A-EYE IN THE SKY: Examining Machine Learning-Based Keystroke Reconstruction from Passive Video Capture

Yong Nathan

ABSTRACT

The rapid growth of artificial intelligence (AI) has allowed for unprecedented progress across disciplines, but has also introduced new cybersecurity threats. Among these are machine-learning (ML) powered side-channel attacks. In this paper, we explore optical side-channel attacks, and the role ML plays in this attack vector. We investigate the feasibility of vision-based keystroke inference as an automated attack vector. Using a publicly available dataset of typing videos from Hugging Face, we frame the problem as a supervised classification task and systematically explore a minimal yet effective Neural Network (NN) pipeline. Our experiments compare five neural network architectures: EfficientNet_B0, ResNet-8, MobileNetV2_100, ConvNeXt_Tiny, along with a Feed-Forward multilayer perceptron (MLP). We also explore ten critical hyperparameters and discuss how variations affect performance. We train 26 models with varying hyperparameters, using Kaggle P100 GPUs. We evaluate models using F1 score, training loss, validation loss, and validation accuracy. Results indicate that a pretrained MobileNetV2-100 architecture, trained on min-max normalized, transformed and generously labeled data without time context, with class balanced training, a hidden layer, dropout regularization, and a time-based learning-rate schedule, achieves the best performance with a peak F1 score of 0.542. Our findings demonstrate that effective keystroke inference can be achieved with relatively modest computational resources and without sophisticated preprocessing or postprocessing. This suggests that such attacks are within reach of moderately skilled adversaries and highlights the need for countermeasures to such attack vectors. We discuss limitations of this work, chiefly dataset frame-label mismatch, and propose directions for future work such as blind keystroke recognition and inference on mobile devices.

CONTENTS PAGE

1 INTRODUCTION

Artificial Intelligence (AI) has progressed rapidly in recent years, surpassing prior expert predictions. AI has been shown to be a transformative force in many fields, with cutting-edge models being able to outperform humans in many tasks. Use of AI to solve complex problems has been instrumental for groundbreaking scientific developments in many expert fields. In biology, Google's AlphaFold and AlphaFold 2 have been revolutionary in predicting protein structure [1]. In chemistry, generative AI methods have helped discover millions of specific chemical structures that were beyond human chemical intuition [2]. In mathematics, recent advancements showcase AI's ability to come up with novel syntax, with models being able to advance the forefront research on advanced mathematical problems [3]. In Physics, physics-informed AI algorithms have been used to solve computational physics problems [4]. literature presents a mere fraction of AI contributions in cutting-edge science. Outside academia, commercially available AI tools such as GPT-5 have also been shown to be, depending on context, superior to the average human's performance in benchmark tests. Results from the Theory-of-Mind reasoning benchmark BigToM [5], and the ARC-AGI benchmark [6], demonstrate that AI's reasoning and comprehension capabilities closely align with human levels. Other benchmark tests such as GPQA [7] show that these tools already far surpass the average human in tasks involving solving complex, PhD-level problems in the domains of biology, physics, and chemistry.

With such advancements, there exist computer security concerns. The field of cybersecurity has rapidly evolved in importance since the internet's inception, specifically, the wide popularization and subsequent adoption of the Internet of Things (IoT) technology. It is defined as the field of protection of "cyberspace and cyberspace-enabled systems from occurrences that misalign de jure from de facto property rights" [8]. The importance of cybersecurity has been highlighted on many occasions, from the far-reaching WannaCry ransomware attack in 2017 [9], to the politically-charged and targeted attack by Russian hackers of the Ukrainian Power Grid in 2015 [10].

Computer security concerns arising from the fast advancement of AI are well-founded and justifiable. AI capabilities enable the scanning for vulnerabilities, development of malicious software, and the ideation of fresh attack vectors, all at a pace previously unattainable by humans. This indicates a dire future whereby zero-day vulnerabilities and sophisticated cyberattacks threaten systems globally. Research into the current and potential capabilities of AI systems indicate that they are more than competent in areas of cyber-attacks, from intelligent target profiling and vulnerabilities detection, to AI-driven self-learning malware. AI can take over multiple stages of the "cybersecurity kill chain" [11]. While current AI threats may be more theoretical than applied, more needs to be done to unearth potential attack vectors.

We posit that this field requires immediate attention because of a series of concerning factors:

1. AI growth is challenging to model, with studies showing that predicted timelines independent of expert competence do not correlate with actual growth [12], [13]. Empirical data consistently demonstrate that AI capabilities are advancing at a rapid

pace. Given the unpredictability of future development trajectories, it is prudent to consider scenarios in which this growth persists. Thus, it is paramount that we investigate the use of AI when it comes to cyberattacks, even if such attacks are presently not being utilized on a large scale. This ensures that appropriate guardrails and countermeasures are in place should AI become a significant threat in cyber space.

2. The issue of AI alignment is an important consideration. Alignment is the technical problem of imbuing AI systems, often black boxes to external observers, with a set of strict ideals and ethical principles that lead to a set goal. Research shows that as virtuous alignment improves, this increases the ease with which malicious models can be misaligned—through model, input, or output tinkering [14]. This presents a catch-22, whereby as AI models and their alignments improve, the easier it is for malicious actors to coerce and utilize such models for more sophisticated cyber-attacks.
3. Lastly, the sheer velocity of AI development leads to a widening in the gap of competencies between cyber-attacks and cybersecurity. This gap reflects a well-documented asymmetry in cybersecurity, where defensive systems are inherently reactive and consistently lag behind the rapidly evolving threat landscape. This gap is steadily widening in the cybersecurity AI arms race, as AI has supercharged the advancement of attack vectors, allowing malicious actors to overcome defenses and find undiscovered techniques, bolstering their ability to stay ahead of defensive developments. A comprehensive study in this area highlights the need for “continuous innovation and collaboration in the cybersecurity field” for AI-powered defenses to keep pace with such attacks [15].

In this paper, we shall explore the use of AI, more specifically, vision-based machine learning, in optical side-channel attacks involving keystroke inference. Machine learning (ML) techniques used involve artificial neural networks (NNs), defined as “nonlinear statistical data models that replicate the role of biological NNs” [16]. Vision-based ML refers to computer vision NN models, which specialize in processing visual data in the form of an image tensor, extracting information from the image, and generating desired output [17]. We treated the problem statement as a supervised classification ML problem, which involves training a model on labeled data, to ideally create a generalized model that can successfully predict discrete categorical outputs for external data. Side-channel attacks are attacks that exploit information leakage indirectly by a system during its operation, rather than breaking the system’s intended security mechanisms directly [18]. Examples include inferring sensitive information like keystrokes through acoustic emanations [19], through wireless network connections [20], or through power supply fluctuations [21].

The focus of study was chosen as it comprises another frontier of cybersecurity that must be sufficiently researched for appropriate defenses to be created to counter future threats. While the traditional space of cybersecurity will be faced with novel and increasingly sophisticated attacks, these attacks are limited to within cyberspace. However, side-channel attacks reveal another dimension of attacks that can be carried out—those outside of cyberspace. Information extraction from physical leakages of the system is difficult to discover and counter effectively. Furthermore, side-channel attacks are uniquely positioned to be empowered by increasingly sophisticated ML algorithms [22]. This is due to these systems' unparalleled ability to analyze, comprehend, and interpret data that may seem to a human like random and incoherent. ML is able to achieve intelligent human imitation, with NNs able to model the flexibility and ingenuity of human pattern recognition [23], but with superior processing speeds, higher information storage, and more efficient information transmission [24]. AI can sift through noise and find patterns at a rate previously not computationally possible, leading to novel side-channel attacks that are more effective in capturing sensitive data.

Thus, AI poses a large threat to the field of side-channel attacks, and research into the field is needed.

This paper focuses on one type of side-channel attack: optical keystroke inference attacks. We examine the use of ML for automated visual hacking, also colloquially known as “shoulder surfing” or “visual eavesdropping”, where passive video collection of keystrokes is used for data extraction. This is notably different from typical side-channel attacks, which tend to be more covert in nature, such as the aforementioned acoustic emanations or power fluctuation analysis. However, as the system itself does not need to be breached to engage in such an attack, it still falls under the classification of a side-channel attack. The rationale behind this choice of attack was twofold.

First, computer vision is an area that is rapidly developing and could grow to pose significant dangers. The problem of computer vision has intrigued computer scientists for years, as it entails giving algorithms the subjective ability of vision. However, in recent years there has been great success in this area, with the shift towards Convolutional Neural Networks (CNNs) [25] that are better able to extract features from images. However, with ML that is more accurately able to comprehend images comes a large array of potential cases of misuse. Issues of surveillance and privacy is one [26], and in this paper, we explore another alternative malicious use case, that being computer vision analysis of keystrokes. With computer vision, attack vectors that would have had to be manually conducted, such as human keystroke inference, can now be automated to be undertaken at far higher speeds and accuracies. What would have been an unfeasible attack vector due to limited scalability and human error, can now be a viable strategy for malicious actors.

Second, keystroke inference from passive video capture presents a real-world problem. Visual inference of keystrokes is an attack vector that can be used by malicious actors in public areas. This issue is exacerbated by growing trends of sensitive operations being conducted in public, on personal devices such as laptops and mobile phones. Such behaviors, like online banking, are now often conducted where passive video data can be easily recovered. For example, the prevalence of security cameras could potentially provide footage for such attacks.

Since both the method of attack is developing in sophistication and the difficulty of attack is increasing in ease, we thus chose to investigate this problem.

1.1 Related Work

Conducting a literature review, we examined 3 papers that explored a similar threat space. These papers were chosen to provide a range of different methodologies, conclusions and focuses.

1.1.1 Leveraging Disentangled Representations to Improve Vision-Based Keystroke Inference Attacks Under Low Data Constraints (Lim et al., 2022) [27]

This paper explores video domain adaptation by introducing synthetic data into training. The purpose is to showcase a realistic use case of keystroke inference that does not require an unrealistic amount of input data. The study was able to achieve 95% accuracy in character detection with only 100 videos of 300 frames each. This indicates an increased practicality of such keystroke inference attacks.

In terms of methodology, they utilized 10% of the training set as a validation set. Labels were limited to 26 letters and 4 additional tokens. They scored the model on character detection accuracy, and on postprocessing of their output on language models. They utilized Bleu-n, ROUGE, and METEOR, which are metrics for word coherence and precision.

Their conclusion was that deep learning methods could be used by adversaries to achieve accurate models with limited data. The authors also argued that, holding all other hyperparameters constant, an attacker using deep learning methods would outperform one with a shallow method. This is an observation in other fields of computer vision problems.

1.1.2 ClearShot: Eavesdropping on Keyboard Input from Video (Balzarotti et al., 2008) [28]

This paper presents a, at the time, novel approach to keystroke inference from video, utilizing motion tracking, sentence reconstruction, and error correction. The researchers constructed a tool, called ClearShot, that can extract a substantial portion of typed data from video.

Their methodology involved a frame-by-frame analysis, whereby each frame of data was the input for the NN, with output being a solution space of possible keystrokes. This was then input for text analysis, which, utilizing context and language-sensitive error correction, produced an array of possible words for each word in a reconstructed sentence.

Researchers attempted optical flow analysis, using temporal context via previous frames to make an informed analysis of a certain frame. However, due to the proposed reasons of noise and complexity in typing patterns, this proved unsuccessful. Researchers also disregarded using skin tone analysis as it was seen as inefficient and sensitive to the whole hand placement, rather than to only movement in the fingertips of video subjects.

They concluded that this was a viable vector of attack with a reasonable recovery rate near that of other side-channel attacks of the time, and comparable to humans tasked with the same visual keystroke inference.

1.1.3 Towards a General Video-based Keystroke Inference Attack (Yang et al., 2023) [29]

This paper aims to increase the feasibility of keystroke inference attacks by training models that function in realistic attack scenarios. Researchers created a pipeline for an attack that assumes no pretraining with training data, no knowledge of keyboard layout nor typing style, and no external sensors. Using purely video data collected in noisy environments with a commercially available camera, the study was able to create an accurate keystroke inference model.

A two-layer self-supervised system was used, where initial hand tracking is used to run keystroke detection/clustering for the video, which is processed by a language-based model to recognize keystrokes based on occurrence. This provides the labels for CNN model training, which took single-frame data as input along with a label assigned from the first layer. This self-supervised model was able to predict characters with a 92-97% accuracy, depending on the character, with only about 3000 keystrokes, equivalent to 15 minutes of typing. The model was able to adapt to different environments, keyboards, and users with different speeds and styles of typing.

This paper was able to show that a general, vision-based keystroke inference attack could be a feasible attack vector despite short footage and a lack of pretraining.

1.2 Our Contribution

This paper intends to explore a similar problem space to the three studies mentioned above. As a synthesis paper, we plan to examine the process of creating a NN to solve this problem, sharing the considerations that factored into the final model. We will also explore the permutations of different parameters and how they affected performance. For example, we will compare various easily accessible vision models, conducting a parallel study to compare how each model performs.

The pipeline we intend to use is one simpler in nature, with just a CNN without significant preprocessing nor postprocessing. The goal of this project is to **synthesize the fundamentals of keystroke inference models, and construct a baseline model with a reduction in complexity, comprising only the most critical components**. This was partially a practical decision due to a lack of expertise in the area, along with the desire to build our pipeline from the ground up in order to document the entire process. From a significance perspective, this can provide the scientific community with a grasp on what an amateur can achieve using publicly available resources, a case study that could be more indicative of what the majority layman population of malicious actors are capable of.

1.3 Hypothesis

Our hypothesis is that a suitably generalized model can be trained from a limited dataset, using an unsophisticated model. We define suitably generalized as a model that can perform well with fresh test data, with the metric for this performance being the F1 score. A limited dataset is defined as a video dataset that has a total length of less than 2 hours. This number was chosen as it is rather feasible for malicious actors to obtain footage of such length, for example through security cameras, or via manual filming in a public space such as a work cafe. Lastly an unsophisticated model refers to one that comprises just the NN, without significant preprocessing nor a complex pipeline as seen in all 3 of the studies we reviewed in Section 1.2.

1.4 Structure of this paper

This paper will be structured as follows. In the above few sections is the introduction, where terms are defined and background information provided. A literature review was also conducted to lay the groundwork for our further research.

The following section is Methodology, where we will discuss the experiments conducted and the different considerations that went into hyperparameter values. We will first go over our dataset and its preparation. Next, we will discuss the network itself, providing a comprehensive explanation as to certain developmental decisions made, with the explicit goal of providing a well-reasoned, coherent, and easily replicable experiment. We will also explore some of the additional components we may have chosen to exclude, and the rationale behind these choices. Then, evaluation criteria will be outlined including our driving motivation behind them.

Next will be the results and analysis section, where we will share the data collected from these experiments in graphical form. This includes results across different variations of the model, organized into 10 different experiments. We will analyze each experiment to draw conclusions about varying hyperparameters.

Lastly, in the Conclusion section, we will discuss the implications of our project, and address whether our hypothesis is to be accepted. We will also explore certain study-wide limitations along with avenues for future research.

2 METHODOLOGY

An overview of our methodology is as follows. We utilized a standard workflow, with our model framework built on PyTorch, a Python library regularly employed by ML researchers. PyTorch, developed by Facebook AI Research lab (FAIR) in 2016, provides significant capabilities such as Graphics Processing Unit (GPU) acceleration through Compute Unified Device Architecture (CUDA), along with easy integration with existing Python modules, granting it flexibility and lasting forward compatibility. It rapidly became a reliable tool for researchers, with the majority of ML research using PyTorch implementations today, as seen from a cursory exploration of papers on arXiv and in top conferences.

The following sections go into the details regarding the considerations and decisions made in our data preparation, model training, and model evaluation phases. Our code base can be found attached in a GitHub Repository at the end of this paper.

2.1 Standard Methodology

To provide background and terms for Sections 2.2-2.8, we will give a brief overview on standard methodology within the space, from our literature review and expert guidance.

Studies begin with the finding or collection of data. For larger-scale operations, this can involve collecting data from experiments directly. However, this is resource-intensive due to the large amount of data required, and is thus often undertaken only by universities, labs, and for-profit enterprises. For individual researchers, this is often less than feasible, and as such many empirical studies in ML mainly rely on the data available on code-hosting platforms like Hugging Face. These provide large, publicly available datasets that can be used for training.

Data preparation is the next step, often taking up most of the actual experimental time. One study suggests such preparation work comprises 80-90% of the project itself [30]. One element of preparation is data labelling. Data must be accurately labelled to ensure the model is being trained on valid information [31]. Otherwise, the common axiom of “Garbage In Garbage Out” holds true, whereby the quality of output from a NN is only as valid as the data that is fed in. Data preparation also involves gauging possible issues with the data and solving it preemptively. An element of this is data normalization, where data is sanitized and standardized. This is to ensure that the scale of features present the dataset is in the same range, such that no one feature contributes disproportionately to the model’s output.

Once the data is prepared, we move on to the loading and training phases.

Loading data involves batching and shuffling. Batching is the grouping of samples together for the model to process concurrently, which improves efficiency and speed of training, while stabilizing gradient estimates. For this step, the batch size is a variable that can and should be adjusted through iterative development—smaller batches are better for memory and generalization, but face longer training times and noisier gradients, and vice versa for larger batches. Meanwhile, shuffling is often automated in Python PyTorch, and involves randomization of samples for training such that models do not overfit to the sequence of data.

Training phase involves a NN that should be designed with the goal in mind. NNs can take many forms, and for this paper we went with both a CNN and a standard feed-forward NN to compare the two. Both architectures employed different commercially available models, and different hyperparameters such as hidden layers and dropout rate. Most of our NNs used a similar input and output structure, that being an input of one frame and an output of a label between 38 classes, comprising 26 letters of the alphabet, 10 number characters, space, and a blank character denoting a frame where no keystrokes are occurring. Lastly, some training hyperparameters such as epoch count, learning rate, and optimizers are chosen.

When training is run, cycles iteratively tweak each parameter in the NN through training loss and back propagation. The validation dataset is used to generate validation loss. This is a good metric for the quality of training, since it is data the model is not trained on. For example, validation loss increasing while training loss decreases is a standard signal of overfitting.

Lastly, the trained model is evaluated, and the whole cycle is repeated with varying parameters to conduct a review of different components of the pipeline.

