Fixation-Guided Visual Attention Models with Transformers

Joan Lafuente Baeza

July 1, 2025

Abstract

Transformer-based models have achieved remarkable success in fields such as Natural Language Processing and Computer Vision, largely because of their self-attention mechanism. However, they also come with significant drawbacks, including substantial data requirements and high computational costs. This work aims to reduce their computational cost in the Computer Vision domain by introducing a visual attention model inspired by the human visual system. The proposed end-to-end architecture processes images sequentially, focusing on small, model-predicted regions (fixations). In contrast to previous approaches, we introduce a new fixation-generation action based on relative movements. We show that this method enables the model to selectively attend to relevant regions of the image, reducing the computational resources spent in the non-informative areas.

Keywords: Computer Vision, Reinforcement Learning, Visual Attention

1 Introduction

In recent years, several Artificial Intelligence (AI) models have been developed that have significantly impacted various aspects of our daily lives. One notable example is chatbots, which leverage Large Language Models (LLMs), such as ChatGPT or, more recently, Deepseek. These models make use of the Transformer architecture [1], which has demonstrated remarkable results for sequential tasks, particularly in Natural Language Processing (NLP), but has also achieved exceptional performance in other domains, such as Computer Vision with Vision Transformer (ViT) [2]. The main characteristic of these networks is the Self-attention mechanism, which enables the model to break down the input and determine the areas in which it should focus. This mechanism boosts performance, but it is not the only advantage, as it can also improve the interpretability of a model. The attention scores of a trained model can be extracted, providing insights into the weight assigned to the different parts of the input, which could be understood as what the model looked at.

But this architecture has some limitations, it requires substantial data and computational resources to achieve significant results on a task. Furthermore, while transformer-based models contain a self-attention mechanism, is the model truly performing attention? Attention should allow the model to efficiently select what is attended and allocate the expensive resources only to those areas. However, transformer-based methods process the entire input in the same manner before deciding how to combine the information and which regions to focus on. This results in a high computational cost that scales exponentially with the input size. An example of this comparison can be seen in Figure 1.

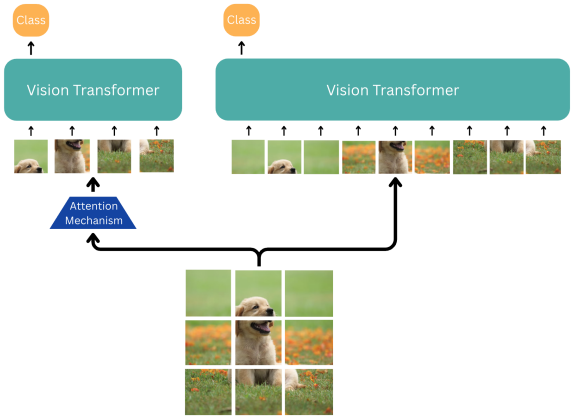


Fig. 1: A comparison between an attention mechanism that uses a simple and efficient module to discard uninformative parts of the image (left), reducing the input size of a computationally expensive model, and a classic Transformer-based method (right) that processes all parts of the input image uniformly.

This work aims to address these challenges in the Computer Vision domain. Inspired by the human visual system, we modify how attention is applied to the image. Instead

• Contact E-mail: joan.lafuente@autonoma.cat
• Supervised by: Dimosthenis Karatzas (Departament de Ciències de la Computació)
• Academic Year 2024/25

of processing the entire image at once, the model sequentially attends to small patches of the image, referred to as fixations. At each step, the model processes one fixation and then predicts the location of the next one, until it has collected enough information to perform the task. This approach allows the model to efficiently handle larger images without an exponential increase in computational cost, enabling it to focus on the most relevant regions.

Building on previous research [3, 4, 5, 6, 7], we propose an end-to-end visual attention model that combines a convolutional neural network (CNN) with a transformer decoder. Our model introduces a novel method for generating fixations, in which the model predicts a relative movement with respect to the previous fixation, which is closer to how the human visual system works. A detailed explanation of the architecture is provided in Section 5.1.

Our main contributions are as follows:

1. We propose an end-to-end architecture with a novel action definition for fixation prediction, inspired by the human visual system.

2. We introduce the use of Actor-Critic reinforcement learning methods in visual attention models, improving the policy optimization process.

3. We show that our method reduces the portion of the image that must be observed for accurate classification in a synthetically generated dataset. Consequently improving efficiency by focusing on the most informative regions.

2 Related Work

This section presents a review of the work related to this project. First, the topic of visual attention is explored, covering recurrent approaches inspired by human vision for efficiency. Next, state space models are reviewed, followed by an analysis of the integration of sequence models, such as Transformers, with reinforcement learning.

2.1 Visual Attention

Recurrent Models of Visual Attention

As input image size increases, computational cost and memory requirements grow rapidly. This raises the question of whether models can adapt to focus only on relevant regions of an image, decoupling computation from input size. To address this, methods inspired by the human visual system have been developed. The human visual system focuses on small, detailed regions of the visual field while perceiving surrounding areas with less detail.

The first work on this topic [3] used Recurrent Neural Networks (RNNs) to process images through sequential fixations, mimicking how humans focus on specific regions. This work framed the problem as a Reinforcement Learning (RL) task, where the image served as the environment and the model as the actor predicting fixation locations. The REINFORCE algorithm [8] was used to train the actor network.

Subsequent RNN-based methods [4, 5, 6] expanded on this idea, though most were tested on the MNIST dataset [9] or similar tasks. For instance, Eslami et al. [5] extended the approach to object detection and introduced Variational Autoencoders for training fixation prediction. Ablavatsk et al. [6] proposed using Spatial Transformers to make the fixation prediction operation differentiable, which was useful in the context of object detection. More recently, Morales et al. [7] introduced the use of Transformers for this task. Their work showed limitations of the REINFORCE algorithm for this task, which might have affected their model performance, indicating the need to explore alternative reinforcement learning approaches. Furthermore, they suggested that a redefinition of the reward function may be required to ensure a better alignment with the model target.

Other Visual Attention Mechanisms

Other visual attention mechanisms have focused on performance improvements of regular models rather than computational cost reduction. In these cases, they incorporate the visual attention mechanisms to add additional information to the original model, not as a single information source. For instance, V* [10] introduced an LLM-guided visual search mechanism that interprets questions and generates efficient visual queries to aid in answering them. Combined with Multimodal LLMs, this forms SEAL (Show, Search, and Tell), a system designed to solve Visual Question Answering (VQA) problems by zooming into image regions likely to contain the answer when the model is uncertain about the it. However, this method still requires processing the entire image before deciding where to focus, resulting in a higher computational cost.

Human Visual Attention

Research on human visual fixations and scan-path prediction [11, 12, 13] provides insights into how humans process visual information. These findings can inform the design of visual attention models, offering a deeper understanding of human-like visual processing.

2.2 State Space Models

State space models (SSMs) provide a structured approach to modeling sequential data by maintaining a latent state that evolves over time through transitions. Recently, Mamba [14] was proposed, introducing input-dependent state updates that enable the model to selectively propagate or forget information. Mamba achieved performance comparable to Transformers, which scales quadratically, while maintaining linear scaling with sequence length.

2.3 Reinforcement Learning

The integration of RL with Transformers has grown significantly in recent years, leveraging Transformers' strengths in RL tasks. Transformer-based agents excel in tasks requiring long-term memory, but struggle with sample efficiency and long-term credit assignment [15].

RL has been widely used to align LLM outputs with user intent, as seen in models like ChatGPT [16]. Additionally, human preferences between trajectory pairs have been used

to solve complex RL tasks without the need of reward functions [17]. Recent advancements, such as GRPO [18], simplify earlier RL methods while significantly improving the reasoning capabilities of LLMs.

However, Transformers face training instability due to residual connections, which amplify noise and parameter perturbations, while supervised learning mitigates this with proper initialization and learning rate schedulers, RL remains unstable [19]. GTrXL [20] addresses this by reordering layer normalization and replacing residual connections with a GRU gating mechanism.

Finally, a variant of Structured State Space Sequence (S5) models for RL [21] has demonstrated strong performance in sequence modeling tasks. S5 models run asymptotically faster than Transformers in sequence length and outperform RNNs in memory-based tasks, offering both efficiency and scalability.

3 Objectives

The project can be divided into four main objectives:

1. **To define a more efficient architecture**: The first objective is to design an architecture and training methodology that allows the model to process an image sequentially, performing recurrent visual attention. This includes defining an action to predict the model fixations.

2. **To refine the defined architecture by solving a simple classification task**: The second objective is to refine the architecture by solving a task in the MNIST dataset using a predefined amount of fixations. This step will ensure a thorough understanding of the model's behavior and capabilities, providing a foundation for scaling to more complex tasks.

3. **To evaluate the architecture in more complex task where exploration is required**: The third objective is to design and solve a more challenging task in which the model must locate an MNIST digit within a larger image containing gradient patterns that guide the search and then classify the digit. This task aims to verify that the model learns meaningful patterns rather than relying on biases or shortcuts, still using a predefined amount of fixations.

4. **To develop the capability to decide when sufficient information for classification has been seen**: The final objective is to add an additional action to the model output that allows it to stop generating further fixations when enough information has been seen. This would reduce the computational cost by allowing the model to stop generating more fixations once it has seen enough information.

These objectives aim to progressively develop and validate the capabilities of the model, ensuring it can manage increasingly complex tasks while maintaining robust performance.

4 Planning

This section provides an overview of how the project has been organized and the work plan followed throughout its development.

4.1 Project Organization

During the execution of this project, an agile methodology has been used for the project organization. This methodology facilitates the decomposition of the project into smaller tasks and is well-suited for projects characterized by high variability and uncertainty, such as the development of Deep Learning models, as it allows for iterative progress and flexibility in response to challenges.

