Leveraging Deep Q-Network Agents with Dynamic Routing Mechanisms in Convolutional Neural Networks for Enhanced and Reliable Classification of Alzheimer's Disease from MRI Scans

Jolanta Podolszanska[a]
Faculty of Science & Technology, Jan Dlugosz Uniwersity, Armii Krajowej 15/17 Avenue, Czestochowa, Poland

Keywords: CapNet, Reinforcement Learning, Agents Learning, Medical Imaging.

Abstract: With limited data and complex image structures, accurate classification of medical images remains a significant challenge in AI-assisted diagnostics. This study presents a hybrid CNN model with a capsule network layer and dynamic routing mechanism, enhanced with a Deep Q-network (DQN) agent, for MRI image classification in Alzheimer's disease detection. The approach combines a capsule network that captures complex spatial patterns with dynamic routing, improving model adaptability. The DQN agent manages the weights and optimizes learning by interacting with the evolving environment. Experiments conducted on popular MRI datasets show that the model outperforms traditional methods, significantly improving classification accuracy and reducing misclassification rates. These results suggest that the approach has great potential for clinical applications, contributing to the accuracy and reliability of automated diagnostic systems.

1 INTRODUCTION

Convolutional Neural Networks (CNNs) revolutionized computer vision by capturing spatial patterns effectively (He et al., 2017). MRI, crucial in Alzheimer's disease (AD) diagnosis, detects neurodegenerative changes like hippocampal atrophy (Leszek, 2012). This work proposes a hybrid CNN-CapsNet model with dynamic routing and a DQN agent to enhance AD classification accuracy.

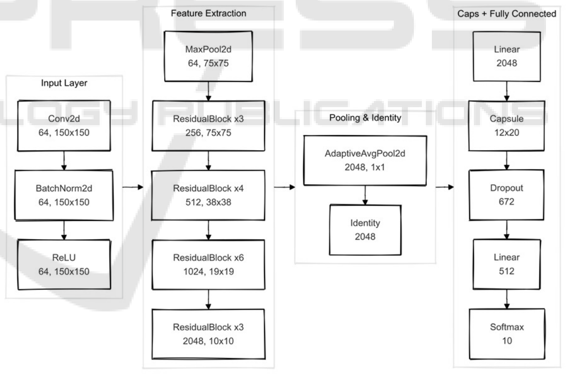


Figure 1: CNN Architecture diagram.

1.1 Related Works

Capsule Networks (CapsNets) effectively model hierarchical spatial patterns, improving classification, especially in medical imaging (Sabour et al., 2017). Enhancements like Efficient-CapsNet (Jia et al., 2022) and Res-CapsNet (Pawan et al., 2023) use mechanisms such as auto-attention and residual connections to boost accuracy and stability. Recent applications in Alzheimer's (Bushara et al., 2024), lung cancer (Bushara et al., 2024), and COVID-19 detection (Afshar et al., 2020) validate their utility in complex datasets.

Techniques like SE-Inception-ResNet (Xi et al., 2023) and TE-CapsNet (Yadav and Dhage, 2024) address challenges such as class imbalance and computational costs. MResCaps (Abhishek et al., 2024) and S-VCNet effectively classify datasets like DermaMNIST and OrganMNIST-S, demonstrating the versatility of CapsNets.

Reinforcement learning-based dynamic routing improves adaptability in tasks like Alzheimer's disease progression analysis (Jiao and et al., 2019), malaria detection (Madhu et al., 2021), and lung cancer classification in CT images (Bushara et al., 2024). Combining pre-trained ResNet weights with CapsNets enables robust spatial feature analysis and accurate diagnostic predictions.

[a] https://orcid.org/0000-0002-6032-5654

Podolszanska, J.
Leveraging Deep Q-Network Agents with Dynamic Routing Mechanisms in Convolutional Neural Networks for Enhanced and Reliable Classification of Alzheimer's Disease from MRI Scans.
DOI: 10.5220/0013301900003890

In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 3*, pages 1172-1179
ISBN: 978-989-758-737-5; ISSN: 2184-433X

2 NETWORK ARCHITECTURE

The efficiency of machine learning systems relies on their architecture. This section details the model's structure, parameters, and optimization techniques designed to enhance accuracy and performance. Unique features distinguish it from conventional methods, contributing to its superior results.