2.2 Dataset

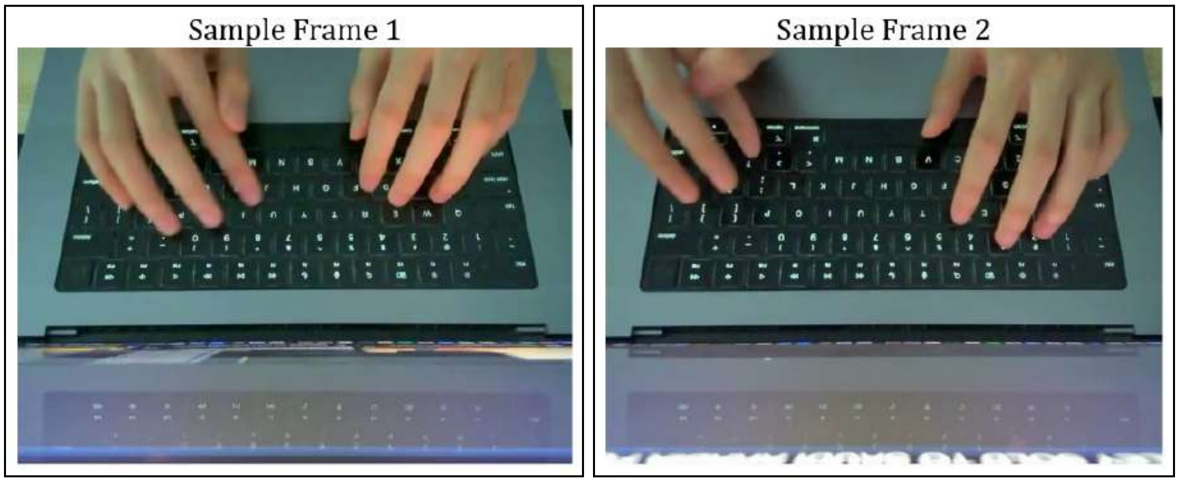
The dataset used in this study was sourced from Hugging Face. This was the “Keystroke-Typing-Videos” dataset published in 2025, by user Andrew Tran (https://huggingface.co/datasets/andrewt28/keystroke-typing-videos). The dataset comprised 800 videos with a mean video length of about 6.73 seconds, and an average of about 29.5 frames per second, giving a database of 158854 frames and 40862 frames with keystrokes. The data was also labelled with a keystrokes array, indicating the key pressed and millisecond timestamp data, along with the full sentence typed.

We separated the dataset into 620 videos for training, and 120 videos for validation and testing.

2.2.1 Why it was chosen

The dataset was chosen as it was the most appropriate dataset publicly available. The video data had a consistent user typing, a consistent keyboard, and a consistent camera angle. Lighting was also largely consistent. Keeping these other factors controlled was imperative to ensure the prospective NN trained on such a dataset would only generalize based on hand placement, and not on other features. The author was also meticulous in labelling the timestamped information of keypresses, thus ensuring that frame-by-frame analysis was possible, as we could infer the frame labels.

Example images of the dataset are shown below in Figures 1a and 1b.



Figures 1a, 2b, respectively: These sample frames of the dataset are two different frames from different videos, and illustrate how lighting, keyboard position, and user are all kept constant throughout the videos.

2.2.2 Frame-Based analysis

The choice to conduct analysis on a frame-by-frame basis was one that we decided upon after extensive research and consideration. Initially, the idea of video input was intriguing to us as it seemed logical from a task goal perspective: The initial input being a video would allow the NN to recognize features such as hand movements, giving the NN temporal context to determine the words being typed. However, when breaking down the problem we found issues with video-based input.

Video-based feature extraction is a possible route for ML, with studies having been done in the area. Action recognition is a common field, with the UCF50 [32] being one of the most famous video datasets comprising over 6000 samples. Models such as MC3_18 are also crafted for this purpose, being able to take in a large amount of input data, that being a 4-dimensional tensor with height, width, channels, and a 4th temporal dimension. These models comprise a 3-dimensional CNN.

However, we chose not to due to the following reasons.

First, resource constraints. The heavy compute power required for video-based analysis, due to sheer size of the data and the required network parameters, was difficult for us to obtain. Studies show that video-based 3D CNNs can take 27 times the amount of compute compared to 2D CNNs doing frame-by-frame analysis [33], and thus 3D CNNs are inefficient a large portion of the time. Since this amount of computing resource was not available to us, using video-based analysis would lead to unfeasibly long training times and thus hinder our ability to conduct iterative development and experiments into the tweaking of different constraints.

Second, the importance of temporal context was limited and could also lead to more issues. Pre-frame context could explain why hand placement was different for different keys, for instance, as seen in Figure 2 where the same letter “b” is typed, but hand placement is different due to the previous keystroke. However, it could lead to overfitting in the cases where the NN learned patterns for certain words, for instance that “u” was often pressed after “q”. Without a way to reliably ensure the NN would be able to distinguish characters, it would leave the NN competent in sentence analysis while less competent in character analysis. Since the goal of this experiment would be to simulate an attack vector, the data to be recovered is unlikely to follow English syntax. Instead it may be a random string of ASCII as seen in many people’s passwords. As such, each key press should be independent of other keystrokes, something that would be difficult for video-based analysis to achieve.

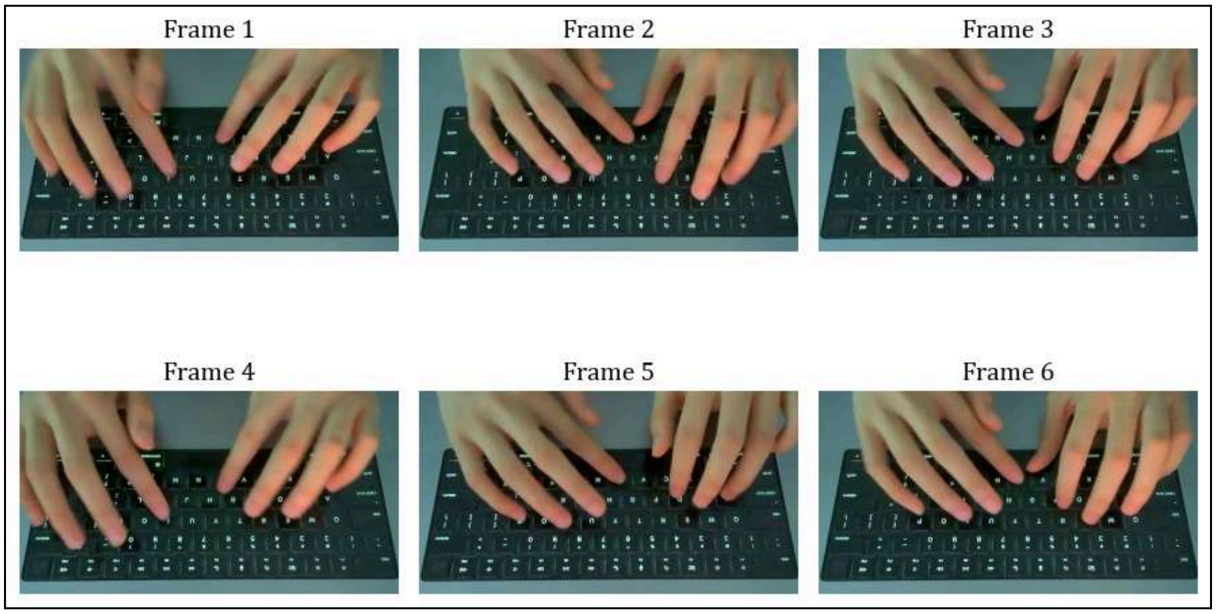


Figure 2: This shows 6 frames from different samples whereby the same letter ‘b’ is typed. This shows the importance of temporal context as due to hand placement before the frame in question, some frames show the left hand used to type ‘b’, while other frames show the right hand being used.

Our literature review further validated this stance, with most studies we found conducting similar experiments also choosing to utilize each singular frame as input.

It is to be noted that we did attempt at a hybrid method, that being frame-based analysis, where a set of three sequential frames would be the input, arranged in chronological order. This was conducted by splitting the dataset into overlapping sets of three frames, with the center frame being where the keystroke occurred. The theoretical idea was to include a brief amount of temporal context, which was hypothesized to help the model better generalize out keystrokes as it could extract information such as the previous and future positions of the fingers to gauge travel. Results from this method can be found in Section 3.1.3.

2.2.3 Limitations of the dataset

The dataset was far from evenly distributed in terms of keystroke types. This is plotted out below in Figure 3. The spread between the most common key and least common key is also rather large at 89210, between 'nothing'=89210 and 'question'=0, suggesting the dataset is far from evenly weighted.

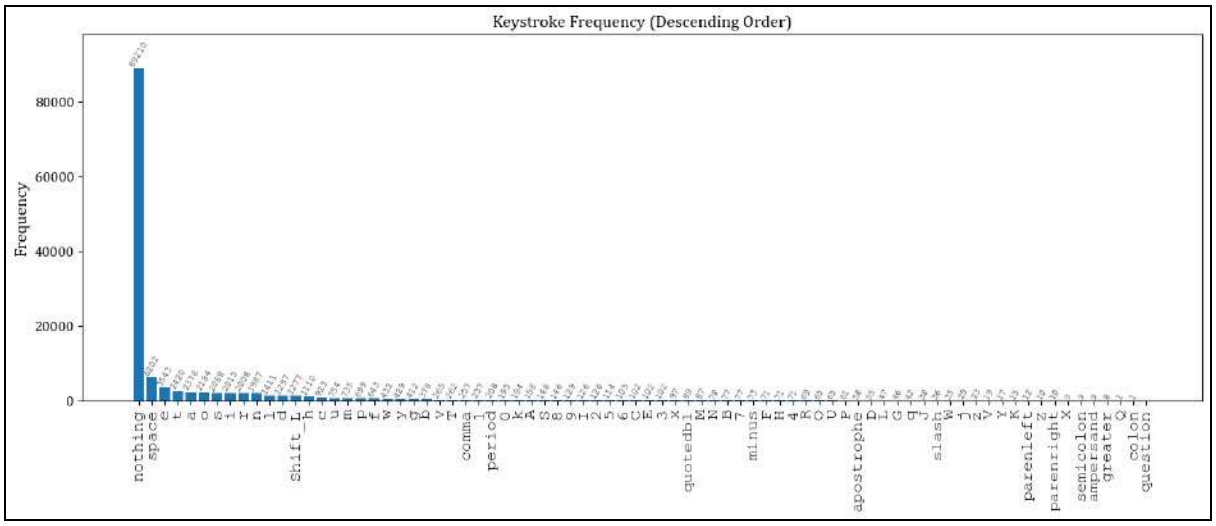


Figure 3: Shows the number of occurrences for each keystroke label, sorted in descending order. The distribution is imbalanced, with "nothing" labels and common keys such as space dominating the dataset, while many other keys appear only rarely or not at all

Furthermore, there is severely low representation of certain keys, such as 'question', 'colon', or 'Q', which have 0, 1 and 1 occurrences respectively. This is very concerning, as with such low numbers of samples, the model will be unlikely to be able to generalize features for these keys.

Next, the videos were far larger in terms of storage usage than necessary. Much of each frame was redundant pixel data that did not contribute to the features needed to discern keystrokes. As seen in figure 4, a significant portion of the image contains information outside the keyboard, where little to nothing changes from frame to frame. Including such redundant information will increase training times, and may even decrease accuracy as it draws model attention away from the focus area, that being the keyboard.

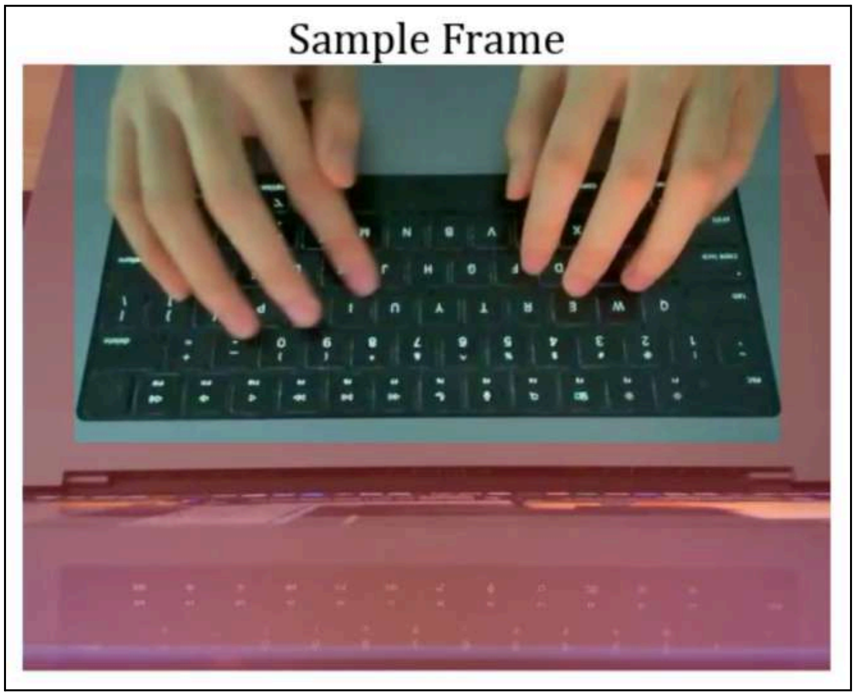


Figure 4: Shows a sample frame. The unnecessary pixel data is highlighted in red. This information is not important for the keystroke inference and just contributes additional noise.

However, by far the most pressing issue was the presence of seemingly random delays in the videos. This delay was present between the timestamp indicated on the keystroke event labels, and the actual frame at which the keystroke occurred, as judged visually by us. Concerningly, the delay appeared to be random and present in both directions. From manual inspection, we found instances of the timestamp preceding the keystroke by ~60ms, over 2 frames. In other instances, the timestamp was labeled more than ~50ms after the frame where the keystroke visually occurred. There was a lack of discernible patterns to these delays: There were multiple videos where positive offsets were initially observed, but as the frames progressed, the offsets decreased and became negative. Other videos exhibited the opposite effect.

This effect is illustrated in the below 5-frame timelines, Figures 5a, 5b, and 5c. Note that “frame pts” refers to the timestamp in milliseconds as recorded by the camera. Δt refers to the difference between “frame pts” and the recorded timestamp of the keypress. The frame highlighted in red is the frame labelled with the keypress by the closest approximation.

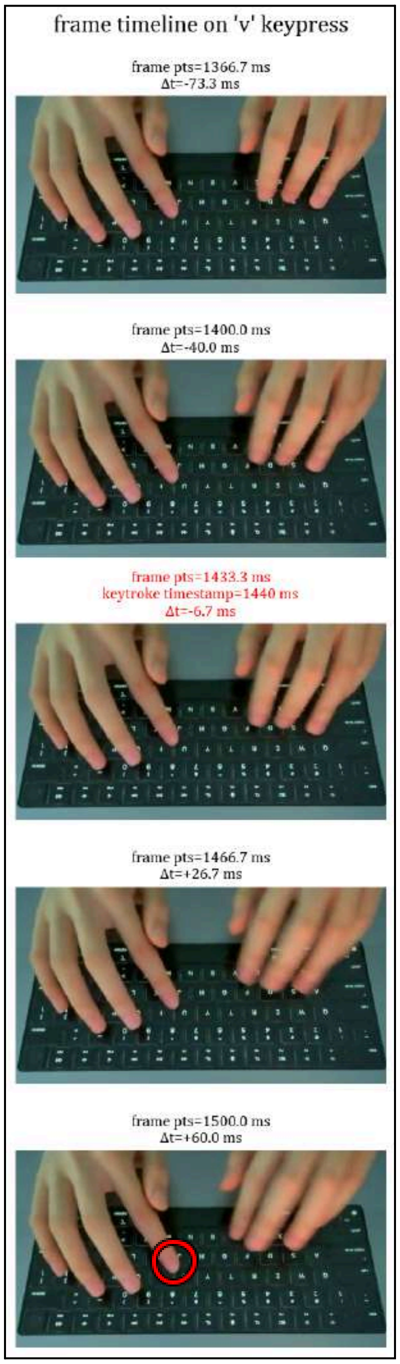


Figure 5a:
Timeline for a random 'v' keypress. In this series of frames, the actual keypress occurs at Δt=+60.0ms, 2 frames after the labeled frame.

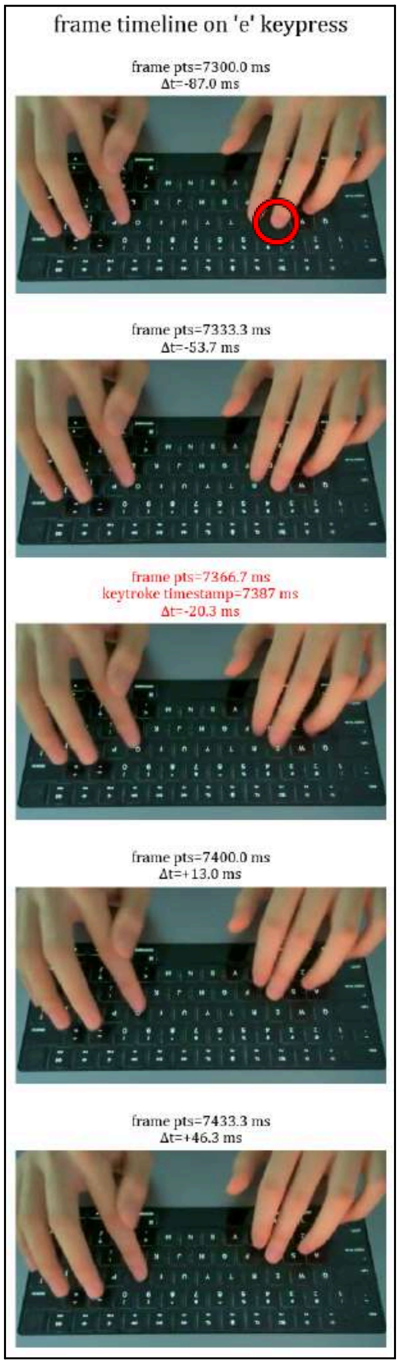


Figure 5b:
Timeline for a random 'e' keypress. In this series of frames, the actual keypress occurs at Δt=-87.0ms, 2 frames before the labeled frame.

