**Report**

1. In this project, we focused on simple Multi-Layer Perceptron (MLP) architectures with 2-3 hidden layers, considering the addition of Dropout and Batch/Layer Normalization for regularization and stability. Our primary goal was to achieve high accuracy while maintaining simplicity and efficiency in the architecture, given the nature of the dataset.

**Reasons for Focusing on MLPs:**

1. **Initial Baseline Results:**

We started by using the default classifier provided, which included only one linear layer. After training with the Adam optimizer and using the ReduceLROnPlateau scheduler, we achieved 82% validation accuracy. This indicated that the data was largely linearly separable.

Given this result, we concluded that more complex architectures might not be necessary for this task, as a single layer was already capturing a significant portion of the decision boundaries.

1. **Supporting Literature:**

We reviewed two papers [1, 2] that applied Implicit Neural Representations (INRs) as input to neural networks for classification tasks. Both papers used simple MLP architectures and reported competitive results compared to state-of-the-art (SOTA) models.

This further motivated our decision to stick with MLPs, as the results in the literature showed that MLPs were effective for classification tasks on datasets like ours.

1. **Suitability of Other Architectures:**

Convolutional Neural Networks (CNNs): We considered whether CNNs might improve performance but concluded that they would not be appropriate for our data. Each sample in our dataset is a dense 512-dimensional vector that has already been trained on the original image and contains embedded information. Convolutions are typically used to extract spatial features from raw image data, and since our input already contains highly abstracted information, CNNs would likely perform similarly to linear layers.

Transformers and Attention: We also ruled out using transformer architectures or attention mechanisms because the data does not lend itself to the standard tokenization process used in transformers. Since each sample is a dense vector and cannot be split into patches or tokens, we concluded that attention mechanisms would not provide any significant advantages for this task.

**Our final architecture**

* Input Layer: 512 → 512 (Fully Connected) + BatchNorm + LeakyReLU + Dropout.
* Hidden Layer 1: 512 → 256 (Fully Connected) + BatchNorm + LeakyReLU + Dropout.
* Hidden Layer 2: 256 → 128 (Fully Connected) + BatchNorm + LeakyReLU + Dropout.
* Output Layer: 128 → 10 (Fully Connected)

Optimizer- AdamW with learning rate=0.001 and weight decay=0.01.

Scheduler – ReduceLROnPlateau with factor=0.1 and patience =3. The scheduler uses validation accuracy as metric.

Dropout probability- 0.1

Batch size- 64

Early stopping mechanism that stops after 5 iterations with no improvement.

The batch normalization layer uses momentum=0.05 which is more suitable for smaller batch.

Checkpoint mechanism to save the set of weights that resulted the highest accuracy on validation set.

Our final network architecture was a good choice because it provides a good balance between simplicity and performance. We used a simple MLP with 4 fully connected layers, gradually reducing the dimensions from 512 → 256 → 128 → 10.

The gradual reduction in dimensionality allows the network to progressively distill the most **Batch Normalization** stabilized training by normalizing activations, and **LeakyReLU** prevented the "dying ReLU" problem, ensuring the neurons remained active throughout training.

**Batch Normalization** stabilized training by normalizing activations, and **LeakyReLU** prevented the "dying ReLU" problem, ensuring the neurons remained active throughout training. The **ReduceLROnPlateau** scheduler helped the model fine-tune after the initial stable learning phase, reducing the learning rate to avoid oscillations or overfitting. Combined with **Dropout** and **weight decay** in the **AdamW** optimizer, overfitting was well controlled. The **early stopping** mechanism ensured that we didn't overtrain the model, stopping when there was no improvement after 5 epochs.

As we’ll explain in the next section, hyperparameter tuning was a critical part of arriving at this final design. The chosen architecture achieved the highest validation accuracy during training at **89.7%**.

**Hyper parameter tuning:**

We performed hyperparameter tuning on the following parameters:

* Learning rate: [0.001, 0.0001]
* Hidden dimensions: [ [512, 256, 128], [256, 128], [128, 128], [512,512,512], [128,64] ]
* Dropout probability: [0,0.1,0.3]
* Batch size: [32,64,128,256]
* Normalization after linear layer: [None, layer normalization, batch normalization]
* Activation: [Relu, LeakyRelu]
* Early Stopping:[5,10]

There were 1,440 possible combinations, which was too large for comprehensive training. To handle this, we used Random Search, where we randomly selected 150 combinations to train and evaluate. Each combination was trained on the training set and evaluated on the validation set. We ultimately selected the combination that achieved the highest accuracy on the validation set (as described in the final architecture section).

The maximum number of epochs was set to 500, and this parameter was not tuned. Prior to the random search, we manually experimented with different optimizers and schedulers, but none performed as well as the chosen ones, so we did not tune the optimizer or scheduler further.

