Application of Neural Networks to a Geophysical Well-Log Regression and Pest Image Classification
FYS-STK4155 - Project 3

Erik Berthelsen, Morten Taraldsten Brunes
Department of Geoscience, University of Oslo

Insects and pests significantly impact agriculture through crop damage and economic losses, making accurate species identification critical for effective control strategies. Emerging technologies such as drone-based monitoring, automated crop protection, and early outbreak prediction rely on robust image-based classification. In this study, we implemented insect and pest classification using PyTorch and convolutional neural networks through three approaches: a custom model, hyperparameter optimization with Optuna, and transfer learning. Transfer learning achieved the best performance due to its pretrained feature extraction capabilities and the limited size of our dataset, resulting in an overall accuracy of 86% and class-level precision between 72% and 100%. These results demonstrate the potential of image classification for pest management, though further improvements are needed to enhance accuracy, particularly for classes with lower precision. In addition, we analyzed a geophysical well-log data from the North Sea exploration well 1/9-7 and applied an FFNN to predict the sonic log from available petro-physical measurements. The neural network outperformed the Ridge baseline, and hyperparameter optimization with Optuna further improved the predictive performance. Our work is limited to one well-log, and further work should include more data from multiple well locations to create a more generalizing model for new geological well log formations.

I. INTRODUCTION

The development of machine learning algorithms, the availability of flexible and powerful libraries such as PyTorch [11], and improved access to computational resources have created new opportunities for data-rich domains to analyze large datasets and extract valuable insights.

Drilling and analysis of well logs are a fundamental part of hydrocarbon exploration and carbon storage. While seismic data can only give an overview of what lies beneath the sea floor, high-resolution well data are regarded as the ground truth of the geology, helping to identify porous, sandstone-rich formations that act as reservoir rocks for CO_2 storage and hydrocarbons. Since well logs are often incomplete due to NaN values in the data, there is a need for reconstructing missing values. To research how to enhance the subsurface characterization, we analyze well 1/9-7 [4] and train a feed-forward neural network to predict the missing log values.

Insects and pests cause significant crop damage and economic losses. Their identification through image classification using convolutional neural networks (CNNs) provides a foundation for solutions such as drone-based monitoring, automated crop protection systems, and early pest outbreak prediction. We utilized the AgroPest-12 [10] dataset and applied CNNs through three different approaches: a custom model, hyperparameter optimization with Optuna [2], and transfer learning from a pretrained model. This study research how accurately insects and pests can be classified in the AgroPest-12 dataset with different methods and outline directions for future work.

The methods applied in this report, covering both well-log analysis and insect classification, are described in section II. Results and insights for these two parts are presented in sections III A and III B. Finally, conclusions integrating findings from both domains are summarized in Section IV.

II. METHODS

A. Analysis of well-log with neural network

The dataset is based on the geophysical data from well 1/9-7 [4]. The dataset was obtained with help from the Institute of Geoscience at the University of Oslo, and PyTorch was used for the regression analysis. The regression was carried out by creating a feed-forward neural network (FFNN) model and optimizing its parameters with Optuna [2]. In this analysis, the goal is to use the available well-log measurements to predict the sonic log (AC), which is an important indicator of lithology and porosity variations in the subsurface.

The well was drilled in the North Sea by Phillips at the edge of the Feda Graben, about 1.5 km from the UK border [4]. The purpose of the drilling was exploration for hydrocarbons, and the well-log measurements provide continuous in-situ information about rock properties such as density, porosity, mineralogy and fluid content. The dataset consists of 18 105 data points recorded between 281.1 and 3 040 meters depth. The logs include depth, sonic transit time (slowness), nominal bit size, caliper, bulk density, gamma ray, potassium, neutron porosity, photo-electric factor, deep resistivity, medium resistivity and spectral gamma ray. These are typical petrophysical parameters used to infer lithology and the presence of hydrocarbons in porous sandstones.

The raw LAS file was imported into the processing script using the lasio library, which provides the numerical log values indexed by depth. The dataset contained several null markers (-999.25, -999 and -9999.25), which were replaced with NaN values and removed before further processing. Depth was kept as a reference but removed as an input variable to the model, since it is a strictly increasing index that does not represent a physical predictor and would risk introducing data leakage. After cleaning the dataset, 11 log channels remained as usable input features.

All features and the target variable were standardized using only the training data statistics to ensure stable gradient-based optimization during training. Instead of a depth split, the dataset was randomly split into 70 % training, 15 % validation and 15 % test. This was done since the rock-formations varies with depth (time), so to simplify, we instead focused on only this well instead of a general well data predictor. The validation set was used for hyperparameter optimization, while the test set was kept completely separate for the final evaluation of the model. A depth-based split was also tested and is discussed in the section III A, but was not used as the primary evaluation due to its instability and reduced performance for this specific single-well prediction task.

In this report, the sonic transit time (AC) is used as the target variable. It reflects how fast a compressional wave travels through the rock, which depends on lithology, porosity and mechanical strength. Rocks with high porosity generally show slower transit times, while compact, well-cemented rocks show higher velocities. In a hydrocarbon exploration context, this helps identify potential reservoir zones, as porous and fluid-filled formations typically have lower acoustic velocities. The FFNN model was designed to learn these relationships from the available log features and produce continuous predictions of the AC curve along the borehole.

1. FFNN

Feed Forward Neural Networks, hereby refered to as FFNN in this report, are one of the most fundamental artificial neural networks, and it forms the basis of many more advanced NNs. The FFNN takes the input vector and processes in layers of neurons, and produces a output. The term "feed forward" refers to the fact that each layer serves as the input to another layer without loops, which distinguish FFNNs from Recurrent or Convolutional Networks [14].