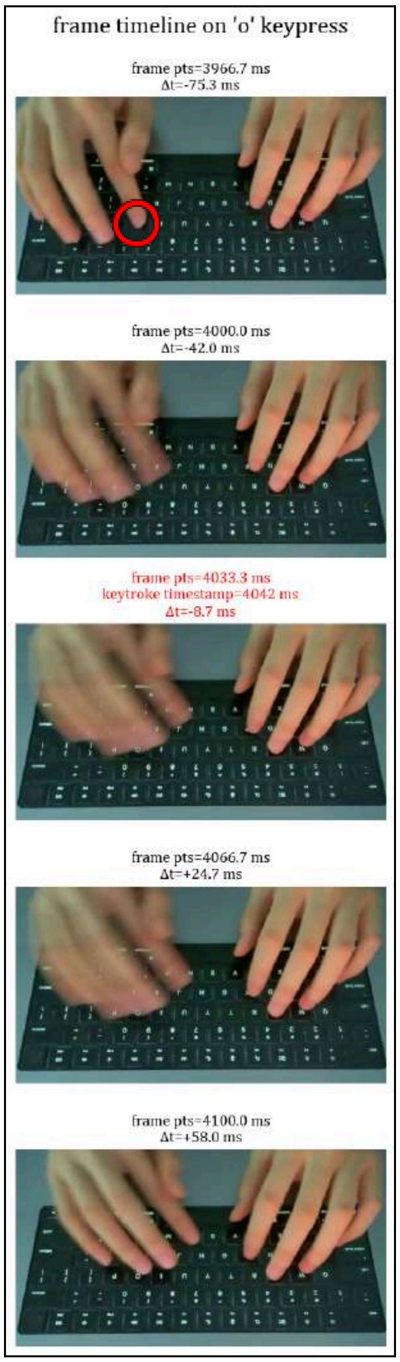


Figure 5c:
Timeline for a random '0' keypress. In this series of frames, the actual keypress occurs at Δt=-75.3ms, 2 frames before the labeled frame.

A suspected reason is the method for data collection. The keystrokes recording was likely conducted on the computer, allowing for little to no delay due to hardware wiring. Conversely, the image-capture device was likely an external camera, and thus would face delays in capturing keystrokes if linked wirelessly or with a slower hardware cable. However, this would explain a fixed delay between label and actual keystroke, but does not adequately explain the randomness observed. A possible further explanation is that the camera experienced variable buffering, whereby images were stored before being written to memory. Since this is determined by the speed of the write operation, it can vary from frame to frame, leading to a certain randomness, or jitter, in delays.

Another possible reason is Variable Frame Rate (VFR), whereby the presentation timestamp (PTS) of frames drifts and is corrected. This is a common capture mode, and is the default on commercial cameras and mobile phones. VFR and PTS are reliable capture methods. However, PTS is based on the camera's internal clock, which may run at variable speeds to the keylogger clock. We suspect that small drifts built up, leading to offsets. Some drivers periodically resync timestamps, producing sudden ±1–2 frame shifts. This can explain the random nature and changing direction of offsets.

Yet another hypothesis is an operating system problem that would have led to inaccurate labels. Keystroke event timestamping is polled at a dynamic refresh rate. If under load, the system may buffer event timestamps, delaying them by milliseconds. When the system later processes them, this can introduce random delays. These delays would likely be going in one direction, which is the camera capture being frames ahead of the labels. This can explain why, at times, the keystroke frame occurs before the label.

The superposition of several sources is likely at play in this phenomenon. Do note that we attempted, with some success, to mitigate this issue via labelling frames within a larger temporal window, that being ±50ms. This means that more frames, around three on average, would be labelled with a certain label for a given keystroke. This aimed to increase the generosity of the labelling, and ensure that keystrokes that occurred a temporal distance away from the attached timestamp would still be captured with the correct label. The results will be covered in Section 3.1.4

2.3 Data Preparation & Considerations

Below are the core steps of data preparation that we executed. Included also are our additional considerations that went into this phase.

2.3.1 Data Normalization

Each frame in the dataset was represented by a tensor [C, H, W] of shape [3, 480, 640], where 3 denotes the RGB color channels, 480 the image height in pixels, and 640 the image width in pixels.

Each RGB channel comprised integers from [0-255]. This required normalization, the research standard for mapping [0-255] values to [0-1] floating representation. We chose to use min-max normalization, known as rescaling. This involves a quick mathematical operation of dividing every value by 255.

The rationale behind this normalization is to provide numerical stabilization in optimization. Numerical stabilization is where the optimization algorithm can maintain accuracy without amplifying data features. An unstable numerical method, conversely, displays significant changes in outputs for minor changes in inputs, affecting model generalization [34]. High input values (>1.0) can cause overflow or underflow issues for optimizers, leading to numerical instability. This can be explained by common activation functions being designed with a stable range of [0-1] in mind.

It should be noted that another method of normalization is z-scaling, also termed as standardization, which we did attempt. This is where each value is rescaled such that the resulting distribution has mean of zero and a unit variance. We compared the two methods of normalization and their impact on model performance. Results of this comparison are in Section 3.1.1.

2.3.2 Data Transformation

Some transformation was done on the data.

First, the 3-Channel RGB frame data was flattened into a 1-Channel black and white sample. This was a complicated process as it also involved changing the use of certain models, which had to be augmented to accept single-channel data. One reason this was done is that single-channel analysis helps limit the complexity of the space, thus reducing memory and compute requirements. Another reason is that single-channel analysis may be better for pattern recognition, especially for tasks like keystroke inference, where color is unimportant. For such tasks, color data may dilute the number of features the NN can be attentive to, thus increasing the unnecessary noise the model must learn to ignore. This is why such data augmentation is seen in famous datasets such as the Modified National Institute of Standards and Technology (MNIST) dataset [25]. This transformation is seen below in Figure 6.

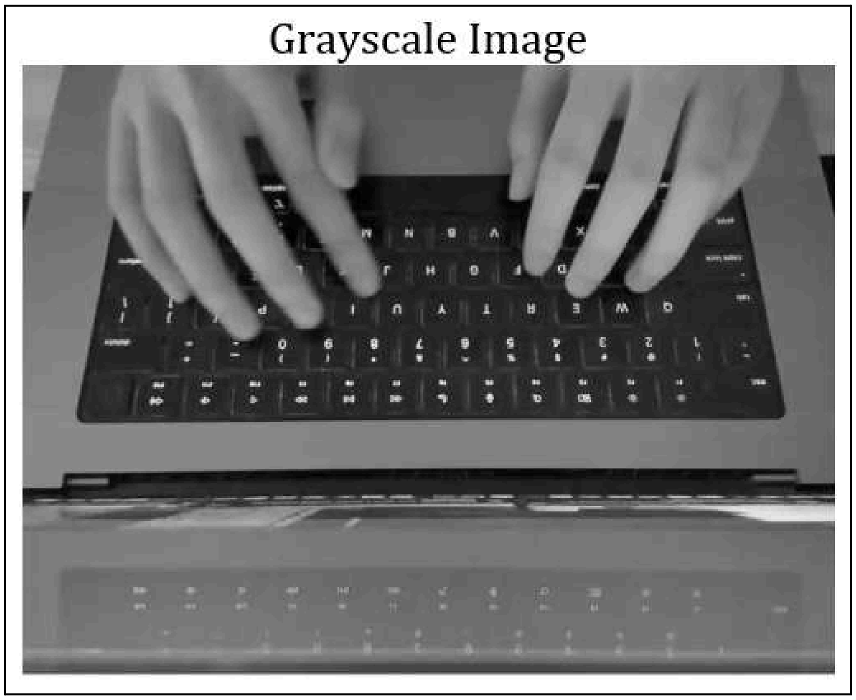


Figure 6: Shows a sample frame that has undergone the grayscale transformation.

Second, a contrast boost is applied to the image. This increases the differences in pixel values, thus amplifying existing trends. The mathematical patterns present will have more prominent differences, making it easier for the NN to extract relevant features. In this project, a value of 2.0 was chosen for the contrast factor. The transformation is seen below in Figure 7.

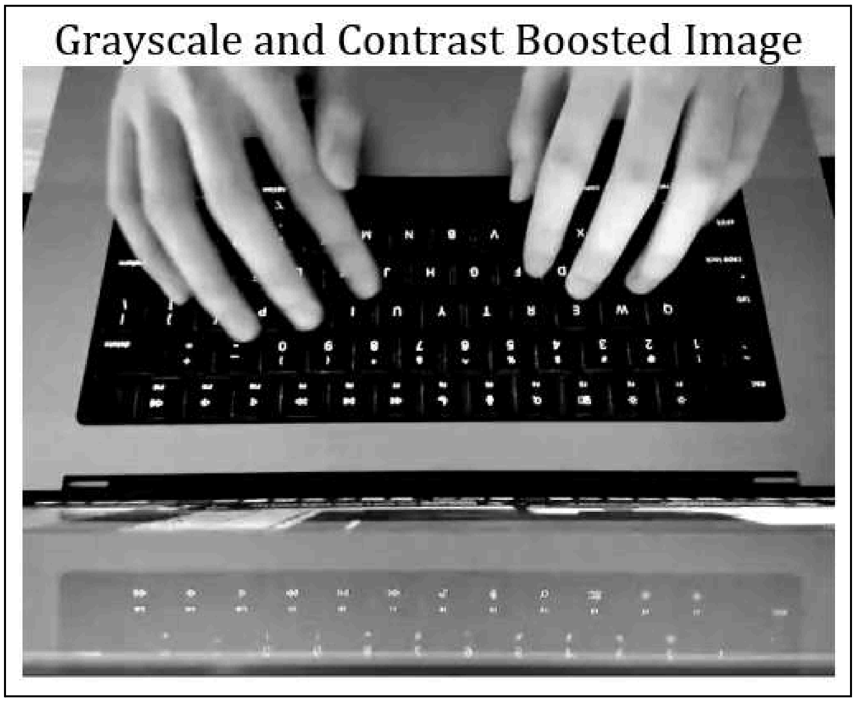


Figure 7: Shows a sample frame that has undergone the grayscale transformation and the subsequent contrast boost.

Third, the image is cropped. This addresses the limitation highlighted in 2.2.3 of redundant pixel data by cropping out sections of the video that do not contain necessary information for keystroke inference. The rationale is to reduce the amount of storage used by the training set, thus speeding up loading and training times. Another rationale would be to reduce the noise fed into the NN, ensuring it is better able to pick up on actual features such as changes in hand placement. The crop is illustrated in Figure 8.

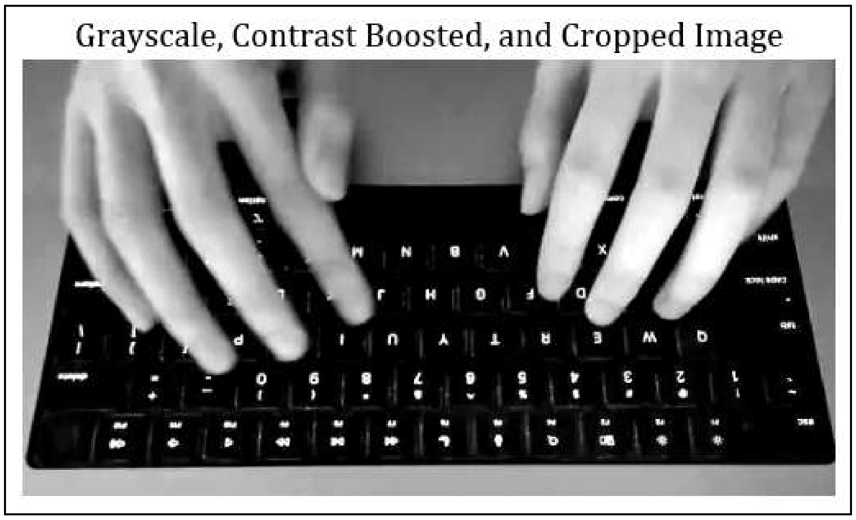
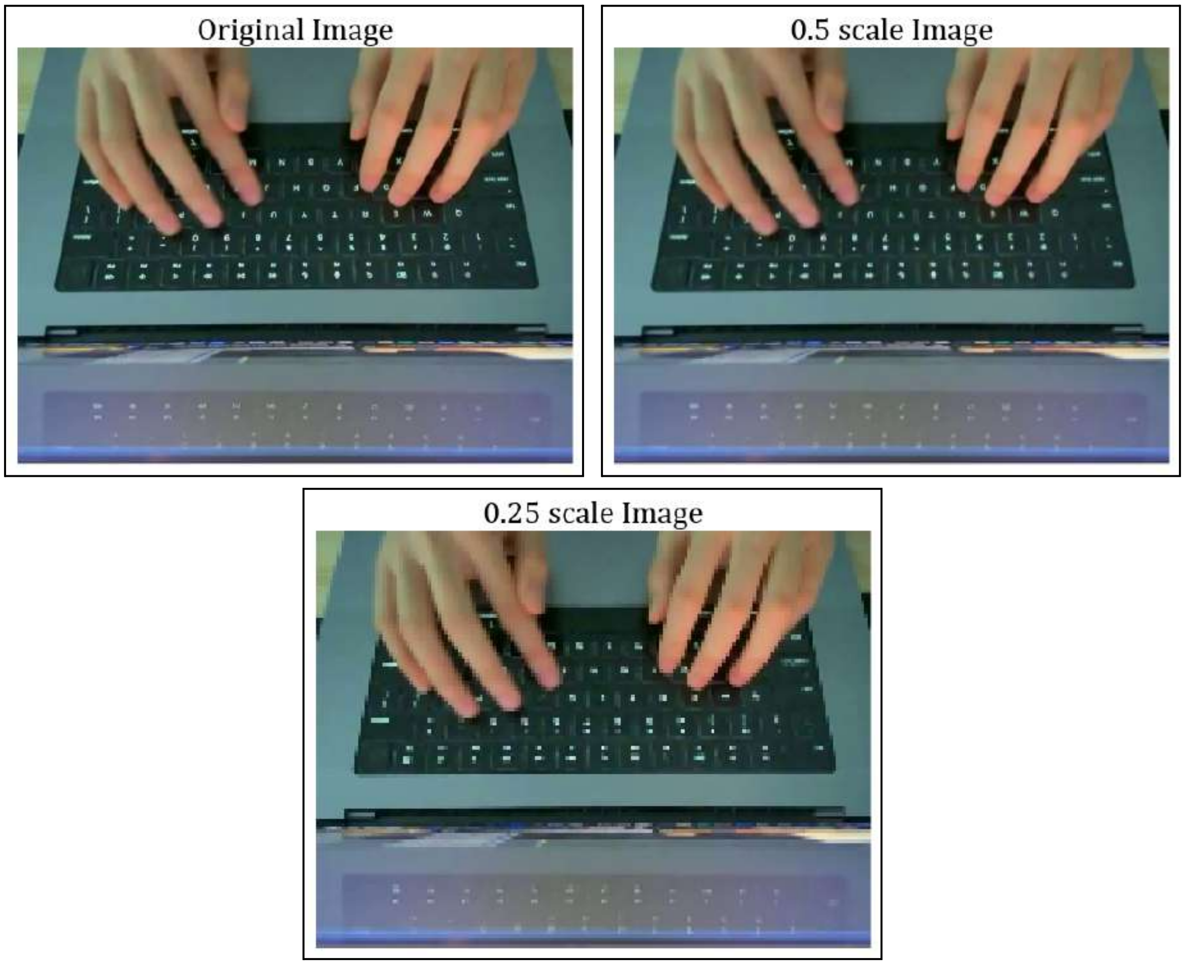


Figure 8: Shows a sample frame that has undergone the grayscale transformation, the subsequent contrast boost, as well as the crop.

Fourth, downsampling of the video. This was a reduction in the pixel density of the image. This was largely to decrease the size of the samples to reduce Random Access Memory (RAM) usage and training times. This was deemed as practical as the initial resolution of the video, [480, 640], was far higher than required for feature extraction. A scale factor is the size of the output image as a fraction of the original image. We chose a downsampling scale factor of 0.5 after visually comparing other scales. This comparison is shown below in figures 9a, 9b and 9c. As such, the final uncropped frame resolution was [240, 320], while the cropped frame resolution was [150, 280].



Figures 9a, 9b, 9c, respectively. This set of images illustrate the different downsampling scale factors, of 1.0, 0.5 and 0.25. As can be observed visually, a scale of 0.5 preserves the core features without compromising resolution, while 0.25 results in a loss of significant details, such as keyboard letters. As such, 0.5 scale is the most suitable.

Overall, these transformations reduced the training sample size from 58.31GB to 6.38GB. This allowed for superior training speeds and accuracy.

2.3.3 Label and Frame Selection

This is to address the limitation highlighted in 2.2.3 of poor distribution of labels. Initially, the dataset comprised 76 labels, including lower and uppercase alphabets, numbers, punctuation, and other standard keyboard keys such as "Shift_L".

First, we merged lower-case and upper-case letters. This decision was due to the low count of certain upper-case letters such as 'Q', 'Z', and 'X'. This was also decided as we felt that capital letters may confuse the NN, considering it would have to extract the feature of the shift key being held down when typing capital letters.

Next, we removed all classes involving punctuation, excluding 'Space'. This was because we wanted to focus on just the alphanumeric characters that were more prominent in the dataset. Removed labels had their frames relabeled to 'nothing'.

Lastly, 'nothing' frames constituted the majority of the dataset (69.1% of all frames). Thus, we applied random undersampling to reduce their dominance. We retained one out of every 30 'nothing' frames to preserve representativeness while preventing class imbalance, discarding the remaining 29. This reduced the number of 'nothing' frames to 2880, resulting in a more balanced dataset. Without this selection, the model could trivially achieve 69.1% accuracy by always predicting 'nothing', without learning to generalize to actual keystrokes. Thus, this step was required.

These changes led to a better distribution, shown below in Figure 10. We were left with 38 classes.

