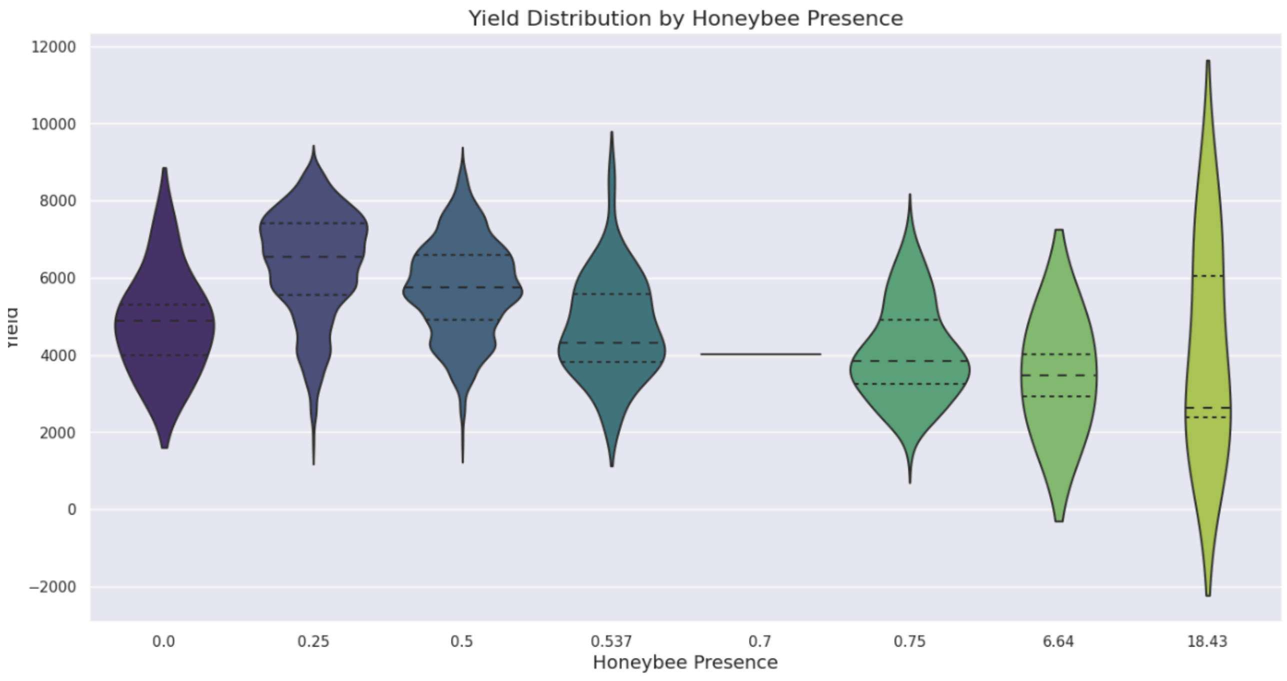
Deep Learning for Blueberry Yield Prediction

For our final project, we join a Kaggle competition, namely DeepBlueberry Challenge 2025. In this competition, we build several deep learning models, such as CNN, RNN, LSTM, Transformer, and so on, mainly to solve the problem of predicting blueberry yield based on various environmental and pollination factors. We chose to join this competition because not only we found it more interesting than some of the other competitions, but we also think that this project might be able to contribute to precision agriculture and optimize harvest logistics of blueberries produce. All of the models we built achieved an MAE of less than 300, which means they all perform well. Our model with highest performance was able to achieve a mean absolute error of 255.544 on the leaderboard. The most interesting thing we find when we were working on the project is that the presence of bees is extremely important to the yield of blueberries. Below is a plot of the yield distribution by one of the bee kinds, honeybee presence.