Each layer consist of a set of neurons, where each one does the same thing. It takes a input, adds a bias term and passes the results through a activation function. The activation function is essential for the models ability to learn complex data, as it introduces non-linearity to the model. In this project, ReLu activation function will be used because of its efficiency, fast and stable learning.

For the regression task, we designed a FFNN with 11 input features, removing the depth feature since it is strictly increasing and does not carry physical predictive information. The model architecture implemented in our FFNN model script consists of two fully connected hidden layers with 64 neurons each, followed by ReLU activation and dropout regularization. Dropout was included with a probability of 0.2 to reduce overfitting by randomly deactivating neurons during training. The output layer contains a single neuron, representing the predicted sonic log (AC) value.

The architecture can be summarized as follows:

Algorithm 1 FFNN Architecture for Well-Log Regression

```
**Input:** Feature vector $x \in \mathbb{R}^{11}$
**Output:** Predicted sonic value $\hat{y} \in \mathbb{R}$
**Layer 1:** Linear$(11 \to 64) \to$ ReLU $\to$ Dropout$(p = 0.2)$
**Layer 2:** Linear$(64 \to 64) \to$ ReLU $\to$ Dropout$(p = 0.2)$
**Output Layer:** Linear$(64 \to 1)$
**Return:** $\hat{y}$
```

This network architecture was implemented using a nn.sequential container that assembles all layers into a clean and modular structure. The FFNN was trained using the Adam optimizer and mean squared error (MSE) loss function 1, which is standard for continuous regression problems. The model receives mini-batches of training data through PyTorch's DataLoader, and the weights are iteratively updated through backpropagation.

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2, \tag{1}$$

[5]

Overall, the FFNN serves as a flexible function approximator for learning the relationship between the petrophysical logs and the sonic transit time. By combining non-linear activation functions, dropout regularization, and standardized input features, the network is capable of capturing the underlying physical trends present in the well-log data.

2. Hyperparameter tuning

The hyperparameters for the FFNN were evaluated with the help of Optuna Trial from the Optuna library [2]. The point of the hyperparameter tuning is to search for combinations that minimizes the validation error, instead of manually assume layer size and learning rate. Where the Optuna tested different architectures for layers, nodes, learning rate and dropout and chose the one that produced the best MSE score. These parameters where selected since they control the model capacity, regularization strength and the optimizer behavior.

B. Classification of insects and pests

We used the AgroPest-12: Image Dataset for Crop Pest Detection [10] available at Kaggle and the PyTorch library [11] for classification analysis. To identify the most effective approach for classification, we first design a custom CNN architecture, then optimize hyperparameters using Optuna [2], and finally apply transfer learning by initializing the model with pretrained weights from ResNet-18 [6].

The AgroPest-12 dataset consists of 12 classes of crop insects and pests. The dataset is organized into three subsets: training, validation, and test and contains 11 502, 1 095, and 546 images. The train subset available at Kaggle contains 3 834 original images, each image have been augmented by rotation and brightness adjustments during dataset preparation, resulting in a total of 11 502 training images. The classes and the number of images in the training dataset are as follows: ants – 1 029; bees – 1 101; beetles – 857; caterpillars – 906; earthworms – 717; earwigs – 939; grasshoppers – 1 044; moths – 1 059; slugs – 797; wasps – 1 011; weevils – 960. Every image is accompanied by a label file that specifies the class and the bounding box coordinates of the object(s). Notably, a single image may contain multiple objects. A sample image from one of the classes in the dataset is shown in figure 1.

Figure 1: Snail as a sample of one class from the AgroPest-12 dataset.

1. *Fundamentals of Convolutional Neural Networks*

Convolution is a core operation in CNNs. In this paper, we use 2D convolution as illustrated in figure 2, with a 3×3 kernel applied to an 8×8 input image using padding of 1 to preserve dimensions. Kernels define local receptive fields and start with random weights that are updated during training to learn useful features. Padding ensures edge regions are processed, while stride controls how far the kernel moves, affecting output size and computational cost. Together, kernel size, padding, and stride determine the dimensions of the resulting feature map [15]. The first layers of convolution capture low-level features like edges, corners and simple features, while deeper layers learn more complex and abstract patterns such as object parts. By stacking multiple convolutional layers, the model progressively extracts different types of features [3].

Additional components include pooling for spatial downsampling and local invariance, dropout to reduce overfitting, batch normalization to stabilize and accelerate training, and adaptive average pooling to produce fixed-size outputs and reduce parameters before classification. The network is trained using cross-entropy loss, the standard for multi-class classification. Detailed CNN theory is outside the scope of this paper; see [15] for further reading.

We have used accuracy, per class precision and confusion matrix as performance metrics. The accuracy metric is given by

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

and gives one metric for the whole dataset.

Precision is calculated for each class and is given by

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

where TP and FP represent the number of true positives and false positives.

The confusion matrix is a NxN matrix and is constructed such that $C_{i,j}$ is equal to the number of observations known to be in class i but predicted to be in class j. Thus, the row indices of the confusion matrix correspond to the true class labels, and the column indices correspond to the predicted class labels [9].

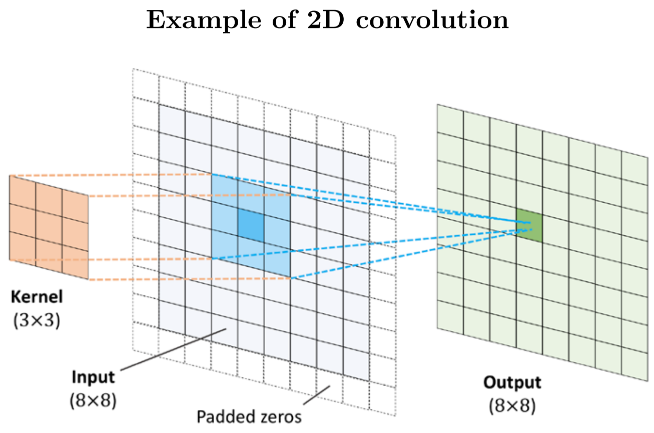


Figure 2: Example of 2D convolution with padding 1 and stride 2x2 [15].