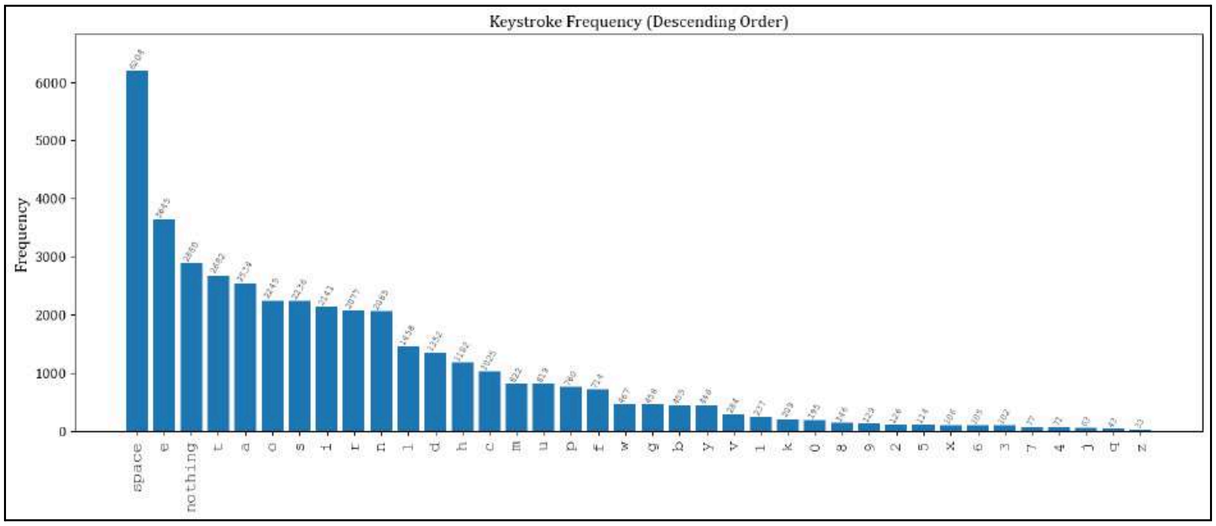


Figure 10: Shows the number of occurrences for each keystroke label, sorted in descending order. The distribution has been improved from Figure 3.

While distribution was improved, it is evident that it is still far from perfectly balanced. We did not want any further data manipulation out of concern that it could impact training outcomes. As such, we did attempt weighted training, also called class balanced training, which will be reviewed in Section 3. Class balanced training is where loss contribution of each sample is scaled by a weight inversely proportional to its class frequency. This prevents the model from minimizing loss by favoring majority classes, incentivizing the model to generalize for rare classes. This approach balances the loss so that each class contributes approximately equally to the gradient update, regardless of its prevalence.

```
class_weights = [len(train_samples) / (num_classes *
d[i]) for i in range(num_classes)]

weights_tensor = torch.tensor(class_weights,
dtype=torch.float).to(device)

weights_tensor = torch.clamp(weights_tensor,
max=10.0, min=1.0)
```

We implemented this with the code snippet above. We used a basic inverse function to compute inverse-frequency weights for all classes. We then convert this into a tensor for PyTorch and clamp the weights to prevent destabilization due to extreme weights.

2.3.4 Data Loading

We chose a batch size of 16 for data loading. This was empirically determined as the largest size that could be accommodated within the available GPU memory while maintaining stable training dynamics. This is a smaller batch size, which helps introduce beneficial noise into gradient descent. This allows for more changes in gradient per epoch, something important for datasets of limited scale such as ours to achieve generalization.

2.4 Models

We evaluated multiple convolutional neural network (CNN) model architectures. Each model had a different design paradigm, which would yield varying results on the keystroke inference task. By benchmarking across a range of models, we sought to compare performance and identify the most appropriate architecture. In this section, we will give a brief overview of the models we chose to use, along with their features. For definition purposes, Floating Point Operations (FLOPs) refers to a mathematical operation conducted on a floating-point number. The total required FLOPs for a model's inference refers to the total number of operations the model conducts to evaluate an input and produce an output, and is thus a measure of model efficiency.

In addition to convolutional backbones, a feed-forward NN was implemented as a baseline. This architecture is also called a Multi-Layer Perceptron (MLP).

We report parameters and total FLOPs at 224×224 resolution with 3-Channels for all models (CNNs and MLP) to ensure fair comparability.

2.4.1 EfficientNet_b0 (CNN)

EfficientNet was introduced by Tan and Le (2019) [35] as a family of CNNs derived through Neural Architecture Search (NAS), an automated process for designing highly optimized architectures. This allowed for innovative optimization improvements; one example is depth-wise convolutions, where features are extracted on a per-channel basis, before later being merged to capture information on the whole image, thus reducing compute used. EfficientNet-B0 is the baseline model.

EfficientNet-B0 achieves competitive ImageNet accuracy with substantially fewer parameters and FLOPs compared to earlier CNNs. It is a strong base model and is often use in vision problems as an initial baseline. Its emphasis on parameter efficiency also reduces the risk of overfitting on small datasets.

EfficientNet-B0 has approximately 5.3 million parameters and requires 0.39 billion FLOPs for inference at standard 224×224 resolution with 3-Channels [36].

2.4.2 ResNet18 (CNN)

ResNet, introduced in Kaiming He et al. in 2015 [37], was the first major architecture to popularize the concept of residual learning through skip connections. This is where input layer features are combined with features of the output layer, allowing for the “skipping over” of layers in between. Thus, ResNet is easier to optimize, and benefits from less gradient degradation with increased depth. ResNet-18 is a standard variant in the ResNet family, with 18 layers.

ResNet-18 is also widely used in research. Its architecture enables efficient training and strong convergence even without sophisticated scaling.

ResNet-18 has approximately 11.7 million parameters and requires 1.8 billion FLOPs at 224×224 resolution with 3-Channels [38], [39].

2.4.3 MobileNetV2_100 (CNN)

MobileNetV2, introduced by Sandler et al. in 2018 [40], is a CNN designed for efficient deployment on mobile and other resource-constrained devices. This architecture takes an inverse approach to ResNet, introducing inverted residuals and linear bottlenecks, where input and output layers are limited in size while intermediate layers are allowed to increase dimensionality. Intermediate layers are processed with depth-wise convolutions, similar to MobileNetV1 and EfficientNet. This significantly reduces computation while maintaining representational capacity. The “100” variant denotes the default width multiplier.

MobileNetV2-100 is a lightweight and powerful model, with fewer parameters than ResNet-18 or EfficientNet. It has been widely used in medical imaging analysis. This model was used to test the feasibility of keystroke inference being deployed on resource-constrained devices. We hypothesize this to be the main vector of future attacks, given the more covert nature of utilizing a mobile phone or a laptop to conduct the inference.

MobileNetV2-100 has approximately 3.5 million parameters and requires 0.3 billion FLOPs at 224×224 resolution with 3-Channels [41].

2.4.4 ConvNeXt_Tiny (CNN)

ConvNeXt, developed by Liu et al [42]. in 2022, is a CNN architecture inspired by design elements of Vision Transformers (ViTs), a recent alternative to CNNs. Researches updated traditional CNN structures with modern architecture choices. For example, the model uses large, 7x7 kernels so CNNs can capture broader context, similar to the global attention present in ViTs. The Tiny variant is the smallest model in the ConvNeXt family.

ConvNeXt-Tiny achieves high accuracy with improved representational power. It thus provides performance on par with the superior ViTs, while retaining the inductive bias and efficiency of CNNs.

ConvNeXt-Tiny has approximately 28.0 million parameters and requires 4.47 billion FLOPs at 224×224 resolution with 3-Channels [43].

2.4.5 Feed-Forward (MLP)

MLPs are the earliest form of artificial NNs, originating from a 1958 paper by F. Rosenblatt [44]. They are more basic than modern architectures. In vision problems, feed-forward is often seen as inferior to CNNs, as they are unable to evaluate features spatially. This architecture consists solely of fully connected layers applied to flattened image tensors, without the use of convolutional techniques to extract features. However, this does make MLPs less computationally expensive. An example is how CNNs have sliding window analysis which substantially increases the number of operations for a given image input size.

MLPs learn direct mappings from raw pixels to class labels. This means that every output label in the output layer is directly connected to every pixel through a chain of dense weights. This makes pattern recognition difficult as there is no explicit notion of locality.

Its inclusion served as a control to evaluate the extent to which spatial feature extraction is necessary for keystroke classification. While computationally efficient and straightforward to train, the model is hypothesized to underperform CNNs.

Parameter and FLOPs count can be calculated easily for a MLP. A 224x224 resolution image tensor with 3-Channels can be flattened into a 150528 dimensional vector, and passed through a stack of dense layers. Assuming one hidden layer, as we have implemented, this would contain 77.4 million parameters, requiring around 0.15 billion FLOPs per forward pass.

2.5 Architecture Hyperparameters

Model hyperparameters refer to the initial configuration of the architecture that is set prior to training. This section outlines the principal hyperparameters chosen in this study and their rationale.

2.5.1 Pretrained

We attempted using both pretrained (for CNN models) and non-pretrained hyperparameters for the architectures used. We hypothesized that pretrained models were lacking in accuracy. This was due to our reasoning that many commercial models are trained on very large and varied datasets, and are not suited for a task requiring only a specific classification ability. For instance, the reputable ImageNet-1k that was used to train EfficientNet comprises 1000 object classes and 1281167 training images [45]. Many features learned within the pretrained EfficientNet are likely not helpful for this study, and only further confuse the model.

We considered layer freezing with pretrained models. This involves freezing the lower layers within the CNN. Theoretically, this can capture low-level feature recognition that trained CNNs have already mastered, while allowing the model to adjust later parameters corresponding to higher-level features to extract finer details from samples. However, we hypothesized that this would be unhelpful in this problem. This was due to the complex nature of the task, where existing parameters likely had to be completely retuned to be used in such a specific domain. We hypothesized that keystroke recognition requires significant retuning across the network, as even low-level pretrained filters would not be sufficiently aligned with the visual patterns in this dataset.

Non-pretrained architectures were used for the majority of experiments. We did run experiments to compare pretrained architectures with pretrained architectures trained with freezing, along with non-pretrained architectures. Results will be reported in Section 3.

2.5.2 Hidden Layer

A hidden fully connected layer of size 512 units was introduced in the classifier head. This was to account for more complex feature extraction, and is often a standard design choice for vision problems. Performance with the hidden layer was better than without, and these results will again be shared in Section 3.

2.5.3 Dropout Rate

Dropout is a regularization technique, often used in vision problems and the wider ML space. During training, a random selection of neurons are set to zero, the proportion of which is determined by the dropout rate. This does not carry on to validation nor testing phases. This encourages generalization and prevents overfitting, as it prevents the NN from relying on certain specific pathways.

A dropout factor of 0.4 was chosen. This is rather aggressive but was chosen due to the observed tendency to overfit and the small dataset size.

2.5.4 Rectified Linear Unit

Rectified Linear Unit (ReLU) is a widely-used activation function in vision problems. Defined as $f(x) = max(0, x)$, it was utilized to introduce non-linearity into feature extraction. It involves shutting off neurons whose parameters are negative. This enables the network to model complex, non-linear relationships in the data that would not be possible with purely linear functions. This helps increase the ability of the CNN to learn complex features.

2.6 Training Hyperparameters

Training was conducted on Kaggle hardware, a NVIDIA Tesla P100 GPU with 16 GB VRAM, and 8 vCPUs.

```
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
optimizer = torch.optim.Adam(
    filter(lambda p: p.requires_grad,
model.parameters()),
    lr=1e-3
)
```

The model was trained using cross-entropy loss with label smoothing as the objective function. Cross-entropy loss is the standard for classification problems. This function penalizes divergence between the predicted probability distribution and the ground truth label. Label smoothing, set at a factor of 0.1, was incorporated to prevent the model from becoming overconfident. Label smoothing functions by distributing a fraction of probability mass across incorrect classes, promoting generalization. This is important for our project, where classes may be visually similar. For example, the difference in frames between keystrokes adjacent on the keyboard may prove difficult to discern for NNs.

Optimization was performed with the Adam optimizer. Adam combines the benefits of Adaptive Gradient Algorithm and Root Mean Square Propagation [46]. Adam maintains per-parameter learning rates that adapt based on both the mean and variance of past gradients. This makes it especially effective for problems with sparse or noisy gradients, and suitable for training models from scratch on limited data. Compared to optimizers such as stochastic gradient descent (SGD), Adam typically converges faster and requires less manual learning rate tuning.

A learning rate of 1×10^{-3} was used as the initial value.

2.6.1 Checkpoint system

A checkpoint system was introduced into the training loop. This was in response to empirical trends of overfitting, whereby after a certain number of epochs the validation loss was observed to increase rapidly while training loss continued to decrease.

The system would choose to save the model when a new highest F1 score was attained. Capturing the best model produced by training when improvements cease ensured that training could run for a fixed number of epochs without risking degradation in generalization.

2.6.2 Learning Rate Scheduler

```
scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer,
    step_size=10,
    gamma=0.5
)
```

Initially, we chose to use the basic time interval stepping for learning rate, shown above. his schedule decays the learning rate by a factor of 0.5 every 10 epochs. However, we wanted to try other learning rate schedulers as well.

```
scheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode='min',
    patience=5,
    factor=0.5
)

scheduler.step(val_loss) #in training loop
```

We then attempted to implement progress-based intervals for stepping the learning rate. This is seen in the snippet above, where the scheduler is set to ReduceLROnPlateau, and scheduler.step is called every epoch to ensure learning rate is stepped proportional to validation loss. When validation loss plateaued for five consecutive epochs, the learning rate was halved. This was hypothesized to create a more effective training regime and is a common practice to prevent overfitting.

Results comparing the two different methods are examined in Section 3.

2.7 Post-Processing

Post-processing is commonly employed in related work to improve the accuracy of NN inference. For instance, some studies (see Section 1.1.2) incorporate text analysis algorithms as an error-correction layer, leveraging linguistic patterns and contextual dependencies to correct misclassified keystroke inferences.

However, similar to our decision as to the inessential need for temporal context, we chose not to include a post-processing step. Our main intention in this project is to target keystroke inference of sensitive data, in particular inputs such as passwords. These are not context nor language dependent. Text-analysis algorithms assume, and are constructed for, the analysis of linguistically structured input. As such, these post-processing steps do not align with our study's goals.

2.8 Evaluation

2.8.1 F1 Score

The F1 score was chosen as our primary evaluation metric. It is typically used for imbalanced datasets as a standard accuracy metric could potentially be misleading. This combines precision and recall into a single score, with the final value being their harmonic mean. The F1 score of a class is calculated using precision, the percentage of correct classifications, and recall, the percentage of that class in the whole dataset that was identified. These scores for each class are then combined using a weighted average to give a single score. The score is from 0 to 1, with a higher score indicating better model performance. The derivation of the F1 Score can be seen below in.

$$f_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

(TP = True Positive, FP = False Positive, FN = False Negative)

This was chosen due to dataset imbalance, seen in Figure 3. The accuracy metric, used in multiple similar studies, is insufficient in this case. Accuracy can be misleading when majority classes dominate, as it only reflects percentage of total correct inferences. F1 scores reflect a model's ability to correctly distinguish minority classes and thus are a more sophisticated indicator of generalization, even under imbalance.

2.8.2 Other metrics

In evaluating training, we also employed 3 additional metrics to illustrate other components of the training process. This was training loss, validation loss, and validation accuracy.

Training Loss quantifies the error between model predictions and actual labels when training. Decreasing loss is indicative of a model that is learning. Training loss is expected to converge to a mostly steady value, with the speed of convergence being indicative of speed of learning, while the value towards which training loss converges is indicative of depth of learning. Higher speed of convergence to a lower loss value indicates faster learning and greater depth of learning.

Validation Loss quantifies how the model performs when labelling data not included in the training dataset. This is used primarily as an indicator of overfitting—the phenomenon where the model fails to generalize, instead adapting to recognize the training data and the specific noise associated with each training set. This allows the model to perform exceptionally when classifying training data, but poorly when attempting to classify data it was not trained on. Validation loss is used in studies as a metric for overfitting, with increases in validation loss mathematically indicating poorer accuracy in labelling validation data and thus implying overfitting.

Lastly, Validation Accuracy was used. This refers to the proportion of keystrokes the model correctly labelled in the validation dataset. A higher score is preferred as it indicates the model is better at recognizing patterns and is generalizing. However, note that there is a weight bias nature of validation accuracy, as common classes are given higher weightage due to the presence of more instances of the class within the dataset. As such, the model may achieve a higher validation accuracy by generalizing just a few common classes and thus fail to generalize for less common classes. As such this metric, while useful, is not the main metric used to determine performance.

3 RESULTS AND ANALYSIS

The general results of our experiments was a model that produced satisfactory results indicative of a generalized solution.

We conducted a total of 26 successful training cycles of 60-100 epochs, producing 26 models with differing hyperparameters and results. In total, we spent 75 hours training models, with each model taking between 2 to 13 hours to train due to varying hyperparameters.

The training cycles have been grouped into 10 different experiments, which each compare two or more models. Each experiment in the following sections is used to deduce the importance of certain hyperparameters, as well as analyze the implications of interesting results.

We evaluated the below results using F1 score, outlined in Section 2.8.1, as well as epoch-based graphs of Training Loss, Validation Loss, and Validation Accuracy, outlined in Section 2.8.2.