2.1 Dynamic Routing

Some decisions, like recognizing large objects, are simpler than specialized tasks requiring domain knowledge. Complex tasks benefit from systems that identify subtasks and select suitable algorithms. Research (Jiao and et al., 2019), (Madhu et al., 2021), (Bushara et al., 2024) shows dynamic routing improves accuracy but often neglects computational costs, relying on opaque heuristics for efficiency. This approach leverages ResNet-50.

In dynamic routing, let $b_{ij} \in \mathbb{R}$ represent the initial coefficient, signifying the "belief" of input capsule i about contributing to output capsule j. Initially, b_{ij} indicates no prediction for any output capsule. During routing, these coefficients are iteratively updated to optimize the correspondence between input and output capsules, as defined in (1).

$$b_{ij} = b_{ij} + agreement \quad (1)$$

Agreement concordance is calculated as the scalar product of the prediction vector $\hat{u}_{ij}$ and the output v_j for each capsule. Higher compatibility increases b_{ij}, which is converted to c_{ij} using the softmax function (2). In Equation (2), e is the base of the natural logarithm, essential in exponential functions widely used in machine learning.

$$c_{ij} = \frac{e^{b}_{ij}}{\sum_k e^{b}_{ik}} \quad (2)$$

Optimal routing can be modeled as a Markov decision process (Bengio et al., 2015). The Q-Routing algorithm (Bai et al., 2024), enhanced with backward updates, improves convergence by balancing load and energy via a reward function considering delay. Experiments show it outperforms standard Q-Learning across all metrics (Valadarsky et al., 2017). Additionally, c_{ij} factors summing to unity enable proportional activation allocation to output capsules based on prediction consistency.

Routing strategy selection can be modeled as a Markov chain. The Q-routing algorithm, enhanced with backward updates, improves convergence and optimizes routing by balancing load and energy through a reward function. It outperforms Q-Learning across metrics (Valadarsky et al., 2017). The c_{ij} coefficients ensure proportional activation allocation to output capsules based on prediction consistency.

Let $\hat{u}_{ij} \in \mathbb{R}^d$ be the prediction of the activation vector from input capsule i to output capsule j. The routing process involves iteratively assigning coefficients $c_{ij} \in [0,1]$ which represent the weight or confidence of the input capsule i to the output capsule j. Defined s_j as the weighted sum of the predictions (3).

$$s_j = \sum_i c_{ij}\hat{u}_{ij} \quad (3)$$

where c_{ij} are calculated by applying the softmax function on b_{ij}(4) values.

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} \quad (4)$$

where b_{ik} are initially initialized as zero and iteratively updated based on the correspondence between the prediction $\hat{u}_{ij}$ and the resulting activation vector v_j. In each routing iteration, the value of b_{ij} is updated (5).

$$b_{ij} = b_{ij} + \hat{u}_{ij} \cdot v_j \quad (5)$$

The scalar product $\hat{u}_{ij} \cdot v_j$ measures the correspondence between the prediction $\hat{u}_{ij}$ and the activation vector v_j. When they align, b_{ij} increases, boosting the assignment factor c_{ij} in subsequent iterations. Iterative routing, performed r times, optimizes c_{ij}, focusing on output capsules that aggregate input vector predictions.

2.2 Simulation of a Capsule-Based Environment

The capsule network in this work employs a Dynamic Routing Capsule Layer inspired by (Sabour et al., 2017). This layer utilizes iterative routing-by-agreement to determine the contributions of lower-level capsules to higher-level capsule outputs. A squashing function ensures vector normalization and learnable transformation matrices are used for capsule-to-capsule predictions.

Consider an agent learning environment as a single-step decision-making process that can be modelled as a Markov process. The agent selects an output capsule in the decision environment to maximize the reward function. The state space S, where $s \in S$, is represented as the activation vector of the input capsules (6).

$$s = [a_1, a_2, a_3, ..., \in \mathbb{R}^{C_{\text{in}}}] \quad (6)$$

The activation of input capsule i, denoted as a_i, is randomly initialized at the start of training, making the state s a random vector. An agent selects an action $a \in A$, where $A = \{1, 2, 3, \ldots, C_{out}\}$, representing the selection of one output capsule from C_{out}. The reward function $R(s, a)$ is defined as the return value and is currently random (7).

