ML24 Project

Project type A

Team Members:

- Mostafa Eid (eid.m2@studenti.unipi.it)
- Dieudonne Iyabivuze (d.iyabivuze@studenti.unipi.it)
- Matteo Piredda (m.piredda2@studenti.unipi.com)

Master degree of Computer Science(AI)

Date: 20/06/2025

Università di Pisa

Introduction and Objectives

- The main goal of this project is to implement a Neural Network from scratch using python and use it to solve a classification and regression task and the results were tested on MONK and CUP test dataset.

- Also to test how different Neural Network architectures affect the performance.

Contributions

- The code is organized into 20 Python files: 15 files in the src/ folder, 1 folder for datasets, 2 main notebooks (CUP.ipynb and MONK.ipynb), plus requirements.txt and README.md.

- We made use of OOP (Object Oriented Programming) and used it to create parent classes from which the other classes inherited to implement our models which made adding complex features more streamlined.

Contributions (Model Architecture)

- Feed Forward NN (input layer, 2 hidden layers, output layer).
- Mini Batch Gradient Descent.
- Optimizers tested: Adam, Adagrad, RMSprop, Vanilla SGD
- Regularization: Dropout, L1, L2, Early Stopping, weight decay, and batch normalization.
- Initialization methods: Xavier initialization, He Initialization, Random Initialization, Gaussian.
- Activation Functions: ReLU, Leaky ReLU, Sigmoid, Tanh, Linear

Libraries used

NumPy: For fast and efficient numerical operations like matrix multiplications and array manipulations, which are key to neural network computations.

Pandas: Was utilized for data loading, cleaning, and preprocessing, enabling easy handling of structured data in tabular form.

Matplotlib: Was employed to visualize training metrics like loss curves, helping us monitor model performance over epochs.

Implementation Details

Normalization/Scaling: We mainly used min-max and z-score scaling .

Batch Normalization: We introduced batch normalization and it is usually accompanied with a full batch training (batch_size = 1000).

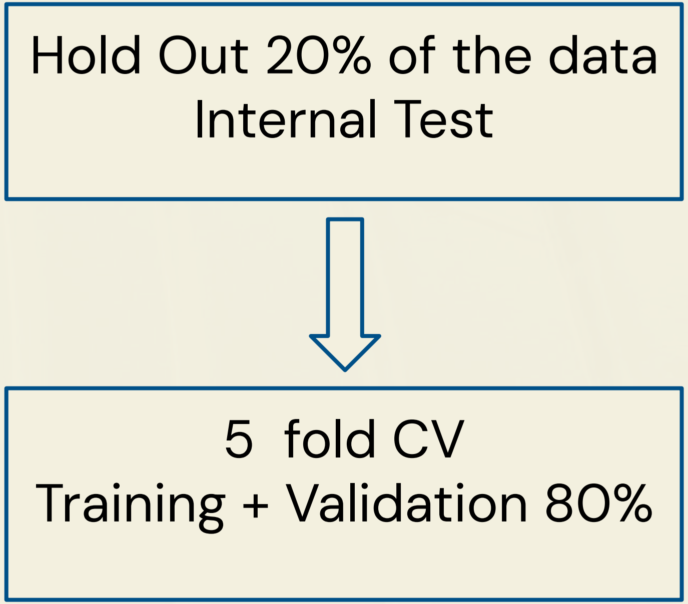
Model Assessment: The process of choosing the best hyperparameters started with several experiments with the parameters by hand, then we use random search with around **500 iterations** which was allowed by early stopping in the training loop.
The hyperparameters use logarithmically scaled spaces with around 10 to 20 values.
We chose logarithmic space over linear space as Searching in log space means that we are exploring orders of magnitude, which is more aligned with how model performance typically responds.

Test-Validation Schema

- In the case of MONK datasets we held out 20 percent of the dataset and used it as a validation after choosing the best model using K-fold CV. We tested on the test set.

- In the case of the CUP dataset we did the same thing but we didn't have an internal testing dataset.

