VNL prediction using Feature Imitation

Mateusz Budzyński, Maciej Ciepiela, Mikołaj Komarnicki

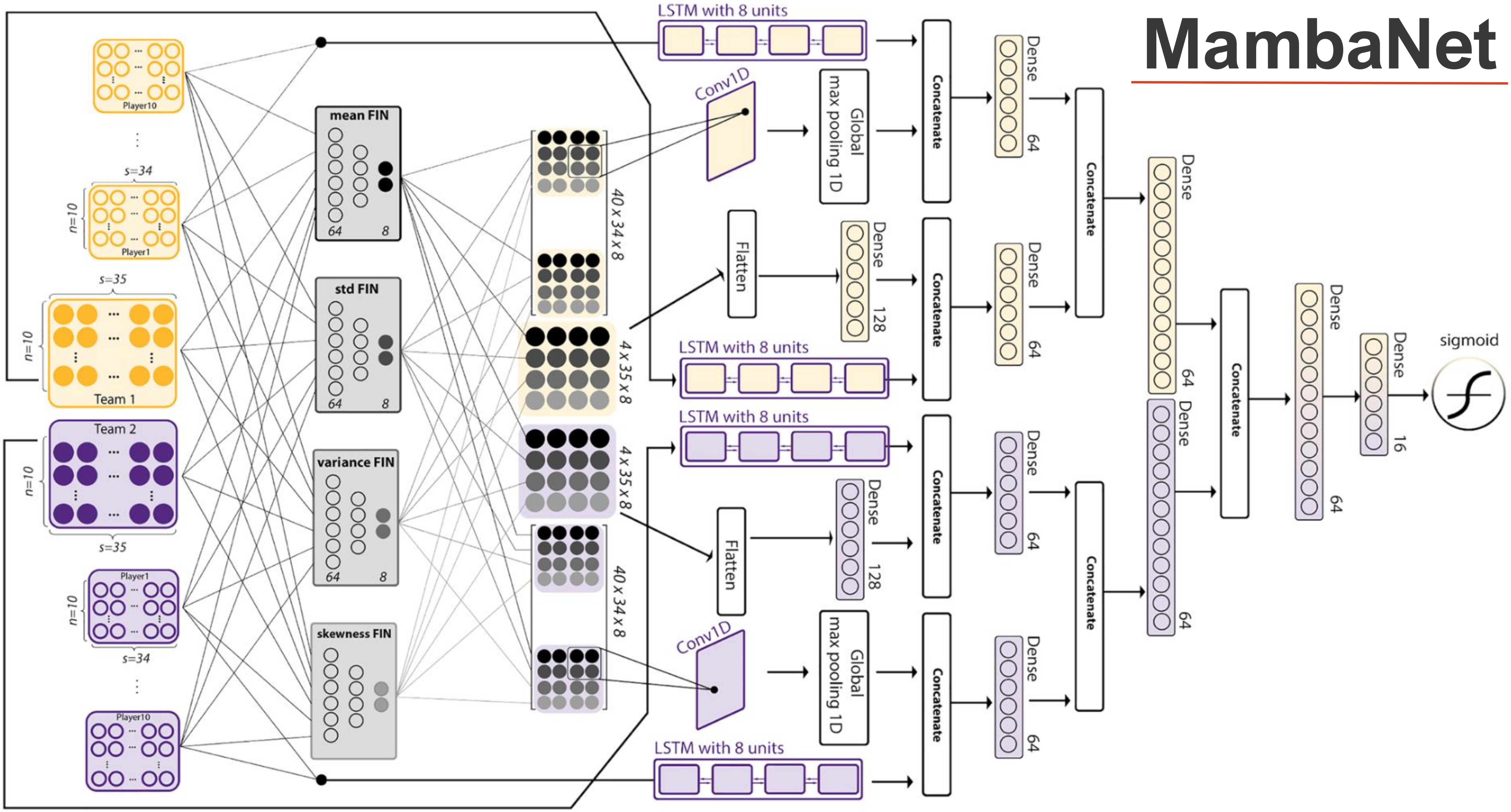
What is this project about?

1. Our main idea was to create simple network that would predict volleyball game outcome based on statistics of the players in each team.