Within the various agile frameworks, *Solo Scrum* is the one that was used. In this framework, the project is divided into sprints, time-boxed periods in which an ordered list of tasks is initially established. At the end of each sprint, the completed work is reviewed and the next steps are determined. In this case, we have executed one-week sprints that ended with a meeting.

4.2 Work Plan

The initial phase of this project involved conducting a literature review, analyzing existing research in visual attention models and related domains. Afterwards, we performed a series of experiments to analyze the behaviors of the different possible architectures on the MNIST dataset. These tasks allowed the definition of an architecture and training methodology capable of effectively addressing this easy task and that theoretically could scale up to a more challenging task.

The next phase of the project involved increasing the difficulty of the task performed by the model. This new task added an additional step of locating the MNIST number within a larger image. See Section 5.2 for a detailed explanation of the data used. After the experimentation in this task, we worked in the addition of a stop action to the model. The last month was dedicated to finalizing the project, writing the report and documentation".

The Gantt chart of the project is available in the Appendix Section A.1.

5 Method

In this section, we explain our proposed method. First, our model architecture is explained, followed by the data and the training methodology used. Finally, we explain the design and formulation of the reward functions that were used during the model training.

5.1 Model Architecture

Our model is a transformer decoder designed for classification, which receives a sequence of fixations as input. These fixations are processed sequentially, and at each step, the model predicts the location of the next fixation it will attend to.

In more detail, the sequence is initialized with a fixation sampled uniformly at random from the image, which

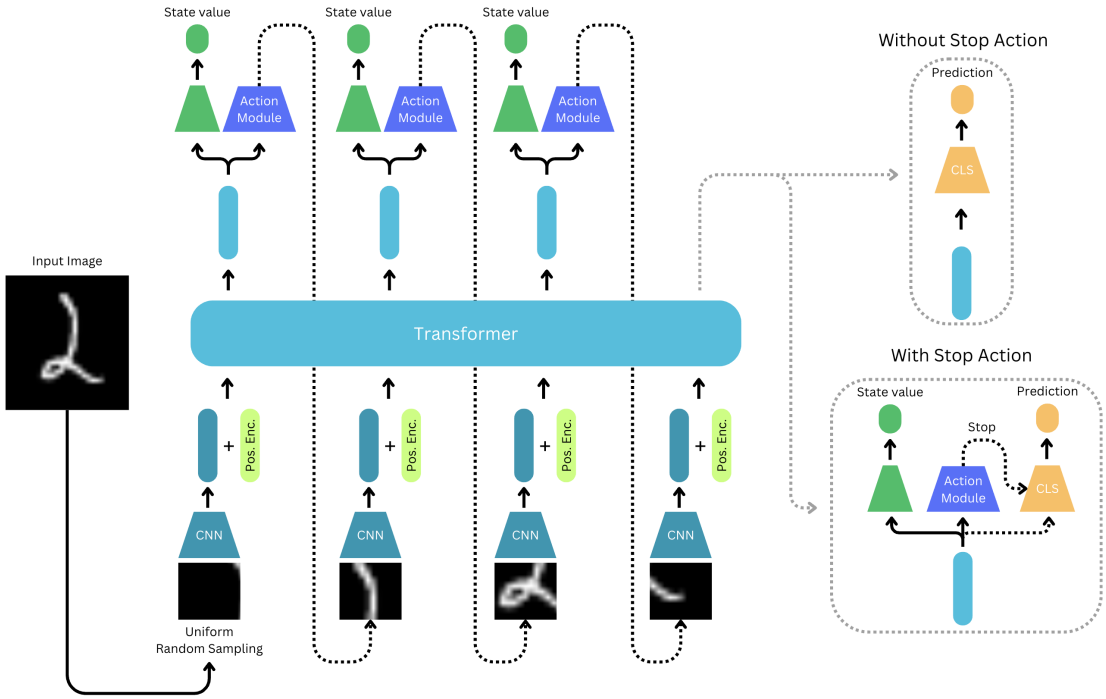


Fig. 2: Overview of the model architecture. The CNN encodes each fixation, which is combined with positional encoding and processed by a transformer decoder. Using the hidden state from the transformer, the action module predicts the next fixation location, the value head estimates the state value, and the classification head produces the final prediction, either after a fixed number of fixations or, when applicable, when the model decides to stop.

provides the initial visual context. This fixation is encoded through a CNN and a three-dimensional positional encoding (x, y, and sequence position) is added.

The positional encoding used is computed using periodic functions and extends the one-dimensional one originally proposed by Vaswani et al. [1]. The x and y encodings are concatenated providing an encoding of the fixation spatial position in the image. Then, the sequence position encoding is added. This allows the Transformer to incorporate both spatial and sequential context, in the experiments done, this approach showed better performance than concatenating or adding the three encodings.

Afterwards, the transformer processes this fixation, and its output is provided as input to the Action Module. The module predicts the location of the next fixation, which is then used to extract the corresponding image patch. This new fixation is subsequently processed with the CNN

This auto-regressive process is repeated iteratively until a predefined maximum number of fixations is reached. Optionally, the action space can be complemented with a stop action that decides when the model should stop generating fixations and perform classification. After processing all fixations in the sequence, a Multi-Layer Perceptron (MLP) classification head produces the final prediction. An overview of the model architecture is shown in Figure 2.

The model also includes a critic network, implemented as an MLP and depicted in dark green in Figure 2. This network estimates the discounted future rewards for the current state based on the hidden state of the transformer. It is required for the Actor-Critic reinforcement learning methods used in this project (see Section 5.3.2 for a detailed explanation).

5.1.1 Action Module

The action module is responsible for generating new fixations for the model and, optionally, deciding whether additional fixations are needed. Each fixation is determined by a movement vector predicted in polar coordinates, representing the transition from the center of the previous fixation to the next one.

This vector is generated by an MLP, which receives as input the hidden state of the transformer corresponding to the previous fixation. The MLP outputs two continuous values: the angle (α) and the distance. Together, these values define a vector in polar coordinates, which is then used to determine the location of the next fixation in the image space. Additionally, when the model includes the capability to stop, it predicts a third value in the range $[0, 1]$, indicating if additional fixations are needed. An overview of this process is presented in Figure 3.

A final detail to note is that the model might predict fixations located outside the image boundaries. To handle this case, the model is designed to receive black pixels at fixation pixels that fall outside the valid image area.

Within the RL framework used in this project, fixations are treated as states (s), while the predicted movement vector and, when applicable, the decision to continue or stop are treated as actions (a). Consequently, the action module serves as the policy network (π_θ), mapping an encoded representation of the current state to an action.

5.2 Data

In this project, the MNIST dataset and some synthetically generated variations from it have been used. MNIST is a

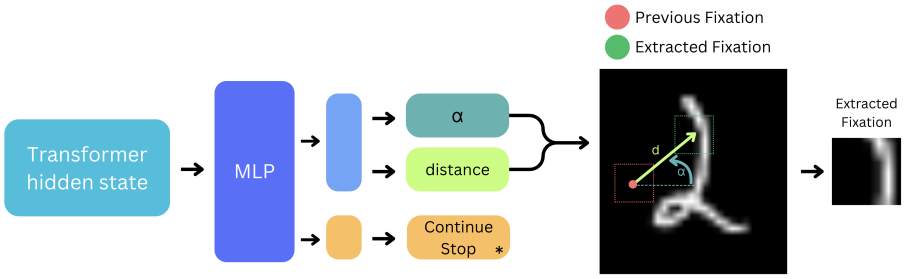


Fig. 3: Architecture of the action module generating movement vectors in polar coordinates from the transformer's hidden state. Optionally, the module predicts a third value, indicated by "*", representing whether to continue generating fixations or to stop. Red and green squares indicate the previous and next fixations, respectively.

simple dataset composed of handwritten images of numbers. The original version of MNIST is a great option to check that all components are working correctly but can be a very simple task for this reason. Two variations of this dataset have been made, which produce a more complex task for the model.

5.2.1 Resized MNIST

The first variation, Resized MNIST, involves generating a new 50x50 black image with an MNIST digit placed at a random location. Unlike the original MNIST, where the digits are centered, here the digits can appear anywhere within the image, requiring the model to actively explore the image to locate the digit before classifying it. An example image is shown in Figure 4.

5.2.2 Resized MNIST with Spatial Information

This variation builds upon the Resized MNIST dataset by incorporating global spatial information about the digit location at each fixation. This is achieved by overlaying a surface based on a Gaussian distribution centered on the digit position. To increase the variability of the samples, the covariance matrix of the Gaussian distribution is different for each instance, the standard deviations along each axis are independently sampled from a uniform distribution over $[0.25, 1]$, and the correlation coefficient ρ is sampled from a uniform distribution over $[-0.4, 0.4]$. Points near the digit have a value close to 1 (depicted as black), while points farther away approach a value of 0 (depicted as white). The Gaussian nature of the surface creates a smooth gradient from black to white, providing visual information about the digit location relative to the current fixation. An example image is shown in Figure 4.

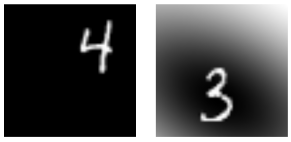


Fig. 4: Example images from Resized MNIST (left) and Resized MNIST with Spatial Information (right) datasets.

5.3 Learning Algorithm

The algorithm used to train our model is largely based on RL. This approach is necessary because the model must learn how to locate fixations within the image, and there is no ground truth specifying the "best" sequence of fixations. However, the model is not trained solely with RL, as we do have ground truth information about the final classification that must be performed for each image. This results in a hybrid loss function that combines supervised learning and reinforcement learning. This loss function is explained in detail in Section 5.3.2.