Data Split across
TR/VAL/TEST



Hold Out 20% of the data
Internal Test

5 fold CV
Training + Validation 80%

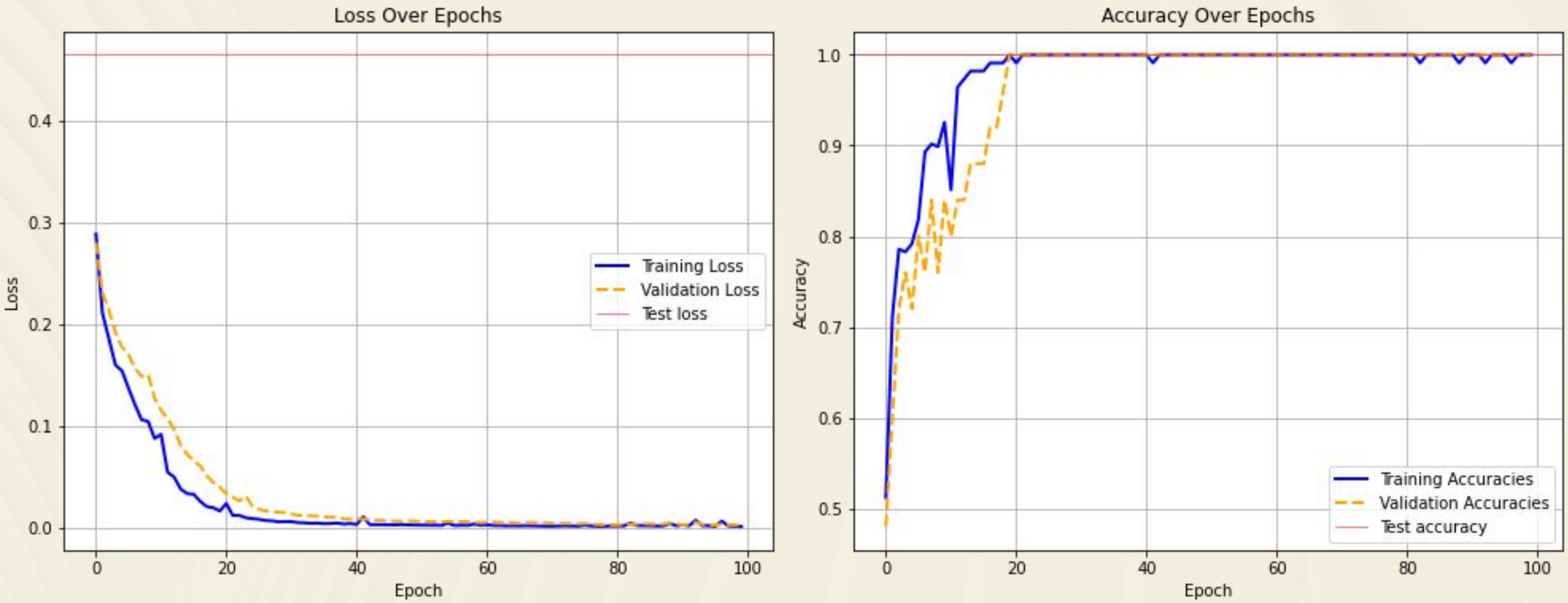
MONK Random Search Parameters

Parameter	Values
hidden size	[3], [4], [5], [6]
hidden activation	[Activation_Tanh], [Activation_Leaky_ReLU], [Activation_Sigmoid], [Activation_ReLU]
batch norm	[True], [False]
learning rate	np.logspace(-3, -1, num=30).tolist()
l1	np.logspace(-5, -1, num=20).tolist()
l2	np.logspace(-5, -1, num=20).tolist()
dropout rate	np.logspace(-5, -1, num=20).tolist()
batch size	[8], [16], [32]
number of epochs	[50], [100]
weight decay	[0], [5e-2], [1e-2], [1e-3], [1e-5]
patience	[10], [30], [50]