2. Designing our own custom architecture of CNN

Our custom model architecture follows the principles outlined in section II B 1 and the steps shown in algorithm 2. After four convolutional blocks with batch normalization, ReLU activation, and max pooling, an adaptive average pooling layer reduces the feature maps to size [batch × 256 × 1 × 1]. Two fully connected layers follow: the first maps 256 features to 12 neurons with ReLU and dropout, and the second outputs 12 neurons for classification using cross-entropy loss.

Algorithm 2 CNN Architecture

```
**Input:** Image $I$ of size $224 \times 224$
**Output:** Class label $\hat{y} \in \{1, \ldots, 12\}$
**Layer 1:** Conv2D($3\times3$, 32 filters) $\rightarrow$ BatchNorm $\rightarrow$ ReLU

MaxPool($2 \times 2$)
**Layer 2:** Conv2D($3\times3$, 64 filters) $\rightarrow$ BatchNorm $\rightarrow$ ReLU

MaxPool($2 \times 2$)
**Layer 3:** Conv2D($3 \times 3$, 128 filters) $\rightarrow$ BatchNorm $\rightarrow$ ReLU
MaxPool($2 \times 2$)
**Layer 4:** Conv2D($3 \times 3$, 256 filters) $\rightarrow$ BatchNorm $\rightarrow$ ReLU
MaxPool($2 \times 2$)
AdaptiveAvgPool($1 \times 1$)
**Fully Connected:** Linear($256 \rightarrow 12$) $\rightarrow$ ReLU $\rightarrow$ Dropout($p = 0.3$)
Linear($12 \rightarrow 12$)
**Return:** $\hat{y}$
```

3. Hyperparameter evaluation with Optuna

Optuna is an open-source framework for hyperparameter optimization that automates the search process [2]. It enables users to define a flexible search space within an objective function, covering a wide range of model configurations. For instance, the learning rate can be sampled from a continuous range such as 1×10^{-5}, 1×10^{-2}, while discrete values can be specified for parameters like dropout rate (e.g., [0.3, 0.5] in a fully connected layer). Similarly, other hyperparameters—such as the number of convolutional layers or the number of hidden units—can be included in the search space. Optuna employs sampling strategies to efficiently explore and narrow down this space, improving the likelihood of finding optimal configurations.

Pruning is one of the benefits of using Optuna. It allows trials that are unlikely to outperform previous ones to be stopped early based on a performance metric. We used the median pruner: it compares the intermediate trail results with the median of the best completed trials at the same step. To ensure reliable comparisons, pruning begins only after a specified number of startup trials have finished, providing enough data for the metric. This approach saves computational time and makes Optuna highly efficient for large hyperparameter search spaces.

The standard sampler in Optuna is TPE, Tree-structured Parzen estimator, which is used in our experiment. This sampler evaluates the distribution of hyperparameters that lead to good and poor performance and chooses new hyperparameters that are more likely to perform well than others [16]. Hyperparameter importance is estimated using the fANOVA (functional ANOVA) evaluator [8]. The fANOVA algorithm decomposes the variance of the objective function into contributions from individual hyperparameters and their interactions. The importance of a hyperparameter is defined by the expected change in the objective when varying that hyperparameter across its domain, marginalizing over all other hyperparameters. In our study, the objective function is the validation accuracy achieved by the model given a specific set of hyperparameters.

In our setup for image classification, the Optuna objective was executed for 50 trials and the image size was reduced to 128x128 to reduce computational time. Each trial evaluates a unique combination of hyperparameters by training the model and computing its accuracy on the dataset. After all trials are completed, Optuna selects the configuration that achieved the highest accuracy to use in the final training on the full dataset. For the final training image size of 224x224 was used and match settings for analysis with custom implementation and pretrained.

4. Classification with transfer learning

The PyTorch library provides a variety of models for various tasks. For our work with the AgroPest-12 dataset we selected a classification model. These models are distributed with pretrained weights that enable transfer learning. In our approach, the convolutional feature extractor was kept frozen (weights unchanged), while the fully connected layers responsible for classification were trained and updated on the AgroPest-12 dataset.

When using pretrained models, it is essential to preprocess the image dataset using the same settings as the original training. We used ResNet-18 with ImageNet-1K weights [6], pretrained on the ImageNet dataset [1]. The model and weight names indicate that the network consists of 18 layers and was originally trained to classify 1000 classes. The rationale for using ResNet in transfer learning is that it has learned general visual features, making it an excellent starting point for fine-tuning on smaller datasets. Image size in use are 224x224.

The ResNet-18 architecture begins with a 7×7 convolution, followed by batch normalization and max pooling. Then it includes residual blocks with 3×3 convolutions and progressively increasing filter sizes (64, 128, 256, and 512), each with batch normalization. The network concludes with global average pooling and a fully connected layer mapping 512 features to 1000 output classes. The input images are randomly cropped and resized to 224×224 pixels. Normalization follows the standard ResNet-18 parameters: mean values of [0.485, 0.456, 0.406] and standard deviations of [0.229, 0.224, 0.225] for the red, green and blue channels [13].

C. Use of AI tools

In this project we have used Microsoft Copilot in the following ways in the production of code and the report:

- Improved some sentences in the report.
- Questions about the Python syntax and coding.
- Creating docstrings in python code.
- Specific questions about python libraries documented at project GitHub

III. RESULTS & DISCUSSION

A. Analysis of well-log

The well log regression analysis is based on real-world data from the North Sea, leading to a more demanding data preprocessing compared to textbook examples for machine learning. This challenge was solved with prior geology knowledge, careful splitting and critical assessments of the model along the way.