5.3.1 Classification Pretraining

Reinforcement learning can be unstable and computationally inefficient, as it requires many interactions with the environment to learn the optimal policy. To address these issues, we defined a supervised learning pretraining step in which the model learns to classify the objects present in the images using provided sequence of fixations. This pretrained model then provides a more informative signal to the RL algorithm when learning to predict fixation locations.

Since ground truth data for "good" fixations is not available, and this data is needed to train the classification model using fixations, we use the available object location information instead. This allows us to generate "good" trajectories, meaning sequences of fixations that contain enough information to classify the object present in the image. These trajectories were generated in three main ways, each consisting of four fixations crossing the object:

- **Line:** The fixations are generated along a vertical or horizontal line, with the center aligned over the object's center.
- **Diagonal:** The fixations are generated across a diagonal line crossing the object, either from top left to bottom right or from top right to bottom left, centered on the object.
- **Square:** The fixations are positioned to form a square centered on the object.

It is important to note that within each generated trajectory the fixations were shuffled to generate a more diverse set of possible trajectories. See Figure 5 for a visualization of the different sampling strategies used.

At this point, the model could be trained to perform classification, but its confidence scores might not be reliable since it would only learn to select one of the possible categories, without considering that a trajectory might not contain any object. To address this potential issue, in addition to learning the classification task, the model must also learn to predict low confidence scores when a given trajectory

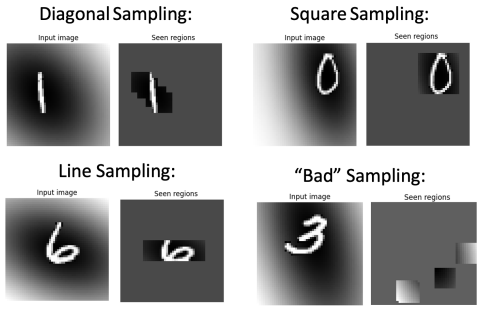


Fig. 5: Visualization of the fixation sampling strategies used during pretraining. Line sampling can also occur vertically, and diagonal sampling can occur along the opposite diagonal (starting from the top-right part of the object).

does not provide enough information to perform a correct classification.

To achieve this, we generate "bad" trajectories from the same set of images by sampling fixations far from the objects. The model is then trained to maximize the entropy (see Equation 2) of the classifier's output, hence maximizing uncertainty by flattening the output logits, when these "bad" trajectories are presented.

The loss function is defined as follows:

$$\mathcal{L}_{\text{good}} = \frac{1}{N_{\text{good}}} \sum_{i=1}^{N_{\text{good}}} \left(-\sum_{j=1}^{K} y_{ij} \log(\hat{y}_{ij}) \right) \tag{1}$$

$$\mathcal{H} = \frac{1}{N_{\text{bad}}} \sum_{i=1}^{N_{\text{bad}}} \left(-\sum_{j=1}^{K} P_{ij} \log(P_{ij}) \right) \tag{2}$$

$$\mathcal{L}_{\text{pretraining}} = \mathcal{L}_{\text{good}} - \alpha \cdot \mathcal{H} \tag{3}$$

where N_{good} is the number of trajectories showing a number, N_{bad} is the number of trajectories which do not show the object, K is the number of classes and α is a scaling factor for the entropy in the loss function.

5.3.2 Loss Function

In this section, there is an overview of each of the components of our loss function and their task when optimizing the model.

Actor Loss

The actor loss ($\mathcal{L}_A$) is used to train the policy responsible for generating the model fixations. In this project, we adopt actor-critic methods, which integrate the strengths of both policy-based (actor) and value-based (critic) approaches. This methods perform simultaneous learning of both the policy and the value function. Compared to purely policy-gradient methods such as REINFORCE, actor-critic approaches benefit from a more stable learning signal provided by the critic, leading to improved training dynamics.

Specifically, the actor-critic methods used optimize the policy π_θ by maximizing the expected advantage of actions, where the advantage quantifies how much better an action is compared to the average value of the current state, considering the actual policy. The advantage is estimated using the value function predicted by the critic, as defined in Equation 4. This approach encourages the selection of actions that are expected to yield higher long-term rewards.

We evaluated two actor-critic methods in this work: Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO).

$$A_t = r + \gamma \cdot V(s_{t+1}) - V(s_t) \tag{4}$$

Here, s_t is the state in the time step t, s_{t+1} is the next state after taking action a_t in s_t, r is the reward received, γ is the discount factor, and $V(s)$ is the estimated value of the state s.

Advantage Actor-Critic (A2C) is an on-policy actor-critic algorithm in which the actor updates the policy π_θ to favor actions with higher estimated advantages. To limit the instability of policy updates, in A2C, the advantages have been normalized using Equation 5 before computing the actor loss in Equation 6.

$$\hat{A}_t = \frac{A_t - \mu_A}{\sigma_A + \epsilon} \tag{5}$$

where μ_A and σ_A are the mean and standard deviation of the advantages over the batch, and ϵ is a small constant for numerical stability.

$$\mathcal{L}_A = -\frac{1}{T} \sum t = 1^T \log \pi_\theta(a_t|s_t) \hat{A}_t \tag{6}$$

where $\hat{A}_t$ is the normalized advantage and T is the number of time steps.

Proximal Policy Optimization (PPO) is an on-policy actor-critic algorithm designed to improve training robustness by limiting how much the policy can change in a single update. PPO introduces a clipped surrogate objective to ensure that updates remain within a bounded region, theoretically guaranteeing convergence.

The actor loss for PPO is defined as follows:

$$\mathcal{L}_A = -\frac{1}{T} \sum_{t=1}^{T} \min\left(r_t A_t,\ \text{clip}(r_t,\ 1-\epsilon,\ 1+\epsilon)\, A_t\right) \tag{7}$$

where $r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies, and ϵ is a clipping hyperparameter.

Critic Loss

Actor-critic methods use a critic network to estimate the value of each state, which represents the expected discounted sum of future rewards from that state.

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s \right] \tag{8}$$

where $V^\pi(s)$ represents the discounted sum of future rewards, starting from state s and following policy π.

The critic network is trained to minimize the difference between its predicted state values and the corresponding target values. In the reinforcement learning algorithms implemented in this work, the critic loss ($\mathcal{L}_C$) is defined as the mean squared error (MSE) between the predicted state values and their respective target values, as shown in Equation 9.

$$\mathcal{L}_C = \frac{1}{T}\sum_{t=1}^{T}(r + \gamma * V(s_{t+1}) - V(s_t))^2 \tag{9}$$

where s_t is the state at time step t, s_{t+1} is the state reached after performing an action a in the state s, r is the reward received for an action a performed in the state s, $V(s)$ is the predicted state value for state s, γ is the discount factor, and T is the total amount of steps done.

Classification Loss

The classification head is trained using the Cross-Entropy loss ($\mathcal{L}_{CE}$). During training, classification is performed at the final step of the trajectory generated by the policy when the stop action is not used. If the stop action is used, classification is performed at the time step in which the model predicted to stop generating fixations.

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}\log(\hat{y}_{ij}) \tag{10}$$

where N is the number of samples, K is the number of classes, y the ground truth and $\hat{y}$ the model prediction.

Loss Function Used

The final loss function used to train the different models is composed of the addition of the different individual losses, as shown in Equation 11.

$$\mathcal{L} = \mathcal{L}_A + \mathcal{L}_C + \mathcal{L}_{CE} \tag{11}$$

5.4 Rewards

To train a model using reinforcement learning, it is necessary to define a set of rewards that indicate the agent goal within the environment. These rewards guide the agent in learning the actions needed to achieve that goal. The following reward functions have been used in this project:

Classification Reward. The model receives a reward at the classification step (cls. state) of the episode if it performs a correct classification. This reward is used in all the experiments conducted. The value of the reward corresponds to the softmax classification confidence (conf) of the predicted class. Optionally, and only for models with the stop capability, a mistake penalty (λ_m) is applied when the model makes an incorrect classification.

$$\mathrm{R}(s) = \begin{cases} max(\mathrm{conf}(s)) & \text{if cls. state \& correct} \\ -\lambda_m & \text{if cls. state \& incorrect} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Surface Reward. In the Resized MNIST with Spatial Information dataset, a surface pointing towards the number location was defined. On this surface, the value is 0 when a point is far from the number and smoothly increases to 1 as it gets closer to it (see Section 5.2.2 for a detailed explanation).

To provide a more explicit guidance, a reward was defined based on this surface information. The reward corresponds to the difference in the mean surface value ($\mathrm{msv}(s_t)$) within the fixation region (s) compared to the previous state (s'). If this difference is negative, the length of the predicted vector ($\hat{d}$) is added to the reward to encourage exploration when the agent movements are not directed to the number.

$$\mathrm{R}(s) = \begin{cases} \mathrm{VD}(s,s') & \text{if } \mathrm{VD}(s,s') \geq 0 \\ \mathrm{VD}(s,s') + \hat{d}\cdot\lambda_d & \text{otherwise} \end{cases} \tag{13}$$

where $\mathrm{VD}(s,s') = \mathrm{msv}(s) - \mathrm{msv}(s')$

Variance Reward. In the case where no surface is added to provide general information about the number location, a reward was defined to inform the agent when a number has been reached. This reward is defined as the difference in the mean variance (fv) of the current fixation compared to the preceding one.

$$\mathrm{R}(s) = \mathrm{fv}(s) - \mathrm{fv}(s') \tag{14}$$

Confidence Reward. This reward was defined to provide an insight into the generated fixations without explicitly explaining how the task is solved, unlike the Surface Reward. It is based on the variation in confidence of the most probable category from the classification head outputs at each step. This approach makes use of the knowledge gained during model pretraining.

