOpLambda	?, ?	0
tf.__operators__.getitem_9	SlicingOpLambda		0
tf.compat.v1.pad_2	TFOpLambda	?, ?	0
tf.compat.v1.pad_5	TFOpLambda	?, ?	0
tf.math.subtract_1	TFOpLambda		0
tf.math.not_equal	TFOpLambda	?, ?	0
tf.compat.v1.transpose_3	TFOpLambda	?, ?	0
tf.zeros_1	TFOpLambda	?, ?	0
tf.cast_3	TFOpLambda	?, ?	0
tf.cast_4	TFOpLambda	?, ?	0
tf.compat.v1.pad_4	TFOpLambda	?, ?	0
tf.__operators__.eq	TFOpLambda	?, ?	0
tf.where_1	TFOpLambda	?, ?	0
tf.compat.v1.transpose_2	TFOpLambda	?, ?	0
tf.where_2	TFOpLambda	?, ?	0
tf.cast_1	TFOpLambda	?, ?	0

Extracted features, attention mask, and position embedding input to the Transformer

vtn_longformer_layer	VTNLongformerLayer	?, ?, 384	19667712
tf.__operators__.getitem_11	SlicingOpLambda	?, 384	0
layer_normalization_1	LayerNormalization	?, 384	768

MLP head to output the classification

dense_1	Dense	?, 384	147840
dropout_8	Dropout	?, 384	0
dense_2	Dense	?, 16	6160

Total params: 23,582,899
Trainable params: 23,542,668
Non-trainable params: 40,231