1. Preprocessing

The preprocessing proved to be particularly challenging due to the non-linearity and stratigraphic structure of the well-log data. As described in Section II, both a random 70%, 15%, and 15% split and a depth-based split were considered and implemented.

A well log is stratigraphically organized rather than random, meaning that it contains rock formations deposited in time, that can resemble each other. And in a sedimentary basin such as the North Sea, lithologies like sandstones, shales, and limestones occur in repeated depth intervals. A depth-based split therefore often results in the validation or test set being dominated by a limited set of lithologies, while the training set contains others. In addition, petrophysical properties such as porosity vary systematically with depth due to increasing pressure, leading to significant differences between shallow and deep formations [7]. In practice, this made the depth-based split highly unstable and resulted in significantly poorer model performance.

A random split, on the other hand, introduces a risk of data leakage, as similar lithologies may appear in both the training and evaluation sets. However, given that the objective of this study is to predict missing values within a single well rather than to generalize to unseen wells, the random split was found to be the most practical and robust choice for this task.

2. FFNN model

The training and validation loss curves for the FFNN model are shown in figure 3. Both curves exhibit a steep decrease during the first few epochs, indicating effective learning of the relationship between the input well logs and the target sonic log.

After the initial learning phase, the loss curves gradually flatten and converge towards stable values. The validation loss remains consistently close to, and slightly below, the training loss throughout the training process. This behavior suggests good generalization and indicates that the model does not suffer from significant overfitting. The smooth convergence of both curves further implies that the chosen network architecture is sufficiently expressive to capture the underlying non-linear relationships in the data, while remaining stable during training.

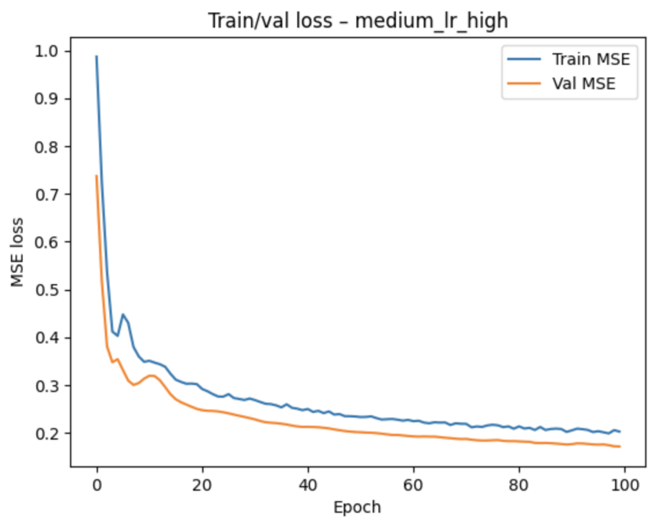


Figure 3: Train and validating of the FFNN model

3. *FFNN vs Ridge*

To evaluate the performance of the FFNN model, its predictions are compared to a linear ridge regression baseline. Figure 5 shows scatter plots of predicted versus true sonic values on the test set for the FFNN model figure 5a and the ridge baseline figure 5b.

The ridge regression model captures the overall linear trend in the data but exhibits substantial scatter around the 1:1 line, particularly for extreme sonic values. This indicates limited ability to model complex relationships between the input logs and the target variable. .

In contrast, the FFNN predictions are more tightly clustered around the 1:1 line across the full range of sonic values. The neural network shows improved performance for both low and high sonic values, suggesting that it is better able to capture non-linear dependencies present in the well-log data.

The Ridge model performed a $R^2 - Score$ of 0.68 and an MSE of 0.32, compared to the FFNNs $R^2 - Score$ of 0.84-0.85 for different seeds and 0.13 MSE. Further establishing the superiority of the FFNN model (Figure 4).

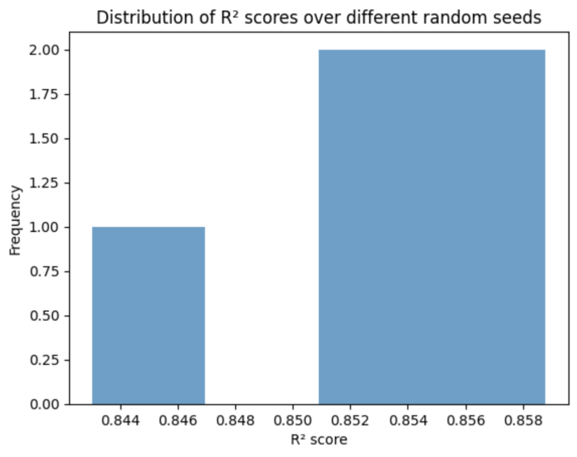
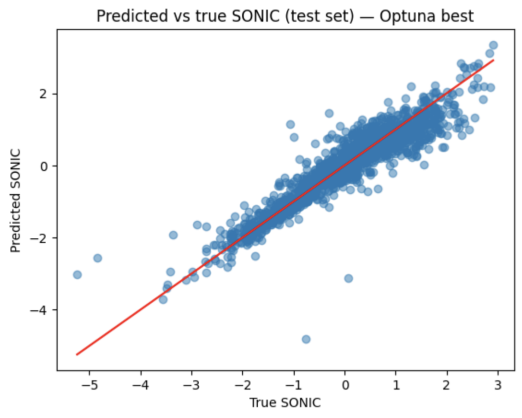
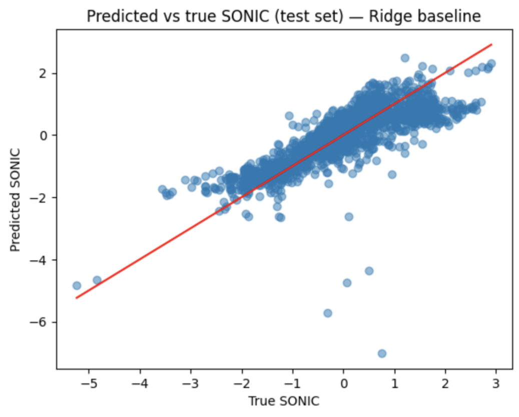


Figure 4: Plot of R^2 scores for 5 different seeds

This comparison highlights the advantage of using a non-linear model for predicting the sonic log, as the relationship between petrophysical measurements and acoustic response is inherently non-linear.