Group Members: Qingyuan Liu, Weiqi Zhang

1. Introduction

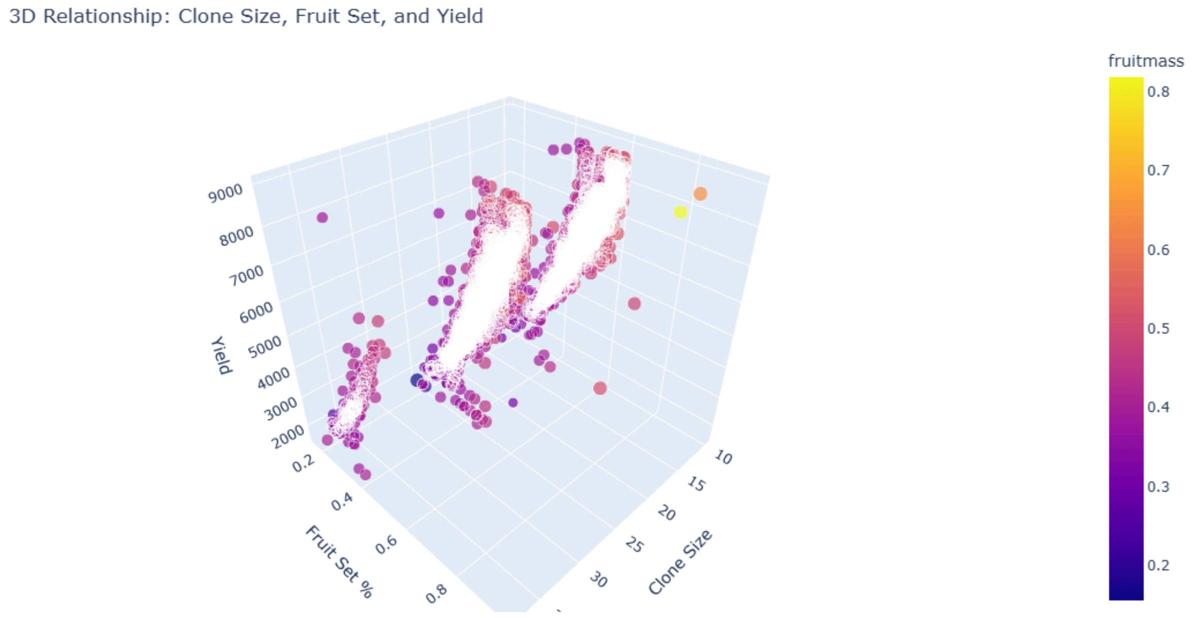
We have already seen a lot of applications of deep learning neural networks in our daily life. The most popular examples will be self-driving cars, natural language processing(GPT and Deepseek) and robotics in all types of industrial applications. With the background of all those applications, we are willing to push the boundaries further in the farming industry. In this Kaggle competition of deep learning for predicting the blueberry yield, we will focus on utilizing only deep learning neural networks to do this prediction and reduce the prediction error as low as possible. This competition offers us an opportunity to apply the state-of-art neural network techniques to a real-world agricultural problem. Our goal is to drive AI-driven advancements in precision farming and sustainable agriculture.

The rule of the competition is simple: given a training and a testing dataset of environmental and agricultural factors of blueberry produce, the participants need to use a pre-trained or build their own deep learning model to predict blueberry yield. The data includes plant and pollinator features, such as size of the plant clone, presence of different kinds of bees, temperature data, such as maximum, minimum, and average temperature, rainfall data, such as the total and average number of rainy days in a given period, yield and productivity data, such as percentage of flowers that develop into fruit, average mass of individual fruits, and number of seeds per fruit. One thing to note is that only the training dataset contains data for yield of blueberries. Therefore, we have to build models to predict the test dataset's values.