$$R(s,a) = \text{random} \sim \mathcal{U}(0,1) \tag{7}$$

where $U(0,1)$ for uniform distribution. In the future, an extended feature may be available that will be based on state-to-state correspondence, and the feature may be available as an early activation and actual output capsule feature. At the beginning of the section, the state is randomly initialized (8)

$$s = [a_1, a_2, \ldots, a_{C_{\text{in}}}], a_i \sim \mathcal{N}(0,1) \tag{8}$$

where $\mathcal{N}(0,1)$ is a normal distribution with expected value 0 and variance 1. The agent chooses action $a \in A$, which represents the choice of output capsule. After action a is executed, state s is re-initialized(9).

$$s' = [a'_1, a'_2, \ldots, a'_{C_{\text{in}}}], a'_i \sim \mathcal{N}(0,1) \tag{9}$$

In the future, this environment will be extended to allow the software to deal with more complex classifications.

2.3 Agent Model

The agent model approximates action values $Q(s,a)$ as in the Deep Q-Learning (DQN) algorithm. The state $s \in \mathbb{R}^d$ represents the environment, where d_s is the state space dimension. The action $a \in A$, with $A = \{1, 2, 3, \ldots, d_a\}$, belongs to a finite action space of size d_a. The network aims to approximate $Q(s,a)$, the expected cumulative reward for taking action a in state s (10).

$$Q(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right] \tag{10}$$

where r_t is the reward at step t, and $\gamma \in [0,1)$ is the discount factor. The agent approximates the value function $Q(s,a)$ using a neural network with three fully connected layers. Let $W_1 \in \mathbb{R}^{128 \times d_s}$ and $b_1 \in \mathbb{R}^{128}$ represent the weight matrix and bias vector for the first layer. The input vector s is transformed as follows (11).

$$h_1 = RELU(W_1 s + b_1) \tag{11}$$

where $h_1 \in \mathbb{R}^{128}$ is the output of the first layer after applying the ReLU activation function. Let $W_2 \in \mathbb{R}^{128 \times 128}$ and $b_2 \in \mathbb{R}^{128}$ be the weight matrix and bias vector for the second layer. The transformation of the vector h_1 is defined by the following equation(12).

$$h_2 = RELU(W_2 h_1 + b_2) \tag{12}$$

where $h_2 \in \mathbb{R}^{128}$ is the output of the second layer after applying the ReLU activation function.

2.4 Convolutional Neural Network with Capsule Layers

Let $f(x)$ represent the transformation performed by the ResNet50 network up to the Fully Connected layer, with the output replaced by the identity function (1). After feature extraction, the result is transformed by a fully connected layer to align with the capsule layer requirements. The attention layer $W_{fc} \in \mathbb{R}^{(\text{in capsules} \times \text{in dim} \times 512)}$ is defined by equation (13).

$$z = W_{fc} f(x) \tag{13}$$

This result is then reshaped to the dimensions required by the capsule layer(14).

$$z \to \tilde{z} \in \mathbb{R}^{(B \times \text{in capsules} \times \text{in dim})} \tag{14}$$

Next, $\tilde{z}$ is processed by the capsule layer, which converts input capsules into output capsules with specified dimensions. Using dynamic routing, the capsule layer transforms $\tilde{z} \in \mathbb{R}^{(B \times \text{in capsules} \times \text{in dim})}$ into $v \in \mathbb{R}^{(B \times \text{out capsules} \times \text{out dim})}$, as defined by equation (15).

$$v = CapsuleLayer(\hat{z}) \tag{15}$$

The result is then flattened to $v_{\text{flat}} \in \mathbb{R}^{(B \times \text{resnet out features} + \text{out capsules} \times \text{out dim})}$. The output features from ResNet50 $f(x)$ are flattened along with the capsule layer features v_{flat}, and then combined (16).

$$c = \text{Concat}(f(x), v_{\text{flat}}) \in \mathbb{R}^{(B \times \text{resnet out features} + \text{out capsules} \times \text{out dim})} \tag{16}$$

where c is a vector of connected features. The CNN model uses the Focal Loss function, which is defined by equation (17).