The reward gives feedback to the model by indicating whether each fixation has provided valuable information to the classification head, evaluating each generated fixation.

$$\mathrm{R}(s) = abs(max(\mathrm{conf}(s)) - max(\mathrm{conf}(s'))) \tag{15}$$

Overlap Penalty. This penalty was defined to discourage fixations that significantly overlap, which increases computational cost without providing useful information. It allows partial overlap with previous fixations without penalizing the model. However, when the overlap exceeds a threshold ($\tau_{\max}$), a penalty is applied. The penalty increases linearly and reaches a defined maximum λ_o when there is total overlap.

$$\mathrm{R}(s) = \begin{cases} 0, & \text{if } \mathrm{ov}(s,S) \leq \tau_{\max} \\ -\lambda_o \cdot \frac{\mathrm{ov}(s,S)-\tau_{\max}}{1-\tau_{\max}}, & \text{otherwise} \end{cases} \tag{16}$$

where S means the set of states previously seen, s the current state, $\tau_{\max}$ the maximum overlap possible without receiving a penalty and ov is a function which computes the overlap between s and S.

Step Penalty. This penalty is designed for models capable of stopping and introduces the concept that each additional fixation produces a cost. It encourages the model to minimize the number of fixations used during training. It is defined as a constant penalty applied at each step.

$$\mathrm{R}(s) = -0.1 \tag{17}$$

Budget Penalty. This penalty is also intended for models that are capable of stopping. It allows the model to be aware of the maximum number of fixations it is allowed to generate. A penalty is applied if the model reaches this maximum without predicting to stop before. In all other cases, the penalty is zero. This encourages the model to learn when to stop before exhausting the fixation limit.

$$R(s) = \begin{cases} -\lambda_b, & \text{if } t == N_{\text{max}} \text{ and } \text{stop}_t == 0 \\ 0, & \text{otherwise} \end{cases} \quad (18)$$

where λ_b is the budget penalty coefficient, t is the current time step in the sequence (equivalent to the number of fixations seen), N_{max} is the maximum number of allowed fixations, and $\text{stop}_t \in \{0, 1\}$ indicates whether the stop action was taken at the time step t.

6 Number Localization and Recognition

In this task, the model must first locate an MNIST digit within a larger, resized image before classifying it. Two variations are considered: one with background surfaces providing global spatial cues about the digit location, and another with a plain black background, where the model must explore without guidance.

This section presents the experiments and results of our method for number localization and recognition, including the experiments done on MNIST to define the proposed action module, and pretraining results.

6.1 Action Definition

The first step in this project was to define how the model predicts the regions to focus on (fixations). To explore different possible action types, we used the MNIST dataset, which offers a suitable environment to test the feasibility of the model. In these experiments, no pretraining was performed, and PPO was used as the actor loss. Moreover, the model also predicted the initial fixation location from a start token. Among the different reward functions described in Section 5.4, only the classification reward was used.

We experimented with three different types of actions. In the first approach, the model predicted the x and y coordinates of the next fixation separately. This prediction could be made by either classifying within a discrete set of positions for each axis or by producing continuous values. However, this approach often got stuck in local minima when trained on the MNIST dataset. Even when noise was added, the model tended to fix the y coordinate approximately in the middle of the image and only varied the x coordinate. We hypothesized that this behavior might be due to the x dimension being more informative than y. To test this, we rotated the input images and observed that the model then fixed the x coordinate and moved only along y. Figure 6 illustrates this behavior.

To address this issue, a second action type was defined, where instead of predicting x and y coordinates separately, the model selected a fixation location from a list of possible combinations of x and y. This approach resolved the previous problem but introduced new drawbacks. Both action types required retraining the model if the input image size changed. Additionally, this second action type was computationally expensive because the number of possible fixation locations grows exponentially with the image size.

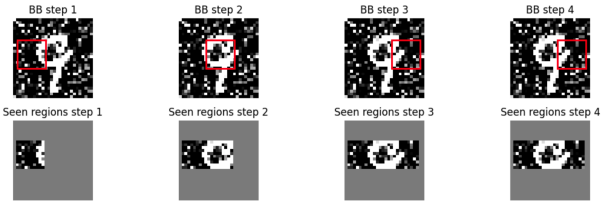


Fig. 6: Visualization of the model behavior which used independent prediction of x and y. The first row shows the predicted fixation at each step, and the second row shows the accumulated fixations the model has seen at each step.

To overcome these limitations, we desigend a third action type where the model predicts a relative movement from the current fixation position to the next fixation, rather than an absolute position. Section 5.1.1 provides a detailed explanation of this approach. This relative movement action is more generalizable, as it should work across different input image sizes without retraining the model. Moreover, since this task is more challenging, it encourages the model to learn meaningful movement strategies instead of relying on trivial solutions.

A visual comparison of the different action types explored is shown in Figure 7.

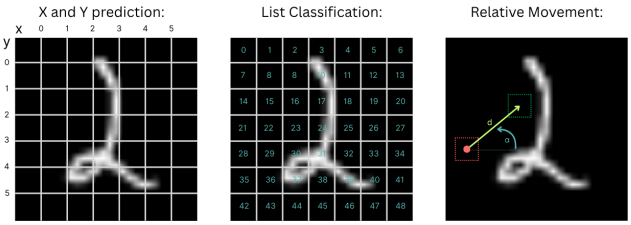


Fig. 7: Visualization of the different types of actions explored for fixation prediction.

6.2 Classification Pretraining

Before training the full visual attention model, we first pretrain the model, except for the action module. This ensures that, given an informative set of fixations, the model can correctly classify the digit. The complete details of the training procedure are described in Section 5.3.1.

Table 1 presents the results of the classification pretraining step, which can serve as an approximate upper bound for task performance when using fixation-based inputs. The results indicate that the model is capable of learning to classify the digits using the generated fixation trajectories. Also, it has been seen that the scale factor for entropy, α, does not significantly affect the model performance.

Analyzing qualitative examples, we observe that the model has learned to produce reliable classification confidence scores, which was the intended outcome of incorporating entropy into the training process. When the model was provided with uninformative fixations and performed classification, it predicted low confidence scores (see Figure 13, located in the Appendix). On the other hand, when the

N° Good Trajectories	N° Bad Trajectories	Scale Entropy	Acc.
512	512	0.25	96.61
512	512	0.50	96.51
512	512	0.75	96.49
512	512	1.00	**96.68**

TABLE 1: Results of the pretraining step on the Resized MNIST with Spatial Information dataset. Accuracy is reported only for trajectories with "good" fixations.

fixations provide enough information to perform the digit classification, the model predicts high confidence scores (see Figure 12, located in the Appendix).

6.3 Ablation Study

This section presents an ablation study to analyze the contribution of different components of our method. We used the generated variations of the MNIST dataset to compare the effects of the reinforcement learning algorithm, pretraining, reward function, and the presence of global image information. The results, shown in Table 2, show the role each component plays in the model performance and behavior. The classification reward was used in all experiments.

PPO vs A2C. A comparison between the two reinforcement learning algorithms used can be seen in Table 2. The results show that A2C outperforms PPO, achieving higher accuracy (95.73% vs. 94.34%). The performance gap becomes larger when examining the total reward accumulated by the learned policies. PPO collects a reward of 3975.73 across the 10,000 evaluation examples, while A2C achieves a significantly higher total reward of 5307.29.

In addition of the improvements in terms of accuracy and collected rewards, A2C also results in better trajectory behavior. The predicted actions show greater variability and adaptation to the current fixations seen. For instance, the policy learned using A2C is capable of making larger movements when no useful information is seen in the previous fixation.

These differences in behavior and performance might be caused by the distinct characteristics of each algorithm. PPO makes constrained updates at each step, this leads to a more stable behavior, but in our case, this update rule might have caused the policy to get stuck in a local minimum.

Pretraining Impact. Table 2 illustrates the impact of pretraining the model for classification. Pretraining leads to a notable improvement in final model performance, likely because it enables a more informative and stable reward signal since the beginning of the training.

During the early stages of training, when the model is pretrained, informative fixation trajectories are more likely to yield correct classification rewards. This is because the model has already learned to classify based on fixation sequences, allowing it to produce more reliable and informative confidence scores. As a result, if the fixations are sufficient to identify a digit, the model is more likely to classify it correctly. In contrast, a model trained from scratch may fail to do so, even if a near-optimal trajectory occurs, leading to a noisier and less stable reward signal.

Rewards effect. To scale up to more complex tasks where complete knowledge of the images is not available, the reward function should avoid explicitly guiding the model on how to solve the task. For this reason, it is valuable to evaluate the impact of removing the surface reward, forcing the model to learn the task without direct information about the solution.

Among the results shown in Table 2, three models are of particular interest for this comparison. These models were trained with identical parameters but differ in their reward functions: one uses the surface and overlap rewards, another uses the confidence reward, and a third uses neither.

The results demonstrate that the model can still solve the task without the surface reward, with only a minor decrease in accuracy. The model trained only with the classification reward achieves an accuracy of 92.49%, which is close to the 95.73% accuracy achieved when the surface reward is included. In contrast, the model trained with the confidence reward exhibits a larger drop in performance, reaching only 89.39% accuracy. This suggests that, in this context, the confidence reward does not provide useful insights and may even degrade the quality of the reward signal.

Qualitatively, removing the surface reward affects the generated fixation trajectories. The model struggles more to learn the image boundaries and is more likely to produce fixations outside the image. When this behavior occurs, usually several fixations have already been made over the digit, and consequently the model already has sufficient information for classification.

Overall, these results show that the model can learn to solve the task without an explicitly encoded strategy in the reward function. This makes it easier to extend the approach to other tasks where the solution cannot be explicitly defined.

Spatial Information Impact. In addition to the reward variation experiments, it is also important to evaluate the effect of removing the background surface, which provides global context about the digit location in each fixation. In this scenario, the model must learn to explore the image and locate the digit without any positional cues. This increases the task complexity and tests the robustness of our method under different conditions.