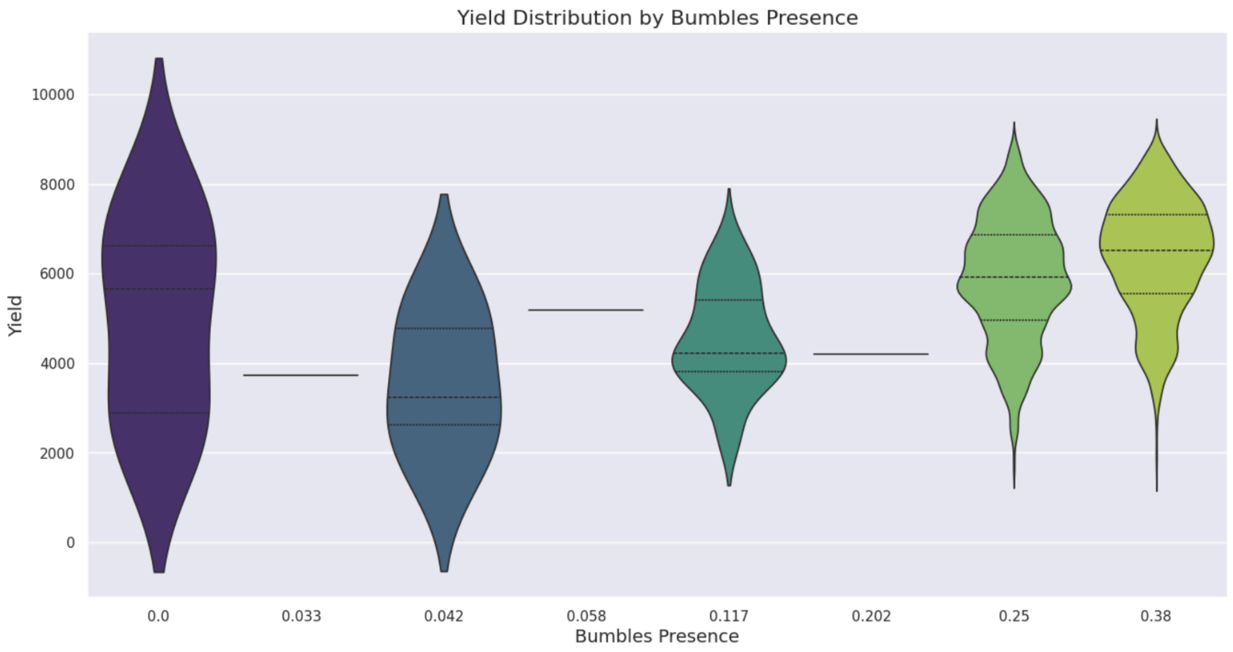
We started by searching online for resources that may previously have done similar research. However, all the papers we found are about applying deep learning or neural networks on imagery data while the data we are dealing with is numerical. Nonetheless, most of the concepts are similar and can be applied to our project. Therefore, we decided to build the five following models: CNN, RNN, LSTM, Transformer, and some kind of hybrid neural network.

2. Details of the approach

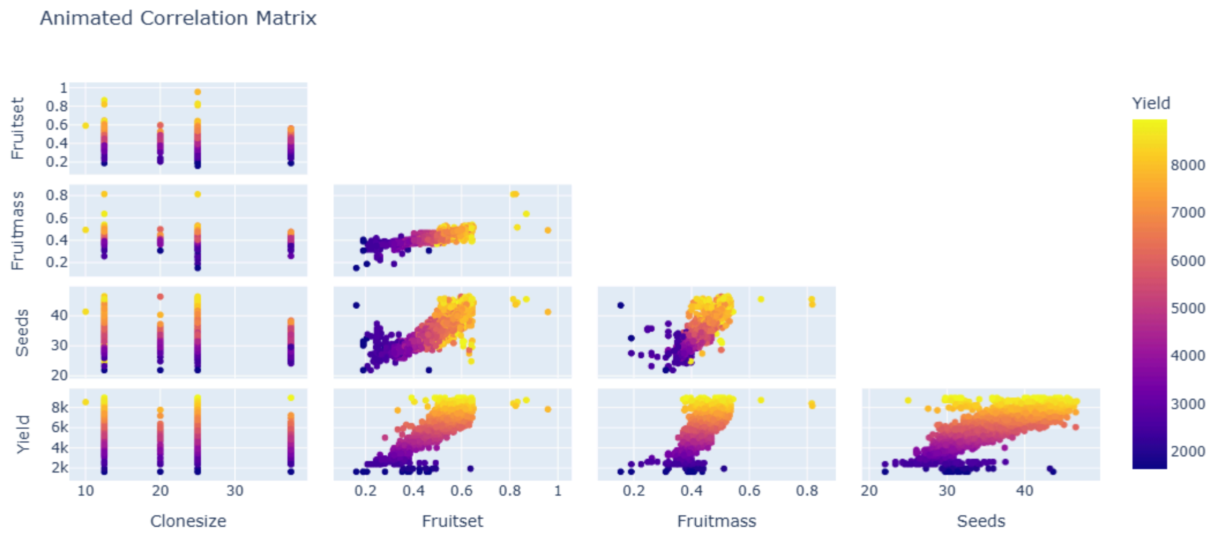
Before building the models, we did a little data preprocessing, pre-engineering, and visualization. We plot a 3D relationship between clone size, fruit set, and yield. On the plot, we can see that within each clone size band, higher fruit set percentage corresponds to higher yields. Moreover, within the same fruit set, larger clone size corresponds to higher yields. On the other hand, fruit mass plays a less important role in the yield outcome.



