Model Architecture and Training

The model we used is a K-Nearest Neighbors (KNN) classifier, with k=1, implemented using PyTorch tensor operations on GPU and Batching for memory efficiency to optimize the overall running time.

The key architectural component of KNN is the fact that it is a non-parametric, instance-based method, so contrary to other models like Neural Networks, KNN doesn’t learn explicit model weights. Instead, predictions are made by comparing test samples directly with stored training samples.

Indeed, the model uses proximity to make classification or prediction about the grouping of an individual point. When running the model:

1. KNN calculates the Euclidean distance between all the data points and the X_test points with this optimized formula $\| x - y \|^2 = \| x \|^2 + \| y \|^2 - 2 * x * y$ that avoids explicit pairwise subtraction and leverages fast matrix multiplication on GPU.
2. KNN finds and retrieves the closest neighbor from the distance matrix with PyTorch's torch.topk.
3. KNN assigns the label of the majority vote among the k datapoints to determine the final prediction. In our case k=1, so the model just assigns the label of the nearest neighboring point.

Training methodology

KNN is non-trainable, so there is no gradient-based training, no epochs, and no learning rate. The "training" phase consists solely of:

- Loading and storing the full training dataset in GPU memory.
- Preprocessing features and labels into Torch tensors.

Because KNN makes predictions directly from stored instances, there is no loss function, no backpropagation, and no weight updates.

Optimization

The only hyperparameter which can be optimized in KNN is the k and in our case, we've run several types of KNNs: KNNs with normalization or without + regular KNN or weighted KNN + tested a lot of Ks for each type of KNN to find the best model possible.

We'd run a model and then try different k's until we start seeing the accuracy decreasing, and then start a new model (regular, normalized or weighted)

Ks	Regular KNN	KNN with data normalized	Weighted KNN
1	0.868	0.784	0.868
2	0.811	0.740	0.868
3	0.811	0.712	0.858

4	0.808	X	X
5	0.804	0.670	0.844
6	0.793	0.651	X

We also had run different KNN models in parallel and then did a majority vote between them to get the label (Ensemble Learning), but its efficiency had not been proven.

The k's chosen for Ensemble Learning from weighted KNN models	**Accuracy**
k=1,3,5,7,11	0.851
k=1,3,7	0.856
k=1,3,5	0.861
k=1,3,4	0.861
k=1,2,3	0.868
k=1,2	0.868

As we can see the maximum accuracy for the KNN model is 0.868. Thus the model we selected as our best model is regular KNN with k=1, since the efficiency of the other methods don't exceed the one done by regular KNN, so no need to complexify our model for no reason.

Evaluation

Metrics

To evaluate our model’s performance, we used **classification accuracy** as the primary metric. Accuracy measures the proportion of correctly predicted labels out of the total predictions, making it a straightforward and reliable indicator of overall model performance for a balanced multi-class problem such as ours (50 classes, each well-represented). Because KNN directly assigns the label of the nearest training sample (k = 1), accuracy allows us to clearly quantify how often this nearest-neighbor rule leads to a correct classification when tested on our validation splits. Alternative metrics such as precision, recall, or F1-score can provide class-specific insights, but our comparative experiments across models (Decision Trees, Random Forests, MLP, Normalized KNN, Weighted KNN, and Ensembles) were centered on finding the configuration that produced the **highest global accuracy**, since this aligns directly with the competition’s evaluation criteria. Using accuracy allowed us to consistently compare different values of k, different preprocessing choices, and different model families, ultimately leading us to identify **KNN with k = 1** as the strongest-performing model with an accuracy of 0.868.

A comparative look at model performance

In this project, we use the KNN model with the simplest possible configuration: the number of neighbors (k = 1), that is, the model looks only at the nearest sample in the training data. With this simple method, we get a total accuracy of approximately 86.8%, which is a good amount.

The main advantage of this model was a high recall value, that is, it was sensitive in detecting the correct samples for each category, and therefore, the number of false negative errors was small; it rarely fails to detect a random sample of a specific category.

In contrast, Precision was slightly less. This means that the model was more enthusiastic than necessary in the classification of the samples: when the boundaries between the categories become unclear, and the samples from different categories are close in the feature space, the model begins to commit false positive errors, i.e., it gives the false category for ambiguous samples.

We also tried other versions such as Weighted KNN and NN for features, but the performance dropped slightly. This gives us a clear indication that the local raw similarity between the samples is the strongest element in our statements; Any modification of this similarity, except by increasing the resolution (k) or changing the scale of all features through naturalization, will falsify the model will lose some of the small and useful differences originally found in the data.