3.1 Results from dataset hyperparameters

3.1.1 Evaluating Normalization

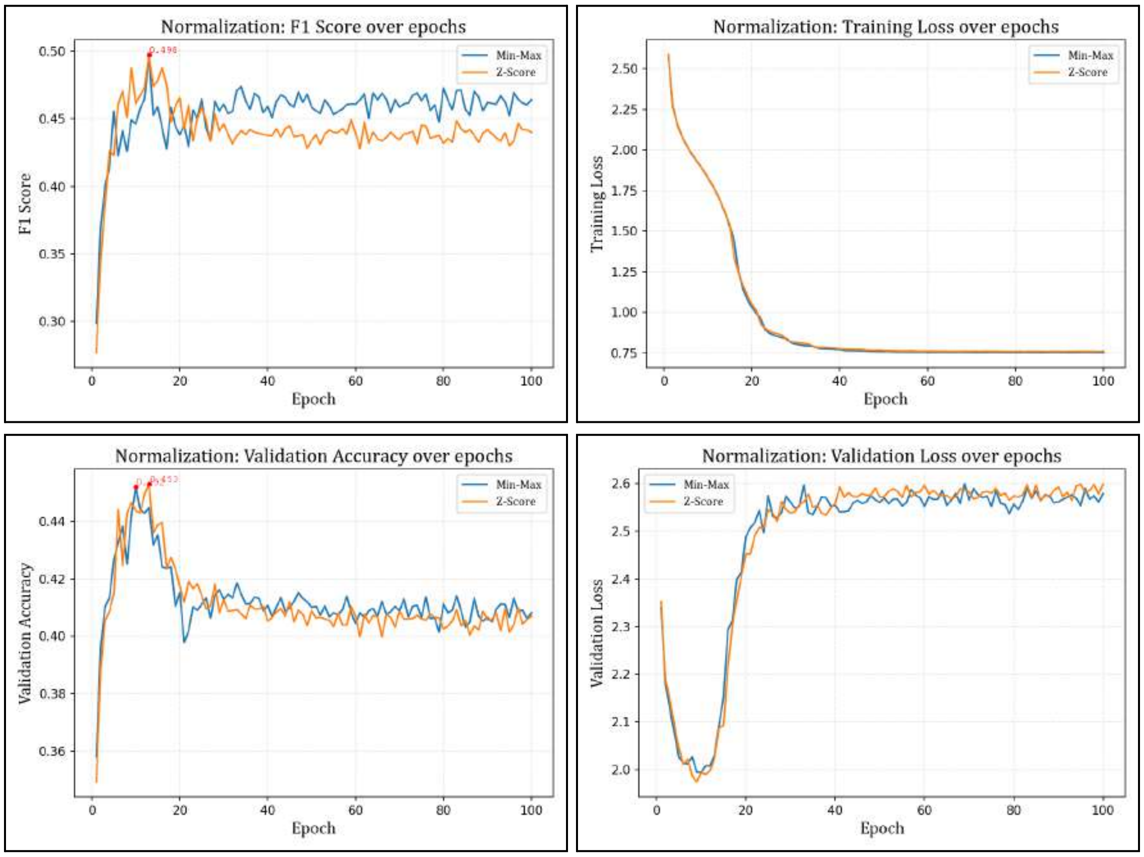


Figure 11a, 11b, 11c, 11d, respectively

This experiment considered two models. The first model was trained with data prepared using Min-Max normalization, and the second model was trained with data prepared using z-scoring. These have been defined in Section 2.3.1. The aim of this experiment was to evaluate the two normalization techniques.

Results show that training loss was nearly identical, seen in Figure 11b. Validation loss in Figure 11d also indicated similar trends, that being overfitting beginning around epoch 10 for both models. Figure 11c shows that validation accuracy remained closely linked together, with z-scoring yielding a slightly higher accuracy of 0.453 compared to 0.452. When we consider F1 score in Figure 11a, we see that although trends diverge slightly as the models overfit, the initial few epochs illustrate a growth closely tied to each other. Both models achieve the same peak F1 score of 0.498.

As such, we can consider that in this experiment, the use of Min-Max normalization versus z-scoring normalization has no significant impact on model performance. We can potentially infer this means that the footage had a significant and rather even spread in pixel brightness values, such that both forms of normalization yielded similar datasets. Another possible conclusion could be that our model is able to extract patterns regardless of normalization method, which could be due to the sharp contrast between hand skin tone and keyboard, which was present regardless of normalization.

3.1.2 Evaluating Data Transformation

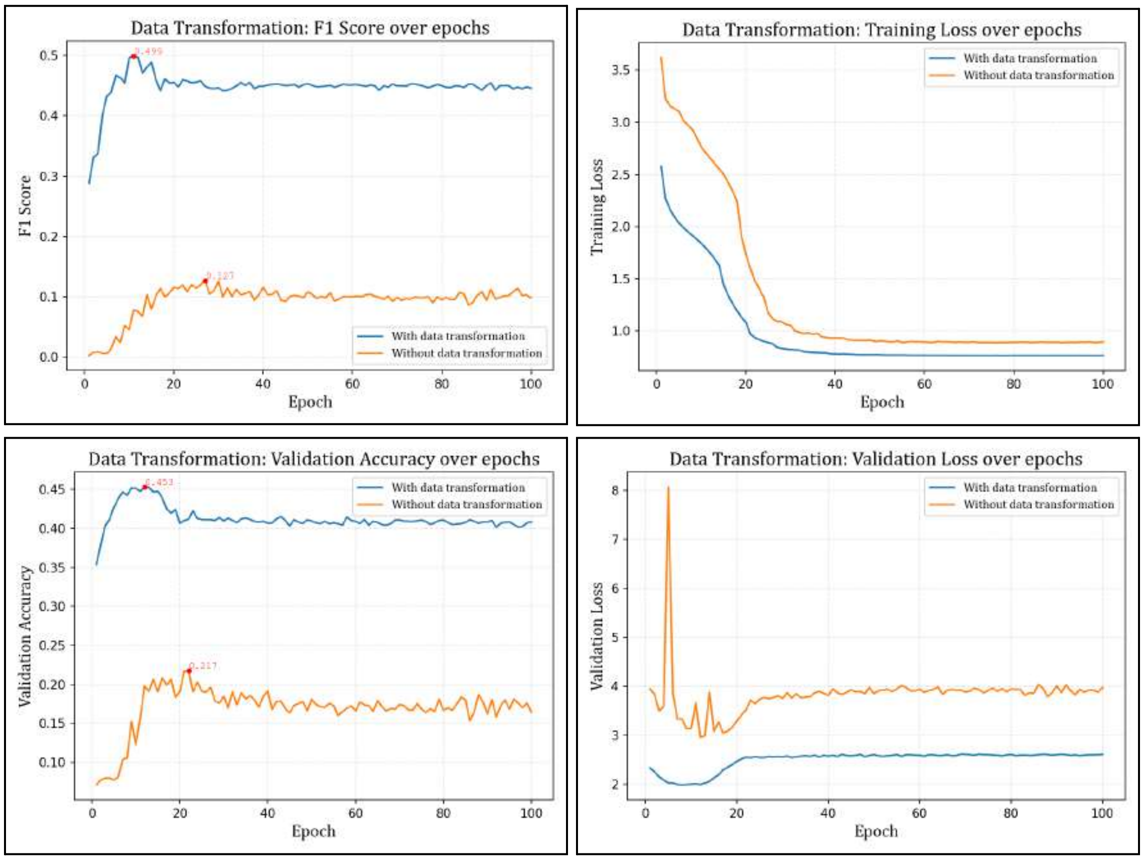


Figure 12a, 12b, 12c, 12d, respectively

This experiment considered two models. The first model was trained with data transformed as outlined in Section 2.3.2, while the second model was trained on data that was not transformed. The aim of this experiment was to evaluate the usefulness of engaging in data transformation in the preparation phase.

Results show large differences. In training loss, data transformation allowed for faster convergence and convergence to a lower loss level, seen in Figure 12b. Validation loss in Figure 12d, however, indicated that both models began overfitting at the same epoch, around epoch 13 for both models. Figure 12c and 12a shows a large discrepancy in validation accuracy and F1 score. The model trained on transformed data has a F1 score and validation accuracy greater than the other model, with a peak F1 score of 0.453 compared to 0.217 for the model trained without transformed data.

As such, we can consider that this experiment showcases the importance of data transformation on model performance. This indicates that our hypothesis when executing data transformation, that it would improve performance as it allows the model to better focus on extracting essential patterns, can likely be accepted. It is also worth noting that the reduction in file time from data transformation led to faster training times, from 8.7 minutes per epoch to 1.4

minutes per epoch. This is another point of consideration for the importance of data transformation.

3.1.3 Evaluating the addition of Time Context

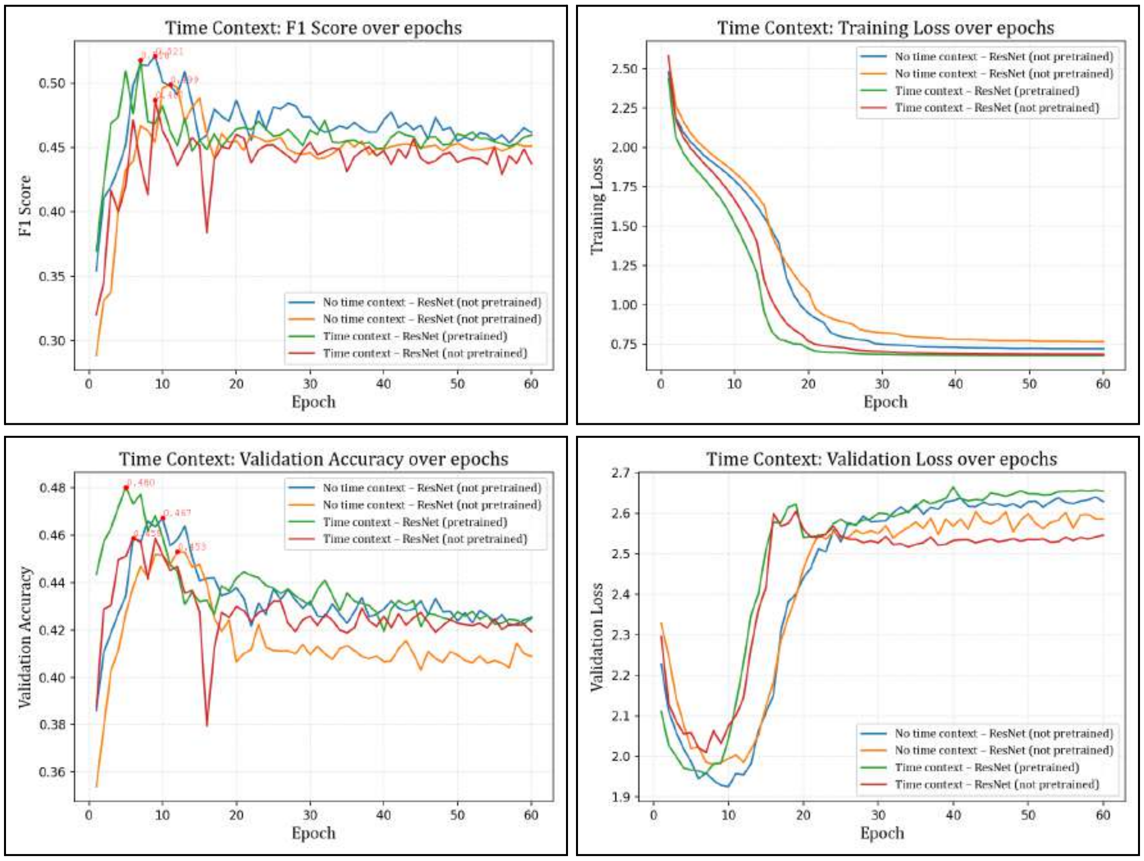


Figure 13a, 13b, 13c, 13d, respectively

Note that the addition of time context was accomplished via the methodology outlined in Section 2.2.2, and in brief involves each data sample consisting of a series of three sequential greyscaled images, stacked together as one tensor creating a 3-Channel sample attached to one label.

This experiment considered four models, that being a non-pretrained ResNet model trained on the regular dataset, a pretrained ResNet model trained on the regular dataset, a non-pretrained ResNet model trained on a time context included dataset, and a non-pretrained ResNet model trained on a time context included dataset. The intention was to have two sets of comparisons between models with and without time context.

When considering Figure 13b, we see that time context allows for faster convergence to a lower training loss value, which could indicate better learning. However, when we look at figure 11d, we see that time context informed models, regardless of pretraining, begin overfitting at earlier epochs, starting around epoch 9, while without time context models only start overfitting around epoch 13. This could explain the lower training loss, as with time context overfitting occurs faster. F1 score in Figure 13a shows a clear trend, that time context informed models perform worse than their non time informed counterpart models. The peak F1 score is attained by the pretrained ResNet without time context, a value of 0.521. Interestingly, Figure 13c illustrates that time context seems to increase validation accuracy, seen by how both pretrained and non-pretrained models improved in validation accuracy by a decent margin when time context was introduced.

Thus, we can conclude that time context has a complicated effect on model performance. We suggest that time context informed models have improved ability to learn common patterns, but an inferior ability to generalize for classes with lower representation. This was inferred from the lower F1 score but higher validation accuracy. We hypothesize this could be because time context informed training requires larger quantities of data, and thus only performed well for classes with larger representation within the dataset. Overall, as per F1 score, models without time context perform better in this experiment.

3.1.4 Evaluating Generous Labelling

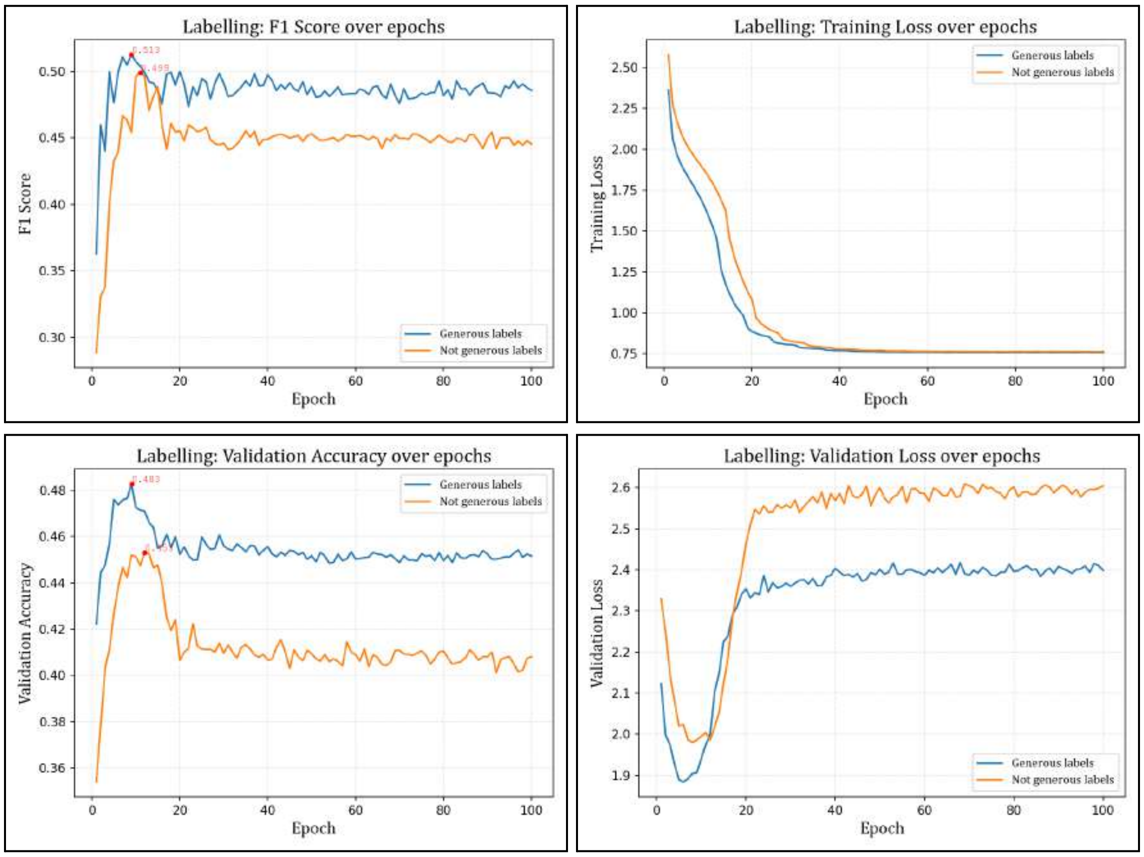


Figure 14a, 14b, 14c, 14d, respectively

This experiment considered two models. The first model was trained with the standard dataset as outlined in the methodology. The second model was trained on a dataset where there was a larger temporal window, that being ±50ms instead of ±30ms. The rationale and method were outlined in section 2.2.3 and involves labelling multiple frames the same label for a given keystroke. This is termed by us as “generous” labelling and was our attempt at mitigating the issue outlined in 2.2.3 of random delay between frame and keystroke timestamps.

In training loss, generous labels allowed for faster convergence, while converging to approximately a similar loss level as the model trained on non-generous labels, seen in Figure 14b. Validation loss in Figure 14d indicates that models trained on generous labels begin overfitting earlier, around epoch 6 versus epoch 11. However, models trained on generous labels overfit to a lesser degree, seen by how the steady state of validation loss was lower for these models than those trained without generous labels. Both Figure 14c and 14a illustrate that models trained on generous labels perform better, both in terms of validation accuracy and F1 score. The model trained on generous labels has a peak F1 score of 0.483 compared to 0.453 for the model trained on data not generously labelled.

As such, we can surmise that this experiment illustrates how generous labels are able to improve model performance, and this is likely due to it achieving our intended purpose of mitigating the random delays, thus allowing for more accurate labelling.