Appendix E Evaluation result

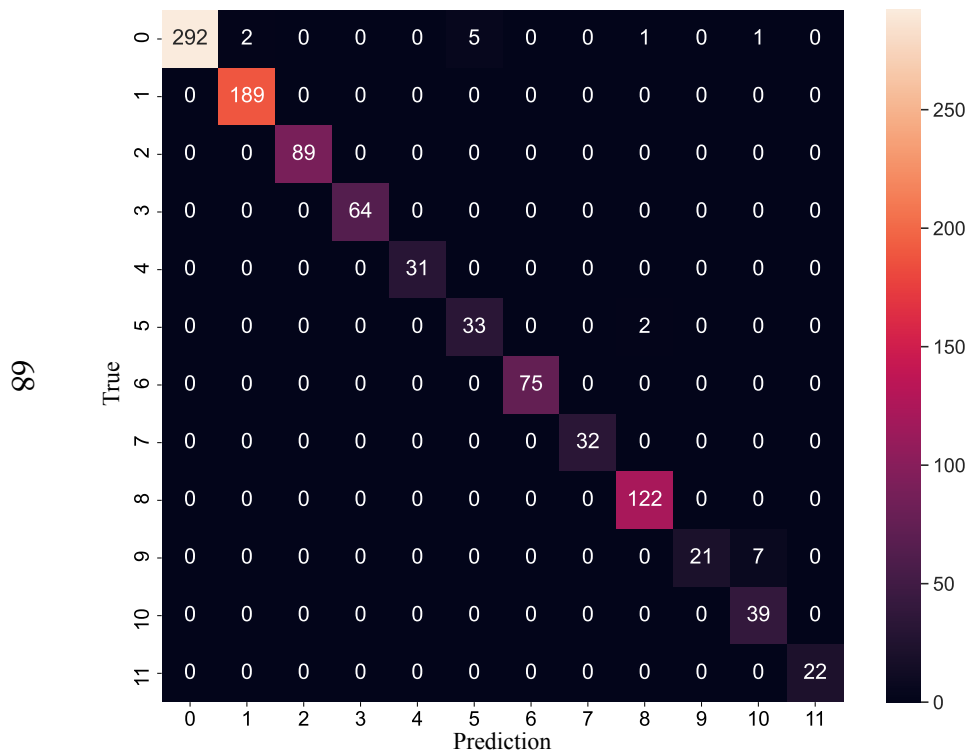


Figure E1: Confusion matrix on validation data set

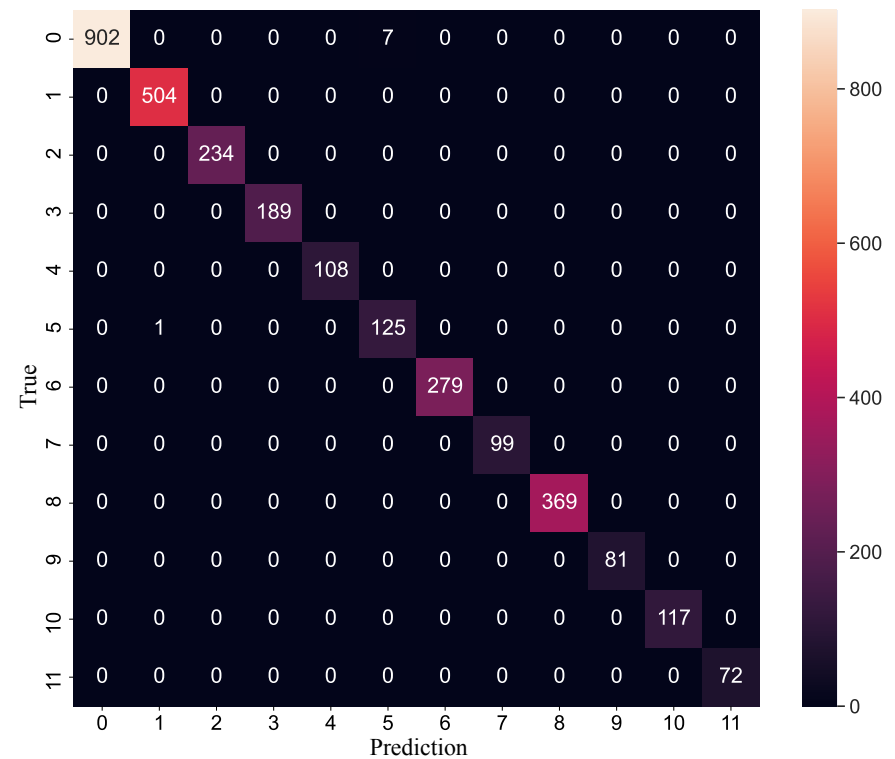


Figure E2: Confusion matrix on training data set

Appendix F App showcase

Appendix F.1 Web-based data collection app

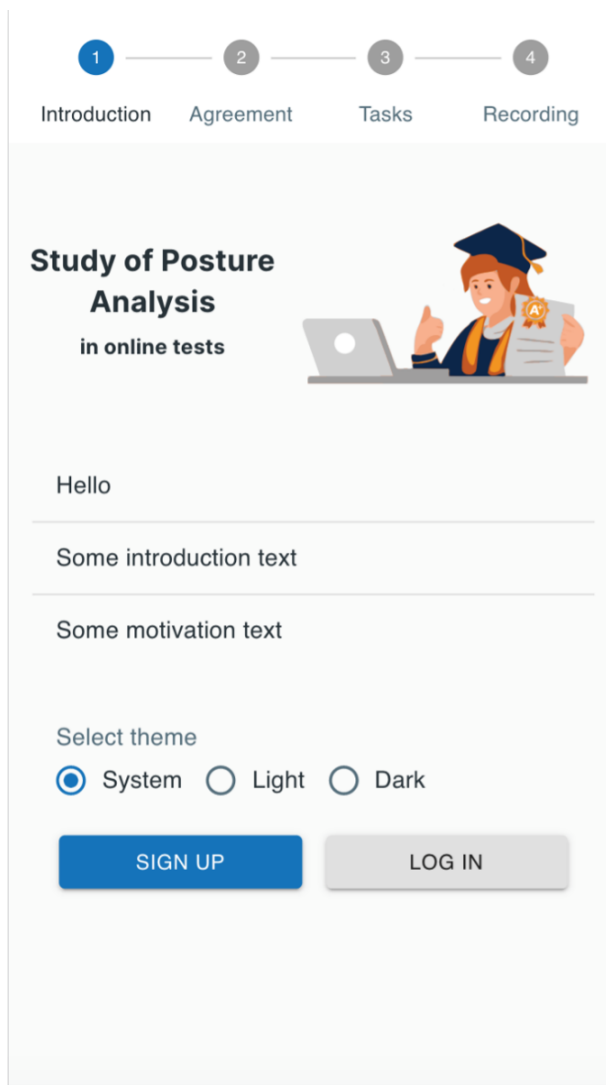


Figure F1: Welcome screen

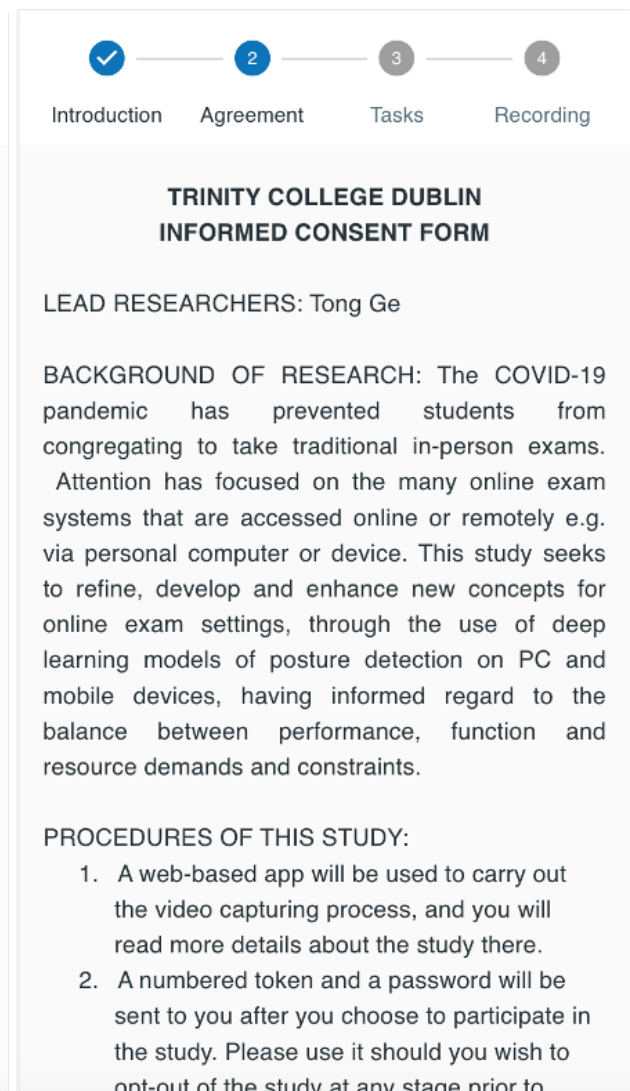


Figure F2: Consent form screen top

PARTICIPANT'S NAME:

PARTICIPANT'S SIGNATURE:

Date:


Statement of investigator's responsibility: I have explained the nature and purpose of this research study, the procedures to be undertaken and any risks that may be involved. I have offered to answer any questions and fully answered such questions. I believe that the participant understands my explanation and has freely given informed consent.


RESEARCHERS CONTACT DETAILS: geto@tcd.ie

RESEARCHER'S SIGNATURE:

Date:

- I am 18 years or older and am competent to provide consent.
- I have completely read and understood the informed consent form above.
- By signing this document, I consent to participate in this study.