Error analysis:

Most of the errors to be noted appeared in cases where the distribution of features between nested categories; That is, when they look similar or more similar in the space of features.
Since we do not use Euclidean distance, if samples from two different groups are very close to each other, even a simple change or noise in the data is enough to change the prediction from one category to another.

The most important reasons to note for errors:

High Dimensionality:

Points from different categories can converge to approximately the same degree. This makes separation between classes much more difficult, a well-known problem in KNN in high-dimensional spaces.

Feature Dominance:

When we did the normalization of the features, the performance was slightly worse. This indicates that some features in their original form were more distinguishable than others, and it was better to be more effective in calculating distance. But naturalization is the same for all and the weakest of this beneficial effect.

No Learned Representations:

Unlike deep models like CNN, the KNN model does not learn new features; It only depends on the distances between the raw data. Therefore, if the actual boundaries between classes are complex and non-linear, KNN will not be able to provide a better representation or find an intelligent crop between them.

To conclude, when the categories are very overlapping in the original feature space, KNN does not perform well, but we do not observe a lot of changes in predictions in these areas full of noise and interference.

Strong categories:

There were categories in which the model achieved outstanding performance, almost perfect recall and precision. These powerful categories had the following characteristics:

Separation is clear from other categories:

The samples of these categories formed clear and compact clusters in the feature space, and were relatively far from the clusters of other categories. This situation is ideal for the KNN model.

Example of training statements:

There was a sufficient number of samples in the training for each of these categories, so that any sample in the test can easily be found in nearly the same category to correctly predict.

Low Variability:

The samples belonging to the same category were very similar to each other, and not scattered or widely separated. This helps completely to make the Euclidean distance a reliable indicator of distance. And since KNN is completely memory-based, the existence of ordered and clear training points like this is ideal. Obtaining outstanding performance in these specific categories, almost without error.

In conclusion, the KNN model with (k = 1) was a strong choice for entities that form clear and compact concepts in the feature space. Our real problems appeared only in the areas where noise is high, and interference between groups is high; There, KNN loses its ability to make decisive decisions without mistakes.

Depth effects

Our chosen model, K-Nearest Neighbors (KNN), is a non-parametric, instance-based algorithm that does not have depth as a hyperparameter. Unlike Decision Trees, MLPs, or CNNs that build hierarchical representations through layers or tree depth, KNN makes predictions by directly comparing test samples to stored training instances using distance metrics. Therefore, depth effects are not applicable to our final model architecture.

Size & kernels

Layer size and kernel size variations are hyperparameters specific to neural network architectures such as MLPs and CNNs. Since our selected model is KNN with k=1, which operates purely on distance-based nearest neighbor lookup without any learned layers or convolutional operations, these architectural considerations do not apply. The only tunable parameter in KNN is k (number of neighbors), which we optimized through systematic testing.

Conclusion

Throughout this project, we evaluated a wide range of models, including Decision Trees, Random Forests, several variants of K-Nearest Neighbors (KNN), and a custom-built Multilayer Perceptron (MLP). Across all experiments, the model that achieved the highest performance was the **regular KNN classifier with k = 1**, which reached a validation accuracy of **0.868**, making it the strongest and most reliable approach for this dataset.

The other models we tested demonstrated noticeably weaker performance. Decision Trees performed the worst, achieving only **0.121 accuracy**, primarily due to their tendency to severely overfit in high-dimensional feature spaces such as ours, which contains 500 continuous features. Random Forests improved substantially over single trees and reached a maximum accuracy of **0.784**, but still fell short compared to KNN. Our custom MLP, trained with multiple hidden layers, dropout, and weight decay, reached **0.656 accuracy**, indicating that the learned parametric representation was not as effective as instance-based similarity for this particular data distribution.

In contrast, KNN performed exceptionally well, especially with **k = 1**, because the dataset exhibits strong local structure: many classes form tight, compact clusters in the feature space, making raw nearest-neighbor distance highly informative. When we increased k, applied weighted distances, normalized features, or attempted ensemble strategies, accuracy consistently decreased, suggesting that additional smoothing blurred the subtle but crucial differences between samples. Similarly, deep models like MLP were unable to learn more useful representations than the natural structure already present in the raw data.

Based on all experimental results, **KNN with k = 1** proved to be the most effective and stable model for this classification task. Its ability to leverage fine-grained local similarity allowed it to outperform all more complex alternatives, and therefore it was selected as our final model for submission.