We also analyzed the relationship between yield and bee presence as pollinators. Take bumbles as an instance. In the following plot, we can see that with no bumbles, the yield ranges roughly from 2000 to 8000 with a median of 6000. With 0.38 of bumble presence, the yield now ranges roughly up to 8000 with an even higher median of 6600. On the plot on the cover page, we can also see a depression in yield when bee percentage is too high. From this, we can conclude that when there are no bees or very few or many bees, it does not improve yield. However, when there is a moderate number of bees, yield rebounds strongly.



Last but not least, we plot a pairwise correlation matrix that captures yield, seeds, fruit mass, clone size and fruit set. On the plot, we can see that as the fruit set increases, yield also increases rapidly. Number of seeds seems positively correlated to fruit set and thus more seeds also boost yield. Clone sizes set the baseline for yield, lifting the overall ceiling. Fruit mass has a mild correlation with yield and can only provide moderately supplementary boost.



Now we will dive deeper into the actual implementation of our models.

a. CNN

The first implementation will be the Convolutional Neural Network(CNN). The structure and basic ideas are to feed the input image to convolutional layers(the number of layers are defined by users) and through the feature maps to flatten the data and to do the further network layer training(shown in the figure 1). Here, we have the input data separated by its features, such as bumbles, seeds, clone sizes and so on. We will then split the data into 80% training and 20% validating for improving the accuracy of the training model. After all the normalization and reshaping for the input data, we define the CNN regression model as following:

1. Use Conv1D with kernel size equal to 5 over the sequential training.
2. Use BatchNorm1d to stabilize the training data.
3. Use MaxPool1d to decrease the overall size of the computation.
4. Use Linear to flatten data for further fully connected layers training.
5. Use an optimizer(AdamW) to improve the weight delay.
6. Get the output and perform validation

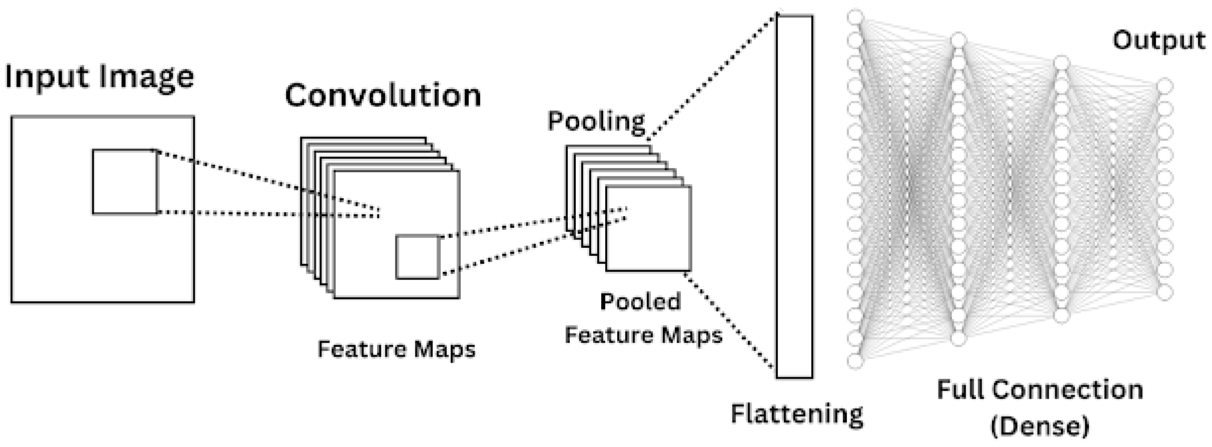


Fig 1: CNN implementation diagram

b. RNN

The second implementation is Recurrent Neural Network(RNN). The basic idea is to feed the sample of each block as a sequence of feature-vectors into the RNN blocks – the users will define its recurrent numbers and sequence length. Again, we will separate the data by its features and split 80% for training, 20% for validating. After all the normalization and reshaping for the input data, we define the RNN regression model as following:

1. Use GRU as our basic block with bidirectional structure so that we can combine forward and backward together
2. For the forward state, we will return hidden states for the next stage until the last one.
3. Use Relu to faster training and introduce non-linearity.