(a) Predicted sonic, using hidden_dims $= (208, 80, 160), dropout = 0.0, lr = 0.007320703417942379$

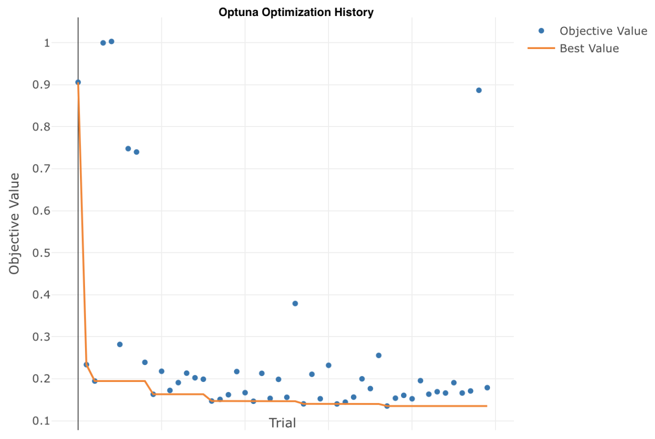


(b) Ridge baseline

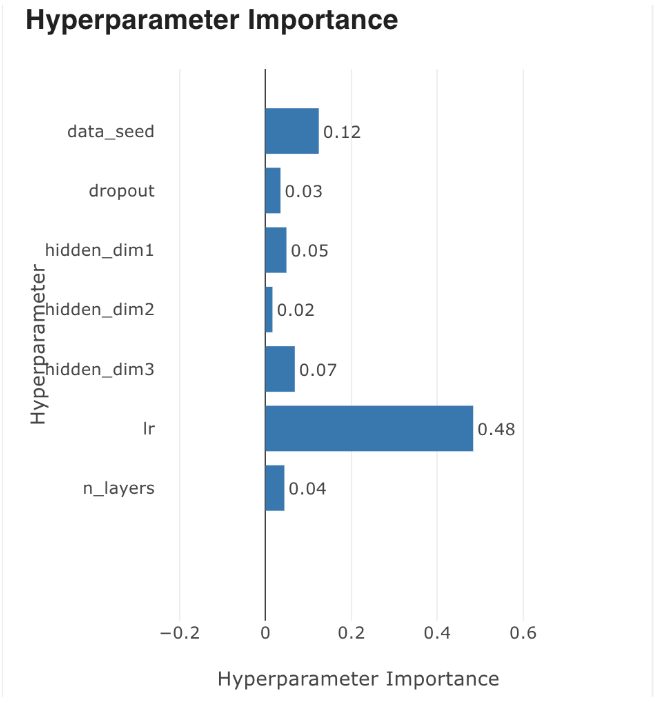
Figure 5: Scatterplot of Predicted Sonic vs Ridge baseline

4. *Optuna hyperparameter search*

Hyperparameter optimization of the FFNN regression model was performed using Optuna, with validation mean squared error (MSE) as the objective function. The optimization history is shown in figure 6a. The results demonstrate a rapid reduction in validation error during the initial trials, followed by a gradual convergence towards a stable minimum. After approximately 15–25 trials, further improvements become marginal, indicating that the sampler has identified a near-optimal region in the hyperparameter space.



(a) Graph of best values, gotten from Optuna Dashboard



(b) Hyperparameter importance

Figure 6: Results from the Optuna tuning, from Optuna dashboard

The hyperparameter importance from Optuna (Fig. 6b) shows that the learning rate (0.48) has the largest effect on the optimization, highlighting how sensitive the training process is to the optimizer settings.

We also see that the data_seed has a noticeable effect on the optimization (Fig. 6b), illustrating that the way the data is split influences the model performance. With a random split, samples from similar depths and lithologies can end up in both the training and validation sets, meaning that different random seeds can lead to slightly different performance.

Interestingly, the data_seed has no effect when using a depth-based split. This is because the geological data are depth- and lithology-based, and the split is fully deterministic once the depth intervals are defined. As a result, the same lithological units are always grouped together, and the model performance no longer depends on random initialization of the data split. This makes the evaluation more stable and geologically consistent, but also significantly more challenging, as the model is forced to generalize to unseen depth intervals rather than interpolating between similar samples.

B. Classification of insects and pests

We used three different approaches for the classification of the AgroPest-12 dataset using CNN. Defining our own custom model architecture, using Optuna for hyperparameter search to determine our own model, and finally using a pretrained model with weights and using transfer learning to our dataset.

1. *Preprocessing and common steps for all CNN approaches*

The AgroPest-12 dataset provides bounding boxes for objects, so images were cropped to focus on the insect before training. The mean cropped size was 395×405 pixels, and all cropped images were resized to 254×254 pixels for consistency and computational efficiency. Preprocessing included rescaling pixel values to [0,1] and per-channel normalization using dataset mean and standard deviation, ensuring stable gradient flow and effective learning—an approach also used in ResNet-18. Models were trained with the Adam optimizer and cross-entropy loss for multi-class classification [15]. To improve convergence, a learning rate scheduler (ReduceLROnPlateau) reduced the learning rate when accuracy stagnated [12].

