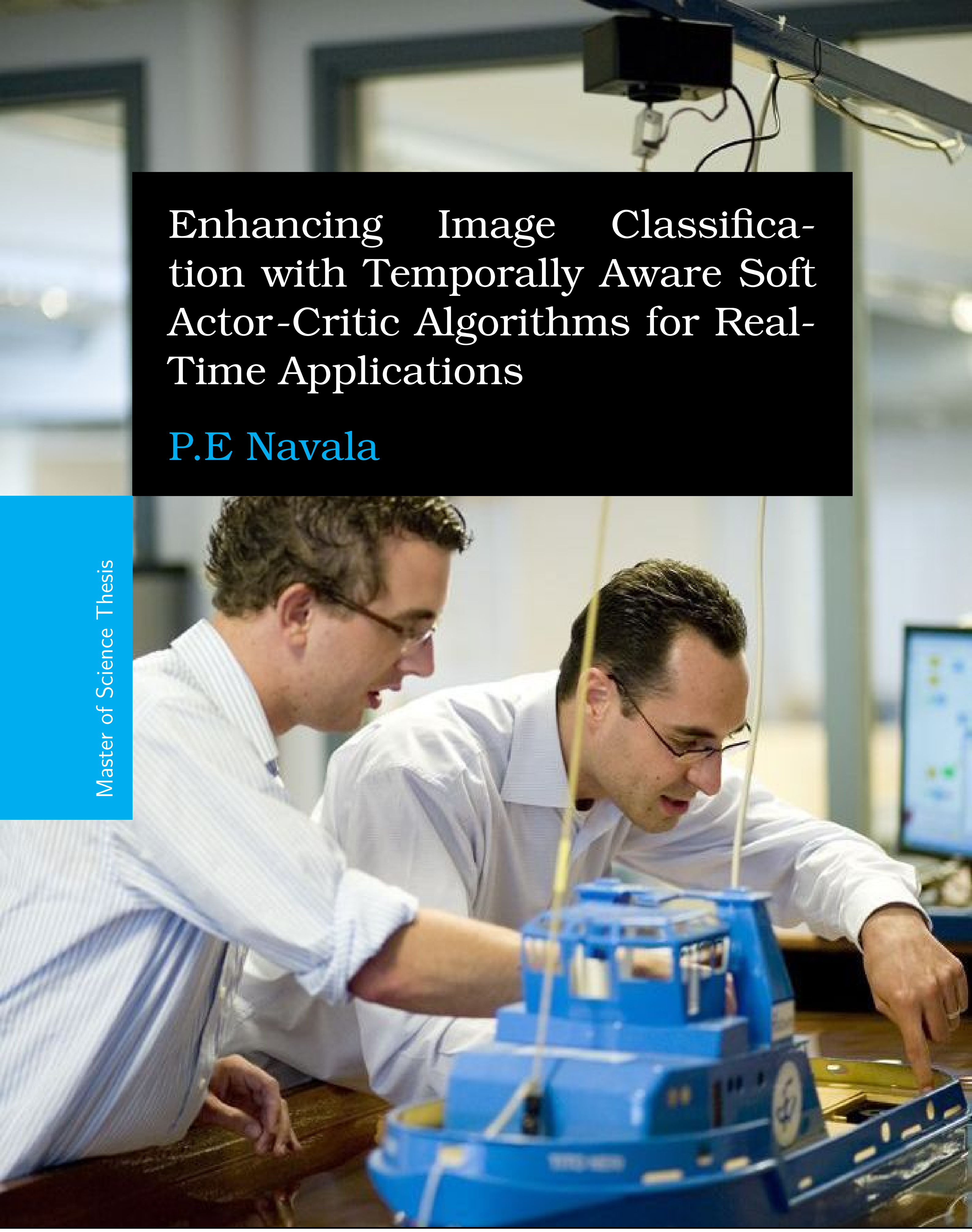


Enhancing Image Classification with Temporally Aware Soft Actor-Critic Algorithms for Real-Time Applications

P.E Navala

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Enhancing Image Classification with Temporally Aware Soft Actor-Critic Algorithms for Real-Time Applications

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

P.E Navala

August 18, 2024

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

The thesis presents a novel approach to optimizing input computation for minimizing classification error in image classification tasks. It leverages the capabilities of the Soft Actor-Critic (SAC) algorithm, a reinforcement learning method tailored for continuous action spaces. The focus is on developing a real-time adaptable feedback loop that continuously learns and adjusts inputs based on classifier output probabilities. Key to this approach is the incorporation of a Gated Recurrent Unit (GRU) architecture within the SAC framework to capture temporal dependencies, addressing the challenge of ever-increasing state dimensions.

Table of Contents

1 Introduction 1

2 Image Classification with Transfer Learning 5

1 Dataset . . . 5
- 1-1 MNIST Dataset . . . 5
- 1-2 Fashion MNIST Dataset . . . 6

2 Imaging model . . . 8
- 2-1 Illumination using Fourier Coefficients . . . 8
- 2-2 Poisson Noise . . . 10

3 Problem formulation . . . 12

4 Classifier model . . . 12

3 Computing the next input with a Soft Actor-Critic algorithm 15

1 Reinforcement Learning . . . 15

2 State Formulation . . . 16
- 2-1 State Dimensionality Problem . . . 17
- 2-2 Novel approach and Research Question . . . 18

3 Gated Recurrent Unit (GRU) . . . 18

4 Soft Actor-Critic . . . 19
- 4-1 MDP . . . 21
- 4-2 Policy (actor) . . . 22
- 4-3 Value functions . . . 22
- 4-4 Bellman Equation . . . 23
- 4-5 Q-function . . . 24
- 4-6 Reward Function . . . 25

5 Soft Actor-Critic (SAC) Training . . . 26

6 Gated Recurrent Unit (GRU) Training . . . 29

4 Results **31**
1 Testing and Validation Scenario 31
2 GRU vs non-GRU approach 32
3 State-of-the-art vs SAC with/without GRU 33
4 Conclusion 35

Bibliography **37**

Chapter 1

Introduction

Motivation and Social Impact

Image recognition, a subfield of artificial intelligence (AI) and computer vision, has emerged as a transformative technology with significant implications for various sectors of society. The importance of image recognition lies in its ability to analyze, interpret, and understand visual data, which constitutes a substantial portion of the information humans process daily. This technology leverages advanced algorithms, machine learning techniques, and deep learning models to classify and identify objects, patterns, and features within images, thereby enabling machines to perceive and respond to visual stimuli in ways previously reserved for human cognition.

One of the primary societal benefits of image recognition is its application in healthcare. Medical image analysis, for instance, has revolutionized diagnostics by providing more accurate and faster interpretations of medical scans such as X-rays, MRIs, and CT scans. Automated systems can detect anomalies and diseases, such as tumors and fractures, at an early stage, thereby enhancing the precision of diagnoses and improving patient outcomes.[1] This technology not only augments the capabilities of healthcare professionals but also mitigates the risk of human error and reduces the time required for analysis.[2]

In the realm of security, image recognition has become a cornerstone for surveillance and crime prevention. Facial recognition systems [3] are widely deployed in public and private spaces to enhance security measures. These systems can identify and track individuals in real-time, aiding law enforcement agencies in monitoring suspicious activities and apprehending criminals. Moreover, image recognition contributes to the development of advanced biometric systems, which are essential for secure access control and authentication processes in various applications, from unlocking smartphones to accessing restricted areas.[4]

The transportation sector has also benefited significantly from image recognition technologies. Autonomous vehicles rely heavily on image recognition to navigate and interpret their surroundings.[5] By recognizing traffic signs, signals, pedestrians, and other vehicles, these systems enable self-driving cars to make informed decisions, thereby enhancing road safety and reducing the likelihood of accidents caused by human error. Additionally, traffic management systems use image recognition to monitor traffic flow, detect congestion, and manage incidents, leading to more efficient and safer urban mobility.

Retail and e-commerce industries utilize image recognition to enhance customer experiences and streamline operations. Visual search engines allow customers to search for products using images rather than text, improving the ease and accuracy of finding desired items. Furthermore, image recognition enables automated inventory management by tracking stock levels and identifying misplaced items, thereby reducing operational costs and minimizing human labor.

Agriculture has seen a profound impact through the application of image recognition in precision farming.[6] By analyzing aerial images captured by drones or satellites, farmers can monitor crop health, detect pest infestations, and assess soil conditions. This data-driven approach enables more efficient use of resources, such as water and fertilizers, and helps in making informed decisions to increase crop yields and sustainability.

In addition to these practical applications, image recognition is crucial in enhancing accessibility for individuals with disabilities. For instance, visually impaired individuals can use assistive technologies powered by image recognition to interpret their surroundings, read text, and identify objects. This empowerment fosters greater independence and improves the quality of life for people with disabilities.

Image recognition is also making significant strides in the field of fault diagnosis, particularly in industrial and manufacturing settings.[7] Fault diagnosis involves identifying and diagnosing defects or malfunctions in machinery and equipment, which is critical for maintaining operational efficiency and preventing costly downtime. Traditional fault diagnosis methods often rely on manual inspections and measurements, which can be time-consuming, labor-intensive, and prone to human error. Image recognition technology, however, offers a powerful alternative by enabling automated and precise detection of faults.

In industrial applications, image recognition systems can analyze images and videos of machinery to detect anomalies such as cracks, wear, misalignment, and other defects.[8] For example, thermal imaging cameras combined with image recognition algorithms can identify overheating components in electrical systems or mechanical equipment, signaling potential failures before they occur. Similarly, in the production lines of various manufacturing industries, high-resolution cameras and image processing software can inspect products for defects, ensuring quality control and reducing the rate of defective products reaching consumers. This automation enhances reliability and accuracy and improves overall productivity by allowing continuous monitoring without the need for human intervention.

The semiconductor industry, which is foundational to modern electronics, has also benefited tremendously from advancements in image recognition technology. Semiconductor manufacturing involves extremely precise processes at microscopic scales, where even the smallest defect can render a chip unusable. Traditional inspection methods are inadequate for detecting minute defects at the necessary speed and scale. Image recognition systems, however, are capable of examining wafers and chips with high precision and speed, identifying defects such as pattern discrepancies, particle contamination, and structural abnormalities.

In semiconductor fabrication, image recognition is integrated into various stages of the production process. During the photolithography stage, for instance, image recognition algorithms analyze patterns on wafers to ensure they match the intended design without deviations.[9] Similarly, in the etching and deposition stages, these systems monitor the surface of the wafers to detect any irregularities. By automating these inspections, semiconductor manufacturers can achieve higher yields and better quality control, ultimately leading to more reliable and efficient electronic devices.

Moreover, image recognition technology in semiconductor manufacturing is not limited to defect detection. It also plays a crucial role in process optimization and equipment maintenance. By continuously monitoring equipment performance and identifying wear and tear through image analysis, manufacturers can implement predictive maintenance strategies, reducing unexpected downtime and extending the lifespan of critical machinery. This holistic approach to quality assurance and maintenance underscores the transformative impact of image recognition technology in the semiconductor industry, driving innovation and efficiency in the production of the building blocks of modern technology.