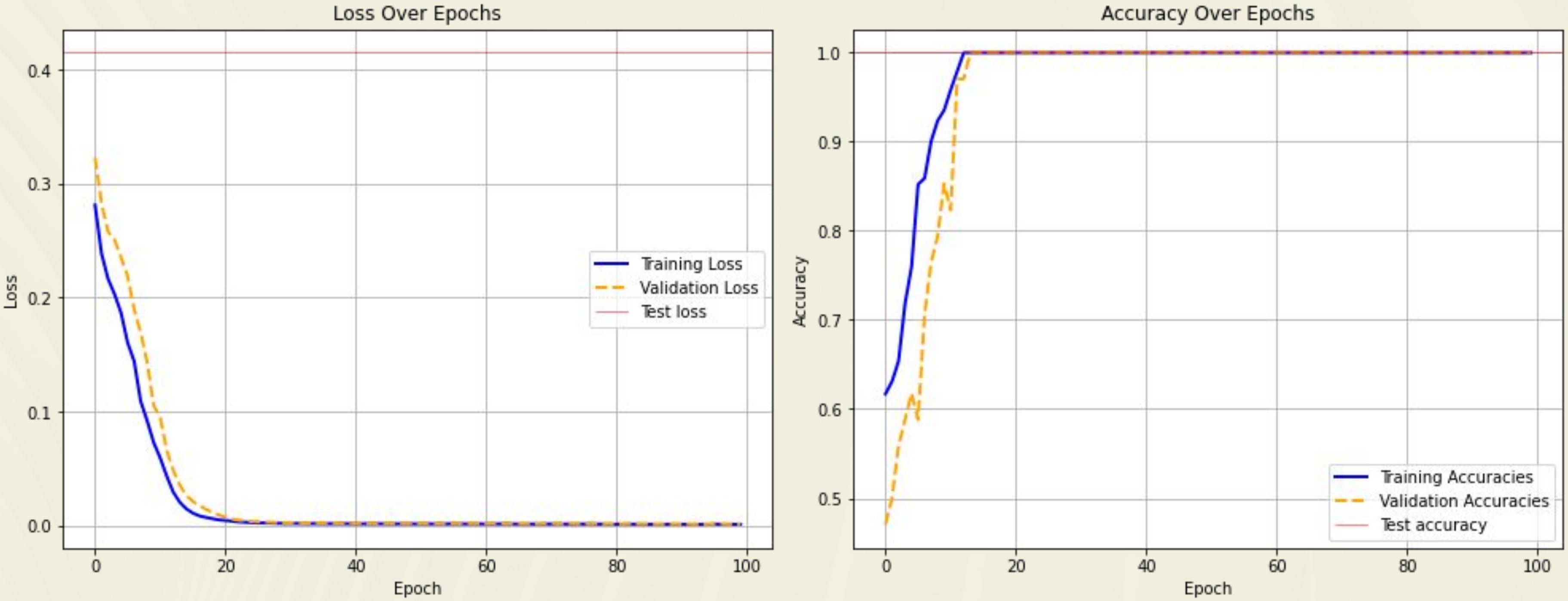
MONK’s Results

Task	hidden_sizes, l1, l2, dropout_rates, use_batch_norm, lr, weight_decay, patience, sched_decay	batch size, activation function, init, epochs	MSE(TR/TS)	Accuracy (TR/TS) (%)
Monk 1	[4], 7.742636826811278e-05, 7.742636826811278e-05, 0.0016, False,0.048, 0.001, 30, 1	16, 'Activation_ReLU', Gaussian, 100	0.0000 / 0.4800	100% / 100%
Monk 2	[5], 1e-05, 0.00021544346900318823, 2.782559402207126e-05, False, 0.0379, 1e-05, 50, 1	16, 'Activation_Leaky_ReLU', Gaussian, 100	0.0000 / 0.4273	100% / 100%
Monk 3	[4], 0.0, 0.0, 0, False, 0.018, 0, 10, 1	32, Activation_Leaky_ReLU, Gaussian, 50	0.0257 / 0.4705	95.88% / 94.4%
Monk 3 (reg.)	[5], 0.0, 0.0, 0.001, True, 0.3, 0.0003, 10, 1	1000, Activation_ReLU, Gaussian, 50	0.0455 / 0.4010	94.85% / 97%