We implemented a stopping criterion in the training loop. If the validation accuracy does not improve for a specified number of consecutive epochs, the training is stopped early. In our implementation, this patience parameter is set to 15 epochs. This avoids unnecessary processing time and prevents overfitting. The training also evaluates improvements on accuracy score in validation dataset and chooses model weights from the epoch with best validation accuracy. Hence, the final model represents the best observed validation performance, which gives a better generalization than using weights from the last epoch.

2. *Custom architecture*

Our custom model architecture is described in pseudocode 2 and the training loop stopped after 82 epochs. In figure 7 it is quite noticeable that the validation accuracy is unstable and fluctuates, while the training accuracy is smooth and gradually improving. In addition, the validation accuracy is better than the training accuracy at the beginning of training, while it is the opposite in the end.

Validation accuracy is based on inference, and early in training this can exceed training accuracy since the model is evaluated without regularization and benefits from generalization. When training accuracy becomes better than validation accuracy, it is a classic sign of overfitting. Training accuracy increases smoothly since epochs are optimized to minimize the cost function and averaged over batches.

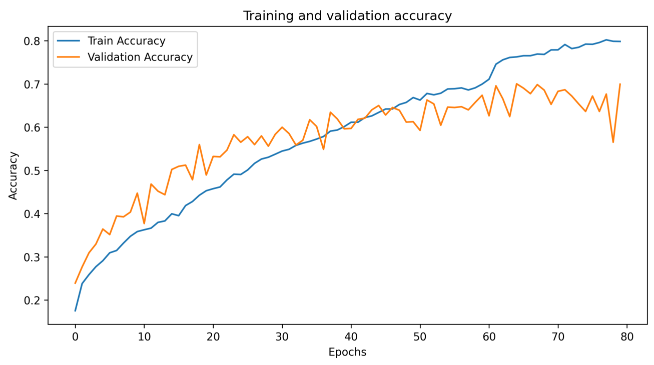


Figure 7: Training and validation accuracy for custom model architecture. Training stopped at 82 epochs.

The accuracy for all classes is 69%, and the precision values for each class range from 42% to 97%, indicating large variability in class level performance. This implies that some classes are inherently more difficult to distinguish than others.

The confusion matrix in figure 8 provides a detailed view of these patterns, showing which classes are consistently classified correctly and which are prone to misclassifications. For example, class 10 (wasps) achieves 38 correct classifications but is misclassified as class 6 (grasshoppers) in 6 cases and only once as classes 0 and 1. In contrast, class 0 (ants) exhibits widespread confusion, with misclassifications spread across eight different classes. Such patterns highlight that while certain classes (e.g., wasps) are relatively well-separated, others (e.g., ants) share visual similarities with multiple categories, making them more challenging for the model to distinguish.

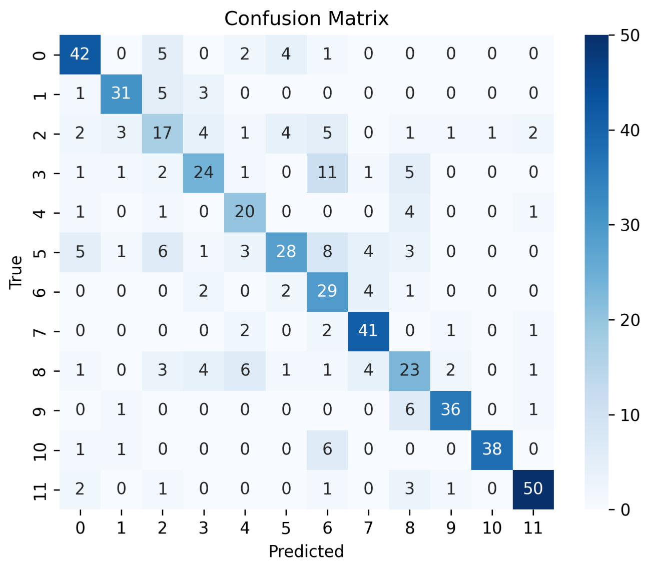
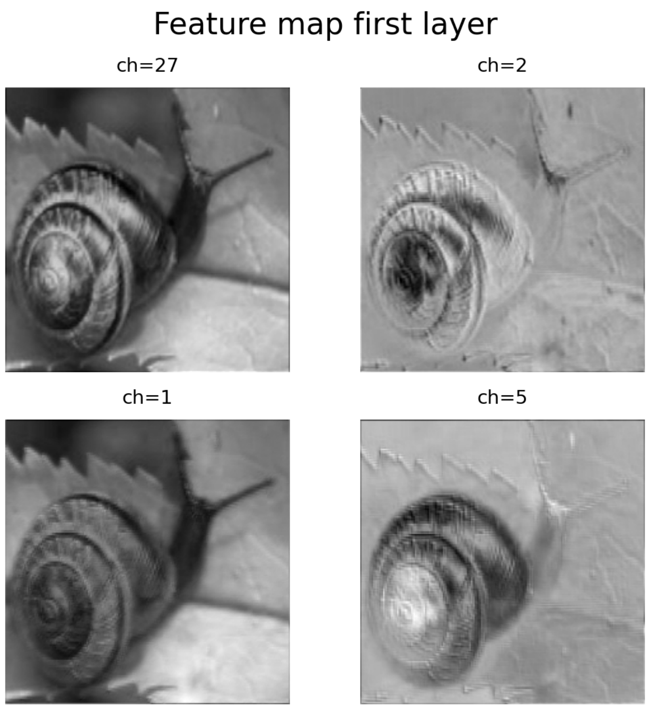