Thus, image recognition is a pivotal technology that addresses a myriad of societal needs by enhancing capabilities in healthcare, security, transportation, retail, agriculture, and accessibility. Its ability to interpret and analyze visual data opens new horizons for innovation and efficiency, making it an indispensable tool in the modern world. As the technology continues to evolve, ongoing research and ethical considerations will be critical to harness its full potential and mitigate associated risks.

Current state-of-the-art for Closed-Loop Active Model Diagnosis

This paradigm shift towards intelligent data acquisition has been effectively implemented in various applications through the methodology known as Closed-Loop Active Model Diagnosis (CLAMD) developed by Noom [10]. This approach dynamically adjusts the imaging system's inputs after each measurement to ensure rapid and accurate diagnosis. Although the theories of active fault diagnosis and auxiliary signal design, which underpin CLAMD, have not yet seen widespread practical application, they hold substantial promise for enhancing efficiency in imaging sys- tems. In the referenced paper, CLAMD is employed for high-performance object recognition with a focus on minimizing phototoxicity. The system architecture includes a neural network cluster tasked with diagnosing objects with a specified level of confidence and a controller designed to determine subsequent, minimally invasive yet effective illumination inputs. A significant portion of the computational processes involved can be executed offline, enabling swift online operational capabilities. The author demonstrates the effectiveness of this approach through simulation experiments using the MNIST handwritten digit dataset, comparing the results with those from a traditional open-loop approach.

Determination of Closed-Loop Inputs

The method will be described briefly using the notation introduced in the paper. The reader is encouraged to read the paper for further clarifications.[10]

The images are divided into n^2 subimages with a uniform input $u_{k,\ell}$ applied to each. For each subimage, s distinct neural networks are trained with varying prospective inputs. The relative probabilities of the neural network outputs are then calculated using the respective PDFs, which were established through Gaussian kernel density estimates during the network training phase. These probabilities are linearly interpolated according to the actual illumination input used to generate y_k. The Bayesian update rule is applied successively for each hypothesis to integrate the results from all subimages:

The probability of misdiagnosis for each input configuration is bounded by:

$$Pe(u_k) \leq \sum_i \sum_{j>i} \sqrt{P_{k-1}(M_i)P_{k-1}(M_j)}B_{ij}(u_k), \tag{1-1}$$

where $B_{ij}(u_k)$ is the Bhattacharyya coefficient defined as:

$$B_{ij}(u_k) = \int \sqrt{p(y_k|M_i, u_k)p(y_k|Mj, u_k)}dy_k, \tag{1-2}$$

An affine approximation of the Bhattacharyya coefficient, obtained through a least-squares fit, is used to simplify the computation of input distributions that minimize the error probability:

$$\hat{B}_{ij,\ell}(u_{k,\ell}) = 1 + a_{ij,\ell}u_{k,\ell}, \tag{1-3}$$

The decision on the input $u_{k,\ell}$ considers the total error probability, with each subimage's input determined by the magnitude of the distinction between models, measured by the derived coefficients from the least-squares fit:

$$u_{k,\ell} = \frac{\sqrt{b_{k,\ell}}\epsilon}{\sum_{i=1}^{n^2} \sqrt{b_{k,i}}} \tag{1-4}$$

It is concluded in the paper that this approach ensures efficient and computationally feasible real-time operations, minimizing the probability of misdiagnosis while adhering to energy constraints.

Addressing Limitations

The CLAMD system, while demonstrating significant advances in optimizing diagnostic processes under illumination constraints, encounters several notable limitations. These limitations not only challenge the system's operational efficiency but also its scalability and practical applicability in real-time scenarios. The three notable limitations are the high input dimensionality, the discrete nature of the illumination input, and the challenges of real-time application.

Image classification is a fundamental task in computer vision with traditional methods using convolutional neural networks (CNNs) trained on large datasets. Transfer learning leverages pre-trained models to improve performance on smaller datasets. However, adjusting input images can further enhance classification accuracy. The Thesis explores the combination of SAC, a reinforcement learning algorithm, with GRU and adjusting Illumination using Fourier coefficients to optimize the classification performance of a pre-trained MobileNetV2 model on the Fashion MNIST dataset.

Chapter 2

Image Classification with Transfer Learning

Transfer learning is a machine learning technique where a model developed for a specific task is reused as the starting point for a model on a second task. It leverages pre-trained models, which have learned a wide variety of features from a large dataset, to improve performance on a new, often smaller, dataset. This approach is beneficial when the new dataset is small or training from scratch is computationally expensive.

1 Dataset

1-1 MNIST Dataset

The dataset of images chosen by Noom to be used for exemplification was the MNIST dataset. The MNIST dataset is a collection of grayscale images (28x28) of handwritten digits from 0 to 9. In addition to the image, a label containing the digit number of the respective image is available. Due to its simplicity and accessibility, it has been widely used as a benchmark for image classification algorithms.

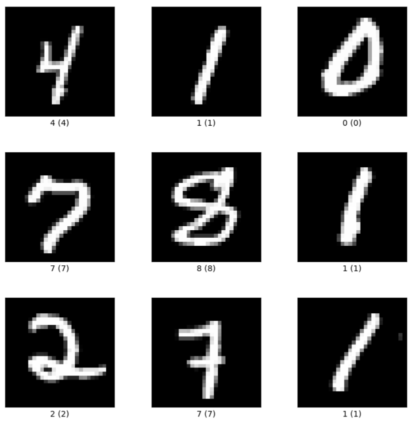


Figure 2-1: MNIST dataset example

However, a more complex dataset was chosen, to both showcase the performance benefits of the proposed approach and to expand on the results of Noom. The chosen dataset is Fashion MNIST.

1-2 Fashion MNIST Dataset

The Fashion MNIST dataset is a collection of grayscale images of various fashion items. It was created as a more challenging drop-in replacement for the MNIST dataset. Each image in Fashion MNIST is 28x28 pixels, with 10 different classes representing different types of clothing and accessories. Similarly to MNIST, a label containing the item category displayed in each image is available. These categories are: 0 - T-shirt/top, 1 - Trouser, 2 - Pullover, 3 - Dress, 4 - Coat, 5 - Sandal, 6 - Shirt, 7 - Sneaker, 8 - Bag, 9 - Ankle boot. An example of an image belonging to this dataset can be seen in Figure 2-2.

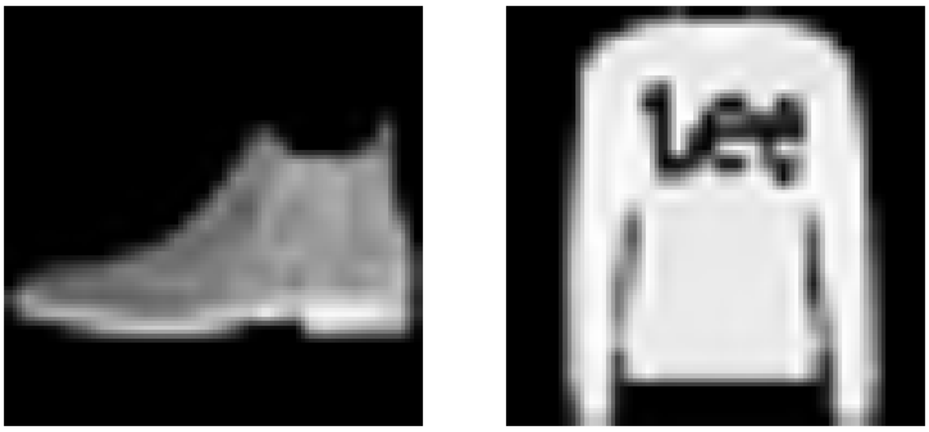


Figure 2-2: Fashion MNIST dataset example showcasing an item of category 9 - Ankle boot and 2 - Pullover

Fashion MNIST images exhibit higher visual complexity due to the textures, patterns, and varying shapes of clothing items. This complexity better mimics the challenges faced in real-world image recognition tasks. The dataset includes items with different silhouettes and sizes, adding another layer of difficulty compared to the uniform shape of digits in MNIST. Furthermore, some classes in Fashion MNIST, such as T-shirts and shirts or sandals and sneakers, have subtle differences that make them harder to distinguish. This encourages the development of models that are more sensitive to fine-grained visual distinctions. Moreover, each class has significant variation due to different styles, shapes, and orientations of the clothing items.

For the rest of the Thesis, Fashion MNIST will be used to illustrate the applicability and the implementation of each component.

2 Imaging model

The image captured by the camera at time step k denoted as I_k is formed by pointwise multiplying the object matrix O with the illumination matrix at time step k, denoted as L_k. In the context of using the Fashion MNIST dataset, the object matrix O is an image from this dataset containing the current object under test. The dimensions of matrix O are then, in this case, 28x28, with each entry representing a pixel with a value between [0,255]. The illumination matrix L_k is a matrix of equal dimensions to O, 28x28 in this case, and contains the coefficients by which each pixel of the original image will be adjusted through illumination. The computation of these coefficients and illumination patterns will be detailed in the coming sections. Additionally, an additive Poisson noise term w_k was added to the model. The dimension of w_k is equal to the object, in this case 28x28. The method by which the noise is computed for each individual pixel is detailed further in this chapter.

As such the imaging model can be formulated as:

$$I_k = O \cdot L_k + w_k, \tag{2-1}$$

where I_k is the image at time step k, O is the object under test, L_k is the illumination vector at time step k and w_k is the noise at time step k.

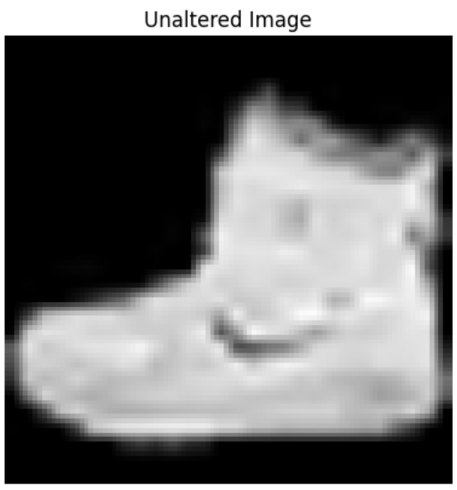


Figure 2-3: Example of object image O of class 9 - Ankle boot from the Fashion MNIST dataset

2-1 Illumination using Fourier Coefficients

The illumination adjustment process modifies the image's brightness and contrast by multiplying an illumination coefficient with the original pixel value. These coefficients are computed in the frequency domain and then transformed in the spatial domain using a Fourier transform before being applied. This technique leverages the Fourier transform's ability to decompose an image into its constituent frequencies, which can then be adjusted to achieve desired visual effects.

A frequency matrix F_k must be defined in order to create the illumination pattern for the 2D image. The frequency matrix will have the dimension of L_k, in this case, 28x28, with the DC component as the first entry in the top left corner. Equation 2-2 showcases F_k for a 28x28 image.

$$F_k = \begin{bmatrix} DC & f_1 & \dots & f_{27} \\ f_1 & \ddots & \ddots & f_{1,27} \\ \vdots & \ddots & \ddots & \vdots \\ f_{27} & \ddots & \ddots & f_{27,27} \end{bmatrix} \tag{2-2}$$

In the case of the implementation presented in this thesis, the first 8 frequency components on both vertical and horizontal directions will be the ones adjusted, thus effectively modifying the top left 8x8 block of F_k.