Table 2 presents a comparison between models trained with and without the surface present in the image. When the spatial information is removed, eliminating the Surface reward without introducing additional rewards, the model still achieves an accuracy of 92.37%, almost the same accuracy obtained as in the equivalent setup with the spatial information. We also observe that adding alternative rewards, such as the confidence or variance rewards, negatively affects performance, as was also seen when using spatial information in each fixation.

These results show that our method succeeds in a task requiring a different behavior, and that the simplest reward function, based solely on the final classification objective, leads to the best performance. This suggests that our method could scale to more complex scenarios, which could be explored in future research.

Dataset with Spatial Information	Classification Pretraining	RL Algorithm	Surface Reward	Variance Reward	Confidence Reward	Overlap Penalty	Acc.
✓	✗	A2C	✓	✗	✗	✓	90.20
✓	✓	A2C	✓	✗	✗	✓	**95.73**
✓	✓	PPO	✓	✗	✗	✓	94.34
✓	✓	A2C	✗	✗	✓	✗	90.82
✓	✓	A2C	✗	✗	✗	✗	92.49
✗	✓	A2C	✗	✗	✗	✗	**92.37**
✗	✓	A2C	✗	✗	✓	✗	91.44
✗	✓	A2C	✗	✓	✗	✗	90.02

TABLE 2: Ablation study in the Resized MNIST with and without spatial information datasets. Model configuration, reward functions, and hyperparameters not shown in the table are fixed across all experiments. All experiments used the classification reward and a fixed amount of 6 fixations.

6.4 Comparison With Baseline

In the Resized MNIST with Spatial Information dataset, a final comparison was conducted against a baseline model that used randomly positioned fixations. This random fixation strategy establishes a lower bound for performance, as the model lacks any knowledge about how the fixations should be located.

In Table 3, we observe that our RL-based method outperforms the baseline on the Resized MNIST with Spatial Information dataset. This suggests that the model has learned to leverage the information from the fixations to generate new ones that are better focused on the digit region.

Fixation Generation	Accuracy
Random	64.25
RL-Based	**95.73**

TABLE 3: Performance comparison in Resized MNIST with Spatial Information dataset

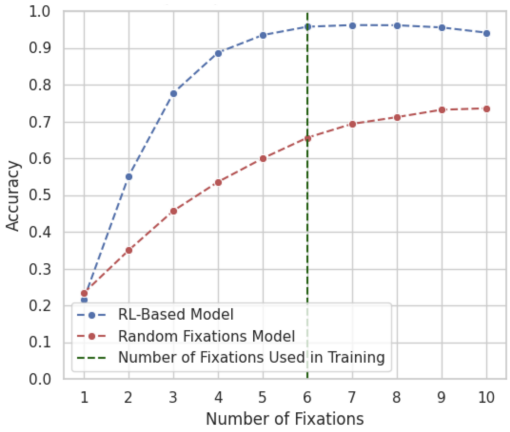


Fig. 8: Performance comparison across different fixation steps between an RL-based model and a model using randomly positioned fixations on the Resized MNIST with Spatial Information dataset.

This behavior is also seen when comparing model performance at each fixation step for both approaches (see Figure 8). For this comparison, we evaluated the classification head using the hidden state of the transformer corresponding to each fixation. Both models perform similarly on the first fixation, which is expected since it is randomly initialized in both cases. However, the performance of the RL model quickly surpasses the one of the baseline, as it has learned to locate fixations in regions containing more relevant information for the classification task.

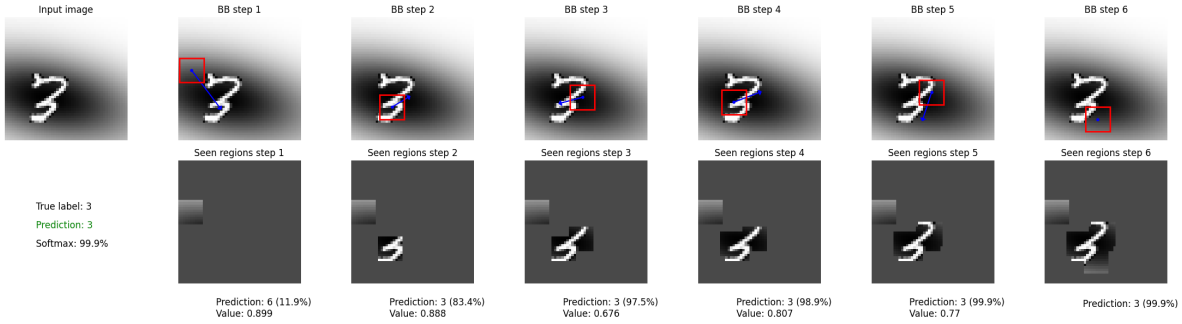
These results demonstrate that our method can effectively focus on the most informative parts of the image, enabling correct classification without the need to observe the entire image, which reduces the computational resources spent on less informative regions.

6.5 Qualitative Results

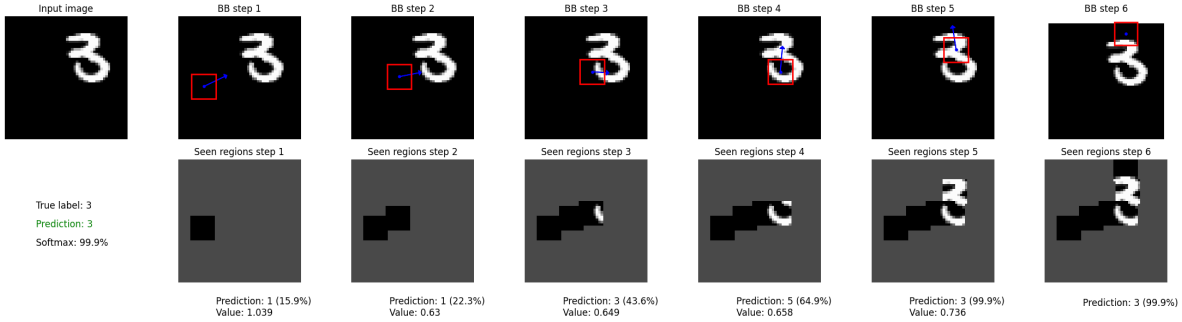
This section presents a qualitative analysis of the best-performing model on the Resized MNIST and Resized MNIST with Spatial Information datasets.

Resized MNIST with Spatial Information. When spatial information is provided through the surfaces, the model makes large jumps between fixations when the current fixation contains little useful information for classification, and smaller, more focused steps when the digit is visible, enabling it to gather the detailed information needed for accurate classification. Additionally, the model predicts low confidence scores when previous fixations lack sufficient information to confidently classify a digit, preserving the knowledge gained during the pretraining step. Similarly, once enough information is collected, the confidence score increases rapidly. An illustration of this behavior can be found in Figure 9a.

Resized MNIST. When the model must learn to perform a pure search without any additional spatial information, as the background surface is, a different behavior emerges. In this case, the model tends to predict only small movements between fixations, even when the current fixation contains no useful information (such as a completely black area). Additionally, the generated trajectories tend to move toward the center of the image, where it is more likely to observe a part of a digit. Similarly to the model trained with spatial information, the model predicts low confidence scores until sufficient information is collected to accurately classify the digit. An example of this behavior can be seen in Figure 9b.



(a) Example of classification and fixations predicted at each step in a Resized MNIST with Spatial Information dataset.



(b) Example of classification and fixations predicted at each step in a Resized MNIST without spatial information dataset.

Fig. 9: Examples of classification and fixation predictions from the best-performing model on each dataset. In the upper row, the fixation locations and predicted movement vectors at each step can be seen. In the second row, the final prediction is shown, along with the accumulated parts of the image observed and the intermediate predictions. (a) Resized MNIST with Spatial Information dataset. (b) Resized MNIST dataset.

7 Incorporating the Stop Action

In the previous task, our model was able to correctly identify and classify the digits using a predefined amount of fixations. However, were all fixations truly necessary? For instance in Figure 9a, the model had already collected enough information by the third fixation to confidently predict the correct digit, although three additional fixations were generated. This raised the question of whether the model could learn to recognize when it had gathered enough information. As a result, its efficiency could be improved by avoiding unnecessary fixations.

7.1 Ablation Study

This section presents an ablation study on models that include the stop action, evaluated on the Resized MNIST with Spatial Information dataset. It evaluates the effect of using an already trained model for number localization and recognition, consequently only needing to learn the stop action, and the impact of different mistake penalty values (λ_m) in the classification reward function.

These experiments used the classification reward, budget penalty, and step penalty. These reward functions were not used in previous experiments, as they were not required.

Training Strategy Impact. Two strategies for training models which had stop action were compared:

1. **Jointly train fixation generation and stop action**, starting from a pretrained checkpoint. This results in a two stage learning, first, the model is pretrained for classification, and then the policy is learned. This follows the same approach that was used in Section 6.

2. **Train only the MLP layers responsible for the stop action**, while keeping the rest of the pretrained model frozen. This involves a three-stage process: classification pretraining, fixation prediction training, and finally training the stop action.

In the second approach, all the pretrained model weights are frozen, and only the layers needed for the stop action

Independent Stop Training	Mistake Penalty	Mean Used Fixations	Acc.
✗	0.0	1.00	22.03
✗	1.5	2.14	64.96
✗	3.0	2.80	74.35
✗	4.5	2.93	78.20
✗	6.0	5.89	21.47
✓	0.0	1.00	21.32
✓	1.5	2.75	67.61
✓	3.0	3.54	80.99
✓	**4.5**	**4.00**	**85.74**
✓	6.0	8.76	91.69

TABLE 4: Ablation study of the stop action on the Resized MNIST with Spatial Information dataset. Model configuration, reward functions, and hyperparameters not shown in the table were fixed across all experiments. During evaluation, models were allowed up to 10 fixations, where fixation generation stopped regardless of the model prediction.

are learned. This setup isolates the stop action and trains it independently, without interfering with the already learned fixation generation and classification policies. As the pre-trained model, we used the model previously trained for number localization and recognition which used only the classification reward, and achieved an accuracy of 92.49% (see Table 2).