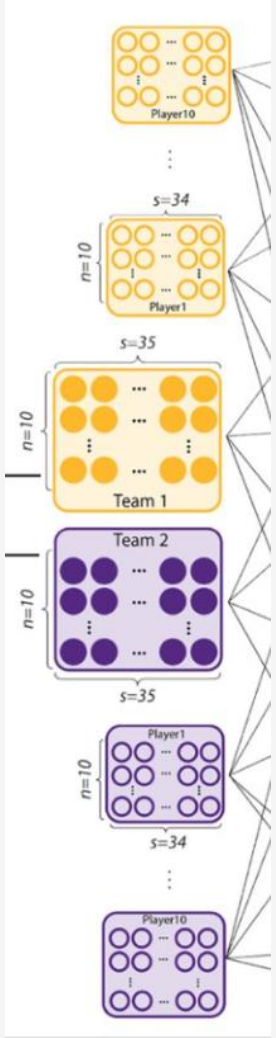
2. During research for this topic we encountered similar problem based on NBA games, that used an interesting structure called FIN.

3. We decided to try, proposed in the paper, neural network architecture and by using different metrics compare the results with simpler methods.

MambaNet


LSTM with 8 units
Player10
s=34
n=10
Player1
s=35
n=10
Team 1
Team 2
n=10
s=35
Player1
n=10
s=34
Player10
mean FIN
64
8
std FIN
64
8
variance FIN
64
8
skewness FIN
Conv1D
Global max pooling 1D
40 x 34 x 8
4 x 35 x 8
4 x 35 x 8
40 x 34 x 8
Flatten
Dense
128
LSTM with 8 units
LSTM with 8 units
Flatten
Dense
128
Conv1D
Global max pooling 1D
LSTM with 8 units
Concatenate
Concatenate
Concatenate
Concatenate
Dense
64
Dense
64
Dense
64
Dense
64
Concatenate
Concatenate
Dense
64
Dense
64
Concatenate
Dense
64
Dense
16
sigmoid

Input preprocessing



- The project is based on Men's Volleyball Nations League from years 2021 – 2023.
- Each input is constructed in the same way as it is in original MambaNet network.
- For each match between two teams. Get their 10 last games and top players performance in them.

Each match record consists of 25 shared features:

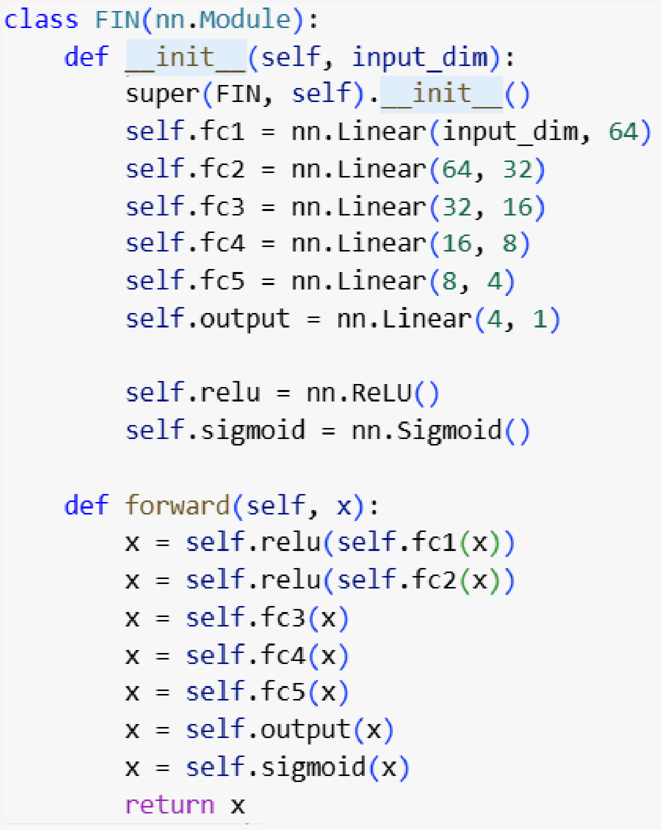
Opponent				
Serve :	Points	Errors	Attempts	Total
Set :	Successes	Errors	Attmpts	Total
Attack :	Successes	Errors	Attempts	Total
Block :	Successes	Errors	Rebounds	Total
Reception :	Successes	Errors	Attempts	Total
Dig :	Successes	Errors	Attempts	Total

Each match team record has 12 exclusive features additionally

Sets Won	Sets Lost	Total Points	Points Diff
Avg set margin	Final set margin	Tight sets	Set 1 Diff
Set 2 Diff	Set 3 Diff	Set 4 Diff	Set 5 Diff