Consider the first 64 components (8x8) of the frequency matrix, which contains the low-frequency components. Let's define $\delta(u, v)$ as the delta pulse in the frequency domain. The aim is to create a matrix $\Delta(u, v)$ that modifies only the first 8×8 frequency components:

$$\Delta(u, v) = \begin{cases} \alpha(u, v) & \text{for } 1 \leq u < 8, 1 \leq v < 8 \\ 0 & \text{otherwise} \end{cases} \tag{2-3}$$

Here, $\alpha(u, v)$ represents the amplitude of the delta pulses, which differ for each frequency component.

Next, the resulting matrix $\Delta(u, v)$ is then pointwise multiplied with F_k and passed through an Inverse Fourier Transform operation in order to get the spatial domain matrix L_k.

$$L_k = |\frac{1}{MN} \sum_{u=1}^{M-1} \sum_{v=1}^{N-1} \Delta_k(u, v) \cdot F_k|^2, \tag{2-4}$$

where M and N are the vertical and horizontal image dimension, in this case (28x28) The resulting L_k is then pointwise multiplied with the object O_k to form the noiseless illuminated image x_k, an example of the resulting image is showcased in Figure 2-4.

$$x_k = O_k \cdot L_k \tag{2-5}$$

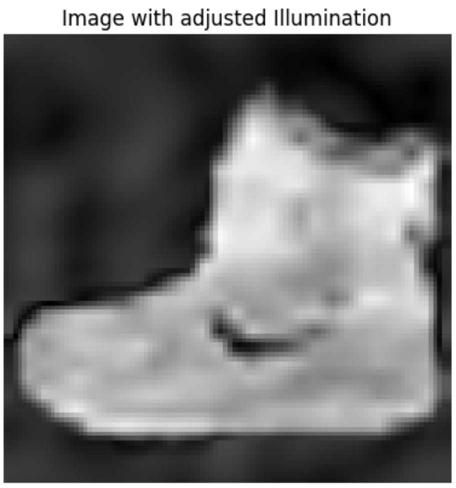


Figure 2-4: Example of Noiseless Illuminated image x_k of class 9 - Ankle boot from the Fashion MNIST dataset

This approach can be extended to multi-channel images (e.g RGB), by applying the same method to each channel individually. Adjusting illumination using Fourier coefficients for each RGB channel allows for controlled and precise manipulation of an image's frequency components, which directly affects its visual appearance. By modifying the first 64 low-frequency components independently for each channel, the overall brightness, contrast, and color balance of the image can be adjusted, potentially enhancing features relevant for classification. This method provides a flexible and powerful approach to image preprocessing in the context of computer vision and can prove beneficial for the classifier in making an accurate classification.

2-2 Poisson Noise

Poisson noise models the acquisition of photons on a photosite, such as a camera sensor. The camera converts incoming photons into electrons with a certain quantum efficiency, and this process occurs over a given exposure time. The accumulated electrons are then converted to an output voltage, which is subsequently quantized. The maximum value of this quantized output corresponds to the camera's specific maximum voltage. Thus, the number of photons is random and depends on the illumination.

For practical purposes, the Fashion MNIST dataset was considered to be a low-intensity image and, thus a value of a 100 electrons for the highest pixel intensity (255) was chosen.

$$\lambda \in [0, 100] \tag{2-6}$$

The noiseless illuminated image x_k can be defined as

$$x_k = O \cdot L_k \tag{2-7}$$

The Poisson distribution $\mathcal{P}(\lambda)$ writes

$$p(a) = \frac{\lambda^a}{a!} e^{-\lambda} \tag{2-8}$$

The mean and variance of the Poisson distribution are both equal λ, which depends on the number of incident photons. So, the noise w depends on the noiseless image x_k.

The corresponding Poisson process has a mean equals to the illumination. The intensity of each pixel (m, n) of the observation I_k is:

$$\forall m, n \quad I_k(m, n) \sim p(x_k(m, n)) \tag{2-9}$$

An example of a noisy image is shown in the Figure 2-5.

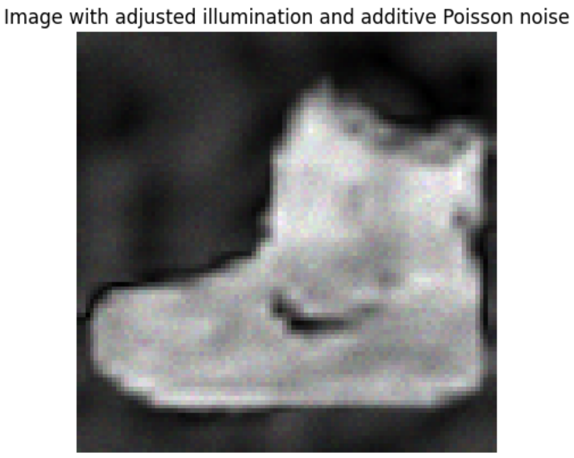


Figure 2-5: Example of Illuminated image with additive Poisson noise, I_k, of class 9 - Ankle boot from the Fashion MNIST dataset

The resulting noisy image I_k simulates the output of the real-life imaging system and will constitute the input to the classifier.

3 Problem formulation

The goal is to maximize the classification performance of an object of a certain class by applying the optimal illumination to the image. This is achieved through a feedback loop, which runs the same object through the classifier multiple times with different illuminations, until the classification performance converges, i.e the classification probability, which indicates how confident the classifer is about the predicted class, is maximized. As a constraint, the total illumination power has to be kept under a certain threshold.

$$\max||\mathbf{P_k}||_\infty \tag{2-10}$$
$$\text{s.t } ||I_k||_2 \leq \epsilon_i, \tag{2-11}$$

where $\mathbf{P_k}$ is the probability vector at time step k and ϵ_i is the illumination power threshold.

The task can be formulated also as a complementary optimization problem, which aims to minimize the illumination power while maintaining the classification probability above a threshold.

$$\min||I_k||_2 \tag{2-12}$$
$$\text{s.t } ||\mathbf{P_k}||_\infty \geq \epsilon_p, \tag{2-13}$$

where ϵ_p is the classification probability threshold.

The focus of the paper will be on the first formulation, as specified in Equations 2-10, 2-11.

4 Classifier model

For object classification and in order to obtain the classification proabilities vector P_k, transfer learning was employed. Transfer learning is a machine learning technique where a model developed for a specific task is reused as the starting point for a model on a second related task. Essentially, it involves taking knowledge gained while solving one problem and applying it to a different but related problem. Image classification tasks often benefit substantially from transfer learning, especially due to the generic applicability of learned features in the lower layers of deep neural networks. As lower layers of convolutional neural networks (CNNs) usually capture universal features like edges, colors, and textures, which are applicable across different image classification tasks. Transfer learning exploits this by allowing these layers to remain largely unchanged, focusing fine-tuning efforts on the higher layers that capture more specific features.

A suitable CNN model for the classification purpose is MobileNetV2, a state-of-the-art convolutional neural network architecture optimized for mobile and embedded vision applications.[11] The MobileNetV2 network is pre-trained by its developer, Google Inc., on the ImageNet dataset, which consists of over a million RGB images and 1,000 classes, ranging in diversity from animals to vehicle parts. This extensive training allows the model to learn a wide variety of features that are transferable to other tasks, especially feature extraction. The pretrained model is available on the developers website for use. It takes as input an RGB image of size 3x96x96 and outputs a vector containing the probabilities that the input image belongs to a certain class. Moreover, MobileNetV2 is known for its efficiency and relatively low computational cost while maintaining high accuracy [1], which constituted the reason it was chosen.

In order for MobileNetv2 to be used on a different dataset, some application specific modifications are required. For example, for the Fashion MNIST dataset, modifications are required to both the MobileNetv2 model structure and the dataset format.

First, the Fashion MNIST dataset, which consists of greyscale 28x28 images, has to be converted to a suitable format that will be accepted by MobileNetv2. As such, an RGB conversion and rescaling function will take as input the greyscale 1x28x28 image and output an RGB 3x96x96. Furthermore, due to the relatively simplistic nature of Fashion MNIST, a data argumentation function will take as input the rescaled RGB image and perform a random zoom of maximum 10%, a random rotation between $[-10, +10]$ and a 10% random translation of the object. Note: The data augmentation steps are applied only in the context of the Fashion MNIST dataset for illustration purposes. In real-life applications, the camera and object would be most likely stationary and as such there would be no rotation, zoom or translation. These properties will be set by the imaging hardware and can be assumed to remain constant.

In the next step the object is illuminated according to the model in Equation 2-7. The computation of L_k will be expanded upon in Chapter 3. The resulting illuminated image is then passed as an input to a function which computes and adds a Poisson distributed noise, according to Section 2. The resulting noisy image I_k is then ready to be passed through the MobileNetv2 Network.

The modifications detailed above pertain to the image itself, nonetheless the structure of MobileNetv2 has to also be adjusted in order to be able to classify a different dataset, in this case Fashion MNIST, than the one it was trained on. The standard procedure as detailed by the developer [2] is to remove the last layer of the MobileNetv2 model, which outputs the probabilities vector, and replace it with an application specific Dense Layer, with a number of nodes equal to the number of classes of the desired dataset and a Softmax activation function. The input of this added layer will be a set of logits outputted by MobileNetv2 and will output a vector of classification probabilities, pertaining to the desired dataset, in this case Fashion MNIST. The conversion from logits to probabilities becomes meaningful in the sense of statistical probability through the loss function used in tuning the model. The loss function used is the Sparse Categorical Cross-Entropy and is defined as:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i \tag{2-14}$$

, where y is the label which will help For the Fashion MNIST, which has a number of 10 classes, the layer will have a size of 10 and the output will be the probabilities vector $\mathbf{P_k}$, containing the probabilities of the 10 classes.

$$\mathbf{P_k} = f_{MobileNetv2}(I_k) = [P_0 \ldots P_9] \tag{2-15}$$

The overall classification pipeline is shown in Figure 2-6

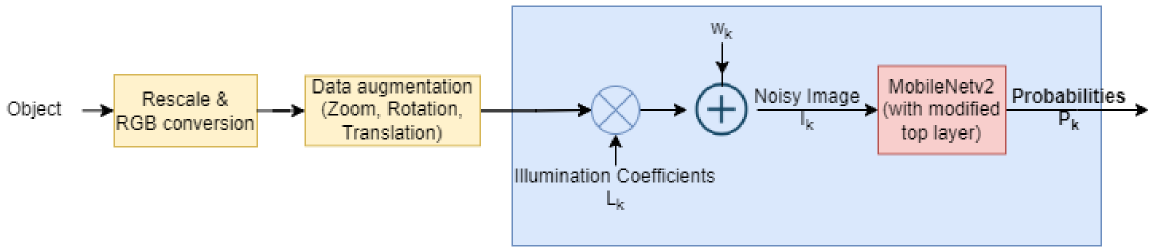

Object
Rescale & RGB conversion
Data augmentation (Zoom, Rotation, Translation)
w_k
Illumination Coefficients L_k
Noisy Image I_k
MobileNetv2 (with modified top layer)
Probabilities P_k

Figure 2-6: Block diagram showcasing the flow of information of the object image O, through the augmentation process, illumination with L_k, Poisson noise addition w_k, and the classification using MobileNetv2 and resulting in the probabilities vector $\mathbf{P_k}$

The training and fine-tuning process is conducted in two main stages of supervised learning.