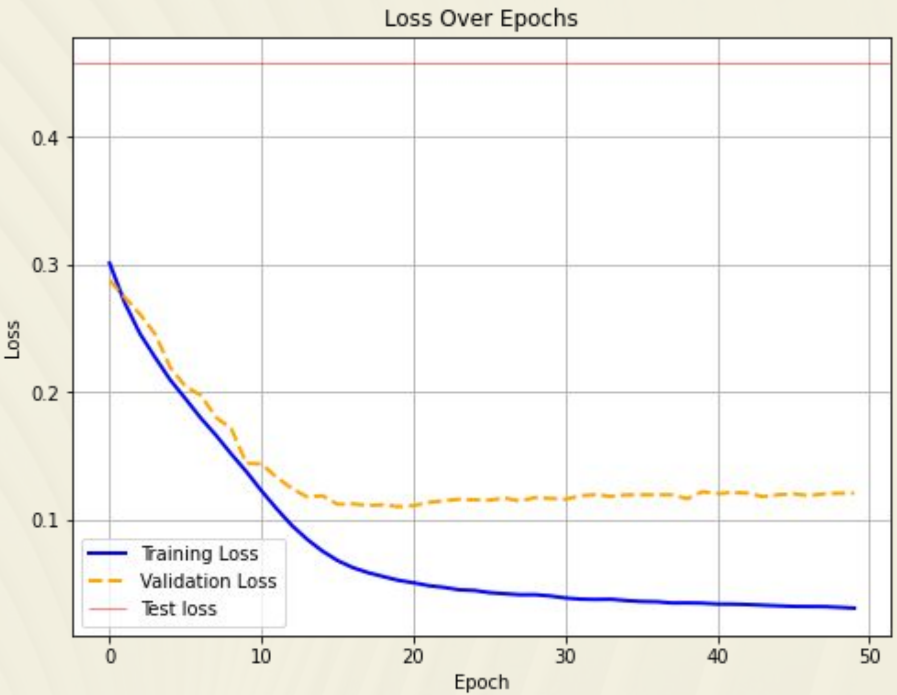
Monk 1 Results

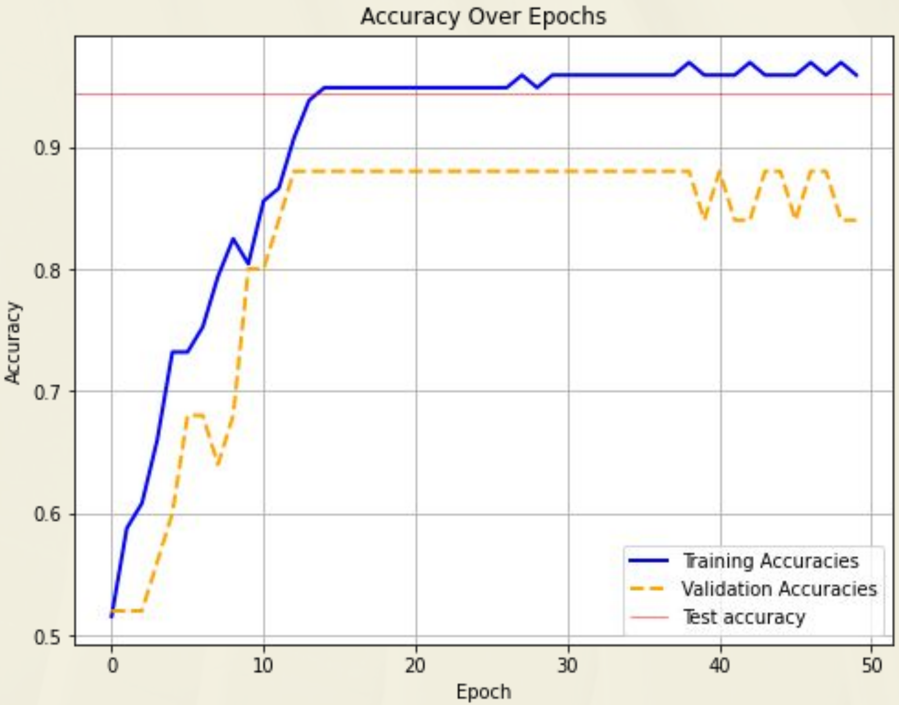


Monk 2 Results

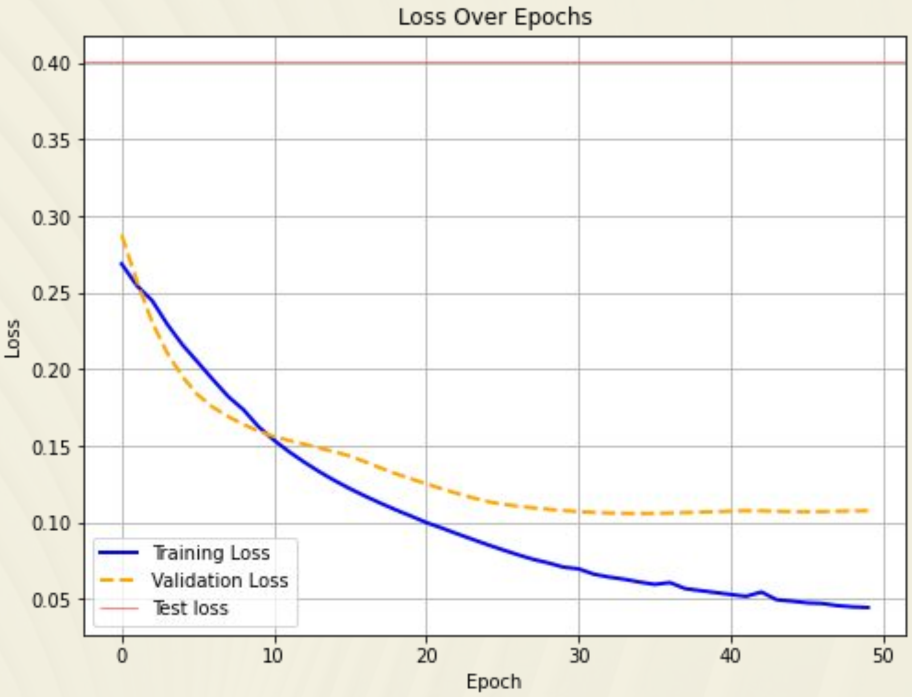


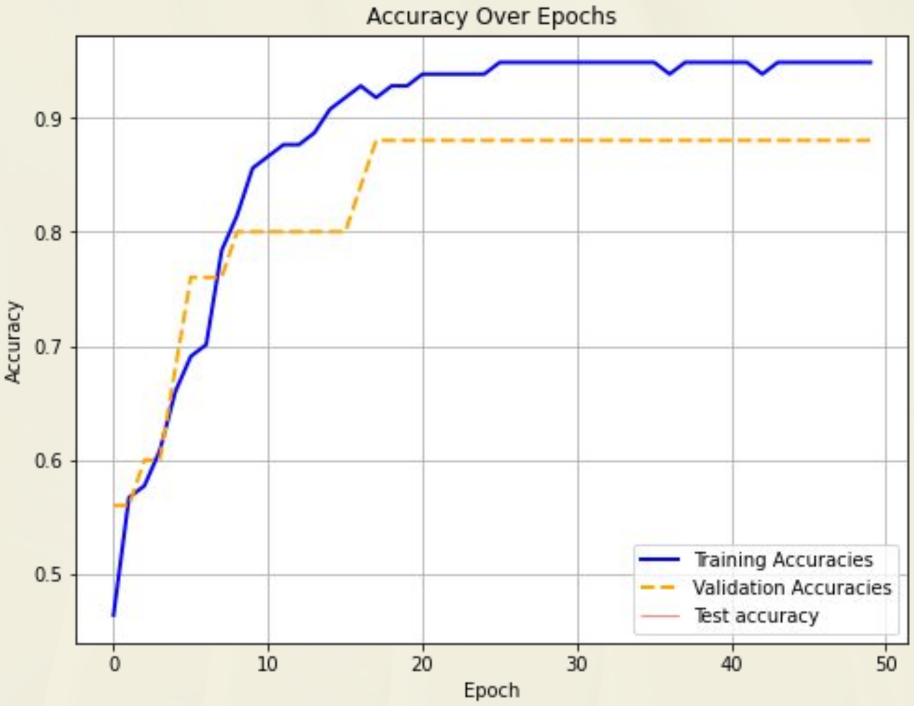
Monk 3 Results





Monk 3 Results With Regularization





CUP’s Architecture and Assessment

Architecture: We explored two architectures for our model:

The first one, used different numbers of neurons for each hidden layer accompanied with different layer related parameters, like activation, batch normalization, and drop out.
The second approach, unified the number of neurons across all hidden layers and gave the same activation, drop out rates, and batch normalization to all of them.

After testing both approaches on the same parameters grid we found out that the First approach produces better results and that is why it was finally used to produce the results.

We **re-trained** the model on all the training data before producing the results of the test set for submission.

CUP Random Search Parameters

Parameter	Values
Number of hidden layers	[2,3]
hidden size	[8, 16, 32, 64]
hidden activation	[Activation_Tanh], [Activation_Leaky_ReLU], [Activation_Sigmoid], [Activation_ReLU]
batch norm	[True], [False]
learning rate	np.logspace(-3, -1, num=30).tolist()
l1	np.logspace(-5, -1, num=20).tolist()
l2	np.logspace(-5, -1, num=20).tolist()
dropout rate	np.logspace(-5, -1, num=20).tolist()
batch size	[8], [16], [32]
number of epochs	[100, 200, 300]
weight decay	[0], [5e-2], [1e-2], [1e-3], [1e-5]
patience	[0, 30, 50]
weight initialization	[gaussian, gaussian_scaled, xavier, he, random]