Feature Imitation Network (FIN)

1 Architecture

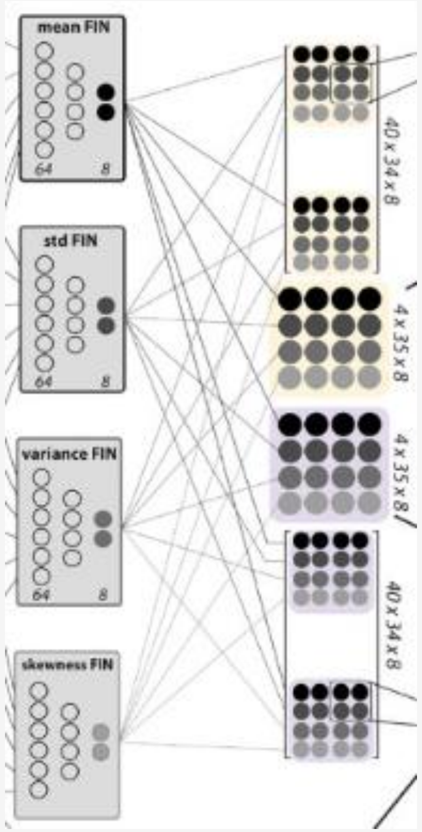


```python
class FIN(nn.Module):
    def __init__(self, input_dim):
        super(FIN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 16)
        self.fc4 = nn.Linear(16, 8)
        self.fc5 = nn.Linear(8, 4)
        self.output = nn.Linear(4, 1)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        x = self.fc4(x)
        x = self.fc5(x)
        x = self.output(x)
        x = self.sigmoid(x)
        return x
```

2 In short

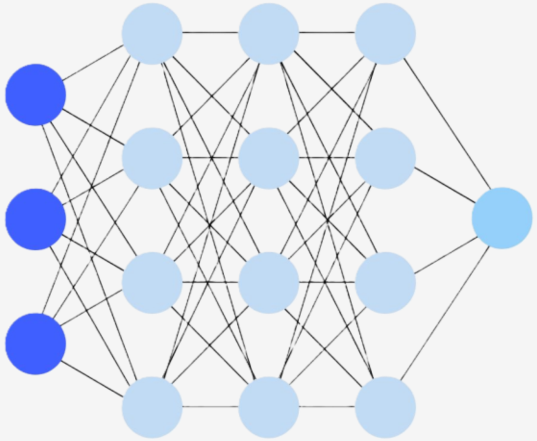
1) Train the model on randomly generated signals.
2) Freeze the first two layers and fine-tune on real data.
3) Remove the last two layers and extract embeddings
4) Process the input data for compatibility.

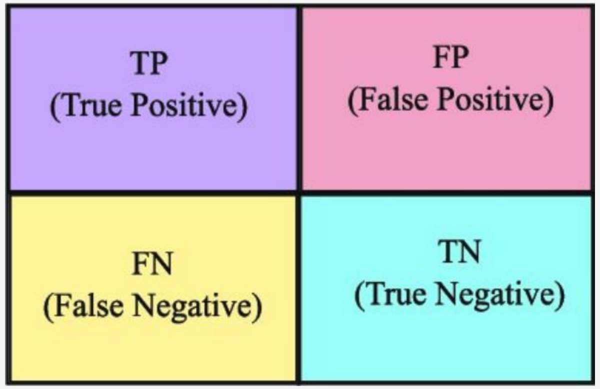
3 Integration*



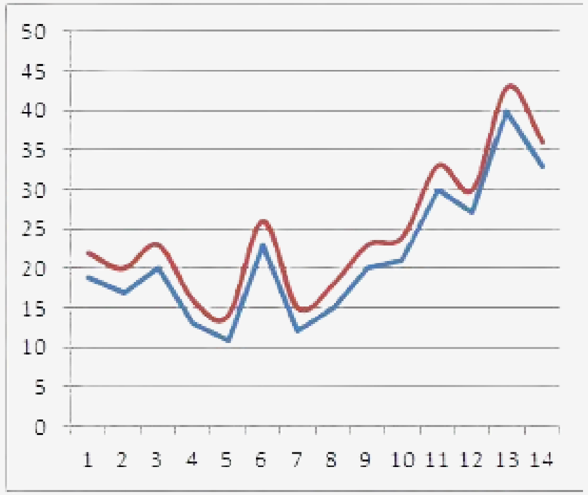
* 34->25, 35->37

Methods for comparision





TP (True Positive)	FP (False Positive)
FN (False Negative)	TN (True Negative)