Initially, all the layers of the MobileNetV2 base model are frozen to retain the pre-trained weights and features learned from ImageNet. Only the newly added dense layer is trained. Namely, the target dataset, in this case Fashion MNIST, is used to train the network, however the weights remain the same for the MobileNetv2 layer, due to them being frozen, with only the Dense Layer weights being updated. This stage adapts the high-level features learned by MobileNetV2 to the Fashion MNIST dataset.

In the fine-tuning stage, a number of layers from the base model of MobileNetv2 are unfrozen and the training process with the target dataset is repeated. The exact number required is application specific with more feature rich datasets requiring a higher number of layers. In the case of Fashion MNIST, 52 layers seemed to be the optimum, as the performance gain plateaued in that region. Specifically, layers from the 100th to the final layer (a total of 52 layers) are made trainable. This allows the model to fine-tune the higher-level feature representations to better match the Fashion MNIST dataset while still benefiting from the pre-trained weights. Fine-Tuning Configuration: The model is compiled with the Adam optimizer, sparse categorical crossentropy loss, and accuracy as a metric. A lower learning rate is used during this stage (0.001) to avoid large updates to the pre-trained weights.

The combination of initial training of custom layers and fine-tuning specific layers of the base model ensures that the classifier achieves high accuracy and robustness.

To summarize, in the proposed application example the image of the object O_k from the Fashion MNIST dataset is rescaled and converted to RGB, 1x28x28 -> 3x96x96, data augmentation in the form of zoom, scaling and translation occurs, illumination and noise are applied according to Equations 2-7 and 2-5. The resulting noisy image I_k constitutes the input to a pretrained classifer, MobileNetv2, which was adapted for this specific dataset. The output of MobileNetv2 will be a vector classification probabilities P_k, which will further constitute the input to the algorithm, used to generate the optimal illumination coefficients matrix Δ for the next step $k + 1$.

Chapter 3

Computing the next input with a Soft Actor-Critic algorithm

The goal is to use P_k, as given by the output of the classifer, to compute the optimal input for the next time step $k + 1$, which will maximize the probability as stated in the Problem Formulation in Equation 2-10. As the feedback loop has to react in real time and adapt to the environment as time progresses, Reinforcement Learning was seen as a suitable approach.

1 Reinforcement Learning

Reinforcement learning distinguishes itself from other machine learning paradigms through its focus on an agent learning from direct interaction with its environment, without needing explicit supervision or complete models of the environment. The agent's learning process revolves around the exploration-exploitation trade-off, where it has to balance between exploring unknown parts of the environment to find new strategies and exploiting known strategies to maximize rewards.

There are many Reinforcement Learning algorithms, however a few key characteristics were identified, that helped in the choice of the algorithm. First, due to the nature of the required input, a condition would be for the algorithm to handle continuous action spaces. Secondly, the real time aspect and overtime adaptability to an everchanging environment, are making a good case for an off-policy algorithm

As such a Soft Actor-Critic (SAC) algorithm was chosen to be investigated. The Soft Actor-Critic algorithm represents a significant advancement in the field of reinforcement learning, offering a robust framework for training agents, here an "agent" refers to the entity that interacts with the environment to achieve a certain goal through trial and error, in complex and high-dimensional environments.[12] This algorithm, developed in the context of deep reinforcement learning, embodies a blend of off-policy actor-critic methods with a maximum entropy framework. This synthesis not only promotes efficient learning but also encourages the exploration of state-action spaces, thus enhancing the adaptability and performance of the agents.

SAC takes as input a state vector $\mathbf{S_k}$, that will use to generate an output, consitituing of an action vector $\mathbf{a_{k+1}}$

$$\mathbf{a_{k+1}} = f_{SAC}(S_k) \tag{3-1}$$

As such, it can be observed that the state formulation is a crucial aspect of the reinforcement learning framework, as it encapsulates the necessary information for the agent to make decisions.

In the context of our application the action a_k will represent the entries in the Δ matrix, representing the amplitude of the desired delta pulses to be used in the illumination adjustment. Thus, $\alpha(u, v)$ from Equation 2-3 will take the value of the action:

$$\alpha_k(u, v) = a_k(u, v) \text{ for } 1 \leq u \leq 8, 1 \leq v \leq 8 \tag{3-2}$$

As the Fashion MNIST is composed of originally greyscale images which had to be artificially converted to RGB for compatibility with MobileNetv2, the illumination will be applied uniformly over the three channels. As such, the action space will be computed as if the image had only one channel and the dimension will remain bounded to 1 channel.

2 State Formulation

For the purpose of our application, the stat, which is the input to the SAC, will contain the probability vector $\mathbf{P}$. Thus, for a time step k, the state can be formulated as follows:

$$S_k = \mathbf{P_k} \tag{3-3}$$

However, taking into consideration only the current probabilities, neglects a potentially benefical feature in the form of memory. Thus, the goal of the thesis is to evaluate whether by taking into account the previous action, that led to the current classification probability vector, the Soft Actor-Critic would achieve the optimal action in fewer iterations. As such, the action a_k, which led to the current probabilities P_k was also included in the state vector S_k, thus effectively becoming an input to the Soft Actor-Critic.

$$S_k = [\mathbf{P_k}, \mathbf{a_k}] \tag{3-4}$$

It can be observed, that the size of S_k is application dependent with both the number of classes and the desired number of Fourier Components varying. In the example case of Fashion MNIST, with 10 classes, and the desired number of Fourier componensts set to 64 (8 vertical and 8 horizontal), S_k is formed as follows:

$$S_k = [P_0, P_1, \ldots, P_9, a_1, a_2, \ldots, a_{64}], \tag{3-5}$$

where P_i are the classification probabilities and a_i are the previous actions.

The total size of the state is:

$$\text{State Size} = \text{Number of Classes} + \text{Number of Coefficients} = 10 + 64 = 74 \tag{3-6}$$

It can be easily observed that the state dimension increases linearly with the number of classes and number of desired Fourier components.

For example if instead of an 8x8 block of Fourier components, a 16x16 block is used, the number of coefficients per channel will increase to $16^2 = 256$. Thus, the new state size would be:

$$\text{State Size} = N_c + 256 = 10 + 256 = 266. \tag{3-7}$$

If a distinct illumination pattern for each channel is desired, the general formula for a multi-channel image would then be defined as:

$$\text{State Size} = N_c + \#\text{Channels} \cdot N_f, \tag{3-8}$$

where N_c is the number of classes and N_f the number of adjusted Fourier Components per channel.

2-1 State Dimensionality Problem

This increase in state size poses a problem of input dimensionality to SAC, which may affect performance and the computational power required. A possible solution might come in the form of data compression in such a way that information is not lost and a memory of the past is still kept. As such, a Recurrent Neural Network(RNN), which purpose is exactly this, could function as an intermediate layer between the state S_k and SAC.

Recurrent Neural Networks are a class of neural networks designed for processing sequential data, making them particularly well-suited for tasks involving temporal dynamics or sequence-dependent information. Unlike feedforward neural networks, where the flow of information is unidirectional and does not loop, RNNs possess the unique characteristic of maintaining a form of memory by incorporating loops within their architecture. This looping mechanism enables RNNs to retain information from previous inputs in the sequence, allow- ing them to make predictions based on both current and historical data.

There are two main types of RNNs, Long Short-Term Memory and Gated Recurrent Unit (GRU). Both achieve the same functionality, however the GRU is simpler in structure, which would benefit the real time performance requirement, and was thus chosen. An additional benefit of the GRU is the fixed output size, independent of the input and as such it would be versatile in the context of our application, as it can be easily repurposed on the fly for other scenarios with different number of classes or actions.

The GRU will take thus as input the current state $\mathbf{S_k}$ and output a fixed size hidden state h_k, which will contain all the information until the current time step. h_k will then constitute the input to the Soft Actor-Critic.

$$h_k = g_{GRU}(\mathbf{S_k}, h_{k-1}), \tag{3-9}$$

where h_{k-1} is the previous step's final hidden state, which contains all the information stored until time step k. Thus, the action is now the result of a compound function:

$$\begin{aligned} a_k &= f_{SAC}(h_k) \qquad (3\text{-}10)\\ &= f_{SAC}(g_{GRU}(\mathbf{S_k}, h_{k-1})) \qquad (3\text{-}11) \end{aligned}$$

2-2 Novel approach and Research Question

The limitations highlighted, ranging from the challenges imposed by high input dimensionality, the continious nature of inputs, to the stringent demands of real-time applications, will be addressed through a combination of a Soft Actor-Critic algorithm with a Gated Recurrent Unit to stand as intermediary between the state vector S_k and the Soft Actor-Critic. This is a novel approach and considering the characteristics of each individual component and individual suitability for the task at hand, it stands to reason that together they might complement each other and lead to an improvement in classification performance. The proposed example application using a simpler dataset, Fashion MNIST, that was mentioned along in the paper is intended to act as a proof of concept and validate the approach.

3 Gated Recurrent Unit (GRU)

The integration of the Gated Recurrent Unit (GRU) and Soft Actor-Critic (SAC) in the reinforcement learning framework allows the agent to make informed decisions by considering temporal dependencies in the state sequence and optimizing actions to maximize rewards while maintaining exploration through entropy regularization.

GRUs are a type of recurrent neural network (RNN) architecture designed to manage sequence data and retain important information over time. GRUs are particularly useful in scenarios where the sequential nature of the data is significant, such as time series analysis, natural language processing, and reinforcement learning.[13]

A GRU cell consists of two main gates:

- Update Gate (z_k)
- Reset Gate (r_k)

These gates control the flow of information and help manage the vanishing gradient problem. The internal structure of the GRU is as follows:

The update gate determines how much of the past information needs to be passed along to the future.

$$z_k = \sigma\left(W_z S_k + U_z h_{k-1} + b_z\right), \qquad (3\text{-}12)$$

where W_z is the weight for the input, U_z is the weight for the previous hidden state, h_{k-1} is the previous hidden state, S_k is the input (state) at the current time step and b_z is the bias of the update gate.

The reset gate decides how much of the past information to forget.

$$r_k = \sigma\left(W_r S_k + U_r h_{k-1} + b_r\right), \tag{3-13}$$

where W_r is the weight for the input, U_r is the weight for the previous hidden state, and b_r is the bias of the reset gate.

A candidate for the new state, influenced by the reset gate.

$$\hat{h}_k = \phi\left(W_h S_k + U_h\left(r_k \odot h_{k-1}\right) + b_h\right), \tag{3-14}$$

where $\hat{h}_k$ is the candidate hidden state, W_h is the weight for the input, U_h is the weight for the Hadamard product of the reset gate with the previous hidden state, and b_h is the bias of the candidate hidden state.

The final output of the GRU cell is a combination of the old state and the candidate new state.

$$h_k = (1 - z_k) \odot h_{k-1} + z_k \odot \hat{h}_k \tag{3-15}$$

where h_k is the hidden state at time step k. h_k will then be further passed to the Soft Actor-Critic.

