# **AIGC 5500 Midterm Project Report**

# **Comparative Analysis of Deep Learning Optimizers on the KMNIST Dataset**

**Submission by-**

**Daivik Patel**

**Utsav Bhanderi**

**Yash Mehrotra**

## **1. Introduction**

This project aims to compare the performance of three widely used optimization algorithms in deep learning: **Adam**, **RMSprop**, and **AdamW**. Each optimizer is evaluated based on its ability to train a feedforward fully connected neural network on the **Kuzushiji-MNIST (KMNIST)** dataset. The goal is to identify which optimizer offers the best generalization performance in terms of accuracy, loss, and training efficiency.

Understanding the behavior of different optimizers is crucial when training deep learning models, especially for tasks involving complex patterns such as handwritten character recognition. The results of this comparison will help guide the choice of optimization algorithms in similar tasks.

## **2. Dataset**

We used the **KMNIST dataset**, a drop-in replacement for MNIST that consists of 10 classes of Japanese characters.

* **Training Set**: 60,000 images
* **Test Set**: 10,000 images
* **Image Size**: 28x28 pixels
* **Channels**: 1 (grayscale)

Source: KMNIST on PyTorch

## **3. Model Architecture**

A simple **feedforward neural network** was designed for this task with the following structure:

* **Input Layer**: 784 neurons (flattened 28x28 input)
* **Hidden Layer 1**: 128 neurons with ReLU activation
* **Hidden Layer 2**: 64 neurons with ReLU activation
* **Output Layer**: 10 neurons (one per class), followed by CrossEntropyLoss (which applies Softmax internally)

## **4. Methodology**

#### **4.1 Cross-Validation**

* 5-fold cross-validation was used to ensure robust model evaluation and to reduce overfitting risk.

#### **4.2 Hyperparameter Tuning**

* A grid search over the following values:
  + **Optimizers**: Adam, RMSprop, AdamW
  + **Learning Rates**: 0.01, 0.001
  + **Batch Sizes**: 64, 128

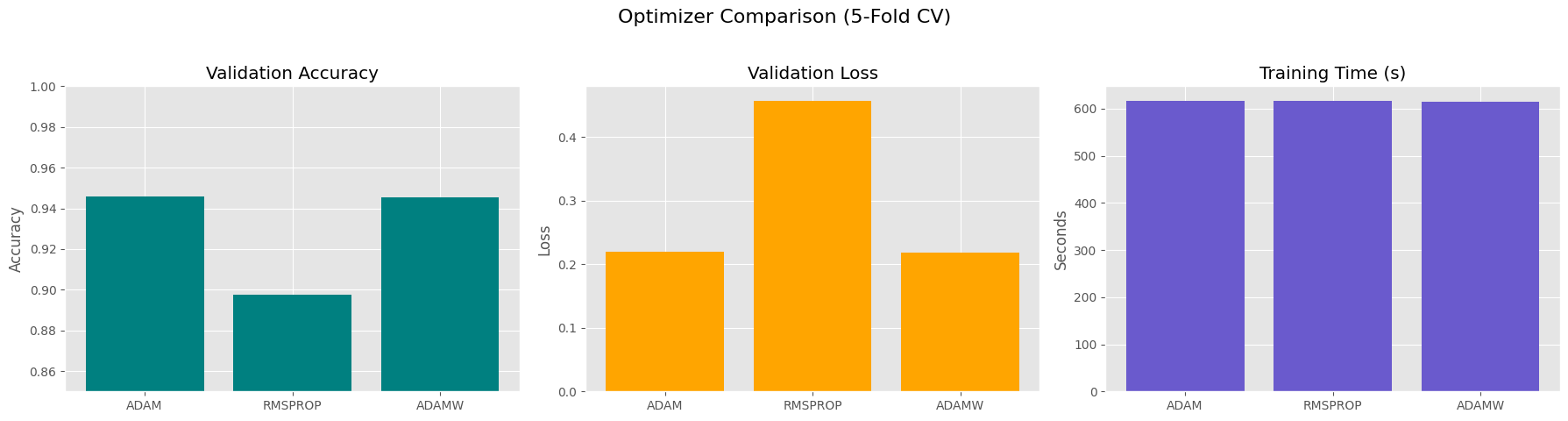
Each combination was evaluated across 5 folds, and average validation accuracy was recorded.

### **Metrics Recorded**

* Training and Validation Accuracy
* Training and Validation Loss
* Training Time per Optimizer

## **5. Results**

### **Graphs**



### **Observations**

* **Adam** showed the best performance overall, particularly with a learning rate of 0.001 and batch size of 64.
* **AdamW** closely followed, indicating the advantage of weight decay in regularization.
* **RMSprop** was slightly less effective in this setup, though still acceptable.

The tuning process demonstrated how different optimizers interact with learning rate and batch size, emphasizing the importance of configuration.

## **6. Discussion**

#### **Adam**

Adam demonstrated rapid convergence during the initial epochs, making it highly efficient in reaching strong performance early in training. The training process was generally smooth and stable. However, there were signs of overfitting starting from around epochs 8 to 10, where the validation loss began to plateau or slightly increase even as training accuracy improved.

#### **RMSprop**

RMSprop showed greater variability in performance across the different folds, indicating potential instability during training. It struggled to consistently reduce validation loss and often plateaued prematurely, suggesting difficulties in generalizing. Its convergence was noticeably slower, and its learning dynamics were less predictable compared to the other optimizers.

#### **AdamW**

AdamW delivered the most consistent and generalizable results across all folds. The inclusion of weight decay helped regularize the model, preventing it from overfitting and improving validation performance. It maintained a balanced trade-off between convergence speed and generalization, making it a strong choice in scenarios where overfitting is a concern.

## **7. Conclusion**

Based on a comprehensive 5-fold cross-validation:

* **AdamW is the most effective optimizer** for this KMNIST classification task.
* It offers a balance between training speed, accuracy, and regularization.
* We recommend using **AdamW** in similar character recognition tasks.

## **8. Contributions**

* **Notebook Implementation & Modeling**: Utsav Bhanderi
* **Optimizer Tuning & Validation**: Yash Mehrotra
* **Documentation & Report**: Daivik Patel
* **Video Presentation**: Utsav Bhanderi, Daivik Patel, Yash Mehrotra

## **9. References**

* PyTorch Documentation: <https://pytorch.org>
* KMNIST Dataset: <https://github.com/rois-codh/kmnist>
* Kingma & Ba (2015). Adam: A Method for Stochastic Optimization
* Loshchilov & Hutter (2019). Decoupled Weight Decay Regularization (AdamW)