4. Use an optimizer(AdamW) to improve the weight delay.
5. Get the output and perform validation.

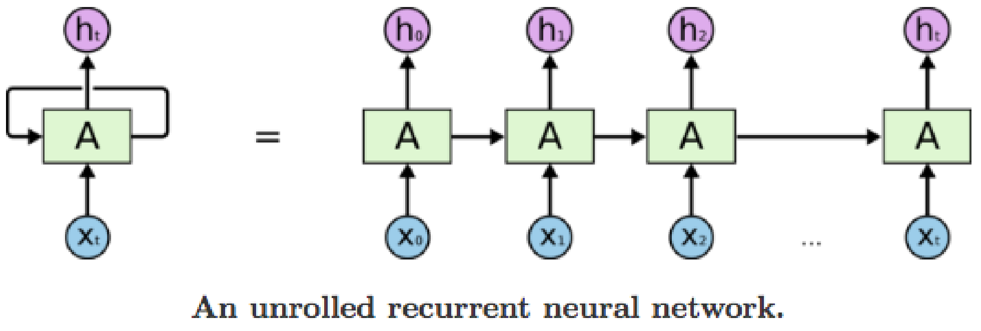


Fig 2: RNN implementation diagram

c. LSTM

The next implementation is the Long short-term memory model(LSTM). Similarly, the LSTM will take in the input vector $x_{□}$, the previous hidden state $h_{□-1}$, and cell state $C_{□-1}$ to compute four transforms output. It will then update the cell from the following equation:

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t$$

And we can have the following hidden states:

$$h_t = o_t \odot tanh(C_t)$$

All the remaining parts should be similar to the implementation of RNN. However, to save the works for the tuning parameters to find the best parameters for the LSTM, we will implement the design by applying the nn.LSTM in the design:

1. Use LSTM with input size equal to number of features, hidden size equal to 128 and number of layers equal to 2
2. Add a dropout
3. Use Relu to speed up training and introduce non-linearity.
4. Use an optimizer(Adam) to improve the weight delay.
5. Get the output and perform validation.

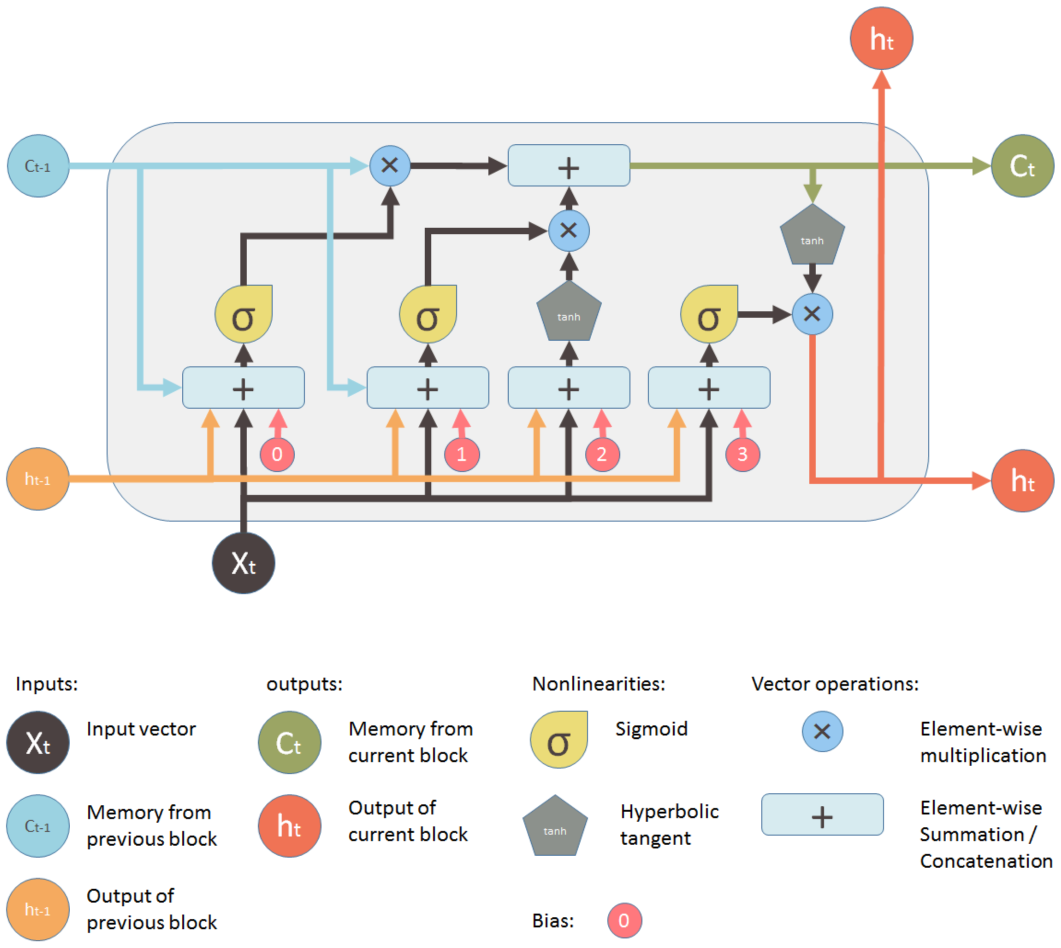


Fig 3: LSTM implementation diagram

d. Transformer

The basic idea of transformer regression is to make every sample as the sequence of the scalar features and embed them to higher dimensional space. We will also include an encoder for the transformer to learn the pairwise interactions parallelly. After the reshaping and normalization, we define the Transformer regression model as follows:

1. Embed each scalar feature into a dimension vector.
2. Add sinusoidal position encoder for the feature extraction.
3. Stack the encoder layers and let each of them have multi-head self attention, layer norm and position-wise forward sublayer.
4. Use Global-Average-Pool to combine all the sequence results into one vector.
5. Use Relu to introduce non-linearity.
6. Use Adam as the optimizer to improve generalization and control weight decay.
7. Monitor the MAE change for each epoch

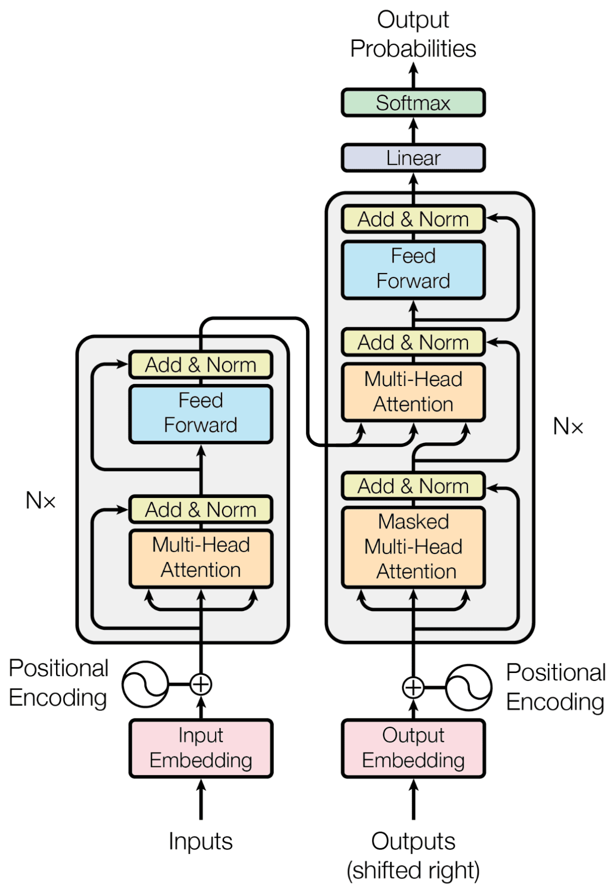


Fig 4: Transformer implementation diagram

e. Hybrid

For the hybrid model implementation, as shown in the figure 5, we have the CNN layers followed by the LSTM layer. The basic idea is to use CNN layer to first produce flattened results and right before the fully connected dense layer, we have another LSTM layer to produce both the memory and current block values. In addition to that, other parts will be similar to CNN implementations