NEXT 



Introduction Agreement Tasks Recording


Study Participation Informed Consent Form

First name *

Last name *

Email *

Captcha

I am human 

BACK NEXT

Figure F3: Consent form screen bottom

Figure F4: New participant register screen

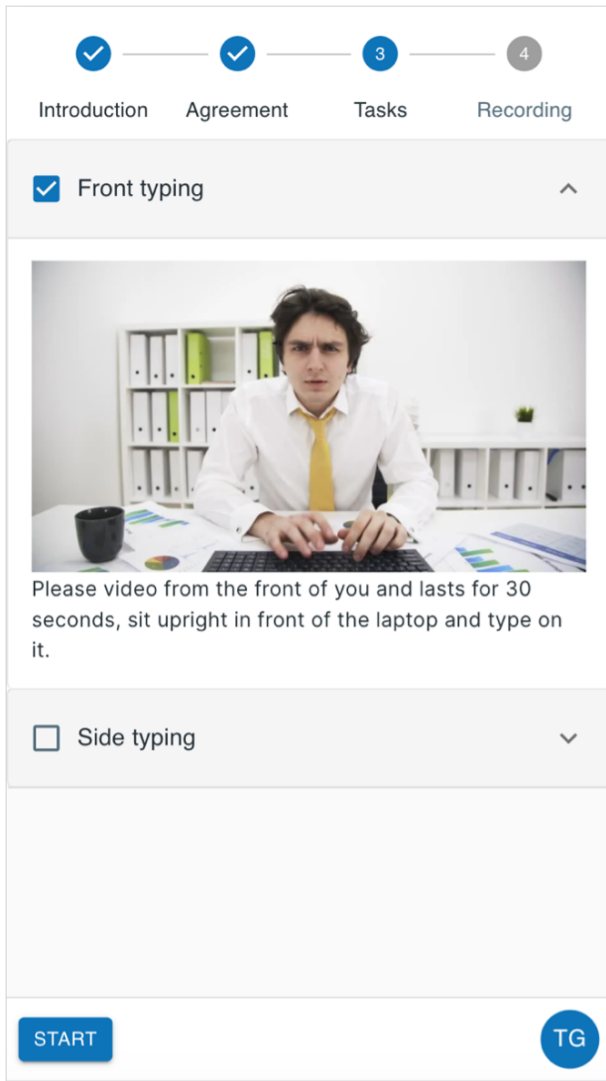


Figure F5: Task selection screen