Table 4 shows that training only the stop action in a frozen model leads to slightly better performance. Jointly training fixation generation and the stop action introduced instability, with performance and convergence speed varying significantly across different random seeds. However, in some cases, such as when $\lambda_m = 1.5$ or $\lambda_m = 3.0$, the joint training approach achieved similar performance while requiring fewer fixations on average, as it learns to generate fixations conditioned on minimizing the amount used.

Mistake Penalty. Incorporating the stop action enables the model to finish the fixation generation once enough information has been collected. In this setting, penalizing incorrect classifications becomes essential. Without such a penalty, the model tends to exploit a suboptimal strategy, always predicting "stop". This behavior minimizes the step penalty received and occasionally results in a correct prediction by chance, and consequently its reward. This happens when the initial fixation is located on top of the number and contains enough information for classification. This issue might not occur in more complex tasks, where accurate classification depends on aggregating information from different regions of the image.

Introducing a mistake penalty helps mitigate this behavior by encouraging the model to gather more information before deciding to stop. As shown in Table 4, for both training setups increasing the penalty results in a higher average number of fixations. Therefore, the mistake penalty serves as a tuning mechanism to balance computational cost against classification accuracy.

However, this reward function interferes with the rest of rewards. When jointly training both the relative movement and stop actions, increasing its weight too much prevents the model from accurately learning how to predict the next fixation location. On the other hand, when training the stop action independently, the model fails to learn when to stop, as it continuously tries to gather more information to minimize the probability of mistakes.

7.2 Qualitative Results

For the qualitative analysis, we used the model trained with the stop action independently and a mistake penalty (λ_m) of 4.5 (highlighted in bold in Table 4), as it achieved the best performance within the fixation budget.

When examining the model's behavior in various examples, we observe that it is generally able to predict when to stop and correctly classify the digit as soon as it has identified a distinctive part of the number. This behavior is illustrated in Figure 10, where the model decides to stop generating fixations once it has seen enough information for classification.

However, the model can sometimes predict to stop too early. For instance, in Figure 16c, located in the Appendix, the model observes a portion of the digit 6 that resembles a 4 and stops, resulting in an incorrect classification. These early stopping cases may explain the performance drop when using the stop action (85.74%) compared to using a fixed number of six fixations (92.49%).

Additional examples can be found in the Appendix Section A.4.

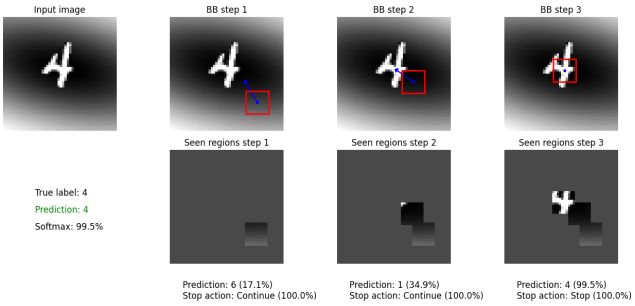


Fig. 10: Example of the model trained with the stop action independently and a mistake penalty (λ_m) of 4.5, model with stop action in the Resized MNIST with Spatial Information dataset. Confidence values are rounded to a single decimal place.

8 Conclusions

In this work, we propose a novel end-to-end architecture for visual attention using transformers, in which the fixation generation has been redefined with respect to previous work, aligning more to human visual system. Our method is capable of performing well on a variation of the MNIST dataset where before classification it must locate the number.

Furthermore, we show that the model learns to focus on informative regions of the image across the datasets used, consequently reducing the computational resources spent processing not informative regions. In addition, the model is capable of predicting when it has gathered enough information and does not require any additional fixations, which can reduce the computational cost at the expense of a small performance drop.

Further work could explore the use of more complex tasks, where the model must attend to multiple regions of the image. This would allow testing the method in scenarios closer to real-world applications, where it is often necessary to focus on more than one area of the image to fully understand its content. Moreover, new reward functions could be explored to ensure that learning the stop action does not interfere with learning the action responsible for generating new fixations.

Acknowledgments

I would like to express my sincere gratitude to Dimosthenis Karatzas, Artemis Llabrés and Carlo Romeo for their support and guidance during the development of the final degree project. I would also like to thank the Computer Vision Center for the Elena Maseras research fellowship and the resources provided.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[3] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/3e456b31302cf8210edd4029292a40ad-Paper.pdf

[4] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.7755

[5] S. M. A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, K. Kavukcuoglu, and G. E. Hinton, "Attend, infer, repeat: Fast scene understanding with generative models." in *NIPS*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 3225–3233.

[6] A. Ablavatski, S. Lu, and J. Cai, "Enriched deep recurrent visual attention model for multiple object recognition," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017, pp. 971–978.

[7] J. M. Casas, "Fixation-guided visual attention models with transformers," 2024.

[8] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning." 1992.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] P. Wu and S. Xie, "V*: Guided visual search as a core mechanism in multimodal llms," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024, pp. 13 084–13 094.

[11] M. Qiu, Q. Rong, D. Liang, and H. Tu, "Visual scanpath transformer: Guiding computers to see the world," in *2023 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2023, pp. 223–232.

[12] Z. Yang, L. Huang, Y. Chen, Z. Wei, S. Ahn, G. Zelinsky, D. Samaras, and M. Hoai, "Predicting goal-directed human attention using inverse reinforcement learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[13] Y. Jiang, Z. Guo, H. R. Tavakoli, L. A. Leiva, and A. Oulasvirta, "Eyeformer: Predicting personalized scanpaths with transformer-guided reinforcement learning," 2024. [Online]. Available: https://doi.org/10.1145/3654777.3676436

[14] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," in *First Conference on Language Modeling*, 2024. [Online]. Available: https://openreview.net/forum?id=tEYskw1VY2

[15] T. Ni, M. Ma, B. Eysenbach, and P.-L. Bacon, "When do transformers shine in rl? decoupling memory from credit assignment," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 50 429–50 452. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/9dc5accb1e4f4a9798eae145f2e4869b-Paper-Conference.pdf

[16] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Gray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=TG8KACxEON

[17] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf

[18] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *CoRR*, vol. abs/2402.03300, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2402.03300

[19] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B1DmUzWAW

[20] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, M. Botvinick, N. Heess, and R. Hadsell, "Stabilizing transformers for reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 7487–7498. [Online]. Available: https://proceedings.mlr.press/v119/parisotto20a.html

[21] C. Lu, Y. Schroecker, A. Gu, E. Parisotto, J. N. Foerster, S. Singh, and F. Behbahani, "Structured state space models for in-context reinforcement learning," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=4W9FVg1j6I

APPENDIX

A.1 Gantt Chart

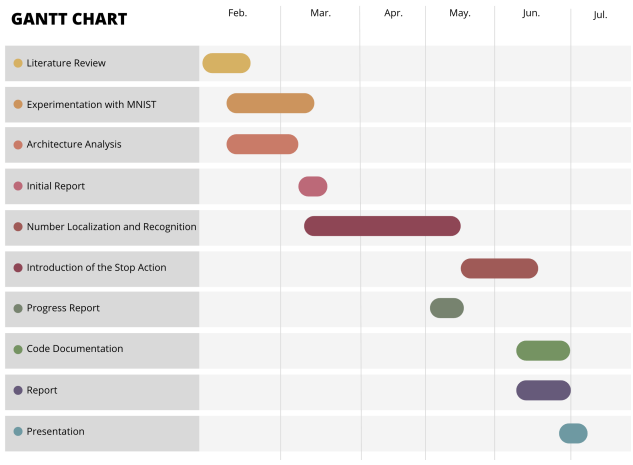


Fig. 11: Gantt chart of the project.

A.2 Pretraining Examples

Figures 12 and 13 show examples of classification by the pretrained model, one using a set of fixations that contains enough information for accurate classification and another where the fixations are insufficient.

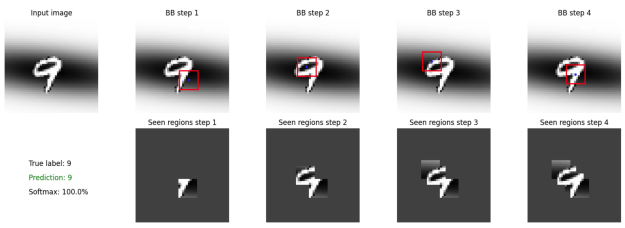


Fig. 12: Example of the pretrained model prediction given a set of fixations which allow to correctly classify the number.