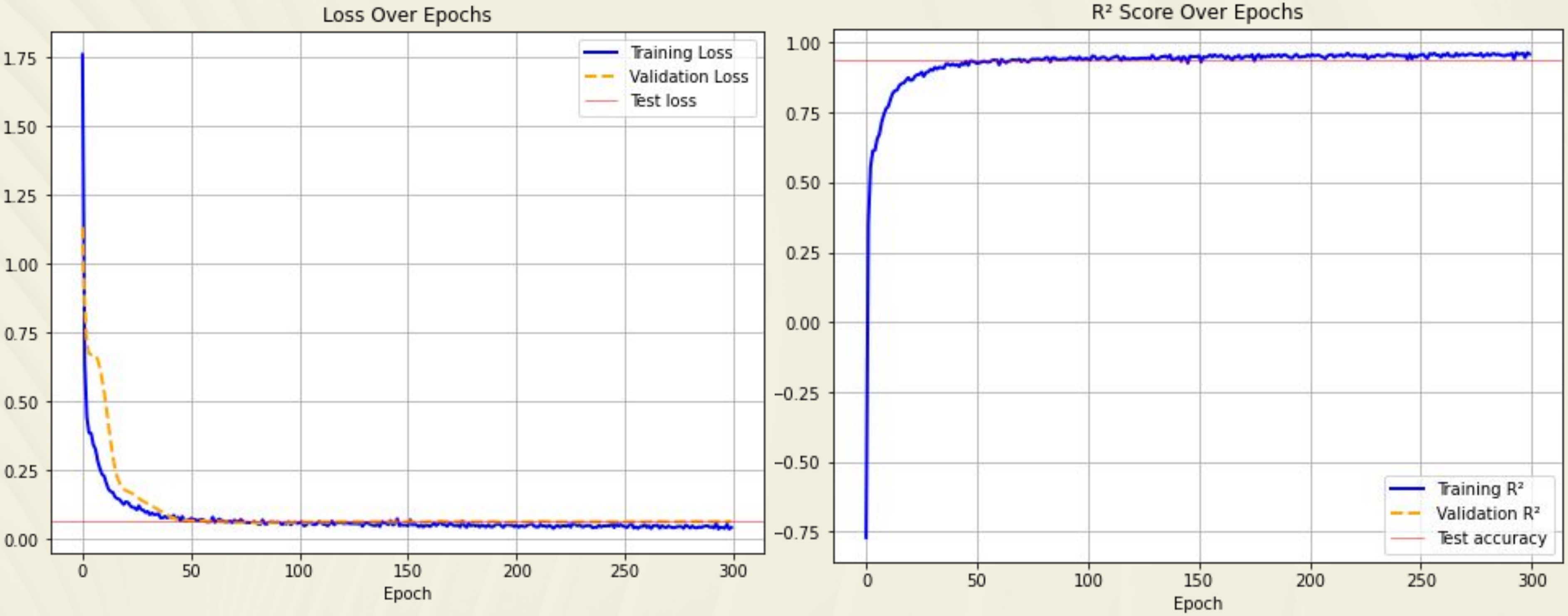
ML Cup 1st Approach

Hyperparameters	Possible Values	Hyperparameters	Possible Values
batch_norm	True,True, True	Hidden sizes	32, 8, 8
l1	1.83E-04	Epsilon	1.00E-07
l2	7.85E-04	momentum	0.9
Batch size	1000 (Full size)	Normalization Type	Z-score
Epochs	300	Dropout Rates	0.0012742749857031334, 0.00011288378916846884, 0.008858667904100823
Patience	20	weight_decay	5.00E-02
CC	FALSE	learning_rate	7.70E-02
sched_decay	0	loss	MSE/MEE
n_h_layers	3	activation function	ReLU, Sigmoid, Tanh
Weights_init	he	output activation	Linear