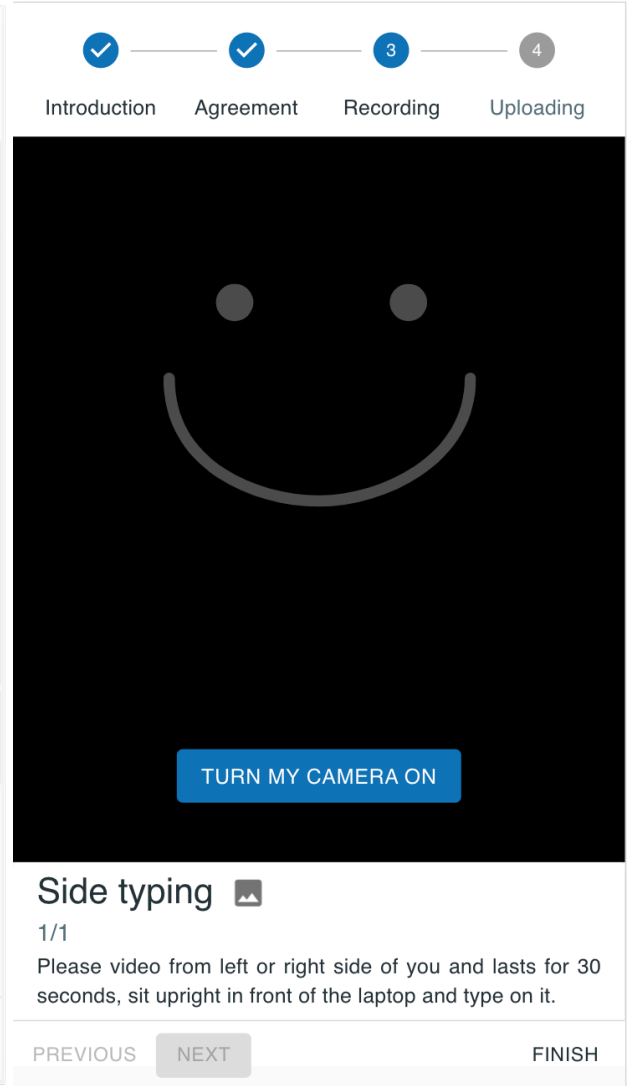


Figure F6: Recording screen

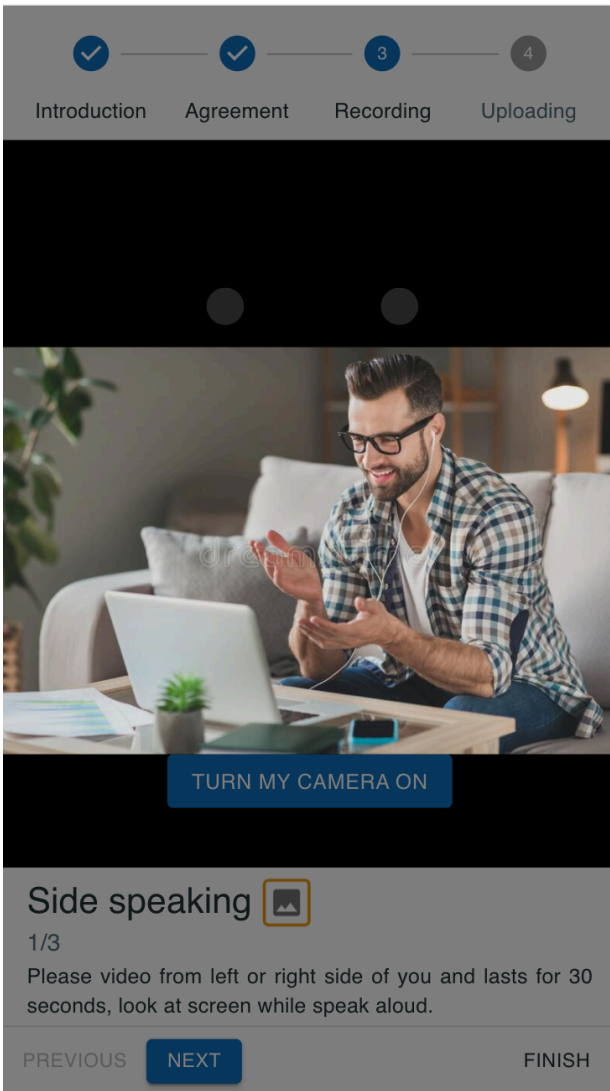


Figure F7: Review task details

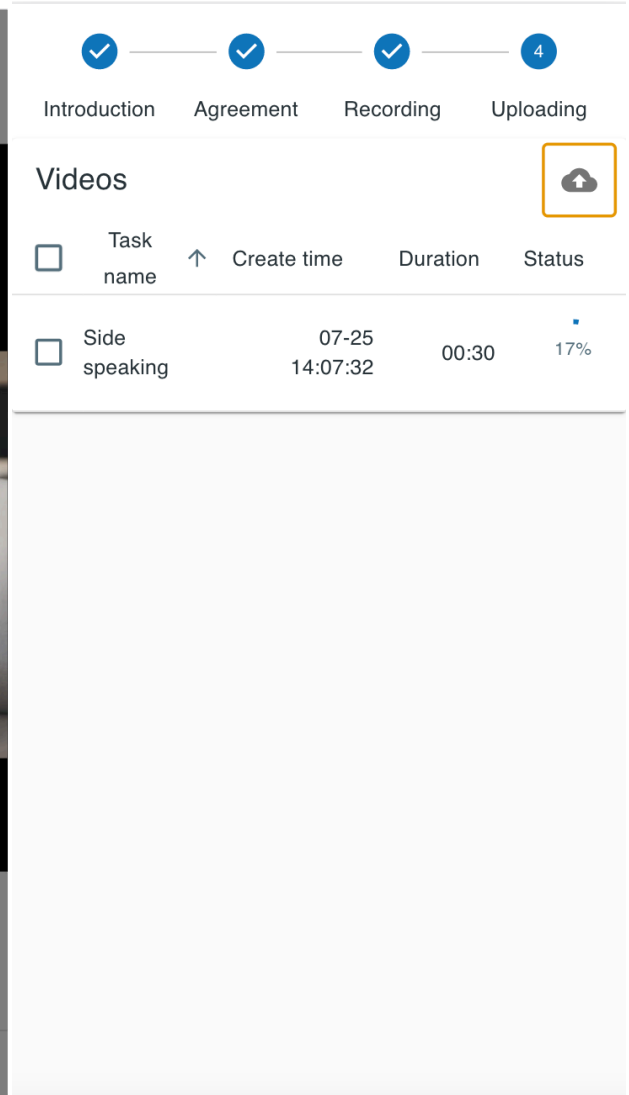


Figure F8: Upload screen

Appendix F.2 Deep model equipped mobile app

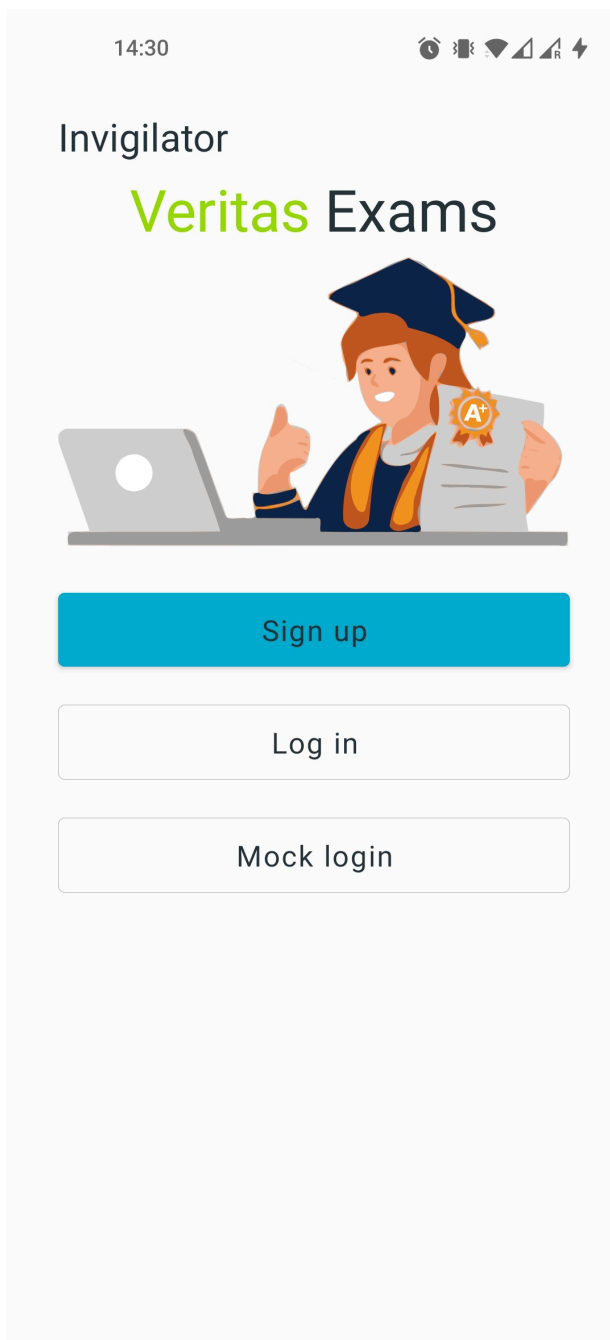


Figure F9: App welcome screen