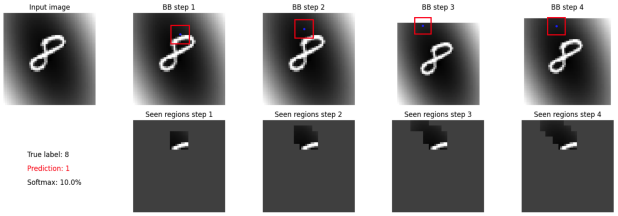
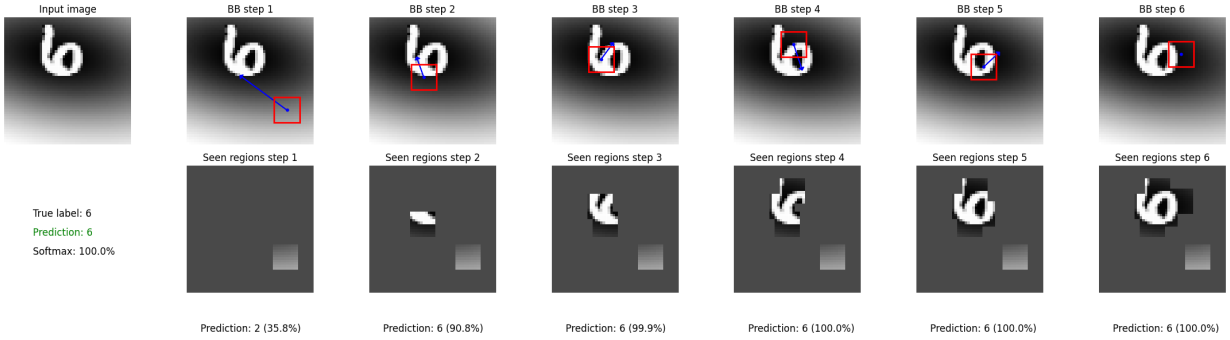


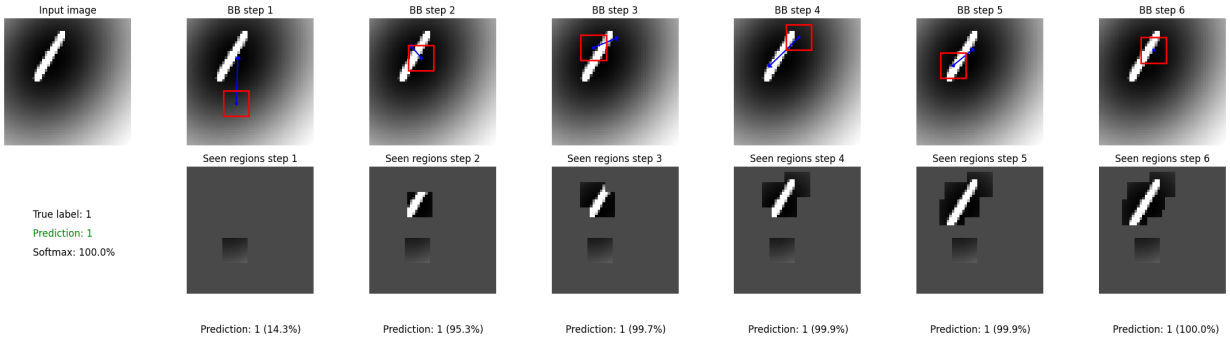
Fig. 13: Example of the pretrained model prediction given a set of fixations which do not allow to correctly classify the number.

A.3 Examples of Number Localization and Recognition

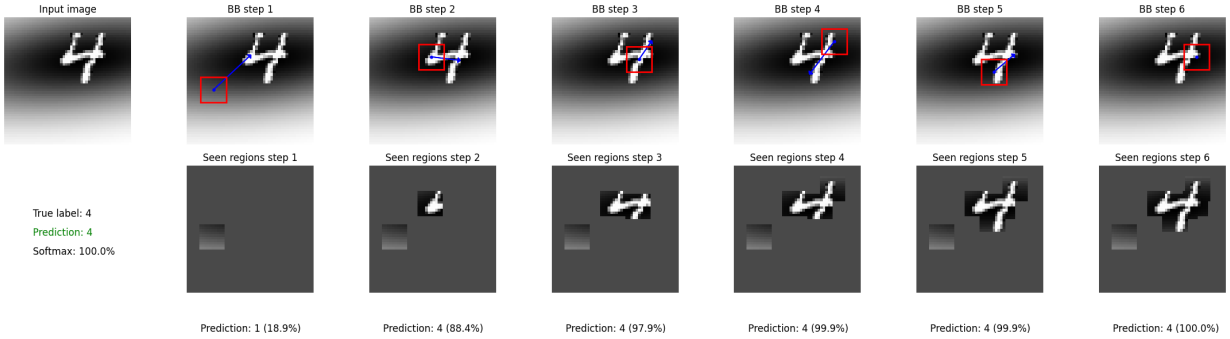
Figure 14 shows additional examples of the fixations generated by the best-performing model on the Resized MNIST with Spatial Information dataset. Similarly, Figure 15 presents examples from the best-performing model on the Resized MNIST dataset.



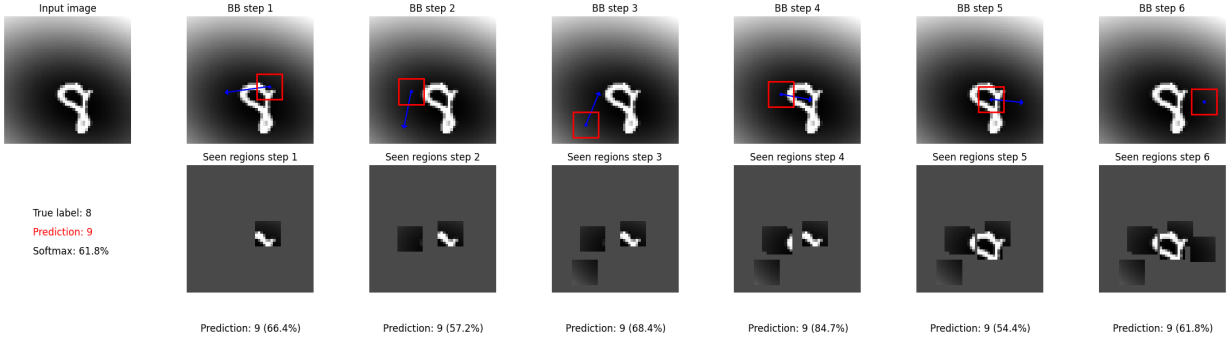
(a) Correct prediction. The model quickly fixates on the number's location using the available spatial information.



(b) Correct prediction. The model quickly fixates on the number's location using the available spatial information.

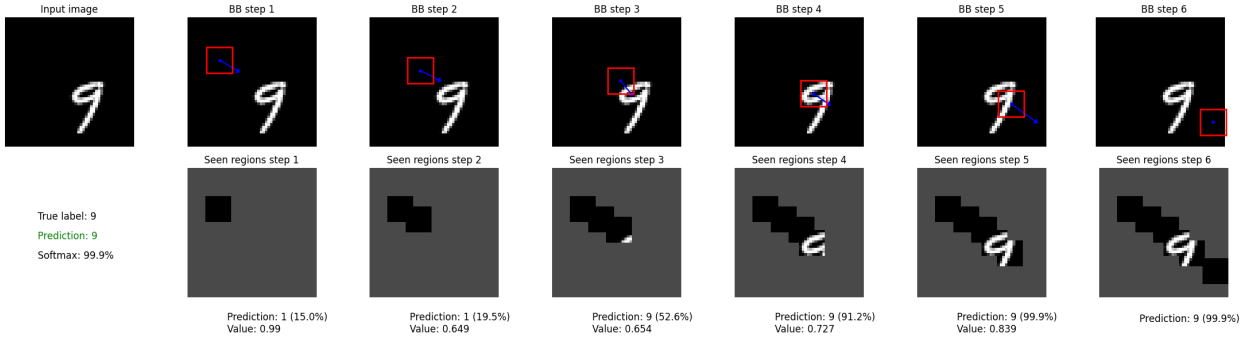


(c) Correct prediction. The model quickly fixates on the number's location using the available spatial information.

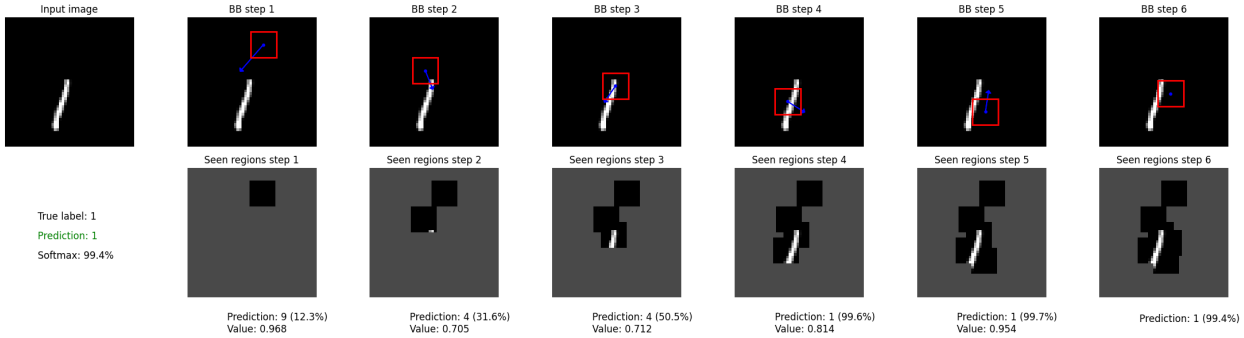


(d) Incorrect prediction. The model fails to fixate on the relevant region, leading to a misclassification. The confusion between "8" and "9" is plausible based on the partial view of the digit.

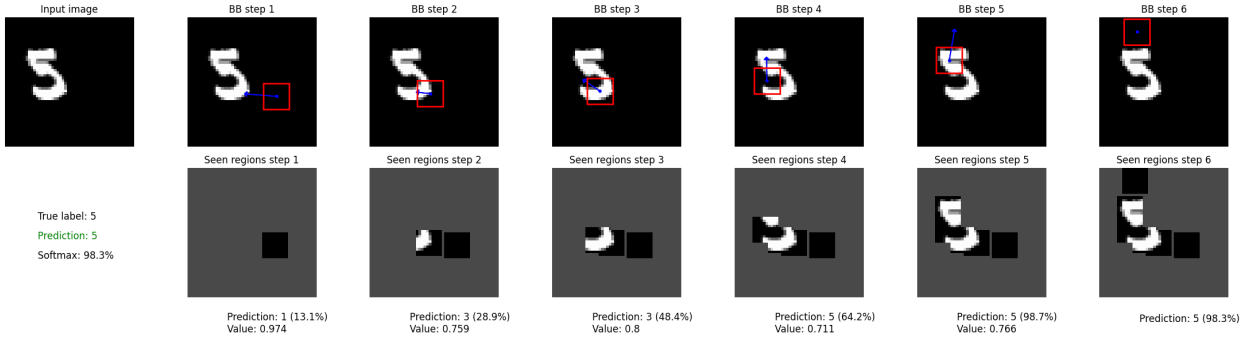
Fig. 14: Examples of classification and fixation predictions from the best-performing model on the Resized MNIST with Spatial Information dataset. The top row shows fixation locations and predicted movement vectors at each step. The bottom row presents the final prediction, the accumulated observed image region, and intermediate predictions.