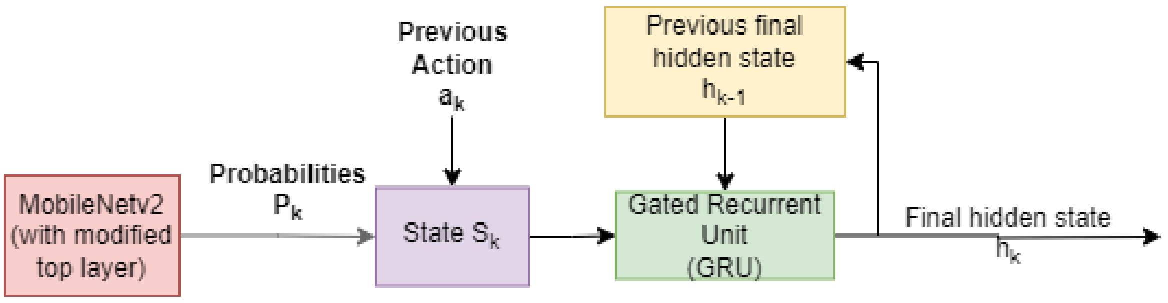

Previous Action a_k
Previous final hidden state h_{k-1}
Probabilities P_k
MobileNetv2 (with modified top layer)
State S_k
Gated Recurrent Unit (GRU)
Final hidden state h_k

Figure 3-1: Block diagram showcasing the flow of information from the classifier using MobileNetv2 to the output of the GRU

In the context of the example application, the size of the final hidden state h was set to 256, the closest power of 2, which is standard common practice, to the actual state S size of 202.

4 Soft Actor-Critic

SAC is an advanced reinforcement learning algorithm that incorporates elements from both actor-critic methods and entropy maximization. The primary goal of SAC is to balance exploration and exploitation by encouraging the agent to maximize both expected return and entropy, which promotes diverse behavior.

The overall data pipeline is shown in Figure 3-2.

The Soft Actor-Critic (SAC) algorithm integrates several key mathematical concepts and notations from reinforcement learning, particularly focusing on the maximum entropy framework. In the case of our application, in which a GRU acts as an intermediate layer between the state vector S_k and the SAC input effectively being the final hidden state of the GRU h_k, the mathematical model of SAC can be described as follows:

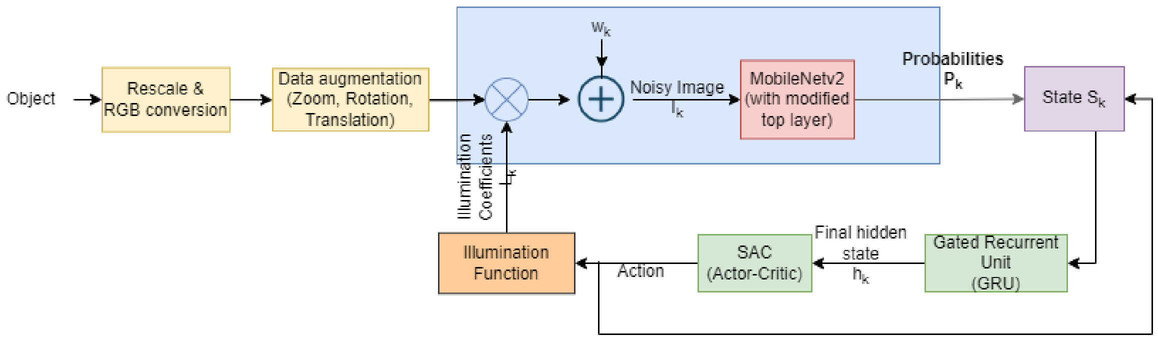


Figure 3-2: Block diagram showcasing the feedback loop with all components present, including the Soft Actor-Criitc

- Markov Decision Process (MDP): MDP is a mathematical framework used to model decision-making in environments where outcomes are partly random and partly under the control of a decision-maker (agent). SAC, like many reinforcement learning algorithms, operates within the context of an MDP defined by the tuple (H, A, P, R, γ), where H is the set of final hidden states, A is the set of actions, $P : H \times A \times H \to [0, 1]$ is the state transition probability function, $R : H \times A \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor for future rewards. However, in the context of the SAC algorithm, the transition probability function $\mathcal{P}(h' \mid h, a)$ is not explicitly estimated or modeled. Instead, SAC operates in a model-free manner, meaning it learns optimal policies and value functions directly from interactions with the environment without explicitly learning the transition dynamics.

- Policy (Actor) π: The policy is a stochastic mapping from states to actions, represented by $\pi : H \times A \to [0, 1]$, where $\pi(a|h)$ denotes the probability of selecting action a given final hidden state h

- Value Functions:
  - State-Value Function $V^\pi(h)$: The expected return when starting in state h and following policy π thereafter, defined as

$$V^\pi(h) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{k+t+1} | H_k = \right] \tag{3-16}$$

  - Action-Value Function $Q^\pi(h, a)$: The expected return of taking action a in state h and following policy π thereafter, defined as

$$Q^\pi(h, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{k+t+1} | H_k = h, A_k = a \right] \tag{3-17}$$

- Entropy Maximization: The SAC algorithm aims not only to maximize the expected cumulative reward but also to maximize the entropy of the policy. The entropy of a policy π at final hidden state h is defined as

$$H(\pi(\cdot|h)) = - \sum_{a \in A} \pi(a|h) \log \pi(a|h), \tag{3-18}$$

which encourages exploration by favoring policies that distribute probability across multiple actions.

- Objective Function: The SAC algorithm optimizes an objective function that combines the expected reward and the entropy of the policy. The objective function for a policy π is given by:

$$J(\pi) = \mathbb{E}_{(h,a)\sim\pi}\left[\sum_{k=0}^{\infty}\gamma^k(R(h_k, a_k) + \alpha H(\pi(\cdot|h_k)))\right], \tag{3-19}$$

  where α is a temperature parameter that controls the trade-off between maximizing the entropy and the expected reward. High values of α encourage more exploration.

4-1 MDP

Expanding on the definition of a Markov Decision Process (MDP) is crucial to understanding the Soft Actor Critic, as it builds upon the standard MDP framework by introducing entropy regularization. This addition encourages exploration and avoids premature convergence to suboptimal policies by promoting stochastic behavior. MDPs provide a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. MDPs are particularly useful in the context of reinforcement learning, where an agent learns to make decisions through trial-and-error interactions with an environment.

The final hidden state space (H) represents all possible configurations or situations in which the agent can find itself. Final Hidden States are the pieces of information used by the agent to make decisions. In a discrete MDP, the state space is a finite set of states, whereas, in continuous domains, it can be represented by a continuous set. The action space (A) defines all possible actions the agent can take. Similar to the state space, the action space can be discrete (a finite set of actions) or continuous. Actions are the means through which the agent interacts with the environment.

The transition probability function, $P(h|h', a)$, represents the probability of moving from state h to state h after taking action a. As mentioned above, in the case of SAC, the transition probability function is not explicitly modeled, as SAC operates in a model-free manner. This function encapsulates the dynamics of the environment, reflecting how the environment responds to the agent's actions. For each state-action pair, the transition function provides a probability distribution over next states. The reward function, $R(h, a, h')$, specifies the immediate reward received after transitioning from state h to state h' due to action a. The reward function is a critical component as it encodes the goal of the agent in the environment. The agent's objective is typically to maximize the cumulative reward it receives over time. Lastly, the discount factor, γ, is a parameter $0 \leq \gamma < 1$ that determines the importance of future rewards. A discount factor of 0 makes the agent myopic, meaning it only cares about immediate rewards. As γ approaches 1, future rewards are valued more equally with immediate rewards, encouraging the agent to consider long-term consequences of its actions.

Solving an MDP involves finding a policy (π), a strategy for the agent to follow, where a policy is a mapping from states to probabilities of selecting each action. The goal is to find an optimal policy, π^*, that maximizes the expected cumulative reward the agent receives over time, starting from any state. This expected cumulative reward is also known as the value function of a state (or state-action pair) under a policy. The solution to an MDP can be found using various algorithms, including dynamic programming methods (like value iteration and policy iteration), Monte Carlo methods, and temporal-difference learning methods (like Q-learning and SARSA), each suitable for different contexts depending on the availability of the model (transition and reward functions) and the nature of the state and action spaces (discrete or continuous).

4-2 Policy (actor)

In the context of Soft Actor-Critic (SAC), the policy is often referred to as the "actor" because it directs the actions of the agent. A stochastic policy provides a probability distribution over actions for each state, denoted as $\pi(a|h)$, which represents the probability of taking action a when in state h. Stochastic policies are particularly useful for exploration and handling uncertainty. In the SAC framework, the policy is typically modeled as a stochastic neural network that outputs parameters of a probability distribution over actions (e.g., the mean and covariance of a Gaussian distribution). The agent then samples from this distribution to select actions, allowing both exploration of the action space and learning a more nuanced control strategy that can adjust to the variability in the environment.

The policy (actor) is updated by taking gradient steps on an objective function that seeks to maximize both the expected return and the entropy of the policy. This process often involves computing gradients of the expected return with respect to the policy parameters and adjusting the parameters to improve both the return and entropy. Furthermore, stochastic policies naturally incorporate exploration, as they allow the agent to try different actions with varying probabilities, which is essential for discovering and learning from new states and rewards.

4-3 Value functions

Value functions in reinforcement learning provide a measure of the expected return (cumulative rewards) that an agent can achieve, starting from a particular state or state-action pair, and following a specific policy. There are two primary types of value functions: the state-value function and the action-value function. Both play a crucial role in evaluating and improving policies.

The state-value function $V^\pi(h)$ estimates the expected return starting from state h and following policy π thereafter. It is denoted as $V^\pi(h)$ and mathematically defined as:

$$V^\pi(h) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{k+t+1} \middle| H_k = h \right] \tag{3-20}$$

where H_k is the state at time k, R_{t+1} is the reward received after transitioning from state H_k due to action A_k, γ is the discount factor, $0 \leq \gamma < 1$, which models the present value of future rewards. The expectation $\mathbb{E}_\pi$ is taken over the distribution of possible trajectories starting from h and following policy π.

The action-value function ($Q^\pi(h,a)$), also known as the Q-function, estimates the expected return starting from state h, taking action a, and then following policy π. It is denoted as $Q^\pi(h,a)$ and defined as:

$$Q^\pi(h,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^k R_{k+t+1} \middle| H_k = h, A_k = a \right] \tag{3-21}$$

where A_k is the action taken at time k.

The state-value function and the action-value function are related through the policy π. For any state h and policy π, the state-value function can be expressed in terms of the action-value function as follows:

$$V^\pi(h) = \sum_{a \in A} \pi(a|h) Q^\pi(h,a) \tag{3-22}$$

This equation states that the value of a state under policy π is the expected value of the action-values under that policy, where the expectation is taken over the actions chosen according to π.

Moreover, both value functions satisfy recursive relationships known as the Bellman equations, which are foundational to many algorithms in reinforcement learning.

Bellman Equation for $V^\pi(h)$:

$$V^\pi(h) = \sum_{a \in A} \pi(a|h) \sum_{h' \in H} P(h'|h,a) \left[R(h,a,h') + \gamma V^\pi(h') \right] \tag{3-23}$$

Bellman Equation for $Q^\pi(h,a)$:

$$Q^\pi(h,a) = \sum_{h' \in H} P(h'|h,a) \left[R(h,a,h') + \gamma \sum_{a' \in A} \pi(a'|h') Q^\pi(h',a') \right] \tag{3-24}$$

These equations provide a way to compute the value functions iteratively, forming the basis for various algorithms that solve reinforcement learning problems, including value iteration, policy iteration, and many approximation methods used in deep reinforcement learning. As such they will be expanded upon in the next subsection.