$$\text{Focal Loss} = -\alpha(1 - p_t)^{\gamma} \log(p_t) \tag{17}$$

where p_t is the probability assigned to the true class. A capsule network layer is defined as follows: let C_{in} and C_{out} denote the number of input and output capsules, respectively, and d_{in} and d_{out} their dimensions. The capsule layer transforms $x \in \mathbb{R}^{(B \times C_{\text{in}} \times d_{\text{in}})}$ to $v \in \mathbb{R}^{(B \times C_{\text{out}} \times d_{\text{out}})}$, where B is the batch size. The transformation matrix $W \in \mathbb{R}^{(1 \times C_{\text{in}} \times C_{\text{out}} \times d_{\text{out}} \times d_{\text{in}})}$ is a learnable parameter. Each input vector x_i for capsule

i is transformed to a prediction vector $\hat{u}_{ij}$ for output capsule j using W, as defined by equation (18).

$$\hat{u}_{ij} = W_{ij} x_i \tag{18}$$

Prediction vectors are matched to output capsules through an iterative routing process. The squash function normalizes output capsule vectors. Let $s_j \in \mathbb{R}^d$ represent the output vector for capsule j, aggregated from input capsule predictions during a routing step. The squash function transforms s_j into v_j with a norm in $[0,1]$, defined as $\mathbb{R}^d \to \mathbb{R}^d$ (19).

$$v_j = \text{squash}(s_j) = \frac{\|s_j\|^2}{1+\|s_j\|^2} \cdot \frac{s_j}{\|s_j\|+\varepsilon} \tag{19}$$

where $\|s_j\|$ denotes the Euclidean norm of the vector s_j and ε is a small scalar value that prevents division by zero. For $s_{ij} = 0$, we have $v_{ij} = 0$. As the norm $\|s_{ij}\|$ increases, the transformation asymptotically approaches a value close to 1 for $\|v_j\|$, allowing for the amplification of activations for output capsules with strong activations while suppressing capsules with weak activations. This transformation is nonlinear, which helps the model better capture dependencies between input elements.

3 PROPOSED METHOD

This study optimizes Alzheimer's disease classification by combining capsule layers with dynamic routing (Sabour et al., 2017) and Focal Loss (Xi et al., 2023), addressing class imbalance and preserving spatial relationships. Dynamic routing enhances hierarchical feature extraction, crucial in medical imaging (Afshar et al., 2018).

Incorporating the CapsuleRoutingEnv algorithm (Bai et al., 2024) with a DQN agent improves routing adaptivity and precision, effectively analyzing complex medical images. The model integrates ResNet, capsule layers, attention mechanisms, and Focal Loss, leveraging their strengths to enhance classification (Afshar et al., 2020), (Sadeghnezhad and Salem, 2024).

3.1 Dataset

The dataset contains 6,400 MRI images, categorized into four classes: Mild dementia (896), Moderate dementia (64), Non-dementia (3,200), and Very mild dementia (2,240). Images were normalized to 128x128 pixels for analysis.

3.2 Model Initialization and Initial Configuration

The model integrates ResNet50 and CapsNet with dynamic routing to leverage spatial information and address data constraints. Pre-trained ResNet weights enhance generalization, and reinforcement learning optimizes routing for improved MRI analysis.

Trained for 50 epochs with a batch size of 64, the model used a learning rate of 0.0001, gradually increased to minimize overfitting. AdamW optimizer ensured stability and handled dynamic structures effectively.

3.3 Training and Validation Procedure

In each training iteration, the model takes a batch of x inputs and their corresponding y labels. The goal is to minimize the loss function L, which has been chosen as Focal Loss to better deal with non-equivalent classes. The loss value for a given batch (x,y) is calculated according to the formula (17), where p_t is the probability assigned to the correct class (20)

$$p_t = \begin{cases} p & \text{for the true class,} \\ 1-p & \text{for the wrong class} \end{cases} \tag{20}$$

we notice that Focal Loss value L_{focal} is minimized using the AdamW optimizer, which allows for stable weight updates of the model. Parameters are updated according to the gradients ∇L_{focal} for each batch to minimize the loss function.

During validation, the model is assessed for its ability to generalize to data that was not used during training. For each batch of validation data, the following metrics are calculated:precision, recall, and F1.

$$P_i = \frac{TP_i}{TP_i + FP_i} \tag{21}$$

These metrics allow for the assessment of the classification quality of various data(21) and (22).

$$R_i = \frac{TP_i}{TP_i + FN_i} \tag{22}$$

where TP_i is the number of true positive examples for class i, FP_i is the number of false positive examples for class i and FN_i is false negative examples for class i. The F1-score for class i is calculated as the harmonic mean of precision and recall (23).

$$F1_i = 2 \cdot \frac{P_i \cdot R_i}{P_i + R_i} \tag{23}$$

These metrics are then averaged across all classes to produce a "macro" score, which ensures that each class is treated equally regardless of its abundance in the data. The AdamW algorithm was used to optimize the model, which updates the weights in each iteration under the rule (24).