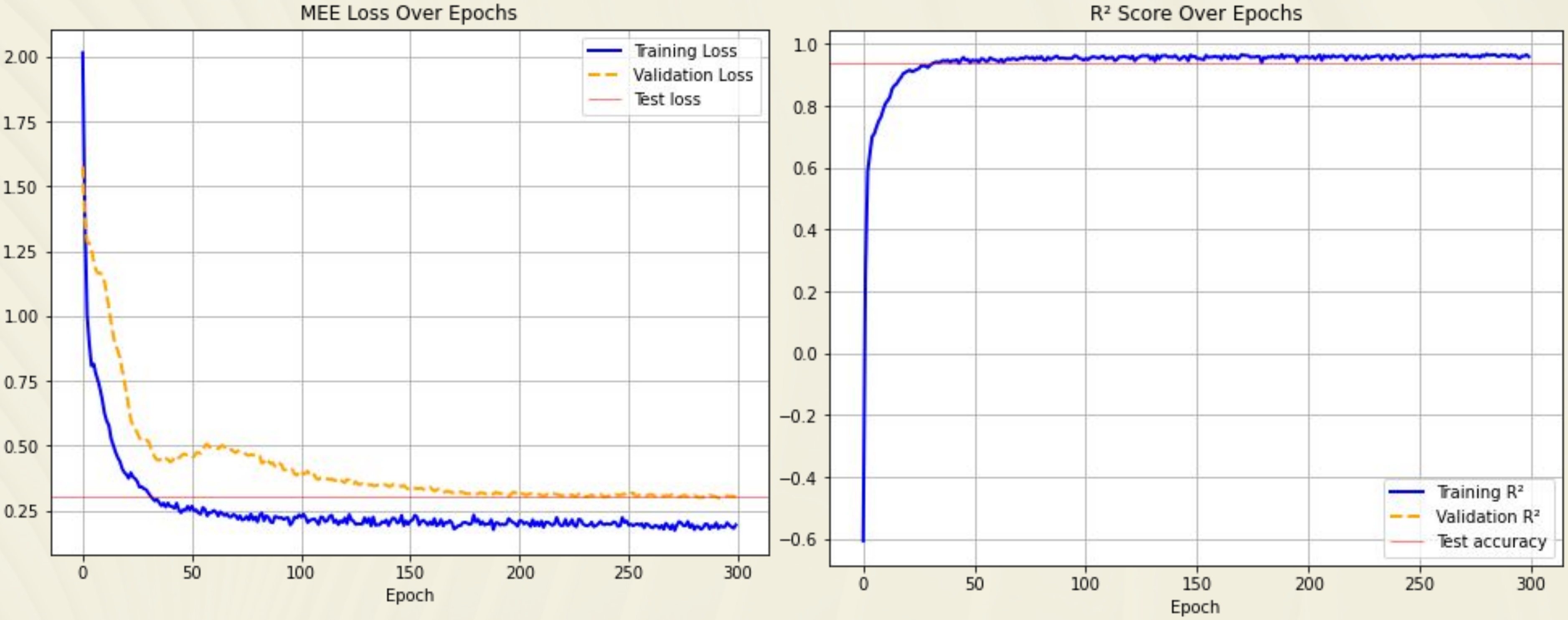
Cup’s final Results 1st approach

Metrics	Values	R2 Score(TR/VL)
MSE (TR/VL)	0.0588 / 0.062	0.940 / 0.917
MEE (TR/VL)	0.222 / 0.300	0.945 / 0.923

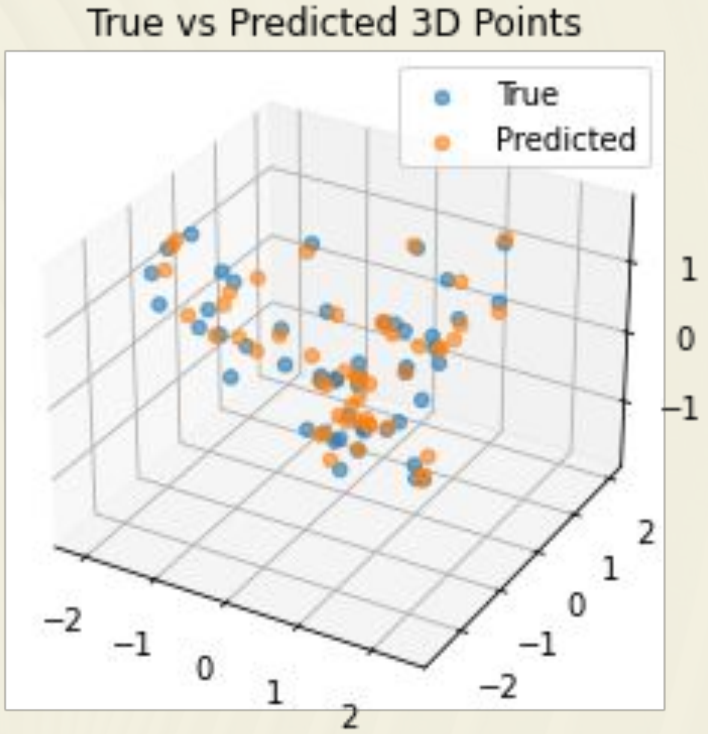
Cup Results MSE

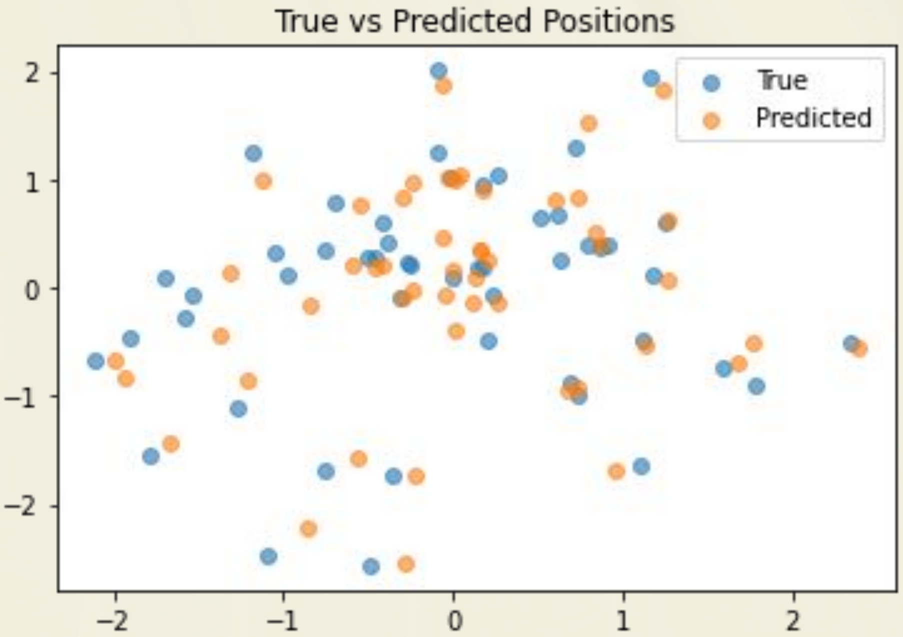


Cup Results MEE



Cup Results - figures





Discussions

We actually tried and implemented two different model structures for cup dataset: one uses same activation functions, number of neurons, drop_out for all layers whereby other chooses different hyperparameters for each layer.

We used different techniques for regularization of our models such as **l1, l2, dropout**, to make our models generalize well and also used **early stopping** to stop the training process before the model fully converges, preventing overfitting to the training data.

We also explored different weight initialization methods such as Xavier, Random, He initialization to make sure that our model does not underfit or experience gradient exploding or vanishing

Discussions - Ensemble learning

We also implemented ensemble learning methods to aim for a higher quality and generalization of the trained model.