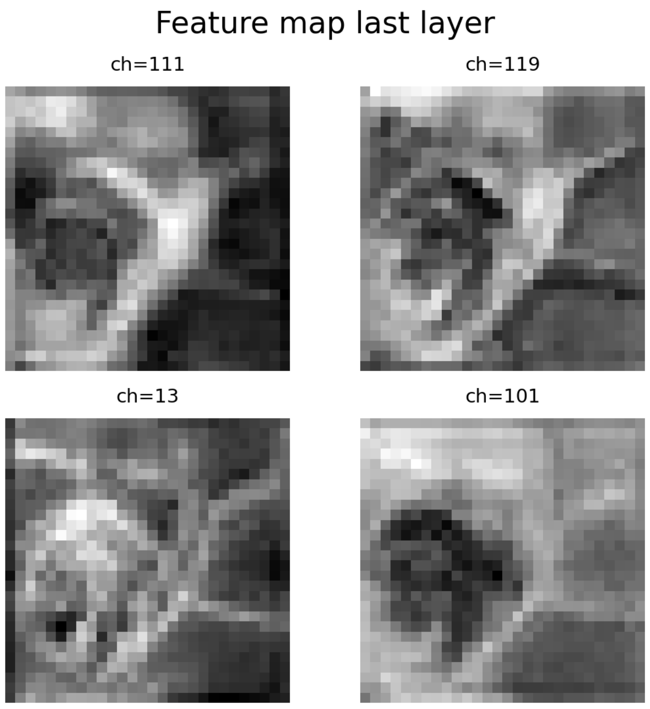
Figure 8: Confusion matrix for custom model architecture.

To visualize the convolution, four randomly selected feature maps are visualized from the first convolutional layer (32 channels in total) and the fourth convolutional layer (256 channels). In figure 9a, channels 27 and 5 emphasize sharp edges and contours, while channels 1 and 2 focus more on intensity gradients and fine textures. The first feature map preserves much of the original image structure. In contrast, the fourth convolutional layer produces feature maps that are more abstract and coarse. This abstraction occurs because deeper layers combine outputs from previous layers and operate on downsampled inputs, reducing spatial resolution. These differences illustrate the hierarchical nature of CNNs: early layers learn diverse filters to capture complementary low-level features, while deeper layers encode higher-level representations essential for classification.

The model summary is computed with torch-summary [17]. This shows that custom model architecture has 392 616 model and trainable parameters, where most of them (389 376 or 99%) are 2D convolutions and batch normalization to extract convolutional features.



(a) Feature map for first convolutional layer



(b) Feature map for fourth, and last, convolutional layer

Figure 9: Convolutional feature maps of snail image

3. *Hyperparameter search with Optuna.*

The results and figures for training and validation accuracy and confusion matrix show the same trends as for custom architecture and training. The general accuracy for all classes is 67% and precision values for each class range from 43% to 88%. These results are marginally poorer than custom architecture and training, and again shows that some classes are harder to classify correctly than others.

The Optuna optimization history plot in figure 10 shows how the validation accuracy evolved over 50 trails by using the Optuna hyperparameter search. Pruners automatically stop unpromising trials in the early stages of training, and the figure shows only completed and not pruned trails. The objective value, validation accuracy, quickly improved to 65% accuracy on trail 16, and then only improved to 67% on trail 50. This shows that the TPESampler quickly approaches the best hyperparameters, and is efficient for discrete spaces and learns from previous trails.

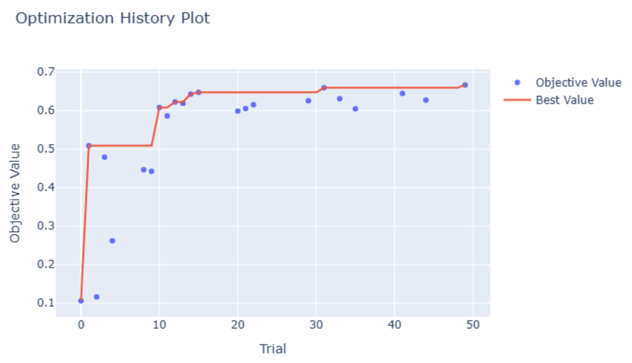


Figure 10: Optuna optimization history with validation accuracy as objective value.

The Optuna hyperparameter importance, 11, shows that the learning rate and dropout of the convolutional layers have the greatest impact on validation accuracy. This suggests that these parameters influence the models ability to generalize the most.

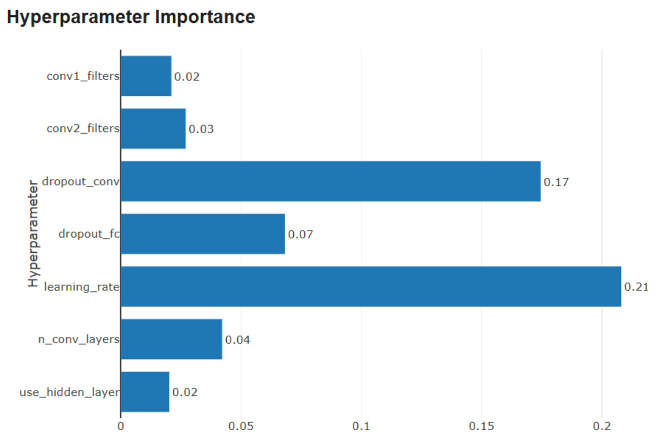


Figure 11: Optuna hyperparameter importance

The model summary shows that Optuna model has 230 376 model and trainable parameters.