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{m_t}{\sqrt{v_t + \varepsilon}} \tag{24}$$

where m_t and v_t are the torque and acceleration of the gradients, respectively, which are tracked to stabilize the optimization process. Additionally, the StepLR schedule is used, which lowers the learning rate every certain number of epochs T (25).

$$\eta_{t+1} = \eta_t + \gamma \tag{25}$$

where $\gamma = 0.1$ satisfying the relationship $\gamma \in [0,1]$. The values of the loss function and metrics (precision, recall, F1-score) are logged after each epoch, which allows for ongoing assessment of the model's quality.

3.4 Regularization and Techniques to Prevent Overfitting

Several regularization techniques were used to improve the model's generalization ability and prevent overfitting. The activation a_i of neuron i after applying dropout with probability p is described as equation (26).

$$\tilde{a}_i = \begin{cases} 0 & \text{with probability } p \\ \frac{a_i}{1-p} & \text{with probability } 1-p \end{cases} \tag{26}$$

The division by $1-p$ in the training phase compensates for maintaining the expected activation value during testing when dropout is not used. L2 regularization involves adding a term to the loss function that penalizes large weight values (27).

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda \sum_i w_i \tag{27}$$

where $\mathcal{L}$ is the base Focal Loss function, and λ is the regularization coefficient. The method of Early Stopping was applied to monitor the validation error during training, which halts the process when errors on the validation set start to increase. In practice, the model trains until there is no improvement in the validation metric (e.g., loss or accuracy) for a specified number of epochs.

3.5 Implementation and Experimental Environment

Experiments were conducted on a Gainward RTX 4090 GPU with Intel i9-12900K processor and 32 GB RAM, using PyTorch Lightning for training. The modular architecture combined ResNet-50 and CapsNet with dynamic routing. Tools like NumPy, scikit-learn, and Matplotlib supported analysis, with metrics monitored in real-time via TensorBoard. Validation ensured stability, and the best weights were saved for reproducibility.

3.6 Computational Complexity

ResNet as a feature extractor has a complexity of $O(L \cdot n^2 \cdot d)$, where n is the spatial dimension, d the channel depth, and L the number of layers. Dynamic routing between capsules has a complexity of $O(n^2 \cdot m \cdot r)$, with m as the output capsule count and $r = 3$ iterations, increasing computational load for larger capsule sizes. The multi-head attention layer operates with a complexity of $O(h \cdot n^2 \cdot d)$, balancing efficient processing with resource demands.

Enabling agent reinforcement learning for capsules incurs an additional learning cost, depending on the number of learning steps t, which gives complexity $O(t \cdot a)$, where a is several shares (target capsules) in each step. In summary, the total complexity of the model is about(29).

$$\begin{aligned} &O(L \cdot n^2 \cdot d) + O(n^2 \cdot m \cdot r) \\ &\quad + O(h \cdot (n^2 \cdot d)) + O(t \cdot a) \end{aligned} \tag{28}$$

which shows the increase in complexity depending on the number of capsules, attention heads and routing iterations.

The relevant parameters in the simulation experiment are shown in Table 1.

Table 1: Model and Training Hyperparameters.