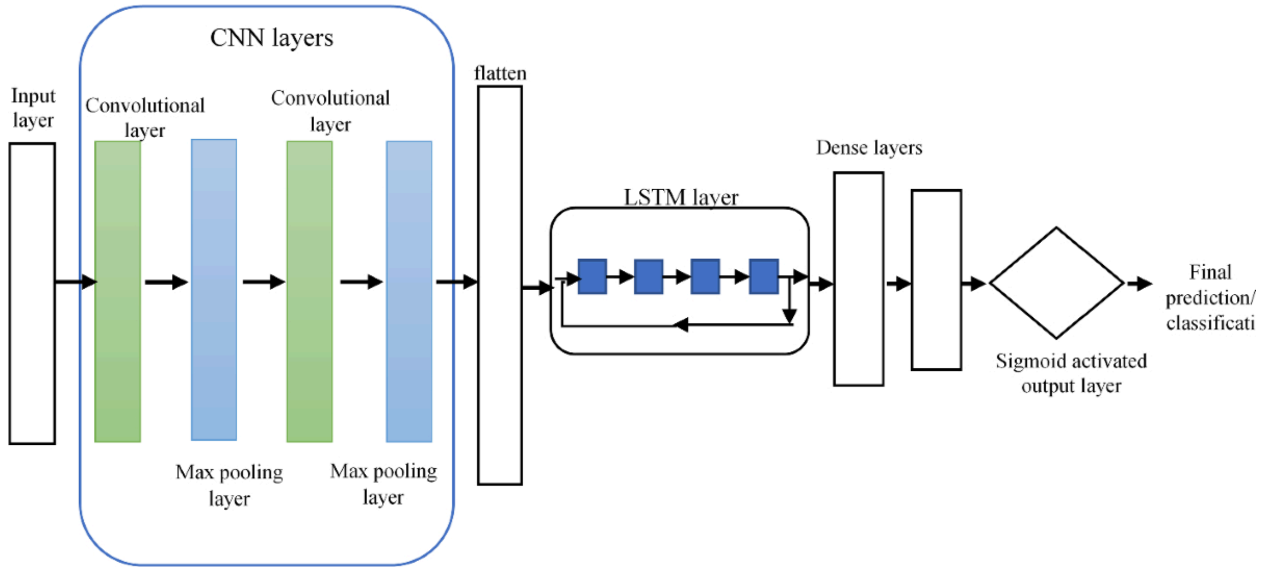
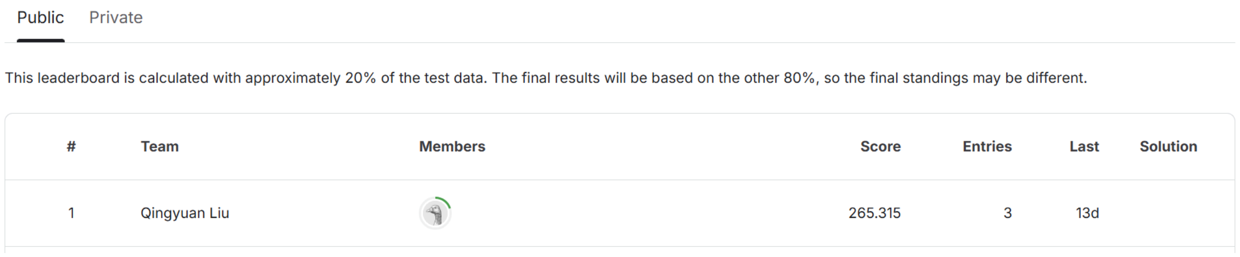


Fig 5: Hybrid model for CNN and LSTM

3. Results

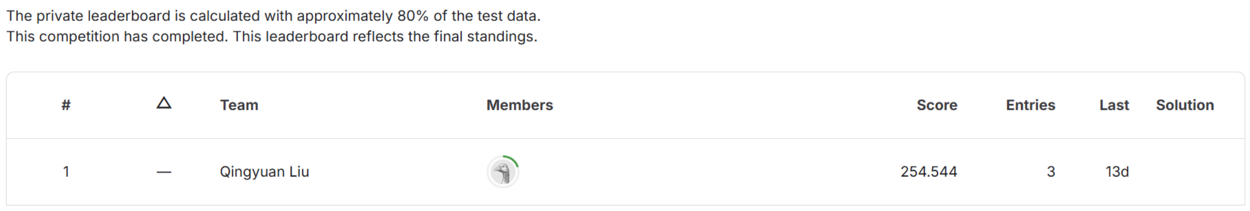
We used mainly PyTorch to build the layers of our models, pandas to process data, matplotlib and other visualization tools to help us draw plots. We used external code only for data visualization and have cited it in the References section and did not use any external code for model building. We have tried training each model for 60, 100, and 500 epochs and adding early stopping. For learning rate, we have tried *1e-4, 1e-3, 5e-4, 5e-3*. For the optimizer, we have tried SGD, Adam, and Adamw. The best results we got locally were 261.5668 for CNN, 258.89 for RNN, 251.76 for LSTM, 264.53 for Transformer, and 265.28 for the hybrid model. We have also submitted our predictions for blueberry yield to Kaggle competition and here are the results we got back. The public leaderboard tests with 20% of the entire dataset while the private leaderboard tests with the rest 80%.



Public Private

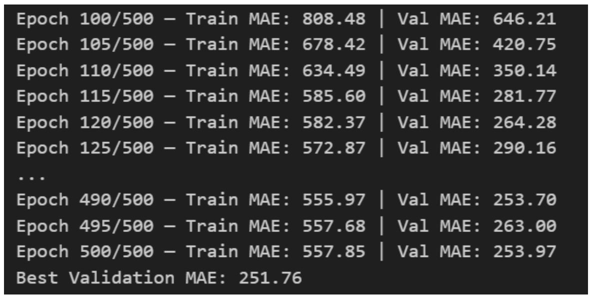
This leaderboard is calculated with approximately 20% of the test data. The final results will be based on the other 80%, so the final standings may be different.