4-4 Bellman Equation

The Bellman equations are fundamental to understanding reinforcement learning, serving as recursive decompositions of value functions into immediate rewards and the discounted value of successor states. These equations capture the essence of the decision-making process in sequential decision problems, like those modeled by Markov Decision Processes (MDPs) or its the ones built upon MDPs, like SAC. Additional conditions are necessary in order to ensure the Bellman Equation can be applied to SAC. These are ergodicity and bounded reward. Ergodicity ensures that the agent can eventually explore the entire state space, which is important for learning an optimal policy. If certain states are inaccessible, the agent's learning may be biased, leading to suboptimal policies that don't generalize well across the state space. Furthermore, the Bellman Equation involves sums of discounted rewards. If the rewards are unbounded, the value functions can become unbounded as well, leading to numerical instability and difficulty in learning. Bounded rewards ensure that the value functions remain well-defined and stable.

Having the conditions defined, let's dive deeper into the Bellman equations for both the state-value function and the action-value function.

The Bellman equation for the state-value function $V^\pi(h)$ under policy π expresses the value of a state as the immediate reward expected from the current state plus the discounted value of the state that follows. Mathematically, it's expressed as:

$$V^\pi(h) = \sum_{a\in A} \pi(a|h) \sum_{h'\in H} P(h'|h,a) \left[R(h,a,h') + \gamma V^\pi(h')\right], \tag{3-25}$$

where $\sum_{a\in A} \pi(a|h)$ iterates over all possible actions according to the policy π, weighting each action by its probability under π. $P(h'|h,a)$ is the probability of transitioning to state h' from state h after taking action a. $R(h,a,h')$ represents the immediate reward received after transitioning from h to h' due to action a. γ is the discount factor, indicating how future rewards are valued compared to immediate rewards. $V^\pi(h')$ is the value of the subsequent state h', reflecting the future expected returns.

The Bellman equation for the action-value function, or Q-function $Q^\pi(h,a)$, under policy π, relates the value of taking a specific action in a specific state to the immediate reward plus the discounted value of the next state, following policy π. It's given by:

$$Q^\pi(h,a) = \sum_{h'\in S} P(h'|h,a) \left[R(h,a,h') + \gamma \sum_{a'\in A} \pi(a'|h') Q^\pi(h',a')\right] \tag{3-26}$$

, where the term $\sum_{a'\in A} \pi(a'|h') Q^\pi(h',a')$ represents the expected value of all possible actions a' in the subsequent state h', according to policy π.

Both Bellman equations have a recursive nature, breaking down the value of a state or state-action pair into the immediate reward plus the discounted value of future states. This recursive decomposition is key to solving MDPs, as it allows for the value functions to be computed iteratively. Furthermore, when applied to the optimal value functions $V^*(h)$ and $Q^*(h,a)$, the Bellman equations transform into Bellman optimality equations, which serve as the basis for finding the optimal policy that maximizes expected returns.

4-5 Q-function

The action-value function, commonly referred to as the Q-function, $Q^\pi(h,a)$, plays a critical role in reinforcement learning by estimating the expected return (cumulative discounted rewards) for taking an action a in a state h and thereafter following a policy π. It serves as a fundamental component in both analyzing policies and devising algorithms for learning optimal behaviors in Markov Decision Processes (MDPs).

The Q-function for a policy π is defined as:

$$Q^\pi(h,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{k+t+1} \middle| H_k = h, A_k = a \right] \tag{3-27}$$

where h is the current state, a is the action taken in state h, $\mathbb{E}_{\pi}[\cdot]$ denotes the expected value given that the agent follows policy π after taking action a, R_{k+t+1} is the reward received after t steps, γ is the discount factor ($0 \leq \gamma < 1$), which represents the difference in importance between future rewards and immediate rewards. The sum accumulates the total expected return starting from time k, considering the immediate action a and subsequent actions as dictated by policy π.

The Q-function is pivotal in evaluating the effectiveness of taking a certain action in a given state under a specific policy. It quantifies the expected utility of action-state pairs, providing a measure to compare different actions. By identifying the action that maximizes the Q-function for a given state, one can improve the policy. Thus, in the context of learning, the Q-function helps in finding the optimal policy (π^*) that maximizes the expected return from any initial state. The optimal Q-function, $Q^*(h, a)$, satisfies the Bellman optimality equation:

$$Q^*(h,a) = \sum_{h' \in H} P(h'|h,a) \left[R(h,a,h') + \gamma \max_{a'} Q^*(h',a') \right] \tag{3-28}$$

In summary, the Q-function is a foundational concept in RL, including SAC, enabling both the assessment of the current policy and the discovery of optimal policies through learning algorithms.

The Equations explained above represent the internal mechanism of SAC and would generally remain true for any kind of application as long as the GRU functions as the intermediary. In the context of our application for the example problem using the Fashion MNIST dataset, the parameter values were set to:

- Actor Network: Two hidden layers with 256 units each, using ReLU activation, and an output layer with sigmoid activation to scale actions in the range [0, 2].
- Critic Network: Two hidden layers with 256 units each, using ReLU activation.
- Discount Factor (γ): 0.99
- Soft Update Coefficient (τ): 0.005
- Learning Rate: 0.001 for both actor and critic optimizers.
- Temperature Parameter (α): Dynamically adjusted to balance exploration and exploitation.

The choice of the parameters is application specific and would require individual tuning.

4-6 Reward Function

The reward function is a critical component of the RL framework, guiding the agent towards optimal behavior. In this project, the reward function is designed to balance the improvement in classification probabilities with the consistency of actions.

Let max_prev_prob be the maximum probability from the previous step, and max_current_prob be the maximum probability from the current step.

$$\text{max\_prev\_prob} = \max(P_{k-1}) \tag{3-29}$$

$$\text{max\_current\_prob} = \max(P_k) \tag{3-30}$$

The improvement in probability, Δprob, is defined as the difference between the current maximum probability and the previous maximum probability.

$$\Delta\text{prob} = \text{max\_current\_prob} - \text{max\_prev\_prob} \tag{3-31}$$

Furthermore, the total illumination power should be kept under a certain threshold, Power Threshold. This parameter should be adjusted according to the application. In the case of the Fashion MNIST a total illumination power threshold of 10 was chosen.

$$\text{Total Power} = ||I_k||_2 \tag{3-32}$$

In the case of Fashion MNIST with an action bounded between [0,2] the maximum achievable total power is:

$$\text{Maximum Total Power} = \sqrt{\sum_{1}^{64} 2^2} = 16 \tag{3-33}$$

An accurate prediction should also be rewarded as such

$$\begin{aligned} &\text{predicted label} = \text{true label, Correct Classification} = 1 \\ &\text{predicted label} \mathrel{!}= \text{true label, Correct Classification} = \text{-1} \end{aligned} \tag{3-34}$$

The reward, R, is a combination of the improvement in probability, the action consistency penalty and the correct classification check.

$$R = \alpha \cdot \Delta\text{prob} + \omega \cdot (\text{Power Threshold} - \text{Total Power}) + \gamma \cdot \text{Correct Classification}, \tag{3-35}$$

where α is a scaling factor for the probability improvement,ω is the scaling coefficient for the Power Restriction ,and γ is the scaling factor for the Correct Classifictation Reward component.

In the context of the example problem, the parameters were set to: α was set to 1 and ω to 0.25, and γ to 1.

5 Soft Actor-Critic (SAC) Training

The Soft Actor-Critic algorithm employs an iterative training process that involves updating the policy (actor), the state value function, and the Q-function (critic) to maximize both the expected return and the entropy of the policy. This section elaborates on the state value function within SAC and describes the iterative training process in detail, incorporating mathematical notations for clarity.

The state value function, denoted as $V^{\pi}(h)$, in the context of SAC, estimates the expected return of being in a state h and acting according to a policy π that is both reward-seeking and entropy-maximizing. As mentioned previously, unlike traditional reinforcement learning algorithms that focus solely on maximizing expected returns, SAC incorporates entropy into the objective function to encourage exploration by the policy. The state value function in SAC is therefore defined to capture both these aspects: the expected sum of rewards and the expected sum of the policy's entropy over future states. To facilitate understanding, let's define it once again as:

$$V^{\pi}(h) = \mathbb{E}_{a \sim \pi, h' \sim p}\left[Q^{\pi}(h,a) - \alpha \log \pi(a|h)\right] \tag{3-36}$$

The iterative training process in SAC involves the following steps, which are performed repeatedly until the policy converges to an optimal policy that maximizes the expected return and entropy:

- Soft Q-Function Update: At each iteration, SAC updates the soft Q-function by minimizing the mean squared Bellman error (MSBE). The target for the soft Q-function is derived from the current estimate of the state value function, adjusted for the reward received and the entropy of the next action:

  $$Q^{\text{target}}(h,a) = r(h,a) + \gamma \mathbb{E}_{h' \sim p}\left[V^{\pi}(h')\right] \tag{3-37}$$

  The soft Q-function is then updated by minimizing the difference between its current estimates and this target.

- State Value Function Update: The state value function $V^{\pi}(h)$ is updated to minimize the difference between its current estimate and the expected value derived from the soft Q-function and the entropy of the policy:

  $$V^{\pi}_{\text{new}}(h) = \mathbb{E}_{a \sim \pi}\left[Q^{\pi}(h,a) - \alpha \log \pi(a|h)\right] \tag{3-38}$$

- Policy Update: The policy π is updated by minimizing the expected KL divergence to a target policy that maximizes the soft Q-function minus the log-policy term, effectively steering the policy towards actions that are expected to yield high returns and are also exploratory:

  $$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} D_{KL}\left(\pi'(\cdot \mid h) \| \frac{\exp\left(Q^{\pi_{\text{old}}}(h,\cdot)\right)}{Z(h)}\right) \tag{3-39}$$

  where Z(h) is a partition function ensuring the policy remains a valid probability distribution. This step ensures that the policy gradually shifts towards more rewarding and exploratory actions.

For the illustrative example using the Fashion MNIST dataset the training of SAC takes the following approach:

First the train dataset is split in batches of a 100 images. The images will be rescaled and converted to RGB, as well as rotated, translated and zoomed, as it was described in Section 2.4

The training loop will iterate through each batch as follows.

1. Take an object from the batch
2. Apply illumination and noise
3. Make a prediction
4. Compute new action using SAC
5. Compute the reward and update SAC
6. Take the subsequent image
7. Once the batch is done, compute the overall accuracy

The loop will run on the same batch until the overall accuracy converges, with a convergence parameter of 1%.

Once convergence has been achieved, the trained SAC with GRU will be tested on a new dataset. In the testing loop, both SAC and the GRU will run in an inference mode in which they won't be updated anymore. The same object will be run multiple times until the convergence of probability is achieved. The metrics for performance will be number of iterations to convergence, the probability per class and accuracy.

After the test has run over the whole test dataset, the training loop will continue with the next batch from the train dataset.

This training loop will run multiple epochs, which for now has been set to 25.