Parameter	Value
Number of Capsules	64
Capsule Dimension	32
Output Capsule (out_capsule)	10
Output Capsule Dimension (out_dim)	16
Number of Routes	3
Number of Attention Heads	4
Batch Size	64
Learning Rate (Agent)	0.0001
Decay Rate	0.98
Focal Loss Alpha (α)	1
Focal Loss Gamma (γ)	2
Agent State Dimension	64
Agent Action Dimension	10
Exploration Rate (ε)	0.7
Experience Replay Size	1000

```
**Data:** Input image $I$ of size $n \times n$
**Result:** Predicted class of $I$
**Step 1: Feature Extraction**
$F \leftarrow \text{ResNet}(I)$ // Extract features using
    ResNet backbone
**Step 2: Attention Mechanism**
$A \leftarrow \text{AttentionLayer}(F)$ // Apply attention
    to enhance significant features
**Step 3: Capsule Transformation**
$C_{in} \leftarrow \text{Transform}(A)$ // Transform attention
    output to capsule input format
**Step 4: Dynamic Capsule Routing**
Initialize routing logits $b_{ij} = 0$ for each capsule
  pair $(i, j)$;
Compute predicted output vectors $u_{ij} = W_{ij} \cdot c_i$ for
  each pair of capsules $(i, j)$, where $W_{ij}$ are
  trainable weights and $c_i$ is the input capsule
  vector;
Define the total number of routing iterations as
  $num\_routes$;
for each routing iteration $r$ from 1 to $num\_routes$
  do
    foreach capsule $c_i$ in $C_{in}$ do
        $c_{ij} \leftarrow \text{softmax}(b_{ij})$;
        $s_j \leftarrow \sum_i c_{ij} \cdot u_{ij}$ // Weighted sum for
            capsule j
        $v_j \leftarrow \text{squash}(s_j)$ // Apply squash
            activation to output
        foreach capsule $c_i$ do
            $b_{ij} \leftarrow b_{ij} + u_{ij} \cdot v_j$ // Update
                logits based on agreement
        end
    end
end
**Step 5: Reinforcement Learning Optimization**
Initialize DQN agent $Q$ with state dimension from
  $C_{in}$ and actions as capsule pairs;
foreach capsule $c_i$ in $C_{in}$ do
    $a_i \leftarrow \text{DQNAgent}(c_i)$ // Select action
        with DQN agent
    Update routing weights based on DQN
      reward;
end
**Step 6: Class Prediction**
Obtain final class prediction from combined
  capsule outputs $C_{out}$;
return Predicted class label
```

Algorithm 1: Hybrid CNN with Capsule Networks and Attention.

4 RESULTS

Figure 4 (3) shows classification results for eight brain MRI samples, while full results are in Figure 3 (2). Seven of the eight samples were correctly classified, highlighting the model's ability to identify class-specific features effectively.

Figure 3 shows Class 1 (Very mild dementia) is mostly accurate, with some misclassification as Class 4 (Non-dementia) due to feature overlap. Class 2 (Mild dementia) performs well despite a smaller sample size, with minimal misclassifications. Class 3 (Moderate dementia) achieves the highest accuracy, with minor misclassification into Class 4. Class 4 also performs strongly, with slight misclassification into Class 3, potentially due to shared features or limited training diversity. Overall classification efficiency is 98.75%.

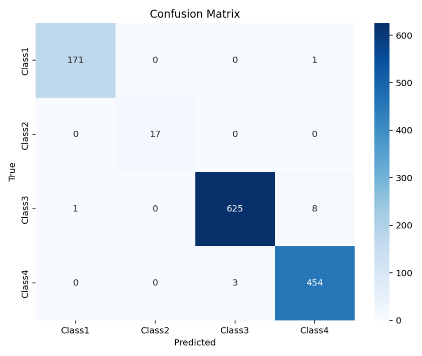


Figure 2: Confusion matrix illustrating the classification performance of the ResNet50-based hybrid CNN model.

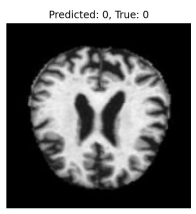
Misclassifications occur where images predicted as Class 2 belong to Classes 1 or 3 (see Figure 4), indicating overlapping features. Class 3 predictions are generally accurate but still prone to confusion due to shared characteristics with other classes. Similarly, Class 1 is occasionally misclassified as Class 2, highlighting challenges in distinguishing subtle patterns between these classes.

Figure 5 (4) illustrates the loss function during training. Initially (0–500 steps), a rapid decrease indicates effective learning and weight adjustments. Subsequently, the loss stabilizes, suggesting convergence. The stable curve, without oscillations, indicates minimal risk of overfitting.

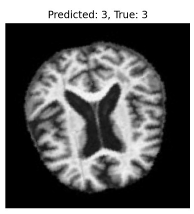
Figure 6 (5) shows the training loss trajectory. Initially, a steep decline reflects rapid parameter adjustments to capture dominant patterns. Later, the curve flattens asymptotically, indicating the model's approach to optimal capacity. The absence of fluctuations suggests a stable training process with appropriate learning rates and model stability.

Figure 7 (6) depicts the Train Loss function over training steps. Initially, a sharp increase suggests a high learning rate or complex parameter adjustments. After step 1500, the loss stabilizes, indicating equilibrium in the optimization process.