3.2 Results from different models

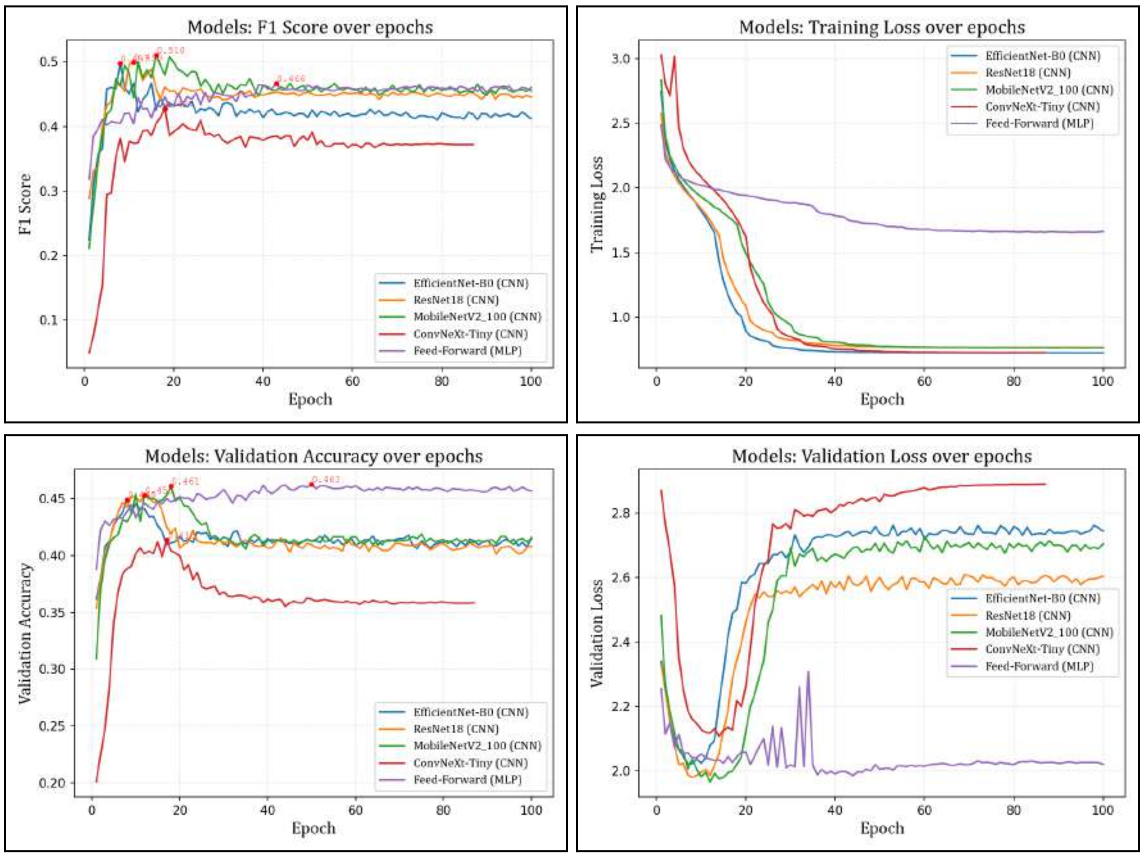


Figure 15a, 15b, 15c, 15d, respectively

This experiment considered five models. These are the models as outlined in Section 2.4, and in brief they consist of EfficientNet_b0, ResNet18, MobileNetV2_100, ConvNeXt_Tiny, and a Feed-Forward (MLP) architecture. It is worth noting that ConvNeXt_Tiny was the heaviest model of these five, with the highest parameter and FLOPs as observed in Section 2.4, and this led to prohibitively long training times. We stopped after 87 epochs as the Kaggle System timed out due to an overrunning of 12 hours of training time. We chose not to resume training, since at epoch 87 the trends were already largely clear, with improvement in the last 13 epochs deemed unlikely and not worth the additional cost of time and compute.

Figure 15b illustrates training loss, where we can observe that EfficientNet converges the fastest while Feed-Forward struggles to converge. Looking at level of loss convergence occurs at, we can deduce that EfficientNet and ConvNeXt reach a deeper level of learning compared to MobileNet and ResNet, while Feed-Forward converges far above the other models, indicating difficulty in learning.

Validation loss in Figure 15d indicates that EfficientNet began overfitting the first, with the other CNN models following in the subsequent epochs. This could explain why EfficientNet has such a fast training loss descent—that being it is overfitting very quickly. Interestingly, Feed-Forward validation loss does not display signs of overfitting. When we consider Figure 15b, we conclude this is likely indicative of underfitting, where the feed-forward model is struggling to extract patterns.

Figure 14c illustrates that MobileNet produces the best accuracy of all CNN models, with ConvNeXt struggling the most. Feed-Forward produces results that are far better than hypothesized, coming in the most accurate model by a thin margin. However, when we consider F1 score in Figure 15a, we can observe that MobileNet is the best performer with a peak F1 score of 0.510, with Feed-Forward only having a peak F1 score of 0.466.

We hypothesize that this discrepancy between Feed-Forward validation accuracy and F1 score performance is due to the Feed-Forward model likely learning to “cheat” by guessing the majority classes that dominated the dataset. This can be inferred as the low peak F1 score indicates it was not generalizing well to less frequent classes. Thus, we can say that Feed-Forward architecture struggled with this problem.

As such, we can conclude that this experiment showcases MobileNetV2 (100%) as the best architecture overall for this problem. The feed-forward model’s underperformance was predicted in Section 2.4.5, and thus likely indicates that spatial feature extraction is needed for this problem. Convolutional architecture, and MobileNetV2 in particular, are more usable for this case of vision-based classification.

3.3 Results from architecture hyperparameters

3.3.1 Evaluating Model's Pretraining

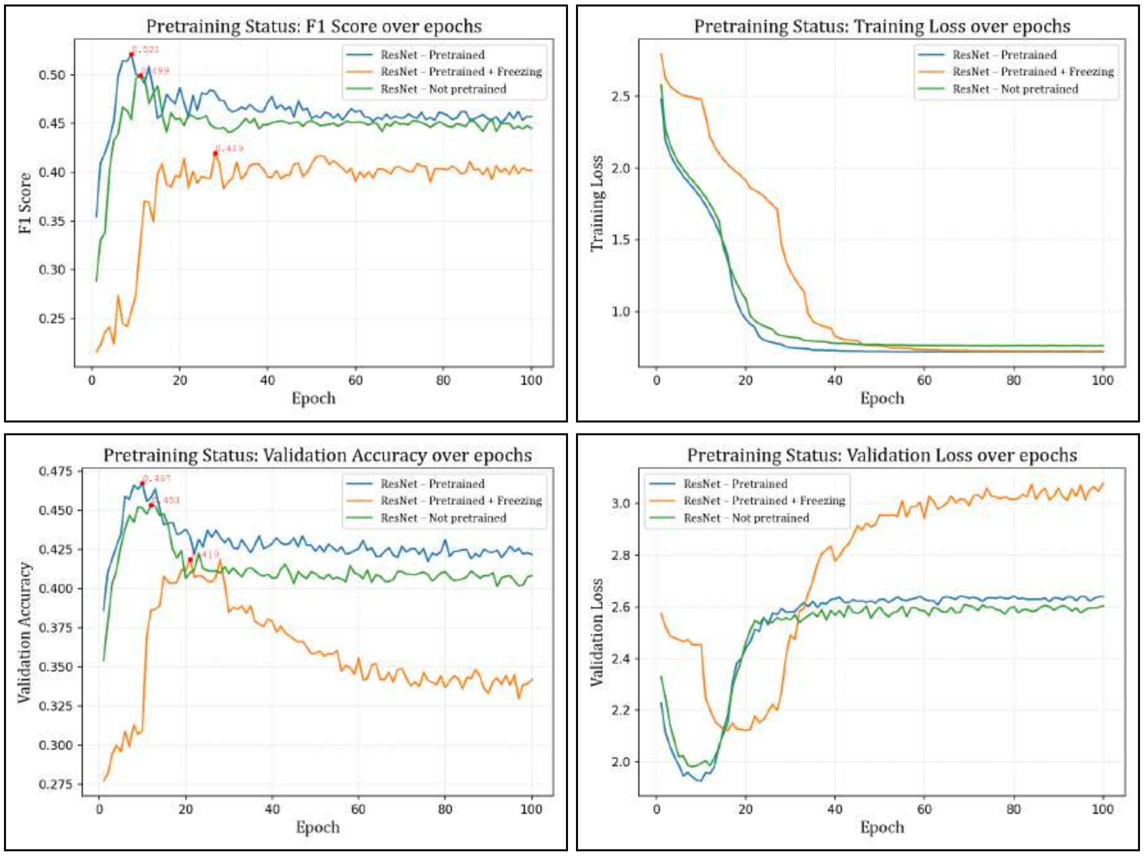


Figure 16a, 16b, 16c, 16d, respectively

As outlined in Section 2.5.1, we wanted to investigate model pretraining and its effects on results, as well as considering the hybrid case where pretrained models were trained with a layer freezing method.

This experiment consisted of 3 models, one pretrained before training, one pretrained and trained with layer freezing, and the last a non-pretrained model. To keep other factors consistent, they were all training on the same dataset and were all ResNet architectures.

When considering Figure 16b depicting training loss, we see that the pretrained ResNet converges faster and to a lower level of loss compared to the non-pretrained one. This indicates that pretraining aids with learning. Interestingly, the model trained with layer freezing lags behind. For the model trained with layer freezing, we observe "steps" in the training loss. We suspect this could indicate freezing was successful, with each sudden increase in descent possibly indicating a fresh layer being unfrozen. However, this method seems to worsen learning speed, seen by the slower convergence, although the model does converge to a lower loss value than non-pretrained ResNet.

Figure 16d shows that regardless of pretraining, the models begin to overfit around epoch 10. However, the pretrained model with layer freezing displays overfitting only at a later epoch, around epoch 20. It is worth noting that this is unlikely to be an intended benefit of layer freezing, and is likely caused by the slow convergence making it difficult to overfit in earlier epochs.

Figure 16a and 16c show that pretrained ResNet outperforms non-pretrained ResNet in both validation accuracy and F1 score, with the pretrained model attaining a peak F1 score of 0.521 compared to 0.499. In both figures, pretrained ResNet with layer freezing underperformed significantly.

As such, this experiment illustrates that counter to our hypothesis, pretrained models outperform non-pretrained models in this problem. Interestingly, the pretrained model with layer freezing was vastly outperformed, which does align with our hypothesis. As such it can be inferred that pretraining is useful in this problem, as patterns learned from general vision problems likely aid with feature extraction like hand placement detection. Layer freezing likely restricts the learning dynamics of the model, thus leading to the large discrepancy between performance of layer freezing trained pretrained models and the pretrained model.

3.3.2 Evaluating Class Balanced Training

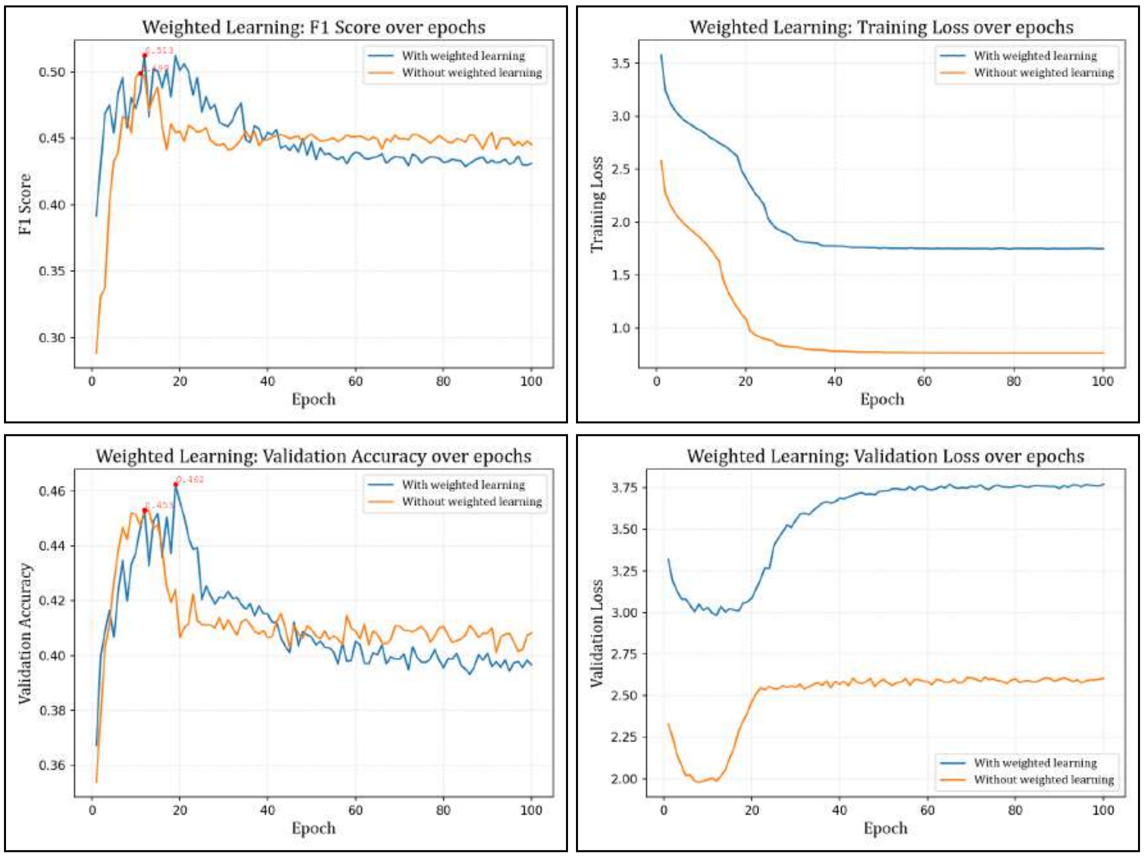


Figure 17a, 17b, 17c, 17d, respectively

This experiment compared two models, one trained with loss weights for class balancing and one trained on unbalanced data. Our methodology for class balanced training was outlined in Section 2.3.3, and it is hypothesized that class balanced training will help mitigate the issue of imbalance dataset.

Figure 17b shows that independent of the presence of class balancing, speed of convergence and thus speed of learning is roughly the same. However, class balanced training does increase the loss value that the model converges too, which could be a concerning indication of poorer depth of learning. When considering Figure 17d, we observe that overfitting occurs around the same epoch, epoch 12, regardless of class balancing. Validation loss for class balanced training remains consistently higher than without class balanced training, which could be concerning as it indicates the model is not performing well and is often getting blind predictions wrong. However, this is also likely due to how class balanced training has a different calculation of loss, that being a weighted calculation. The high training and validation loss could be a reflection of the unbalanced dataset: both models are likely struggling with rarer classes, but for class balanced training this is penalized more with a greater loss than for unweighted training. This is further supported by considering Figures 17a and 17c, where the class balanced trained model outperforms the unbalanced one, with a higher F1 score and validation accuracy.

The class balanced trained model achieves a peak F1 score of 0.513, versus 0.499 for the unbalanced model.

This experiment indicates that our hypothesis holds true—that class balanced training helps improve model performance as it likely partially counters the unbalanced dataset.

3.3.3 Evaluating Hidden Layer

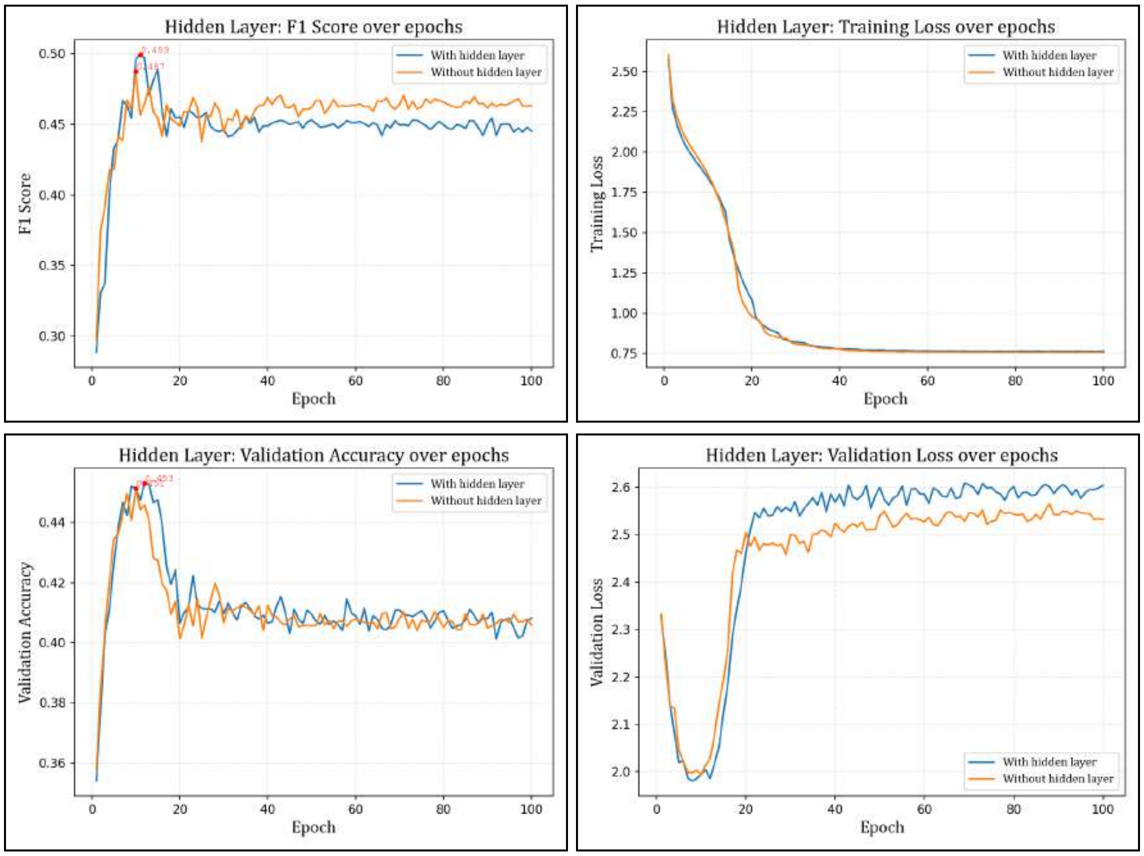


Figure 18a, 18b, 18c, 18d, respectively

The hidden layer, as outlined in Section 2.5.2, aims to add complexity to the feature extraction ability of the network. We ran an experiment to evaluate our hypothesis that adding an hidden layer boosted performance, seen in the above figures where we compare a model with and a model without a hidden layer.

Figure 18b shows that the hidden layer has little impact on training, with the addition of a hidden layer not significantly affecting speed of convergence nor the level of loss convergence occurs at. Figure 18d shows that the hidden layer has a minor impact on overfitting, with overfitting occurring around 2 epochs later when a hidden layer, and validation loss rising to a higher value during overfitting. This can be likely explained as the model with a hidden layer overfitting to a larger degree due to the increased complexity, allowing it to better fully adapt to the training dataset. When we consider figures 18a and 18c, as hypothesized, the hidden layer seems to boost performance. The model with a hidden layer has a higher peak F1 score of

0.499, versus 0.487 for the model without a hidden layer. This increased performance is similarly seen in validation accuracy.

As such, this experiment likely validates our claim about the hidden layer, and showcases how the addition of a hidden layer in an architecture can help boost performance.