#	Team	Members	Score	Entries	Last	Solution
1	Qingyuan Liu		265.315	3	13d	



The private leaderboard is calculated with approximately 80% of the test data.
This competition has completed. This leaderboard reflects the final standings.

#	△	Team	Members	Score	Entries	Last	Solution
1	—	Qingyuan Liu		254.544	3	13d	



```
Epoch 100/500 — Train MAE: 808.48 | Val MAE: 646.21
Epoch 105/500 — Train MAE: 678.42 | Val MAE: 420.75
Epoch 110/500 — Train MAE: 634.49 | Val MAE: 350.14
Epoch 115/500 — Train MAE: 585.60 | Val MAE: 281.77
Epoch 120/500 — Train MAE: 582.37 | Val MAE: 264.28
Epoch 125/500 — Train MAE: 572.87 | Val MAE: 290.16
...
Epoch 490/500 — Train MAE: 555.97 | Val MAE: 253.70
Epoch 495/500 — Train MAE: 557.68 | Val MAE: 263.00
Epoch 500/500 — Train MAE: 557.85 | Val MAE: 253.97
Best Validation MAE: 251.76
```

Fig 6: Results from 20%, 80% and 100% of test data for LSTM model

4. Discussion and Conclusions

Overall, we observed that the LSTM produced the best result with lowest MAE value equal to 251.76 when we trained that with 80% of the data and validated with 20% of the data. Other models have the following MAE results:

- CNN: 261.5668
- RNN: 258.89
- Transformer: 264.53
- Hybrid: 265.28

The reasons why LSTM can outperform all other models can be explained by the following elements. First of all, LSTM has a “memory” portion to store the long range trait from previous training part so that some important features can be used for the later predictions, which is the major difference from other

models that only depend on the most recent training results and get updates. The second reason is that our training data set is not large – 15000 data with all the features and yield rate. Especially for transformer model, it needs a much larger data set for it to have more accurate results in order to avoid overfitting problems. In addition to that, in terms of the optimizer, LSTM could be the best fit for Adam to just have a slightly decaying learning rate, while RNN will suffer from the learning rate decay and Transformer model needs careful learning rate scheduling. Hence, we conclude that given the 15000 training data set, LSTM will be the best one to produce the predictions for blueberry yield. However, all other training models have MAE under 300, which are considered as the good model by the criterion. In the future, if we are provided with more data and a longer timeline, it will be worth to implement the Transformer and LSTM hybrid model to do the predictions for the blueberry yield – it can both have long-range memory and higher efficiency for training(lower the cost) with higher parallelism.
Agriculture-wise, drawing conclusions from the plots, maximizing fruit set is the most powerful and straightforward way to boost blueberry yield while larger clone sizes can raise the ceiling of potential harvest given the fruit set is sufficiently high. The presence of bees is also another big factor. When it reaches a threshold range, it can markedly improve the yield, but it cannot exceed that range.
While working on this project, we learned that a lot of the real-world problems can be solved using AI and AI is going to become a larger part of our life in the future.

5. Statement of individual contributions

Qingyuan Liu implemented the CNN, RNN and Transformer models. Weiqi Zhang implemented the LSTM and hybrid model and data preprocessing and visualization. Report is written by both of us.

6. References

- https://medium.com/latinxinai/convolutional-neural-network-from-scratch-6b1c856e1c07
- https://medium.com/@thisislong/building-a-recurrent-neural-network-from-scratch-ba9b27a42856
- PyTorch documentation — PyTorch 2.6 documentation
- https://www.kaggle.com/competitions/deep-blueberry-challenge-2025/overview
- https://medium.com/towards-data-science/build-your-own-transformer-from-scratch-using-pytorch-84c850470dcb
- Hybrid reference paper: https://link.springer.com/article/10.1007/s11063-024-11687-w
- The Attention Mechanism from Scratch - MachineLearningMastery.com
- NumPy reference — NumPy v2.2 Manual
- API reference — pandas 2.2.3 documentation
- API Reference — Matplotlib 3.10.1 documentation
- External code: Starter DeepBlueberry Challenge 2025