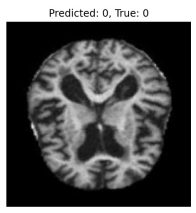
Figure 8 (7) shows the training loss sharply declining during the first 500 steps, reflecting efficient learning. Between steps 500 and 1500, the decrease slows, and the curve levels off near step 1500, indi-



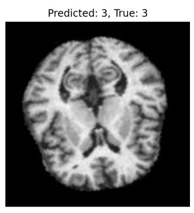
(a) Predicted: 2, True: 2.



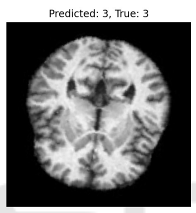
(b) Predicted: 3, True: 3.



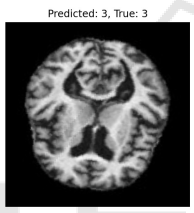
(c) Predicted: 2, True: 2.



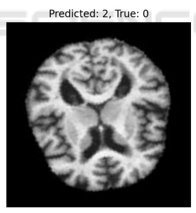
(d) Predicted: 3, True: 3.



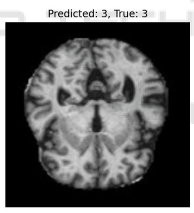
(e) Predicted: 3, True: 3.



(f) Predicted: 3, True: 3.



(g) Predicted: 2, True: 0.



(h) Predicted: 3, True: 3.

Figure 3: Classification results for selected first 8 cases from MRI images using a hybrid CNN model.

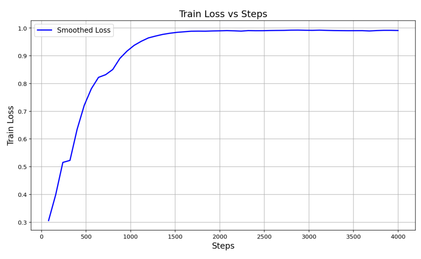


Figure 4: F1 metrics progression of the F1 score during training.

cating convergence. The loss stabilizes close to zero, demonstrating effective error minimization.

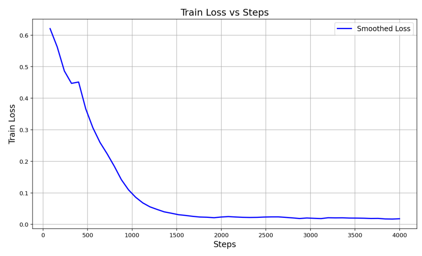


Figure 5: Validation Loss progression of validation loss across training epochs.

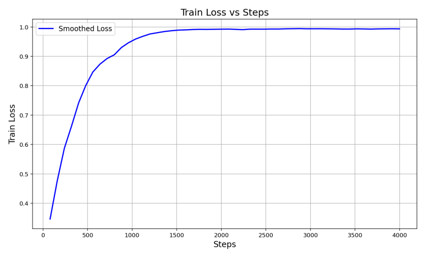


Figure 6: Validation Precision progression of validation precision across training epochs.

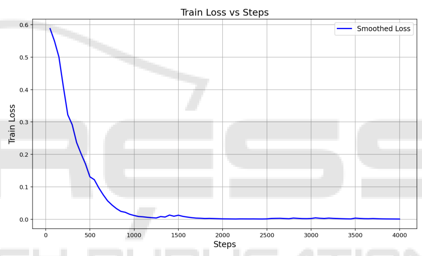


Figure 7: Classification training loss per step.

The classification model was evaluated using a loss function on training and validation data, with the elbow method determining the optimal stopping point. Key metrics such as Precision, Recall, and F1-Score were used for assessment (31). Cross-entropy was employed as the loss function for multiclass classification (29).

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{i,c}) \tag{29}$$

The Training and Validation Loss graphs show a rapid decline at the start, indicating quick pattern recognition. Between 3000-4000 steps, the decline slows, marking the elbow point. The first difference in the loss function, representing the rate of change, is calculated (30).

$$\Delta L(t) = L(t) - L(t-1) \tag{30}$$

The elbow point occurs when the change $\Delta L(t)$ is below the established threshold ε (31).

$$\Delta L(t) < \varepsilon \tag{31}$$

Figure analysis shows $\Delta L(t)$ stabilizing after 3,000 steps, suggesting training can stop to minimize overtraining and optimize generalization. Using the elbow method, training stops at t^* when the average loss change over the last k steps is below ε (32).

$$\frac{1}{k}\sum_{j=0}^{k-1}|\Delta L(t-j)| < \varepsilon \tag{32}$$

Loss charts and evaluation metrics indicate that ε is reached around 3000-4000 steps, suggesting minimal gains from further training. Using the elbow method and metrics like Precision, Recall, and F1-Score, the optimal stopping point was identified, ensuring sufficient accuracy and stability.