As the above paragraph outlines the training loop at a high level, in the following a deeper dive will be taken into the mathematical model of SACs training loop for this example.

The critic networks Q_1 and Q_2 are updated by minimizing the Bellman residual loss. For each experience in the mini-batch, the target Q-value y is computed using the target networks:

$$y_i = r_i + \gamma(1 - d_i) \min(Q'_1(h_{i+1}, a'_{i+1}), Q'_2(h_{i+1}, a'_{i+1})) \tag{3-40}$$

where Q'_1 and Q'_2 are the target critic networks, and a'_{i+1} is the action sampled from the policy at h_{i+1}.

The loss for the critic networks is:

$$J_{Q_k} = \frac{1}{M} \sum_{i=1}^{M} (Q_k(h_i, a_i) - y_i)^2 \quad \text{for } k = 1, 2 \tag{3-41}$$

The actor network π is updated to maximize the expected return and entropy. The loss for the actor network is:

$$J_\pi = \frac{1}{M}\sum_{i=1}^{M}\left(\alpha \log \pi(a_i|h_i) - Q_1(h_i, a_i)\right) \tag{3-42}$$

The temperature parameter α is adjusted to control the trade-off between reward and entropy. The loss for α is:

$$J_\alpha = \frac{1}{M}\sum_{i=1}^{M} -\alpha\left(\log \pi(a_i|h_i) + \mathcal{H}_0\right), \tag{3-43}$$

where $\mathcal{H}_0$ is the target entropy.

The target networks are updated using a soft update with parameter τ:

$$\theta'_k \leftarrow \tau\theta_k + (1-\tau)\theta'_k \quad \text{for } k = 1, 2, \tag{3-44}$$

where θ_k are the parameters of the critic networks, and θ'_k are the parameters of the target critic networks.

6 Gated Recurrent Unit (GRU) Training

The parameters of the GRU are trained as part of the overall RL process. In this integration with the SAC framework, the GRU is used to process sequences of states and generate hidden states that serve as inputs to the actor and critic networks. The GRU's parameters are updated during the training of the actor and critic networks based on the gradients computed from the loss functions of these networks.

The GRU consists of several parameters that need to be trained:

- Update Gate Weights and Biases: W_z,U_z, b_z
- Reset Gate Weights and Biases: W_r, U_r,b_r
- Candidate Hidden State Weights and Biases: W_h, U_h,b_h

These parameters are collectively denoted as θ_{GRU}.

The training of the GRU's parameters occurs through the backpropagation of gradients computed from the loss functions of the actor and critic networks. The key steps in the training process are as follows:

- The GRU processes a sequence of states $\{S_k\}_{k=1}^T$ to generate a sequence of hidden states $\{h_k\}_{k=1}^T$.

$$h_k = \text{GRU}(S_k, h_{k-1}; \theta_{\text{GRU}}) \tag{3-45}$$

- The actor and critic networks use the hidden states h_k to compute the action a_k and the Q-values $Q_k(h_k, a_k)$.

$$a_k = \pi(h_k; \theta_\pi) \tag{3-46}$$

$$Q_i(h_k, a_k; \theta_{Q_i}) \quad \text{for } i = 1, 2 \tag{3-47}$$

  The loss functions for the actor and critic networks are computed based on the sampled mini-batch from the replay buffer.

- Backpropagation: The gradients of the loss functions with respect to the GRU parameters θ_{GRU} are computed using backpropagation through time (BPTT).

$$\nabla_{\theta_{\text{GRU}}} J_\pi \tag{3-48}$$

$$\nabla_{\theta_{\text{GRU}}} J_{Q_i} \quad \text{for } i = 1, 2 \tag{3-49}$$

- Parameter Updates: The GRU parameters are updated using an optimizer (e.g., Adam) based on the computed gradients.

$$\theta_{\text{GRU}} \leftarrow \theta_{\text{GRU}} - \eta \left(\nabla_{\theta_{\text{GRU}}} J_\pi + \sum_{i=1}^{2} \nabla_{\theta_{\text{GRU}}} J_{Q_i} \right), \tag{3-50}$$

  , where η is the learning rate.

The parameters of the GRU are trained as part of the overall SAC training process. By backpropagating the gradients of the actor and critic loss functions through the GRU, the model learns to capture temporal dependencies in the state sequences. The integration of GRU and SAC leverages the strengths of both architectures, allowing the agent to make informed decisions based on past experiences and optimize its actions to maximize rewards.

Due to the nature of the GRU, the above mathematical model will remain unchanged in case of different applications or characteristics, such as number of classes or number of adjusted Fourier Components.

Chapter 4

Results

1 Testing and Validation Scenario

A testing environment was set up to test the methods and hypotheses formulated in the thesis. As such, the dataset used was a subset of Fashion MNIST, a separate 10000-image testing dataset, that the neural networks have not previously seen in training. The labels containing the ground truth class are also available.

The goal is to show that the proposed method can improve classification performance by iteratively adjusting the illumination of the object under test. An additional goal is to show that the proposed method outperforms the state-of-the-art approach proposed by Noom [10].

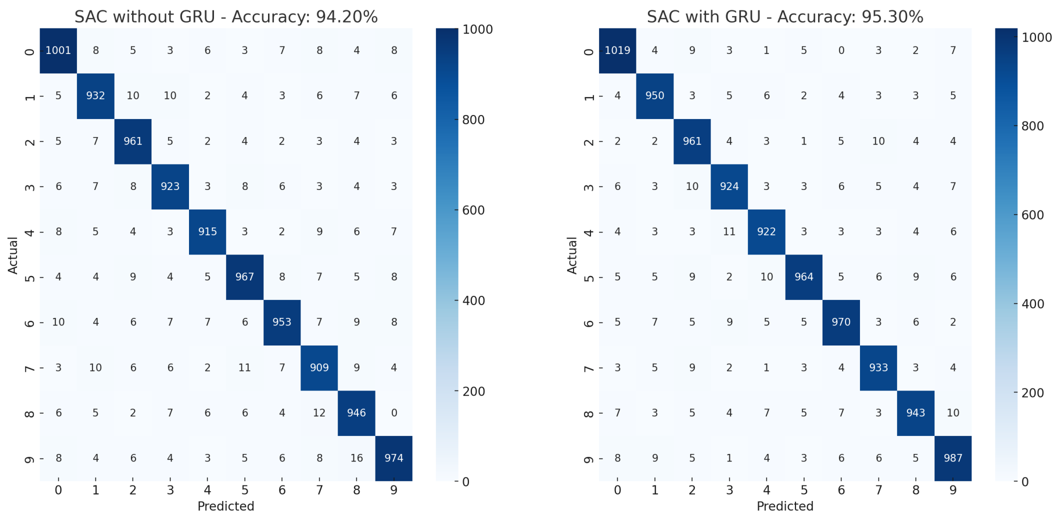
First, a testing scenario was set to test whether the inclusion of GRU benefits the classification performance. In that sense, the Soft Actor-Critic method was run twice: once with a GRU layer between the state and SAC and once without. The first configuration incorporates a GRU with a hidden state size of 128. The hidden state size of 128 was chosen to balance complexity and computational efficiency while making sure to preserve the information integrity The second configuration involves the straightforward SAC architecture without any GRU component. This setup serves as a baseline for evaluating the necessity of the GRU. The benchmarks were chosen to be the average number of iterations to convergence, the probability improvement over iterations, and accuracy improvement.

Furthermore, a second scenario, as one of the goals of the thesis is to improve upon the methods formulated by Noom [10], the proposed algorithm will be tested and benchmarked against the algorithm developed by Noom. As Noom's algorithm was validated on the MNIST dataset, retraining was required in order to accommodate the Fashion MNIST dataset. The illumination coefficients were kept as per the original to $u = [0.2, 0.4, 0.6, 0.8, 1]^T$.

2 GRU vs non-GRU approach

The two approaches discussed above were implemented, and the results after 25 iterations are as follows. On the Fashion MNIST testing dataset, the approach with only SAC achieved an overall accuracy of approximately 94.2%, while the SAC with GRU performed the best, with 95.3% overall accuracy.

The confusion matrices for the two approaches are shown in Figure 4-1. A confusion matrix shows the number of correct predictions for each class on the diagonal and the number of wrong predictions for each class in the rest of the entries. A higher number of correct predictions results in a higher overall accuracy.



SAC without GRU - Accuracy: 94.20%

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1001	8	5	3	6	3	7	8	4	8
1	5	932	10	10	2	4	3	6	7	6
2	5	7	961	5	2	4	2	3	4	3
3	6	7	8	923	3	8	6	3	4	3
4	8	5	4	3	915	3	2	9	6	7
5	4	4	9	4	5	967	8	7	5	8
6	10	4	6	7	7	6	953	7	9	8
7	3	10	6	6	2	11	7	909	9	4
8	6	5	2	7	6	6	4	12	946	0
9	8	4	6	4	3	5	6	8	16	974

SAC with GRU - Accuracy: 95.30%

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1019	4	9	3	1	5	0	3	2	7
1	4	950	3	5	6	2	4	3	3	5
2	2	2	961	4	3	1	5	10	4	4
3	6	3	10	924	3	3	6	5	4	7
4	4	3	3	11	922	3	3	3	4	6
5	5	5	9	2	10	964	5	6	9	6
6	5	7	5	9	5	5	970	3	6	2
7	3	5	9	2	1	3	4	933	3	4
8	7	3	5	4	7	5	7	3	943	10
9	8	9	5	1	4	3	6	6	5	987

Figure 4-1: Confusion Matrices for SAC without GRU and SAC with GRU on the Fashion MNIST test dataset

The classification probability per class also showed a higher increase when using the GRU than the isolated SAC. Table 4-1 and Table 4-2 show the initial classification probability per class and the final one after the 25 iterations. It can be observed that both approaches performed similarly for classes 7 and 8, Sneakers and Bag respectively. A reason for this could be the more simplistic shape of the items, in which case the memory attribute of the GRU will have a lower impact.

Nevertheless, as the goal is to increase classification performance as much as possible, the approach SAC with GRU Layer is considered to have a better overall performance.

Class	0	1	2	3	4	5	6	7	8	9
Initial Prob(%)	85.50	87.13	85.11	84.73	85.35	84.95	85.62	86.51	87.09	85.02
Final Prob(%)	94.30	93.80	94.33	94.95	95.23	93.88	93.90	93.27	93.79	94.56

Table 4-1: Probaiblity improvement per class over 25 iterations of SAC without GRU

Class	0	1	2	3	4	5	6	7	8	9
Initial Prob(%)	84.77	83.45	81.99	83.44	84.46	82.73	84.32	83.06	81.76	84.83
Final Prob(%)	94.19	96.53	97.41	94.48	93.95	93.78	95.73	96.98	96.32	93.62

Table 4-2: Probability improvement per class over 25 iterations of SAC with GRU

3 State-of-the-art vs SAC with/without GRU

Noom's algorithm was tested according to the previously detailed scenario. On the Fashion MNIST dataset, it achieved an overall accuracy of 92.2%. Table 4-3 shows the initial and final classification probabilities per class, as it was the case with SAC and SAC with GRU. It can also be observed that the algorithm performed the best on items from classes 7 and 8, similar to SAC and SAC with GRU. Thus, it can constitute a good baseline for performance benchmarks.