3.3.4 Evaluating Dropout Rate

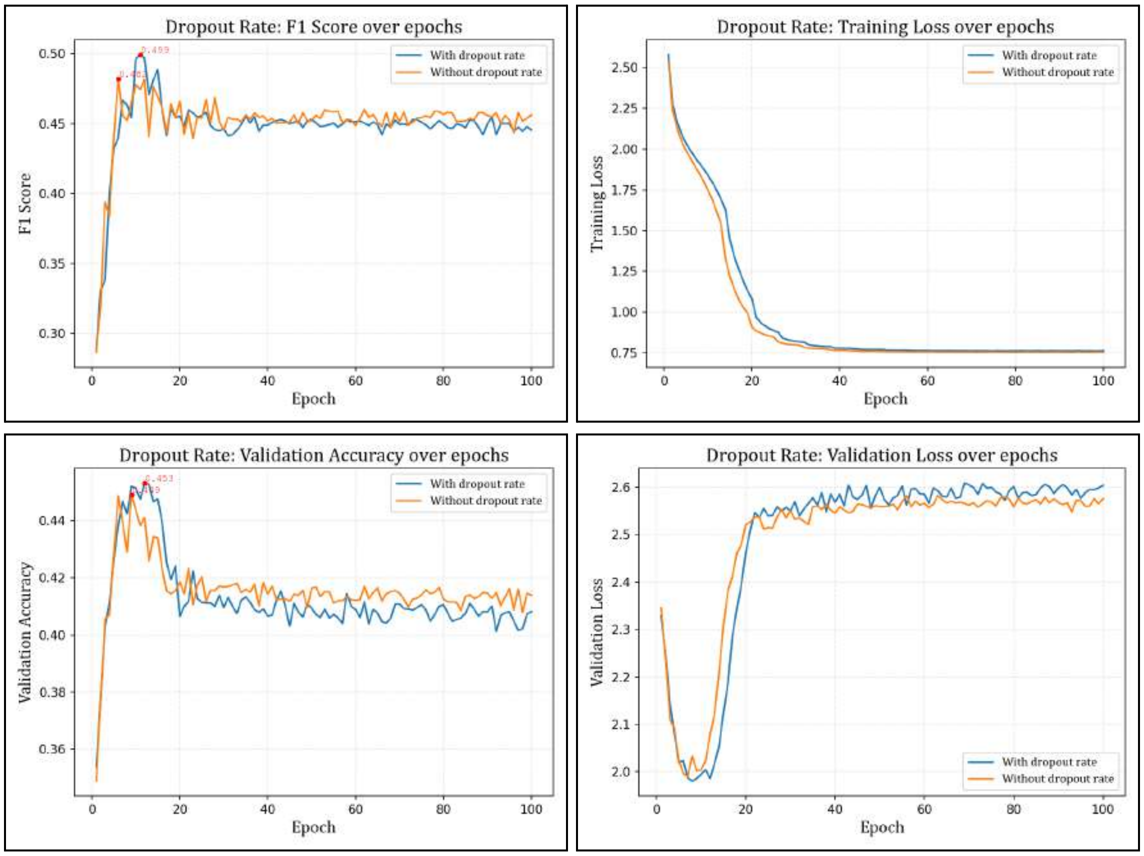


Figure 19a, 19b, 19c, 19d, respectively

Dropout rate, as outlined in Section 2.5.3, aims to improve regularization. This is hypothesized to improve generalization and slow overfitting. We evaluate its usefulness in this experiment, where we consider two models, one with a dropout rate of 0.4, and one without a dropout rate.

Figure 19b shows that the dropout rate had little impact on training, with only a slightly slower convergence to the same level of loss. Figure 19d shows that dropout rate does help slow overfitting, albeit to a small degree. Overfitting occurs only about 3 epochs for the model with a dropout rate. Figures 19a and 19c illustrate a performance improvement with the addition of dropout rate, with a F1 score of 0.499 compared to 0.482.

As such, this experiment corroborates with our hypothesis that dropout rate improves generalization and thus performance.

3.4 Results from training hyperparameters

3.3.1 Evaluating Learning Rate Scheduler

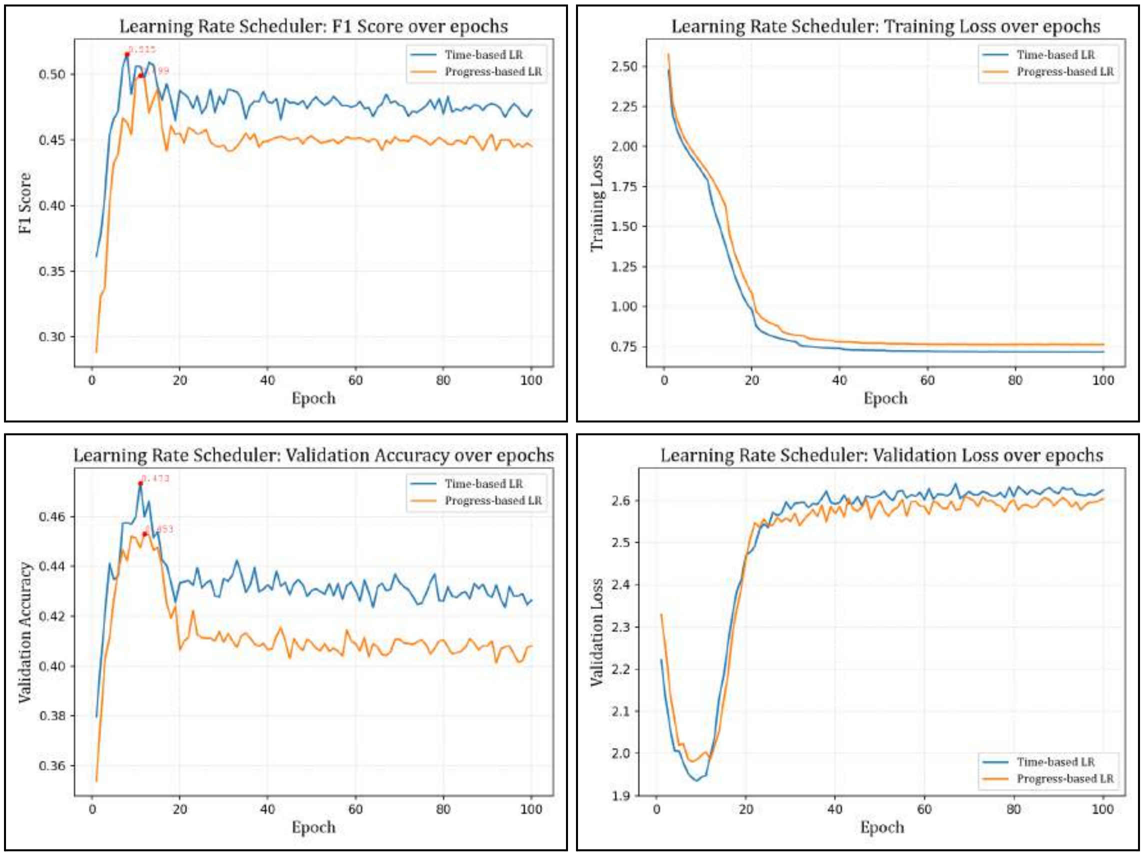


Figure 20a, 20b, 20c, 20d, respectively

This experiment examines two different learning rate schedulers. The methodology was explored in Section 2.6.2. This experiment compares two models, one trained with a time-based learning rate, and the other with a progress-based learning rate. As stated previously, it is hypothesized that progress-based learning rate scheduler will improve training.

Figure 20b shows that a time-based learning rate scheduler allowed for marginally faster convergence to a lower loss level, indicating improved training. Figure 20d shows that independent of learning rate scheduler, overfitting occurred around epoch 10. However, with a time-based learning rate scheduler, there was a higher steady validation loss level reached, which could be indicative of more severe overfitting. Figures 20a and 20c illustrate that time-based learning rate scheduler seems to perform better than a progress-based one, with a F1 score of 0.515 compared to 0.499.

As such, this experiment seemingly contradicts our initial hypothesis. It can be observed from the results that the time-based learning rate scheduler is superior when it comes to model performance. This may be explained by how the progressed-based learning rate scheduler was suboptimally tuned for this experiment. Another possible explanation could be the smooth and deterministic nature of time-based learning rate decay. This thus avoids reliance on noisy validation loss signals and provides a gradual convergence. The progress-based learning rate scheduler could have been susceptive to validation loss fluctuations, leading to suboptimal reductions in learning rate and thus poorer convergence.

3.5 Summary Of Results

The below Figure 21 summarizes the results from all 10 experiments run. Best condition was judged based on F1 score, as outlined in the methodology.

#	Hyperparameter	Best Condition
1	Normalization	[unclear]
2	Data Transformation	Data Transformed
3	Time Context	No Time Context
4	Generous Labelling	Generous Labels
5	Architecture	MobileNetV2_100
6	Pretraining	Pretrained Model
7	Class Balanced Training	Class Balanced Training Present
8	Hidden Layer	Hidden Layer Present
9	Dropout Rate	Dropout Rate Present
10	Learning Rate Scheduler	Time-Based Scheduler

Figure 21

3.6 Best combination of hyperparameters worked the best

Using the results of the 10 experiments, we constructed one final experiment with the best model using a combination of the above hyperparameters. This entails a model trained on a dataset that was min-max normalized, transformed, lacked time context, and was generously labelled. The architecture used was a pretrained MobileNetV2_100, with class balanced training, a hidden layer, a dropout rate of 0.4, using a time-based scheduler. Below are the results, shown in Figures 22a, 22b, 22c and 22d.

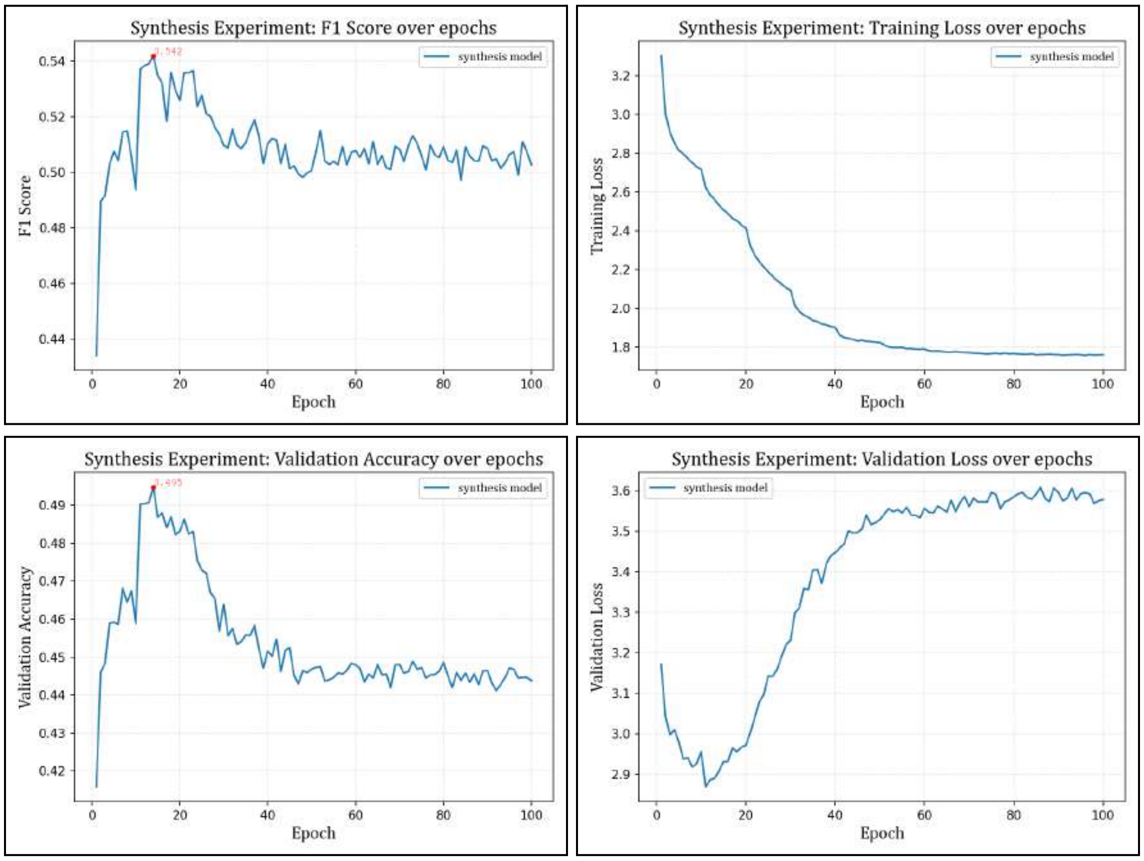


Figure 22a, 22b, 22c, 22d, respectively

This experiment showcases the synthesis model. Model performance is superior to all other training runs conducted, with a peak F1 score of 0.542 and a peak validation accuracy of 0.495, both exceeding all other 26 models. Thus, we can conclude that this combination of the 10 hyperparameters examined produces the most optimal results for this problem and dataset.

4 CONCLUSION

Overall, we were able to accomplish the goal we set for this paper of being able to synthesize the fundamentals of keystroke inference models and construct a baseline model with a reduction in complexity, comprising only the most critical components.

Using only a simple Neural Network, without preprocessing or postprocessing, we were able to produce a suitably generalized model despite training on a limited and flawed dataset. We explored 10 different hyperparameters through 26 experimental training runs, to create a proficient model with a configuration of hyperparameters that were optimized to some extent. Overall, we were able to attain a model with a peak F1 score of 0.542, and a peak validation accuracy of 0.495.

4.1 Implications

These results support our initial hypothesis and demonstrate that keystroke inference remains a viable and replicable attack vector. Importantly, our findings suggest that such attacks could be mounted even by adversaries with limited resources or expertise, underscoring the need for further research into countermeasures and privacy-preserving defenses.

4.2 Potential Countermeasures

We engaged in a cursory review of literature surrounding countermeasures against such attack vectors. We will share them in brief in this section. Several approaches have been proposed [47], with one systemic review collating 55 studies on the matter. In brief, these broadly centered on novel input methods and enhanced authentication interfaces designed to resist observation or recording.

Regarding authentication methods, multiple papers suggested a Graphical Authentication system. This involves incorporating a multi-layer security mechanism, comprising visual elements along with traditional password verification. These visual elements could be color and pattern matching, seen in one paper [48]. Another paper proposed a novel system, PassMatrix [49], which utilized one-time login indicators and dynamically shifting input bars. These systems prevent attackers from keystroke inference even after repeated observations.

Input methods were suggested, such as EyePassword [50] which utilized an on-screen keyboard with input tied to eye pupil movement rather than a physical keyboard. This would thus prevent deduction from passive video capture of physical keyboards. However, we suspect that such a method may be susceptible to a similar attack, that being using passive video capture of the person’s eye movements instead of hand movements.

Overall, while effective to some degree, these countermeasures have not been found to be effective substitutes for traditional password entry. A suggested reason is the inconvenience caused by some of these methods, such as password input from eye-tracking technology which would be foreign to many users. Another impediment is the increased error rates from such password entry or graphical authentication, which can lead to frustration and customer dissatisfaction. There is thus urgent need to research more current and effective methods to counter this form of attack that can and will be rapidly adopted by online services, so as to reduce the feasibility of this attack vector.

4.3 Limitations

Overall, while the results obtained were satisfactory, it is evident that we are unable to reproduce the impressive results seen in other papers in this literature space. We hypothesize this is due, largely, to the major limitation of frame-label mismatch present in our dataset.

The frame-label mismatch caused by the random time delays outlined in Section 2.2.3 are rather severe. The issue led to frames being mislabeled, and we suspect this was the main reason for the model struggling. This is because with mislabeled data, patterns become harder to extract and discern, which greatly affects generalization capabilities.

We did try to address this problem with generous labels, seen in Section 3.1.4. This did improve the model slightly, as more frames where keystrokes actually occurred were captured and labeled. However, this led to the additional problem of increased number mislabeled frames. This can be observed in Figure 23a and 23b below, which illustrates this impact.

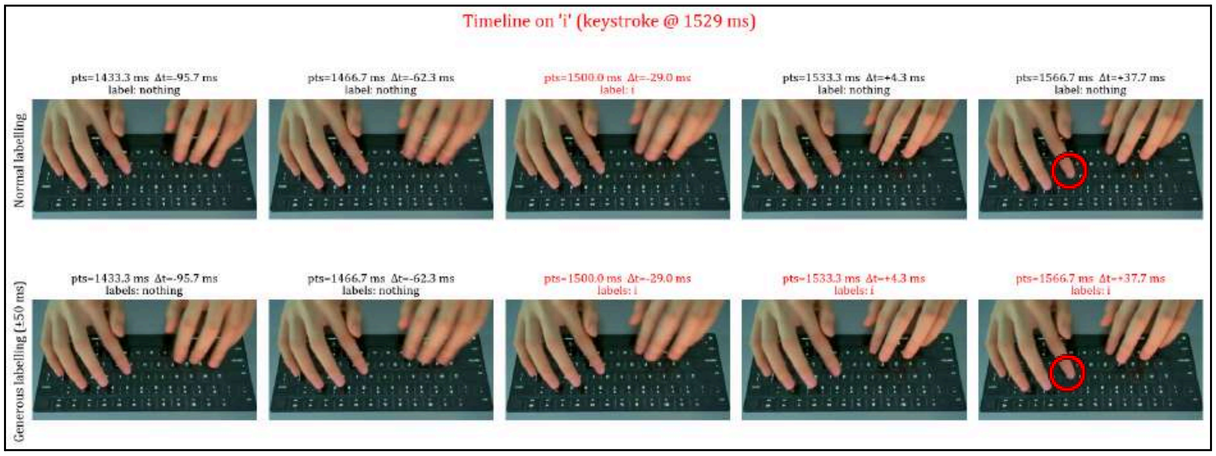

Timeline on 'i' (keystroke @ 1529 ms)
Normal labelling
pts=1433.3 ms Δt=-95.7 ms label: nothing
pts=1466.7 ms Δt=-62.3 ms label: nothing
pts=1500.0 ms Δt=-29.0 ms label: i
pts=1533.3 ms Δt=+4.3 ms label: nothing
pts=1566.7 ms Δt=+37.7 ms label: nothing
Generous labelling (±50 ms)
pts=1433.3 ms Δt=-95.7 ms labels: nothing
pts=1466.7 ms Δt=-62.3 ms labels: nothing
pts=1500.0 ms Δt=-29.0 ms labels: i
pts=1533.3 ms Δt=+4.3 ms labels: i
pts=1566.7 ms Δt=+37.7 ms labels: i

Figure 23a: This showcases a timeline on an ‘i’ keystroke, with timestamp of 1529ms. The first timeline shows normal labelling, while the second shows generous labelling. The red circle shows where, by visual judgment, the actual keystroke occurs.