- **Majority voting**: Each individual network was trained independently with different hyperparameters and then the final prediction for each sample was determined by the most common prediction among the ensemble members.

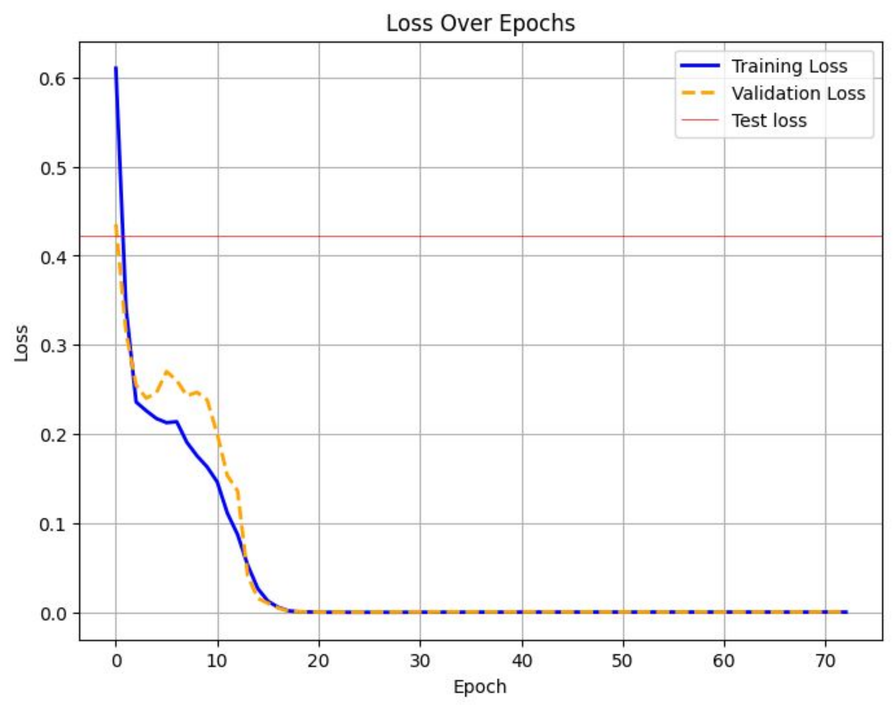
- **Cascade correlation** allows the model to dynamically grow by adding hidden units that maximally reduce error at each stage.

The following graphs will show how these two models behave in the MONK2 dataset.

Majority voting

- **Combines predictions** from multiple models (classifiers).
- Final output is the class **most voted** by the models.
- Helps reduce **variance** and improves **robustness**.
- Works best when individual models are **diverse** and **weakly correlated**.

The following graph shows how majority voting behaves in the MONK2 dataset.

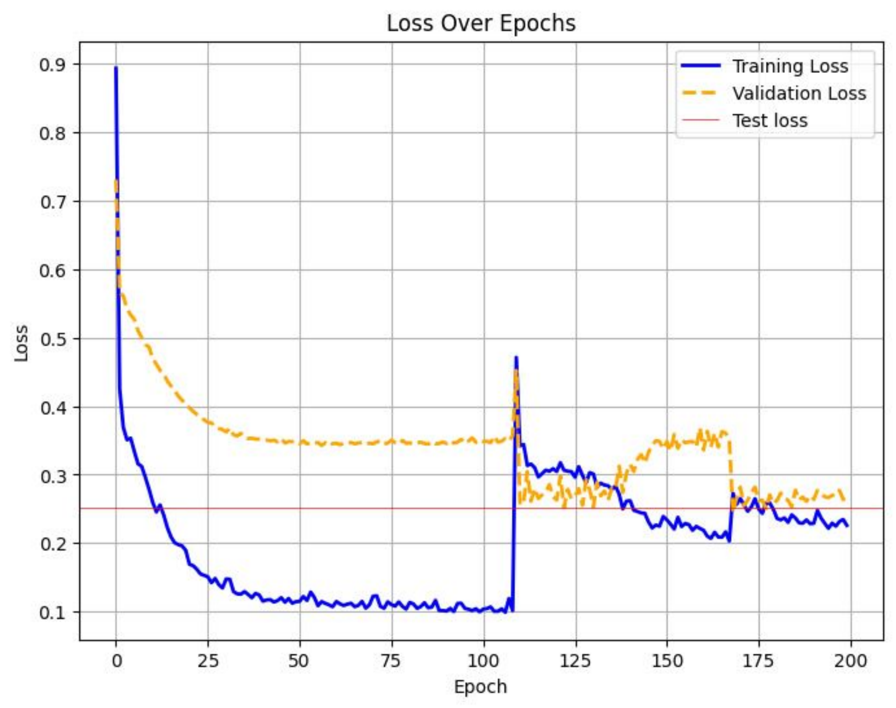


Cascade correlation

- **Dynamic architecture**: Starts small and **grows** by adding hidden units.
- **New neurons** are added one at a time and **frozen** after training.
- Each added neuron **maximizes correlation** with residual error.

Why it has not been chosen:

- **Unstable** validation performance.
- **Non consistent** improvement over time.
- Classic NN likely showed **better generalization** and **smoother learning**.



The following graph shows how cascade correlation behaves in the MONK2 dataset.

Conclusion

This project was a good chance to practice what we learned about machine learning and deep learning in class. It helped us better understand how neural networks work and how machine learning works in general. We also learned why important parts like model selection, evaluation, and choosing the right hyperparameters can greatly affect how well a model performs.