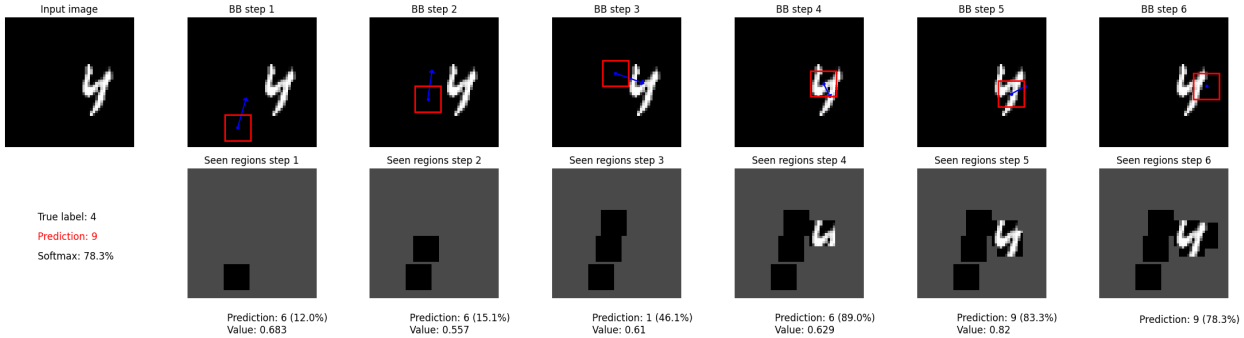
(a) Correct prediction.



(b) Correct prediction.



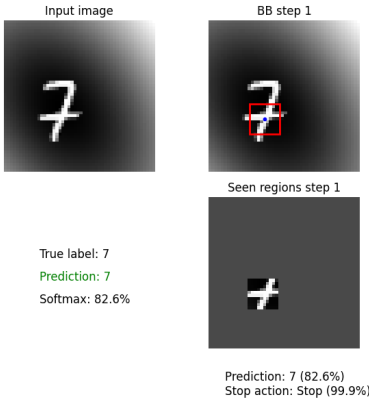
(c) Correct prediction.



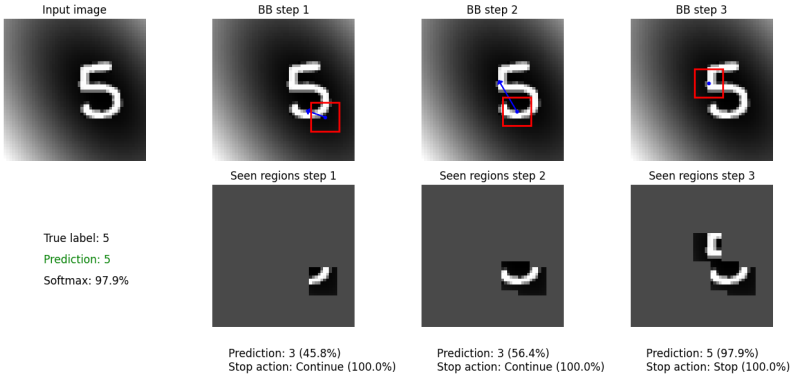
(d) Incorrect prediction. The model would require more fixations to correctly classify the digit. The visible parts of the ”4” resemble a ”9”, making the misclassification plausible.

Fig. 15: Examples of classification and fixation predictions from the best-performing model on the Resized MNIST dataset. The top row shows fixation locations and predicted movement vectors at each step. The bottom row presents the final prediction, the accumulated observed image region, and intermediate predictions.

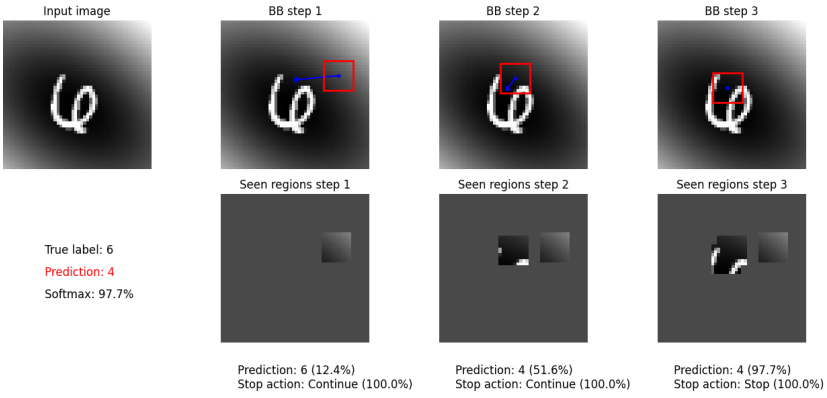
A.4 Examples With Stop Action



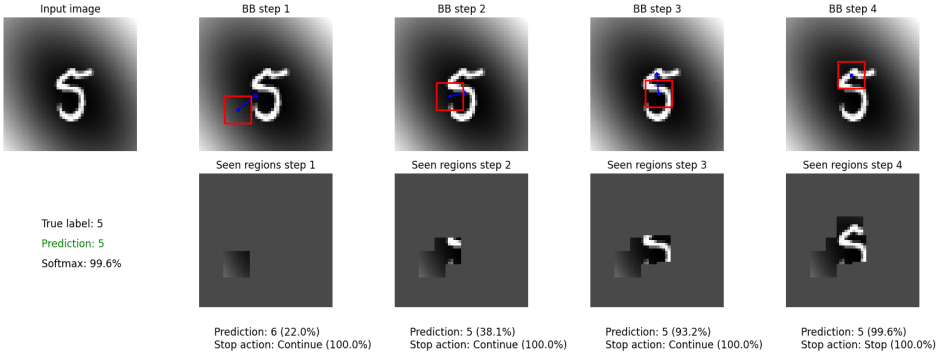
(a) Correct stop prediction behavior.



(b) Correct stop prediction behavior.



(c) Example of the model incorrectly predicting when to stop.



(d) Correct stop prediction behavior.

Fig. 16: Additional examples from the Resized MNIST with Spatial Information dataset, using the model trained with the stop action independently and a mistake penalty (λ_m) of 4.5, as described in Section 7. The top row shows fixation locations and predicted movement vectors at each step. The bottom row presents the final prediction, the accumulated observed image region, and intermediate predictions. Confidence values are rounded to a single decimal place.

A.5 Training Details

This section details the hyperparameters used in the experiments presented in this report, covering pretraining as well as the experiments done in Sections 6 and 7. The configuration files for these experiments are included in the submitted code.

All experiments employed AdamW as the optimizer and a step-based learning rate scheduler, which reduced the learning rate by a decay factor (γ) after a fixed number of epochs.

In the RL experiments, action selection involved a sampling mechanism. Specifically, the predicted angle and distance were sampled from a normal distribution centered on the model's predicted values, with a standard deviation that decreased over time according to a predefined schedule. For the stop action, decisions were sampled based on the model predicted confidence between the "stop" and "continue" options.

Table 6 lists the hyperparameters used during pretraining. Table 5 shows those used in the experiments related to number localization and recognition. Finally, Table 7 shows the ones used used in the experiments done to incorporate the stop action.

Hyperparameter	**Value**
Number of fixations	4
Fixation Size (Pixels)	10x10
Learning Rate (LR)	0.0004
Batch Size	512
Epochs	200
Optimizer	AdamW
LR Step Decay (Epochs)	70
LR Decay Factor (γ)	0.4
Transformer Layers	3
Transformer Hidden Size	128

TABLE 6: Hyperparameters used during pretraining.

Hyperparameter	**Value**
Discount Factor (γ)	0.99
Number of fixations	6
Fixation Size (Pixels)	10x10
Learning Rate (LR)	0.0001
Batch Size	2048
Epochs	500
Optimizer	AdamW
LR Step Decay (Epochs)	200
LR Decay Factor	0.5
Transformer Layers	3
Transformer Hidden Size	128
Initial Sampling STD Distance	0.25
Minimum Sampling STD Distance	0.01
Decay Factor Sampling STD Distance	0.99
Initial Sampling STD Angle	1.0
Minimum Sampling STD Angle	1.0
Decay Factor Sampling STD Angle	1.00
Classification Reward λ_m	0.0
Surface Reward λ_s	0.3
Distance scale in Surface Reward λ_d	0.3
Overlap Penalty λ_o	0.1
Overlap Threshold τ_{max}	0.3

TABLE 5: Hyperparameters used during the experiments of number localization and recognition. Reward-related parameters apply only to experiments using those specific components. "STD" refers to standard deviation.

Hyperparameter	**Value**
Discount Factor (γ)	0.99
Number of fixations	6
Fixation Size (Pixels)	10x10
Learning Rate (LR)	0.0001
Batch Size	2048
Epochs	500
Optimizer	AdamW
LR Step Decay (Epochs)	200
LR Decay Factor	0.5
Transformer Layers	3
Transformer Hidden Size	128
Initial Sampling STD Distance	0.25
Minimum Sampling STD Distance	0.01
Decay Factor Sampling STD Distance	0.99
Initial Sampling STD Angle	1.0
Minimum Sampling STD Angle	1.0
Decay Factor Sampling STD Angle	1.00
Budget Penalty λ_b	0.5
N_{max} in Budget Penalty	6

TABLE 7: Hyperparameters used during the experiments done to Experiment incorporate the stop action. "STD" refers to standard deviation.