Figure 23a showcases that generous labelling can correctly label the frame where the actual keystroke occurs, due to the generous temporal window considered. This is compared to normal labelling which mislabels both the center frame, as well as the actual keystroke frame. However, when we consider that generous labelling labels the third and fourth frame as containing keystrokes when they do not, we see that generous labelling also results in two mislabellings. Thus, in this example, generous labelling results in three correct labels and two incorrect labels (counting 'nothing' frames), while normal labelling results in three correct labels and two incorrect labels as well. Thus, there is no discernible improvement.

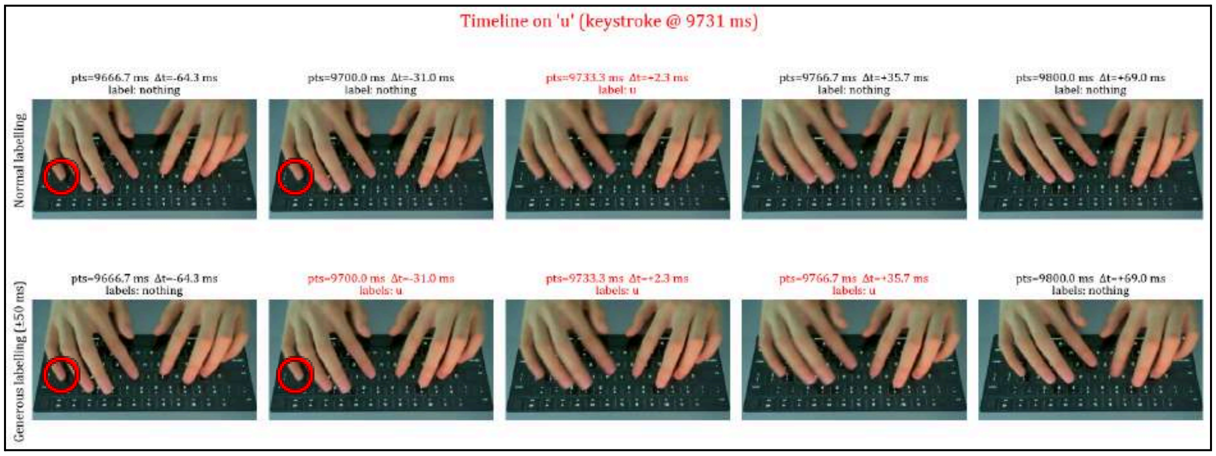

Timeline on 'u' (keystroke @ 9731 ms)
Normal labelling
pts=9666.7 ms Δt=-64.3 ms label: nothing
pts=9700.0 ms Δt=-31.0 ms label: nothing
pts=9733.3 ms Δt=+2.3 ms label: u
pts=9766.7 ms Δt=+35.7 ms label: nothing
pts=9800.0 ms Δt=+69.0 ms label: nothing
Generous labelling (±50 ms)
pts=9666.7 ms Δt=-64.3 ms labels: nothing
pts=9700.0 ms Δt=-31.0 ms labels: u
pts=9733.3 ms Δt=+2.3 ms labels: u
pts=9766.7 ms Δt=+35.7 ms labels: u
pts=9800.0 ms Δt=+69.0 ms labels: nothing

Figure 23b: This showcases a timeline on an 'u' keystroke, with timestamp of 9731ms. The first timeline shows normal labelling, while the second shows generous labelling. The red circles show where, by visual judgment, the actual keystroke could have occurred.

Figure 23b shows a similar issue. Here, generous labelling is also able to correctly label a frame where the actual keystroke occurs. However, when we consider total correct and incorrect labels, we see that generous labelling results in two correct labels and three incorrect labels, while normal labelling results in two correct labels and three incorrect labels as well. Thus, there is also no discernible improvement.

As such, this is a likely reason why generous labels was not a complete solution to the problem. This is thus a possible explanation for why the performance improvement obtained from generous labelling was also not on par with the >95% benchmark set by existing literature.

Furthermore, this raises the concern that the performance boost noticed could also not be due to improved generalization, but instead looser validation criteria as the validation dataset is also generously labelled. This means that if the model guesses the label paired with a mislabeled frame, it will receive a lower loss score despite the label being in fact false. This thus complicates the validity of the F1 score and validation accuracy of generous labelling.

In conclusion, this method to combat this limitation is, itself, severely limited. We could not produce alternative methods, other than a manual relabeling of the dataset which we lacked the time and manpower to conduct.

4.3.1 Further Exploration of Frame-Label mismatch

This limitation does present an interesting point for further analysis, as it could be a common issue that malicious actors will also face when executing such an attack vector. The proposed reasons for this limitation–outlined in Section 2.2.3–likely also apply to any other attack vector involving commercial cameras, unless there is a solution that can be implemented when capturing video. Furthermore, bad actors are unlikely to be able to obtain perfectly frame-label pairs from covert passive video collection.

While other literature likely works around this limitation by having preprocessing, we can assume the average malicious actor lacks such expertise and resources. As such, it is possible that this limitation we faced will also impede their ability to conduct such an attack. If this is the case, it could render this attack vector less feasible.

4.4 Future Direction of Research

We propose two future directions that research can be taken. These were decided upon based on potential future adaptations of this attack vector.

4.4.1 Blind Recognition of Keystrokes

Blind recognition of keystrokes refers to the inference of keystrokes from video of just hand placement. This was something accomplished by some papers mentioned in the literature review, but we would like to revisit the problem with a similar synthesis and minimalist approach as outlined in this paper.

The intention behind such research would be to improve feasibility of the attack vector. Blind keystroke recognition allows for inference without a clear, or any, image of the keyboard. This is more similar to a realistic attack scenario, where capturing a clean overhead shot of the keyboard with keycap labels is unlikely.

If this research is successful, then countermeasures such as obscuring the keyboard from view would become obsolete. This is thus an important future extension of our research.

4.4.2 Keystroke Inference on Mobile Phone Keyboards

Keystroke inference on mobile phone keyboards, while also achieved in current literature [51], is another area that we can apply a similar simplistic methodology to, in order to gain a better understanding of the attack vector as well as what the average malicious actor can achieve.

With the rise of mobile banking, as well as the prevalence of sensitive operations conducted on mobile phones in public, it is worth considering that such keystroke inference may be utilized to attack mobile phone keyboards.

Such a project would also act as a proof of concept for future, more threatening studies. For example, with keystroke inference of mobile keyboards, and the addition of blind inference as mentioned in Section 4.3.1, current privacy methods such as polarized screen filters could be rendered easily bypassable. Such an attack vector would be able to generalize keystrokes based on finger movements alone, leaving mobile phone activity completely vulnerable to such attacks.

5 ACKNOWLEDGEMENT

We, the authors, would like to express sincere gratitude towards Leonard G. for their valuable guidance and constructive feedback throughout this project. We also wish to thank our peers for their insightful discussions, which helped shape the direction of this research.

This work made extensive use of open-source tools, including PyTorch and Hugging Face Datasets, without which this study would not have been possible. We are also grateful to Google's Kaggle platform for providing computational resources that enabled the training and evaluation of our models.

We would like to acknowledge that portions of this work benefited from the use of ChatGPT (OpenAI) for code debugging assistance. All research activities, experimental designs, conceptual contributions, analyses, and final decisions were made solely by the authors.

REFERENCES

[1] J. Jumper *et al.*, “Highly accurate protein structure prediction with AlphaFold,” *Nature*, vol. 596, no. 7873, pp. 583–589, Aug. 2021, doi: 10.1038/s41586-021-03819-2.
[2] A. Merchant, S. Batzner, S. S. Schoenholz, M. Aykol, G. Cheon, and E. D. Cubuk, “Scaling deep learning for materials discovery,” *Nature*, vol. 624, no. 7990, pp. 80–85, Dec. 2023, doi: 10.1038/s41586-023-06735-9.
[3] A. Shehper *et al.*, “What makes math problems hard for reinforcement learning: a case study,” Feb. 11, 2025, *arXiv*: arXiv:2408.15332. doi: 10.48550/arXiv.2408.15332.
[4] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019, doi: 10.1016/j.jcp.2018.10.045.
[5] K. Gandhi, J.-P. Fränken, T. Gerstenberg, and N. D. Goodman, “Understanding Social Reasoning in Language Models with Language Models,” Dec. 04, 2023, *arXiv*: arXiv:2306.15448. doi: 10.48550/arXiv.2306.15448.
[6] “ARC Prize - What is ARC-AGI?,” ARC Prize. Accessed: Sept. 16, 2025. [Online]. Available: https://arcprize.org/arc-agi
[7] D. Rein *et al.*, “GPQA: A Graduate-Level Google-Proof Q&A Benchmark,” Nov. 20, 2023, *arXiv*: arXiv:2311.12022. doi: 10.48550/arXiv.2311.12022.
[8] D. Craigen, N. Diakun-Thibault, and R. Purse, “Defining Cybersecurity,” *Technology Innovation Management Review*, vol. 4, no. 10, pp. 13–21, 2014.
[9] Q. Chen and R. A. Bridges, “Automated Behavioral Analysis of Malware: A Case Study of WannaCry Ransomware,” in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2017, pp. 454–460. doi: 10.1109/ICMLA.2017.0-119.
[10] M. Pollard, “A Case Study of Russian Cyber-Attacks on the Ukrainian Power Grid: Implications and Best Practices for the United States”.
[11] B. Guembe, A. Azeta, S. Misra, V. C. Osamor, L. Fernandez-Sanz, and V. Pospelova, “The Emerging Threat of Ai-driven Cyber Attacks: A Review,” *Applied Artificial Intelligence*, vol. 36, no. 1, p. 2037254, Dec. 2022, doi: 10.1080/08839514.2022.2037254.
[12] S. Armstrong, K. Sotala, and S. S. Ó hÉigeartaigh, “The errors, insights and lessons of famous AI predictions – and what they mean for the future,” *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 26, no. 3, pp. 317–342, July 2014, doi: 10.1080/0952813X.2014.895105.
[13] S. Armstrong and K. Sotala, “How We’re Predicting AI – or Failing to,” in *Beyond Artificial Intelligence*, vol. 9, J. Romportl, E. Zackova, and J. Kelemen, Eds., in Topics in Intelligent Engineering and Informatics, vol. 9. , Cham: Springer International Publishing, 2015, pp. 11–29. doi: 10.1007/978-3-319-09668-1_2.
[14] R. West and R. Aydin, “The AI Alignment Paradox,” Nov. 22, 2024, *arXiv*: arXiv:2405.20806. doi: 10.48550/arXiv.2405.20806.
[15] G. Waizel, “Bridging the AI divide: The evolving arms race between AI- driven cyber attacks and AI-powered cybersecurity defenses,” *International Conference on Machine Intelligence & Security for Smart Cities (TRUST) Proceedings*, vol. 1, pp. 141–156, July 2024.
[16] A. K. Jain, J. Mao, and K. M. Mohiuddin, “Artificial neural networks: a tutorial,” *Computer*, vol. 29, no. 3, pp. 31–44, Mar. 1996, doi: 10.1109/2.485891.
[17] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer Nature, 2022.

[18] F.-X. Standaert, “Introduction to Side-Channel Attacks,” in *Secure Integrated Circuits and Systems*, I. M. R. Verbauwhede, Ed., in Integrated Circuits and Systems. , Boston, MA: Springer US, 2010, pp. 27–42. doi: 10.1007/978-0-387-71829-3_2.
[19] L. Zhuang, F. Zhou, and J. D. Tygar, “Keyboard acoustic emanations revisited,” *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, p. 3:1-3:26, Nov. 2009, doi: 10.1145/1609956.1609959.
[20] M. Li *et al.*, “When CSI Meets Public WiFi: Inferring Your Mobile Phone Password via WiFi Signals,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, in CCS ’16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 1068–1079. doi: 10.1145/2976749.2978397.
[21] S. Saab, A. Leiserson, and M. Tunstall, “Key Extraction from the Primary Side of a Switched-Mode Power Supply,” 2015, 2015/512. Accessed: Sept. 16, 2025. [Online]. Available: https://eprint.iacr.org/2015/512
[22] A. Amrouche, L. Boubchir, and S. Yahiaoui, “Side Channel Attack using Machine Learning,” in *2022 Ninth International Conference on Software Defined Systems (SDS)*, Dec. 2022, pp. 1–5. doi: 10.1109/SDS57574.2022.10062906.
[23] O. I. Abiodun *et al.*, “Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition,” *IEEE Access*, vol. 7, pp. 158820–158846, 2019, doi: 10.1109/ACCESS.2019.2945545.
[24] O. Eluyode, “Scholars Research Library Comparative study of biological and artificial neural networks”, Accessed: Sept. 16, 2025. [Online]. Available: https://www.academia.edu/7938549/Scholars_Research_Library_Comparative_study_of_biological_and_artificial_neural_networks
[25] “Gradient-based learning applied to document recognition | IEEE Journals & Magazine | IEEE Xplore.” Accessed: Sept. 16, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/726791
[26] “d41586-025-01965-5,” *Nature*, no. 634, pp. 839, 840, June 2025.
[27] J. Lim, J.-M. Frahm, and F. Monrose, “Leveraging Disentangled Representations to Improve Vision-Based Keystroke Inference Attacks Under Low Data,” Apr. 05, 2022, *arXiv*: arXiv:2204.02494. doi: 10.48550/arXiv.2204.02494.
[28] D. Balzarotti, M. Cova, and G. Vigna, “ClearShot: Eavesdropping on Keyboard Input from Video,” in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, Oakland, CA, USA: IEEE, May 2008, pp. 170–183. doi: 10.1109/SP.2008.28.
[29] Z. Yang, Y. Chen, Z. Sarwar, and H. Schwartz, “Towards a General Video-based Keystroke Inference Attack”.
[30] S. E. Whang and J.-G. Lee, “Data collection and quality challenges for deep learning,” *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3429–3432, Aug. 2020, doi: 10.14778/3415478.3415562.
[31] S. E. Whang, Y. Roh, H. Song, and J.-G. Lee, “Data collection and quality challenges in deep learning: a data-centric AI perspective,” *The VLDB Journal*, vol. 32, no. 4, pp. 791–813, July 2023, doi: 10.1007/s00778-022-00775-9.
[32] “Recognizing 50 human action categories of web videos | Request PDF,” *ResearchGate*, Aug. 2025, doi: 10.1007/s00138-012-0450-4.
[33] H. Hajimolahoseini, W. Ahmed, A. Wen, and Y. Liu, “Is 3D Convolution with 5D Tensors Really Necessary for Video Analysis?,” July 23, 2024, *arXiv*: arXiv:2407.16514. doi: 10.48550/arXiv.2407.16514.

[34] E. Kloberdanz, K. G. Kloberdanz, and W. Le, "DeepStability: a study of unstable numerical methods and their solutions in deep learning," in *Proceedings of the 44th International Conference on Software Engineering*, Pittsburgh Pennsylvania: ACM, May 2022, pp. 586–597. doi: 10.1145/3510003.3510095.
[35] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," Sept. 11, 2020, *arXiv*: arXiv:1905.11946. doi: 10.48550/arXiv.1905.11946.
[36] H. Ali, N. Shifa, R. Benlamri, A. A. Farooque, and R. Yaqub, "A fine tuned EfficientNet-B0 convolutional neural network for accurate and efficient classification of apple leaf diseases," *Sci Rep*, vol. 15, no. 1, p. 25732, July 2025, doi: 10.1038/s41598-025-04479-2.
[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 10, 2015, *arXiv*: arXiv:1512.03385. doi: 10.48550/arXiv.1512.03385.
[38] T.-B. Xu, P. Yang, X.-Y. Zhang, and C.-L. Liu, "LightweightNet: Toward fast and lightweight convolutional neural networks via architecture distillation," *Pattern Recognition*, vol. 88, pp. 272–284, Apr. 2019, doi: 10.1016/j.patcog.2018.10.029.
[39] "resnet18 — Torchvision main documentation." Accessed: Sept. 16, 2025. [Online]. Available: https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html?utm_source=chatgpt.com
[40] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Mar. 21, 2019, *arXiv*: arXiv:1801.04381. doi: 10.48550/arXiv.1801.04381.
[41] C. Luo, X. He, J. Zhan, L. Wang, W. Gao, and J. Dai, "Comparison and Benchmarking of AI Models and Frameworks on Mobile Devices," May 07, 2020, *arXiv*: arXiv:2005.05085. doi: 10.48550/arXiv.2005.05085.
[42] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," Mar. 02, 2022, *arXiv*: arXiv:2201.03545. doi: 10.48550/arXiv.2201.03545.
[43] F. Wang *et al.*, "E-ConvNeXt: A Lightweight and Efficient ConvNeXt Variant with Cross-Stage Partial Connections," Aug. 28, 2025, *arXiv*: arXiv:2508.20955. doi: 10.48550/arXiv.2508.20955.
[44] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, doi: 10.1037/h0042519.
[45] "ImageNet." Accessed: Sept. 16, 2025. [Online]. Available: https://www.image-net.org/download.php
[46] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 30, 2017, *arXiv*: arXiv:1412.6980. doi: 10.48550/arXiv.1412.6980.
[47] F. Binbeshr, M. L. Mat Kiah, L. Y. Por, and A. A. Zaidan, "A systematic review of PIN-entry methods resistant to shoulder-surfing attacks," *Computers & Security*, vol. 101, p. 102116, Feb. 2021, doi: 10.1016/j.cose.2020.102116.
[48] J. I. D, R. V, T. K. P, A. Iyer, and N. M. S, "Resisting Visual Hacking: A Novel Graphical Password Authentication System," in *2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN)*, June 2023, pp. 910–915. doi: 10.1109/ICPCSN58827.2023.00155.
[49] H.-M. Sun, S.-T. Chen, J.-H. Yeh, and C.-Y. Cheng, "A Shoulder Surfing Resistant Graphical Authentication System," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 2, pp. 180–193, Mar. 2018, doi: 10.1109/TDSC.2016.2539942.

[50] M. Kumar, T. Garfinkel, D. Boneh, and T. Winograd, “Reducing shoulder-surfing by using gaze-based password entry,” in *Proceedings of the 3rd symposium on Usable privacy and security*, Pittsburgh Pennsylvania USA: ACM, July 2007, pp. 13–19. doi: 10.1145/1280680.1280683.
[51] Q. Yue, Z. Ling, X. Fu, B. Liu, K. Ren, and W. Zhao, “Blind Recognition of Touched Keys on Mobile Devices,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, Scottsdale Arizona USA: ACM, Nov. 2014, pp. 1403–1414. doi: 10.1145/2660267.2660288.