Submission file name: last_shot_ML-CUP24-TS.csv

Nick name: last_shot

Bibliography

[1] Neural Networks from scratch

[2] Machine learning A.Micheli lessons and slides

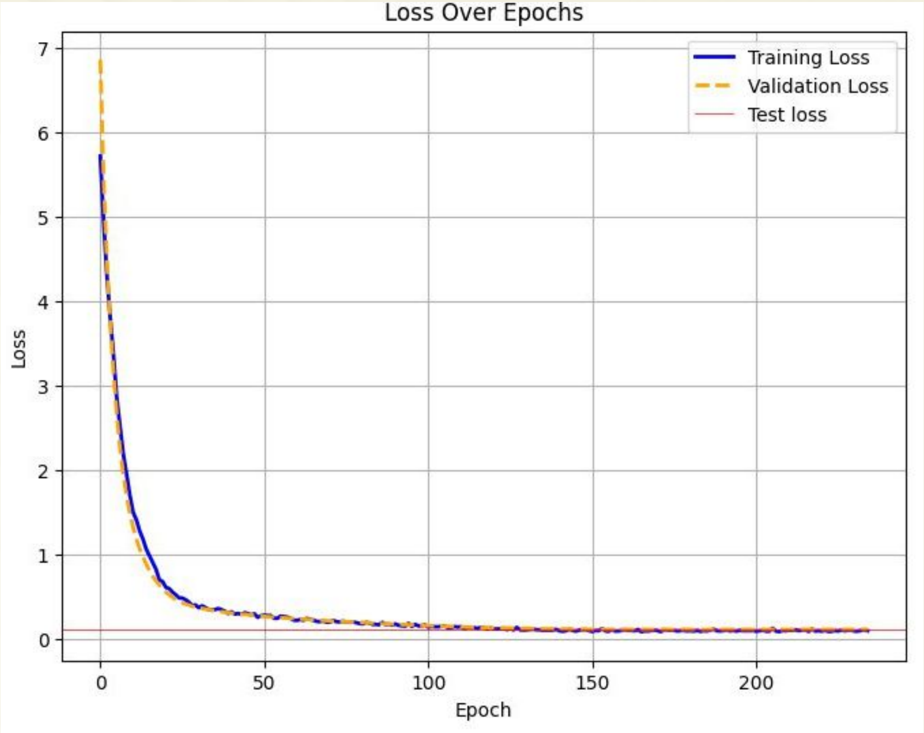
[3] Deep Learning - Ian Goodfellow & Aaron Courville & Yoshua Bengio

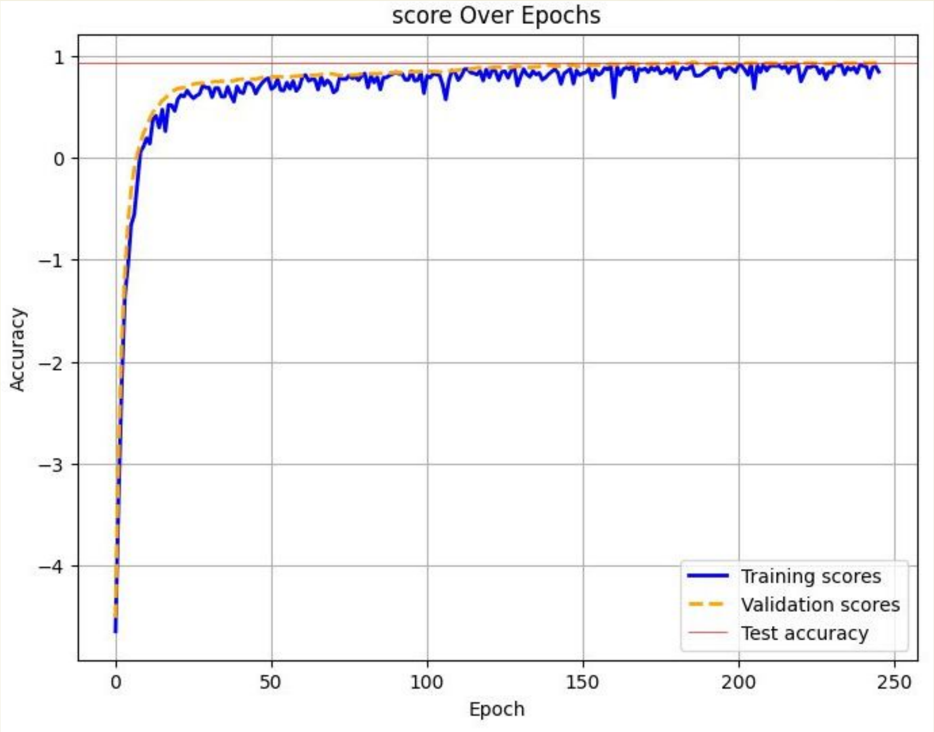
Appendix

ML Cup 2nd Approach

Hyperparameters	Possible Values	Hyperparameters	Possible Values
batch_norm	TRUE	Hidden sizes	10
l1	1.00E-09	Epsilon	1.00E-07
l2	1.50E-03	momentum	0.9
Batch size	64	Normalization Type	Z-score
Epochs	350	Dropout Rates	0.0
Patience	60	weight_decay	1.00E-07
CC	FALSE	learning_rate	7.70E-02
sched_decay	2	loss	MSE, MEE
n_h_layers	2	activation function	Activation_Tanh, Linear
Weights_init	Random		

Cup Results 2nd Approach (figures)





ML Cup 2nd Approach final results

Dataset	MSE (TR/VL)	MEE (TR/VL)
ML CUP 2st Approach	0.0347 / 0.0753	0.2557 / 0.2921

The result of the second approach for the ML cup data: It uses the same activation functions, number of neurons, dropout, for all the layers.