Class	0	1	2	3	4	5	6	7	8	9
Initial Prob(%)	82.2	83.03	82.59	82.65	80.74	81.15	84.31	81.01	81.44	80.79
Final Prob(%)	93.14	91.96	91.21	92.50	93.09	90.97	90.00	93.62	93.58	89.95

Table 4-3: Probaiblity improvement per class over 25 iterations of the baseline algorithm (Noom's)

Besides improved classification accuracy, real-time applications necessitate a higher convergence speed to ensure high throughput in an industrial setting. As such, the 3 approaches were compared in terms of the number of iterations needed for convergence, which can be seen in Figure 4-2. It can be observed that while the baseline algorithm does not reach convergence in the 25 iterations, both SAC and SAC with GRU Layer achieve beforehand. It is noteworthy to mention that while SAC achieves convergence after 12 iterations, SAC with GRU outperforms it in terms of overall accuracy, and also achieves convergences 3 iterations later. The slower convergence speed of the SAC with GRU approach could be because of the increased complexity due to the GRU. There is a clear trade-off between convergence speed and accuracy.

However, both algorithms have outperformed the baseline, both in terms of accuracy and convergence speed.

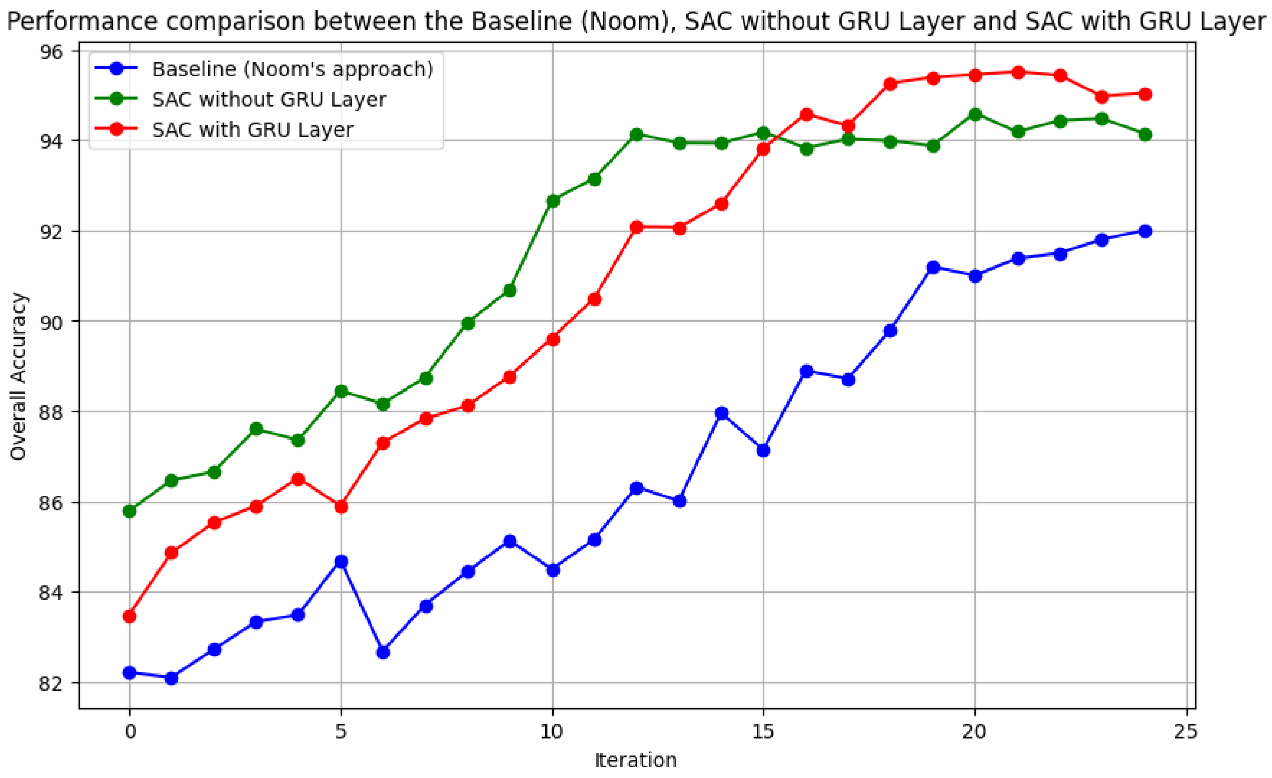


Figure 4-2: Convergence speed of the baseline algorithm, SAC and SAC with GRU Layer on the Fashion MNIST Dataset

Figure 4-3 showcases the iterative illumination process that SAC with GRU applies on an image from the Fashion MNIST dataset. It can be observed by how that illumination changes in consecutive iterations that it balances both exploration and exploitation. Furthermore, towards the last iterations, the illumination changes are more minor and it converges to a setting that increases contrast and outlines the edges, helping the classifier to predict the image accurately.

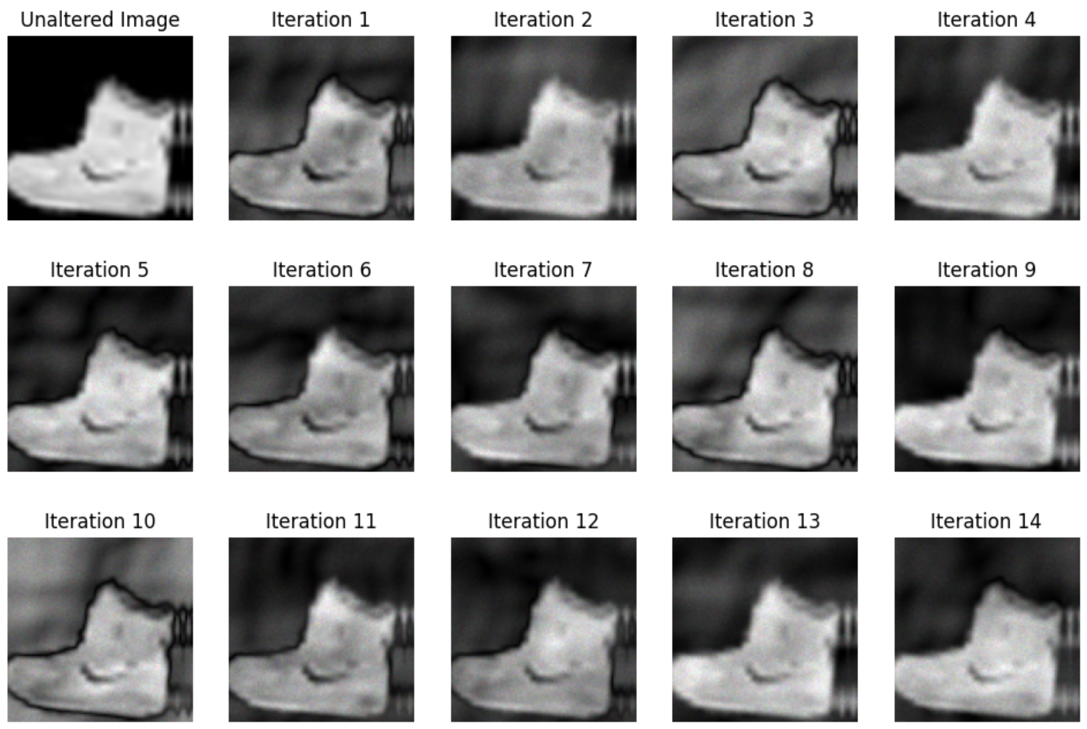


Figure 4-3: Illumination applied iteratively by SAC with GRU on an image from the Fashion MNSIT dataset

4 Conclusion

As shown in the results section above, adjusting illumination using a Soft Actor-Critic with a Gated Recurrent Unit Layer improves the classification performance iteratively. Furthermore, it has been shown to result in a higher improvement of the overall classification accuracy than the state-of-the-art algorithm proposed by Noom, while also achieving faster convergence.

In conclusion, Transfer Learning is a suitable choice for the classification task as the model benefits from the inherited knowledge. Furthermore, the Soft-Actor critic algorithm also represents a solution for computing the next optimal input due to its continuous action space. Finally, the Gated Recurrent Unit's ability to retain information and output a fixed dimension size independent of the input dimension made the GRU a suitable enhancement to SAC.

Thus, the proposed solution explored in the Thesis, namely a combination of Transfer Learning with MobileNetv2 for Image Classification, Soft-Actor Critic for computing the optimal action in continuous space and a Gated Recurrent Unit is a novel approach that proved beneficial to the task at hand.

One point that could be researched further is how the SAC and SAC with GRU approaches would perform on a more complex dataset. As shown, the isolated SAC achieved faster convergence, while SAC with GRU resulted in a higher overall accuracy. It is a clear trade-off scenario, and finding the optimal approach that would balance both speed and performance could be of interest for further research.

Bibliography

[1] Chia Wei Chao, Daniel Winden Hwang, Hung Wen Tsai, Shih Hsuan Lin, Wei Li Chen, Chun Rong Huang, and Pau Choo Chung. Multi-magnification attention convolutional neural networks [ai-explained]. *IEEE Computational Intelligence Magazine*, 18:54–55, 8 2023.

[2] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis, 12 2017.

[3] Lida Hu and Qi Ge. Automatic facial expression recognition based on mobilenetv2 in real-time. volume 1549. Institute of Physics Publishing, 6 2020.

[4] Kezheng Ren, Jun Liu, Zeyang Wu, Xinglei Liu, Yongxin Nie, and Haitao Xu. A data-driven drl-based home energy management system optimization framework considering uncertain household parameters. *Applied Energy*, 355:122258, 2 2024.

[5] Amudhini P. Kalidas, Christy Jackson Joshua, Abdul Quadir Md, Shakila Basheer, Senthilkumar Mohan, and Sapiah Sakri. Deep reinforcement learning for vision-based navigation of uavs in avoiding stationary and mobile obstacles. *Drones*, 7, 4 2023.

[6] Akshay Dheeraj and Satish Chand. Lwdn: lightweight densenet model for plant disease diagnosis. *Journal of Plant Diseases and Protection*, 131:1043–1059, 2024.

[7] Mahdi Shahab and Majid Moavenian. A novel fault diagnosis technique based on model and computational intelligence applied to vehicle active suspension systems. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 32, 5 2019.

[8] Dezhen Yang, Xingshuo Hai, Yi Ren, Jingjing Cui, Kanjing Li, and Shengkui Zeng. A hybrid fault prediction method for control systems based on extended state observer and hidden markov model. *Asian Journal of Control*, 25:418–432, 1 2023.

[9] Ying Zhang, Meng Yue, and Jianhui Wang. Off-policy deep reinforcement learning with automatic entropy adjustment for adaptive online grid emergency control. *Electric Power Systems Research*, 217, 4 2023.

[10] Jacques Noom, Oleg Soloviev, Carlas Smith, and Michel Verhaegen. Closed-loop active object recognition with constrained illumination power. page 6. SPIE-Intl Soc Optical Eng, 5 2022.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84–90, 6 2017.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[13] Luis Alfredo Moctezuma, Yoko Suzuki, Junya Furuki, Marta Molinas, and Takashi Abe. Gru-powered sleep stage classification with permutation-based eeg channel selection. *Scientific Reports*, 14, 12 2024.