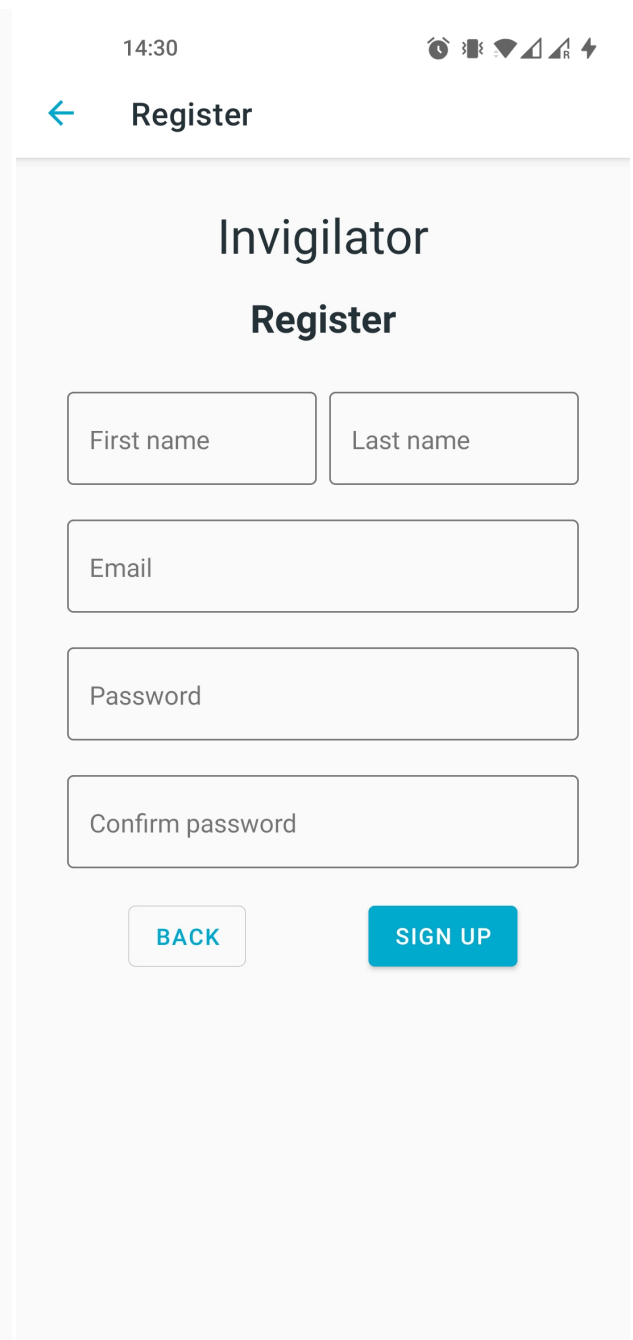


Figure F10: App register

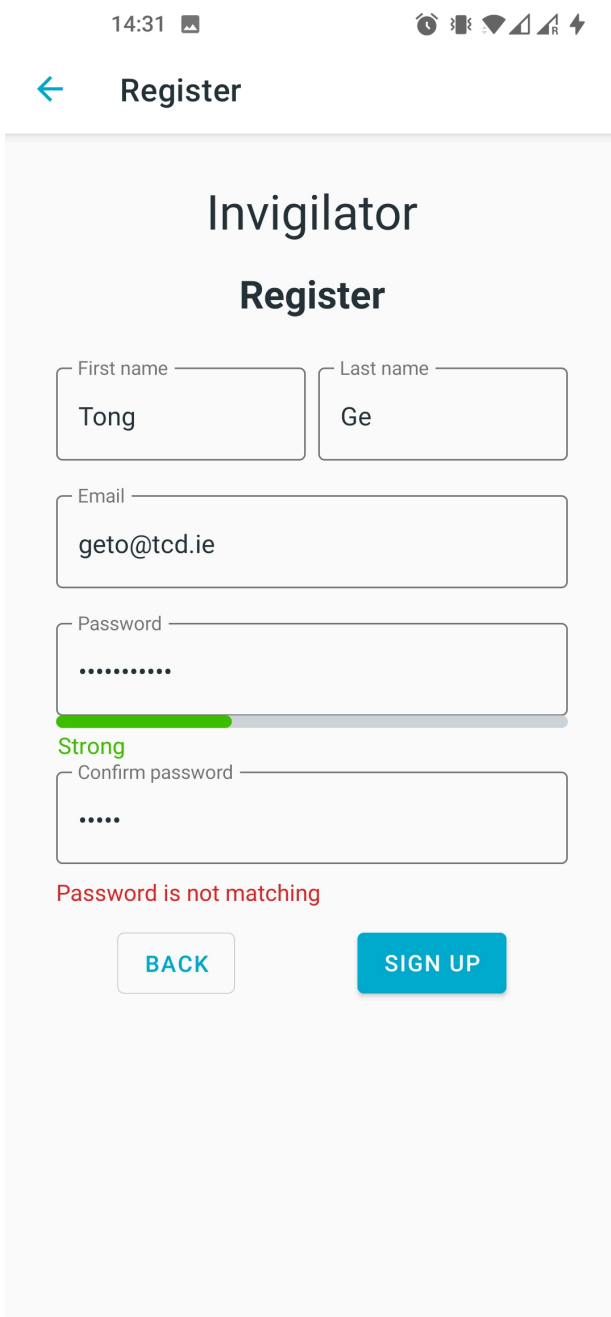


Figure F11: Filled app register

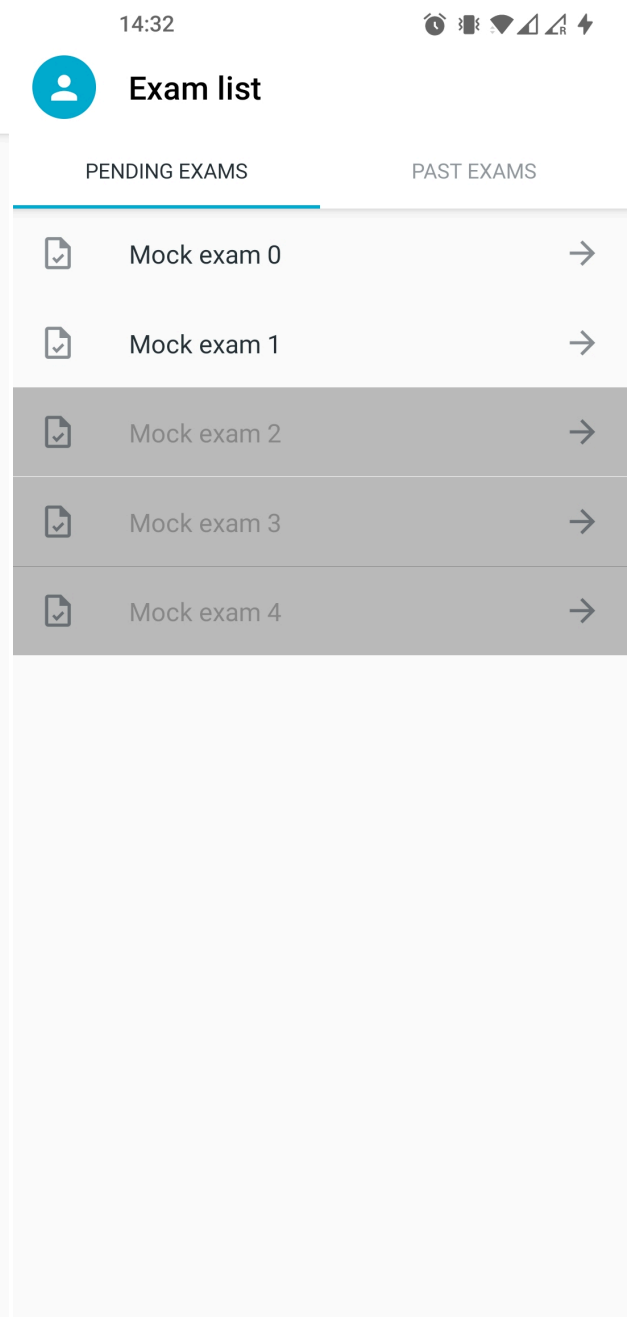


Figure F12: Mock exam list

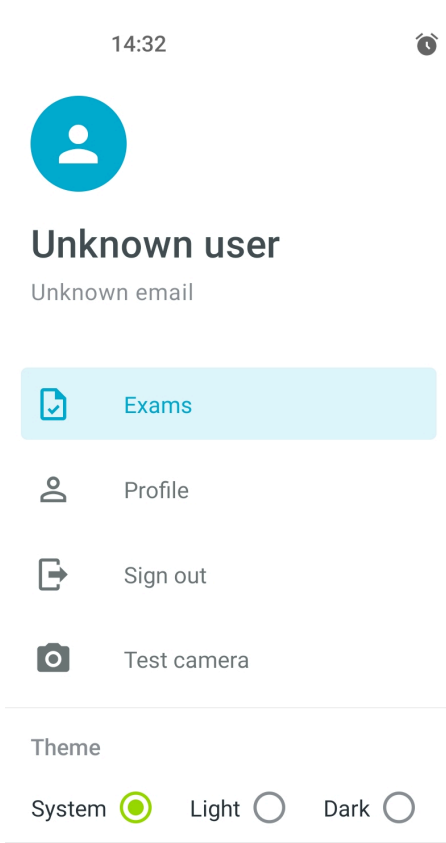


Figure F13: App drawer in mock user