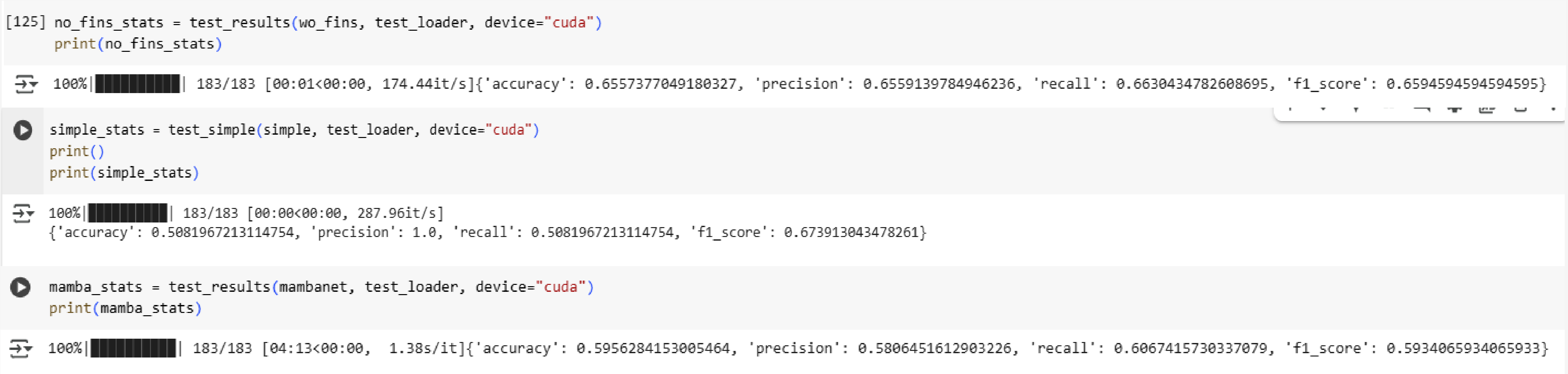


1. We created a neural network based on simple linear layers, avoiding the use of FINs. We also trained a Mamba-based network, both with and without FINs, to compare its performance with the baseline model.

2. Next we prepared functions calculating different metrics, such as accuracy, precision, recall and f1 score.

3. Then we compared and analysed achieved results.

Results and conclusions



```
[125] no_fins_stats = test_results(wo_fins, test_loader, device="cuda")
      print(no_fins_stats)

100%|██████████| 183/183 [00:01<00:00, 174.44it/s]{'accuracy': 0.6557377049180327, 'precision': 0.6559139784946236, 'recall': 0.6630434782608695, 'f1_score': 0.6594594594594595}
```

```
simple_stats = test_simple(simple, test_loader, device="cuda")
print()
print(simple_stats)

100%|██████████| 183/183 [00:00<00:00, 287.96it/s]
{'accuracy': 0.5081967213114754, 'precision': 1.0, 'recall': 0.5081967213114754, 'f1_score': 0.673913043478261}
```

```
mamba_stats = test_results(mambanet, test_loader, device="cuda")
print(mamba_stats)

100%|██████████| 183/183 [04:13<00:00,  1.38s/it]{'accuracy': 0.5956284153005464, 'precision': 0.5806451612903226, 'recall': 0.6067415730337079, 'f1_score': 0.5934065934065933}
```

- FIN-Enhanced Model (Target Architecture)
  - Achieved moderate performance (F1: 0.59)
  - Potential over-engineering: Added complexity from FIN embeddings might have introduced noise rather than useful features.
- Baseline (Linear NN)
  - Weakest results (F1: 0.67* but with precision-recall imbalance, looks like random)
  - Confirms that non-linear patterns exist in the data.
- Main Network *Without* FINs
  - Best performance (F1: 0.66, fastest inference)
  - Suggests:
    - FINs might have overcomplicated the pipeline
    - Raw features were sufficient for this task
    - Possible information loss during FIN's dimensionality reduction

Thanks for attention!

- Inspiration paper
- Notebook with all of the code
- Raport