4.1 Adaptivity of Intelligent Routing Algorithm

Training consists of 30 episodes, each with 2,000 steps, where input data is randomly assigned, and routing paths are refined using rewards based on connection quality.

Routing performance is tested in 50 experiments across 4 scenarios, each lasting 2,000 steps with random topologies. Results are averaged to evaluate routing and classification performance.

5 CONCLUSIONS AND FUTURE WORK

The proposed model achieved 98.75% accuracy in Alzheimer's classification. Future work will focus on incorporating attention mechanisms and testing on diverse datasets to improve generalization and robustness.

REFERENCES

Abhishek, K., Jain, A., and Hamarneh, G. (2024). Investigating the quality of dermamnist and fitzpatrick17k dermatological image datasets. *arXiv preprint arXiv:2401.14497*.

Afshar, P., Heidarian, S., Naderkhani, F., Oikonomou, A., Plataniotis, K. N., and Mohammadi, A. (2020). Covid-caps: A capsule network-based framework for identification of covid-19 cases from x-ray images. *Pattern Recognition Letters*.

Afshar, P., Mohammadi, A., and Plataniotis, K. N. (2018). Brain tumor type classification via capsule networks. In *2018 25th IEEE International Conference on Image Processing (ICIP)*.

Bai, J., Sun, J., Wang, Z., Zhao, X., Wen, A., Zhang, C., and Zhang, J. (2024). An adaptive intelligent routing algorithm based on deep reinforcement learning. *Computer Communications*, 216:195–208.

Bengio, E., Bacon, P.-L., Pineau, J., and Precup, D. (2015). Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*.

Bushara, A. R., Kumar, R. V., and Kumar, S. S. (2024). Classification of benign and malignancy in lung cancer using capsule networks with dynamic routing algorithm on computed tomography images. *Journal of Artificial Intelligence and Technology*, 4(1):40–48.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969.

Jia, X., Li, J., Zhao, B., Guo, Y., and Huang, Y. (2022). Rescapsnet: Residual capsule network for data classification. *Neural Processing Letters*, 54(5):4229–4245.

Jiao, Z. and et al. (2019). Dynamic routing capsule networks for mild cognitive impairment diagnosis. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, volume 11767 of *Lecture Notes in Computer Science*, pages 620–628. Springer, Cham.

Leszek, J. (2012). Choroba alzheimera: obecny stan wiedzy, perspektywy terapeutyczne. *Polski Przeglad Neurologiczny*, 8(3):101–106.

Madhu, G., Govardhan, A., Srinivas, B. S., Sahoo, K. S., Jhanjhi, N. Z., Vardhan, K. S., and Rohit, B. (2021). Imperative dynamic routing between capsules network for malaria classification. *CMC-Computers Materials & Continua*, 68(1):903–919.

Pawan, S. J., Sharma, R., Reddy, H., Vani, M., and Rajan, J. (2023). Widecaps: A wide attention-based capsule network for image classification. *Machine Vision and Applications*, 34(4):52.

Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. arXiv:1710.09829.

Sadeghnezhad, E. and Salem, S. (2024). Inceptioncapsule: Inception-resnet and capsulenet with self-attention for medical image classification. *arXiv preprint arXiv:2402.02274*.

Valadarsky, A., Schapira, M., Shahaf, D., and Tamar, A. (2017). A machine learning approach to routing. arXiv preprint arXiv:1708.03074.

Xi, Y., Li, M., Zhou, F., Tang, X., Li, Z., and Tian, J. (2023). Se-inception-resnet model with focal loss for transmission line fault classification under class imbalance. *IEEE Transactions on Instrumentation and Measurement*.

Yadav, S. and Dhage, S. (2024). Te-capsnet: Time efficient capsule network for automatic disease classification from medical images. *Multimedia Tools and Applications*, 83:49389–49418.