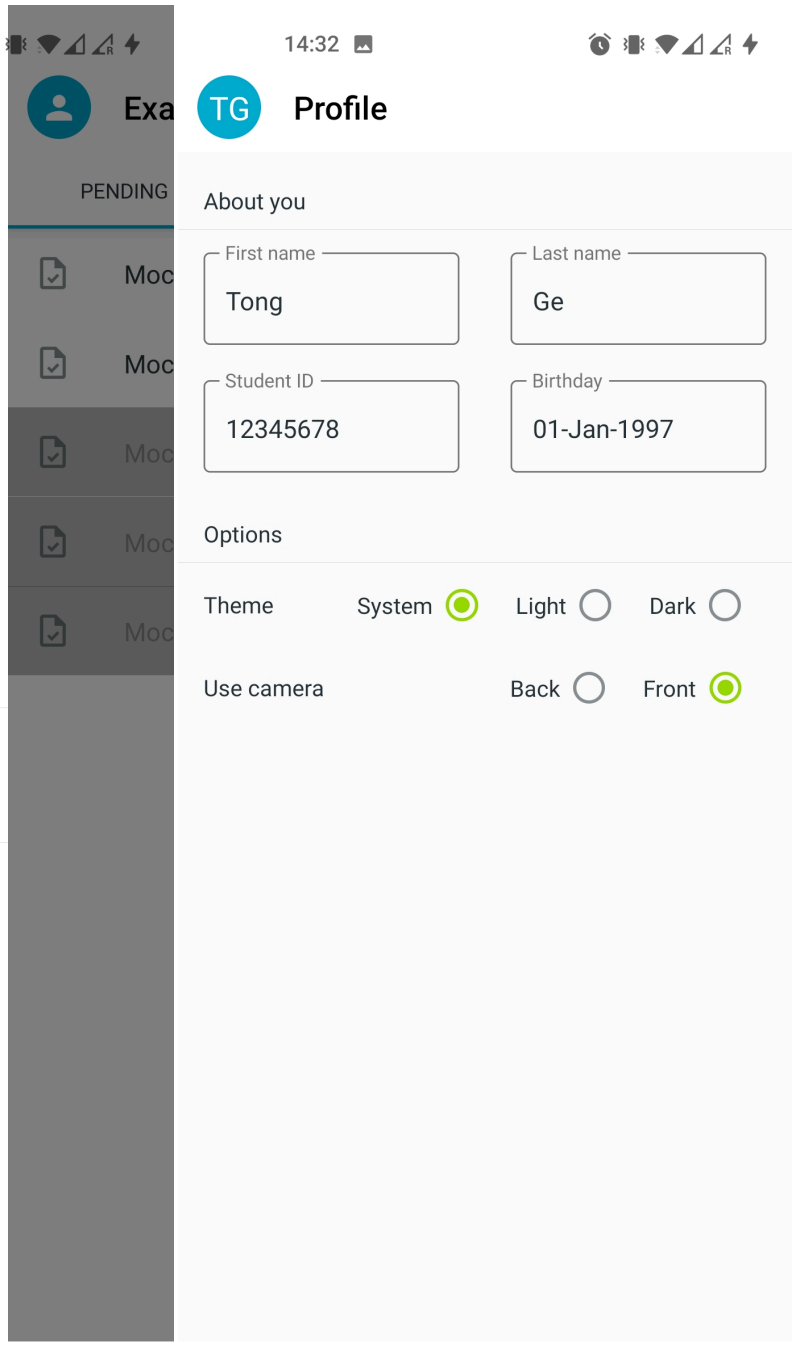


Figure F14: App user profile

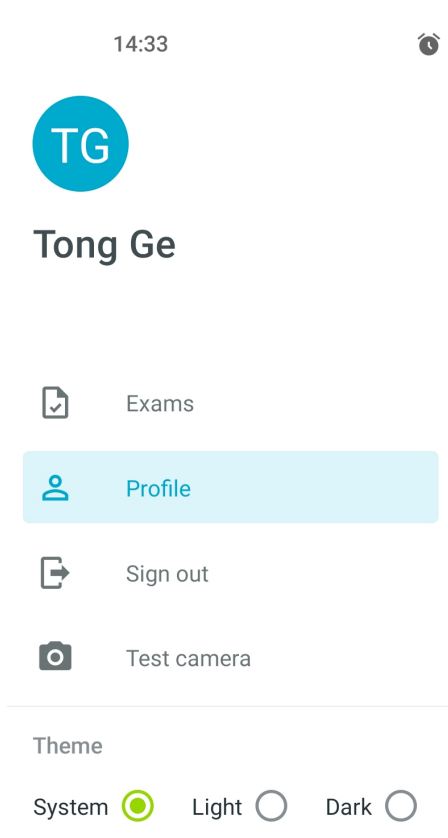


Figure F15: Drawer after profile update

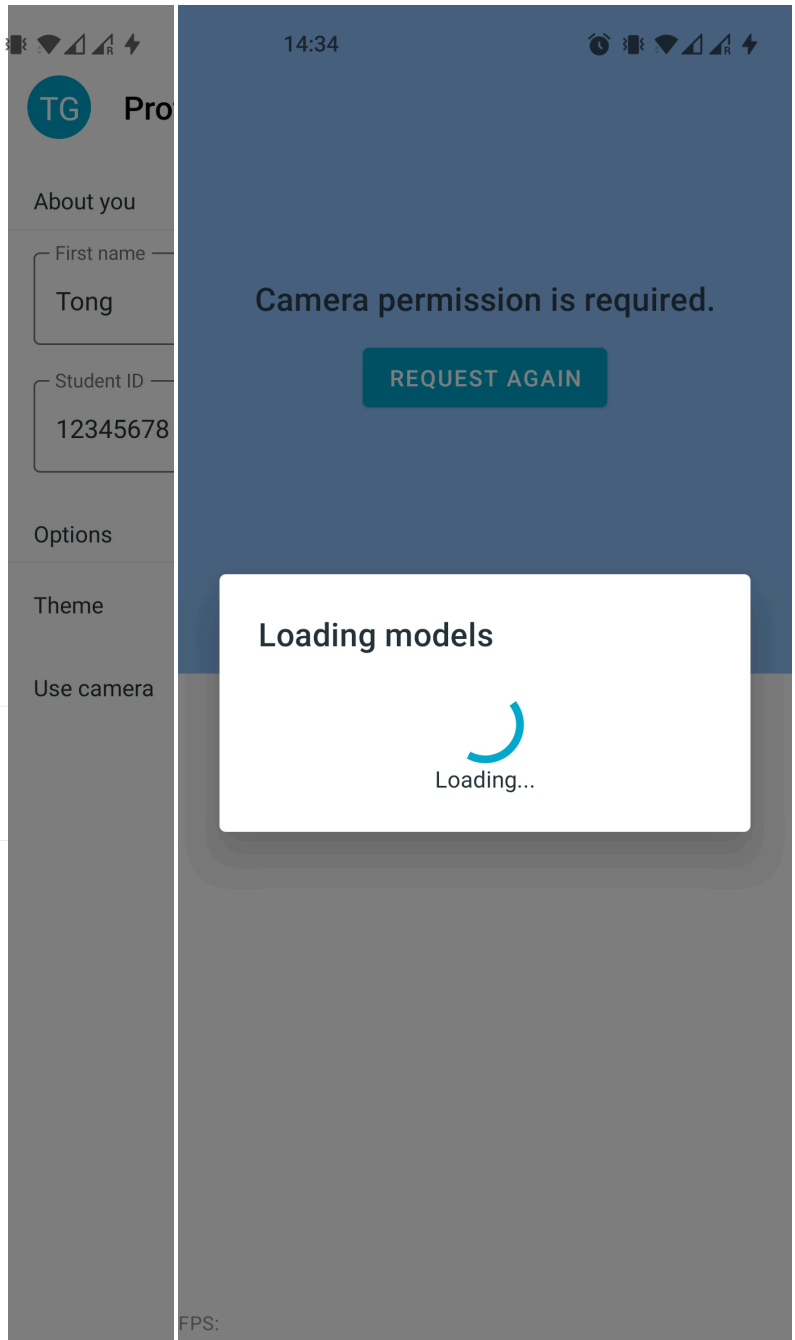


Figure F16: Deep model loading

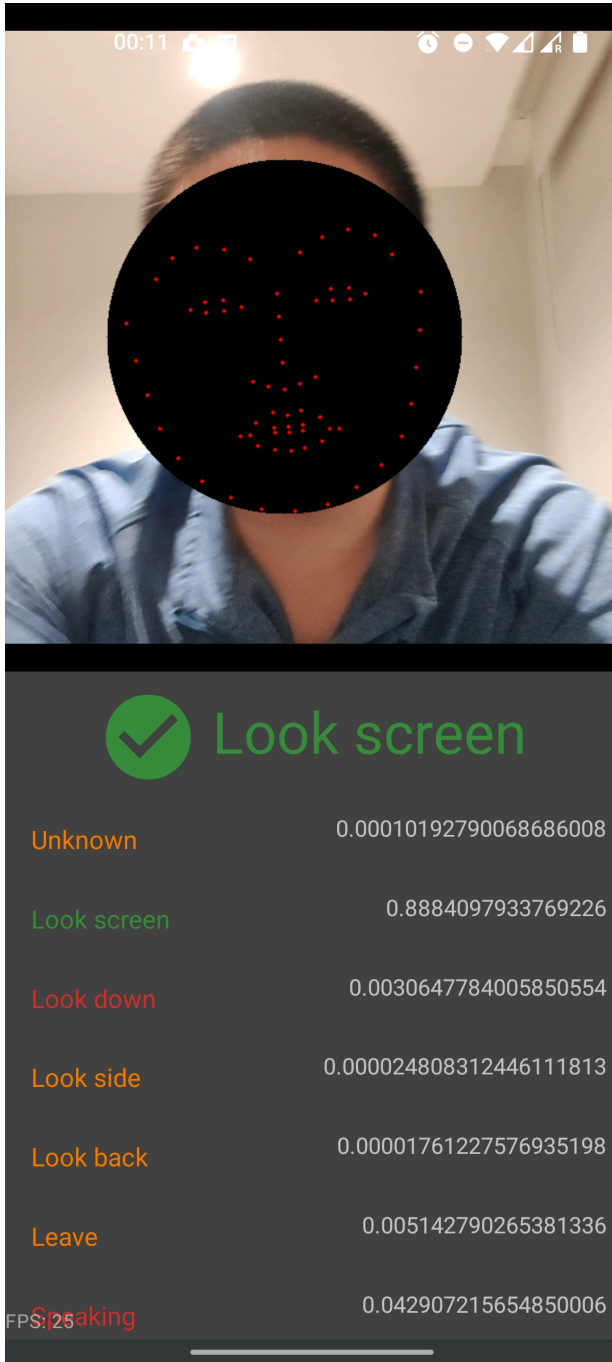


Figure F17: Result in dark theme

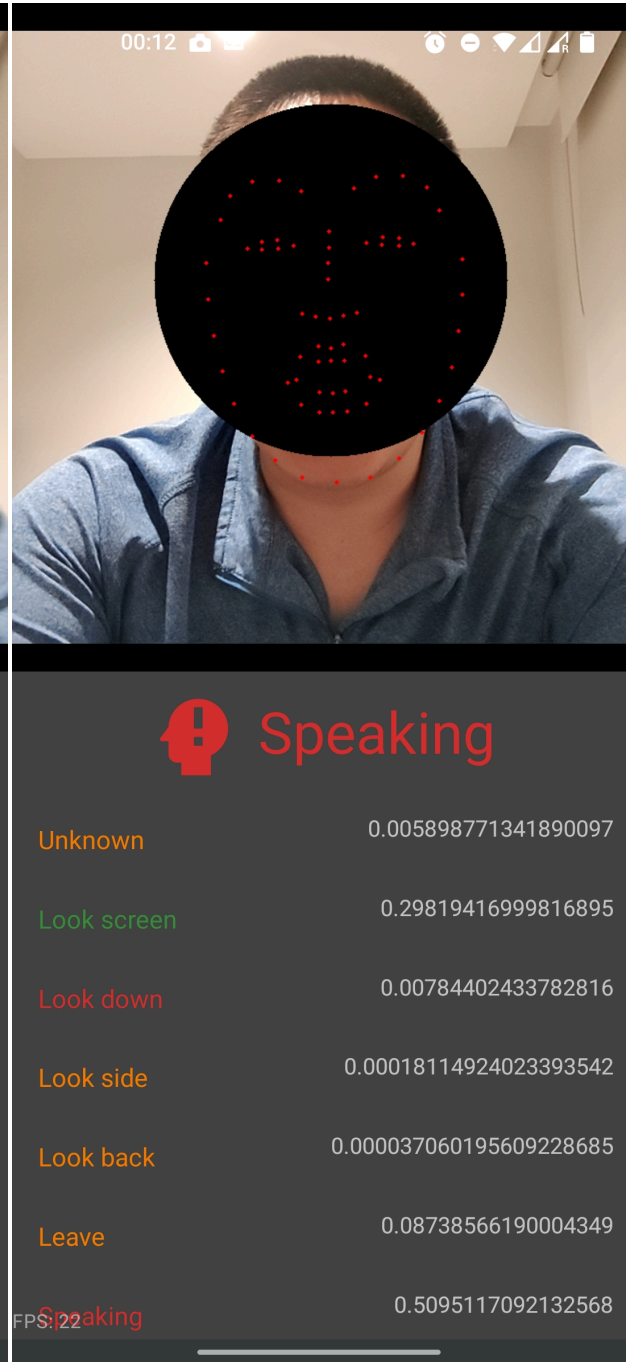