We decided to use AdamW, inspired by [2] and [3], as it provided more stable results compared to the standard Adam optimizer, especially in terms of generalization and convergence.

**Explanation of the choice of parameters**-

Depth+number of neourons:

We considered several options for the depth and hidden dimensions of the neural network. Inspired by [2], we aimed to test MLP architectures with reduced dimensions in each consecutive layer. Additionally, we were interested in architectures with consistent dimensions in each hidden layer, inspired by [1].

Initially, we manually tested a few MLP architectures and observed that networks with 3-4 total layers provided the best performance. Adding more than 4 layers did not lead to any noticeable performance improvement. Based on this, we concluded that the most effective approach would be to focus on MLPs with 3-4 layers during hyperparameter tuning.

Architectures Tested During Tuning:

Reduced dimensions in each consecutive layer architectures:

* (512->512) -(512->256)-(256,128)-(128,10)
* (512->256) -(256,128)-(128,10)
* (512->128)-(128,64) -(128,10)

We chose these architectures to explore how a gradual reduction in neuron count across layers impacts performance. Our goal was to maintain enough neurons in the early layers to capture important features, while progressively reducing dimensions to distill the key information. When manually testing architectures with **more than 512 neurons**, we saw no improvement, and networks with fewer than **64 neurons** didn't perform well enough.

Same size Layers architectures:

* (512->512)-(512->512)-(512->512)- (512->512)
* (512->128)-(128,128) -(128,10)

These architectures were chosen to see if consistency across layers could improve performance by maintaining a more uniform learning process throughout the network. These architectures were inspired by [1]. The specific sizes of 128 and 512 were chosen because we wanted to try the 128 size, same as in [1], and we also wanted to mimic the 4-layer architecture used in [1] but with our input size (512).

Normalization:

We aimed to explore whether normalization applied after the linear layer could improve network performance and accelerate training. Since there wasn't much information available on whether normalization would be beneficial for the INRs dataset, we decided to test whether normalization was necessary at all, and if so, whether Layer Normalization or Batch Normalization would be more effective.

Activations:

Before conducting hyperparameter tuning, we manually tested various activation functions, including Sigmoid, SiLU, ELU, GELU, ReLU, and Leaky ReLU. However, during these experiments, we observed that only ReLU and Leaky ReLUconsistently provided good performance in terms of both accuracy and training stability.

Based on these observations, we decided to focus our hyperparameter tuning specifically on ReLU and Leaky ReLU, as they proved to be the most effective for this task.

Batch size:

We decided to consider [32, 64, 128, 256] as possible batch sizes during hyperparameter tuning. Before this, we manually tested larger batch sizes like 512 and smaller batch sizes like 16, but observed that the performance was suboptimal, and the training was unstable. Larger batch sizes led to slower convergence and reduced generalization, while smaller batch sizes introduced too much noise in the gradients, leading to instability. Based on these preliminary tests, we focused on the more balanced range of 32 to 256 for tuning.

**Learning Rate:**

We chose to explore two learning rates, 0.001 and 0.0001, during hyperparameter tuning. Based on initial experiments, we found that larger learning rates (greater than 0.001) caused the model to oscillate and fail to converge, while smaller learning rates (below 0.0001) resulted in very slow progress.

**Dropout:**

We tested [0, 0.1, 0.3] for dropout during hyperparameter tuning. 0 (no dropout) was included to see if dropout was necessary. The values were chosen based on preliminary manual tests.

**Early Stopping:**

We considered [5, 10] for the early stopping patience. Early stopping is used to halt training if the validation performance doesn’t improve after a certain number of epochs, preventing overfitting and saving computational resources. A patience of 5 epochs was chosen for faster stopping if no improvement was seen, while 10 epochs allowed for more training before halting, in case the model needed extra time to stabilize. This gave us flexibility in balancing between early termination and giving the model enough time to improve.

We didn’t choose a value less than 5 because our scheduler patience is set to 3, meaning the learning rate is reduced after 3 epochs without improvement. By allowing at least 5 epochs for early stopping, we ensure that the effect of the reduced learning rate can be explored before halting the training. This prevents premature stopping and allows the model a chance to improve after the learning rate adjustment.

**Bibliography**

[1] Emilien Dupont, Hyunjik Kim, SM Ali Eslami, Danilo Jimenez Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In International Conference on Machine Learning, pp. 5694–5725. PMLR, 2022.

[2] L. De Luigi, A. Cardace, R. Spezialetti, P. Zama Ramirez, S. Salti, and L. Di Stefano. Deep learning on implicit neural representations of shapes. In International Conference on Learning Representations (ICLR), 2023.

[3] KIM, Taewoon. Generalizing mlps with dropouts, batch normalization, and skip connections. arXiv preprint arXiv:2108.08186, 2021