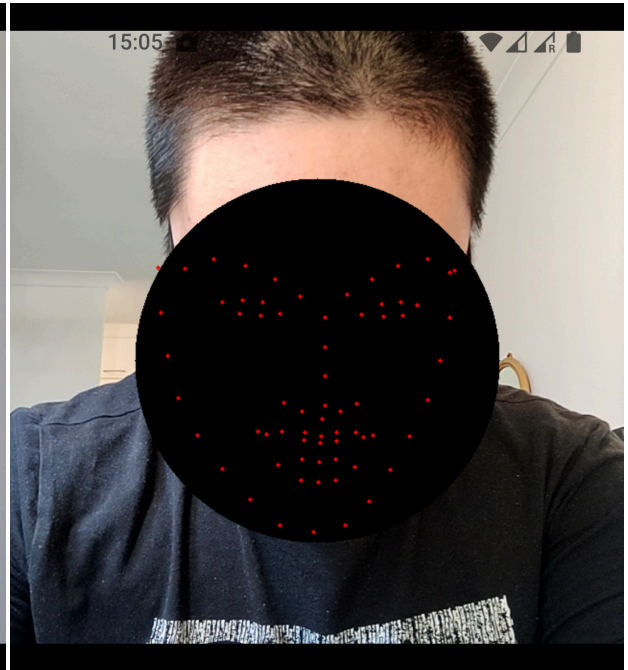
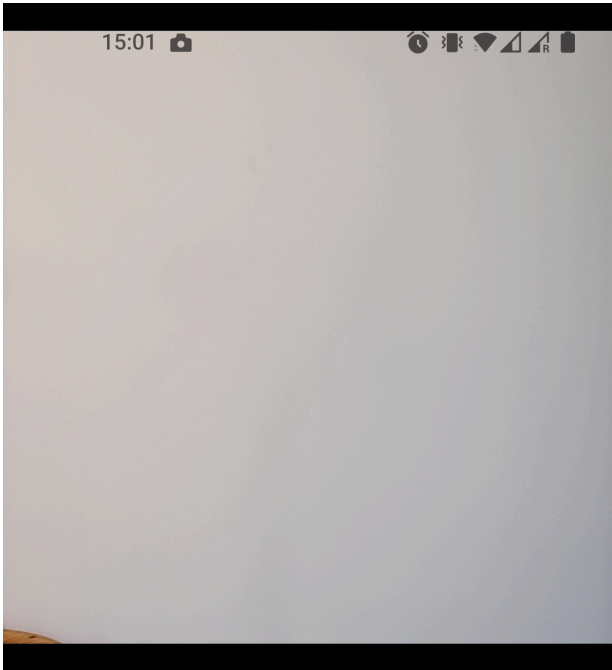


Figure F18: Result with mouth open



Leave

Unknown	0.028107956051826477
Look screen	0.0000024484158984705573
Look down	0.00019070375128649175
Look side	0.00002878105988202151
Look back	0.001038387417793274
Leave	0.9704127907752991
Speaking	0.00005186745329410769



Look down

Unknown	0.010727642104029655
Look screen	0.001036979490891099
Look down	0.9849985241889954
Look side	0.000018793145500239916
Look back	0.000028367736376821995
Leave	0.0006105027277953923
Speaking	0.000049703812692314386

Figure F19: Leave result

Figure F20: Look down result