4. *Transfer learning*

For transfer learning, we used the pretrained ResNet-18 model with ImageNet-1K weights. Only the fully connected layers are trained, leaving the weights for convolutional layers unchanged. The training and validation accuracy in figure 12 shows that only 5 epochs is needed before training shows signs of overfitting, stopping criteria were reached after just 20 epochs, and training is significantly faster than compared to when convolutional weights need to be trained. The model summary shows that the number of model parameters are 11 182 668 and significantly larger than custom model architecture, while trainable parameters are significantly less and only 6 156. The overall accuracy is 86% and precision values for each class range from 72% to 100%. This is significantly better than custom architecture and training of model, but still shows variability in class level performance. It is also noticeable in figure 12 that the validation accuracy fluctuates much less than with our implementation. The reason is that feature extraction is already learned in convolutional layers, fewer parameters need to be trained leading to lower variance in gradients, weights are already initialized at a good level, the dataset is small and our custom model struggle to generalize.

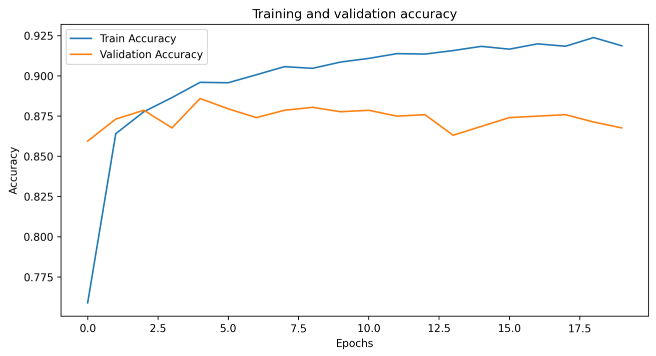
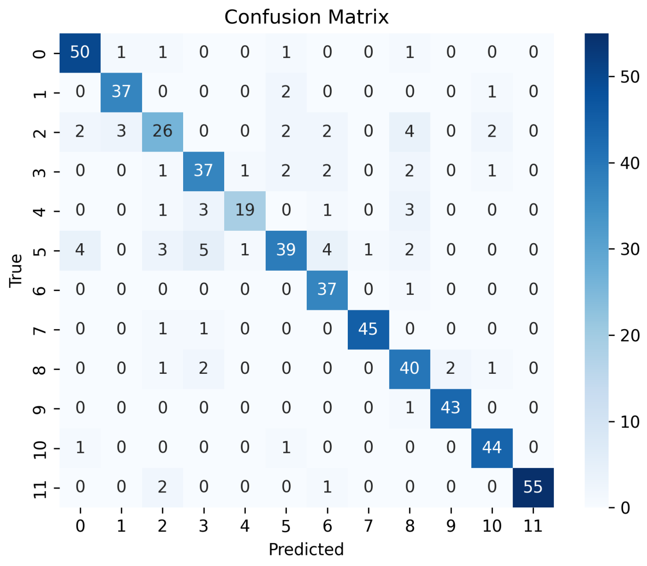


Figure 12: Training and validation accuracy for transfer learning. Training stopped after 20 epochs.

The confusion matrix in figure 13 generally shows good performance for most classes. This also confirms that some classes perform worse than other, and in general it is the same classes that struggle to classify correctly as in custom model architecture and training. Except for class 4 (earthworms), all classes are better classified than using custom architecture and training.



	0	1	2	3	4	5	6	7	8	9	10	11
0	50	1	1	0	0	1	0	0	1	0	0	0
1	0	37	0	0	0	2	0	0	0	0	1	0
2	2	3	26	0	0	2	2	0	4	0	2	0
3	0	0	1	37	1	2	2	0	2	0	1	0
4	0	0	1	3	19	0	1	0	3	0	0	0
5	4	0	3	5	1	39	4	1	2	0	0	0
6	0	0	0	0	0	0	37	0	1	0	0	0
7	0	0	1	1	0	0	0	45	0	0	0	0
8	0	0	1	2	0	0	0	0	40	2	1	0
9	0	0	0	0	0	0	0	0	1	43	0	0
10	1	0	0	0	0	1	0	0	0	0	44	0
11	0	0	2	0	0	0	1	0	0	0	0	55

Figure 13: Confusion matrix for transfer learning.

IV. CONCLUSION

In this study, we have both analyzed geophysical well-log data with a neural network and classified insects and pests from images with convolutional neural networks.

A. Regression of well-log

In this project, the model was able to capture the non-linear relationship between the rock lithologies and the sonic transit time, and it consistently outperforms the Ridge baseline.

Hyperparameter search with Optuna increased the model performance, where the learning rate proved to be the most influential parameter. The analysis of both depth-based and random splits showed that the random split is influenced by how the data are partitioned, while the depth-based split removes this dependency but performs more unstable overall.

This study illustrates that regression analysis on well-log data introduces challenges related to data splitting when working with a single well, making depth-based generalization particularly challenging. The FFNN model is overall suitable for predicting missing sonic values within a single well, but further research should focus on multiple wells in order to develop a more general predictive model.

B. Classification of insects and pests

Custom model architecture and Optuna optimized model achieved comparable performance. The results yielded 69% accuracy and precision per class from 42% to 97% for the custom model, while the Optuna model achieved 67% overall accuracy and precision for each class from 43% to 88%. Transfer learning significantly outperformed both, achieving an overall classification accuracy of 86% and per-class precision between 72% and 100%. This improvement is due to pretrained models that utilize learned feature extraction, better weight initialization, and improved generalization capabilities. Notably, the transfer learning model has about 28 times more parameters, of which only 1.5% are trainable, yet it still delivers significantly better performance than the custom architecture.

The Optuna configuration allowed for disabling of fully connected layers, which is a design flaw because these layers are important for CNN-based classification. Fully connected layers uses features extracted from convolutional layers to learn the non-linear relationship and generalize between features effectively for classification tasks. Both the custom CNN and Optuna models used relatively simple architectures compared to pretrained networks, limiting their ability to generalize. This is evident from the parameter counts: 392 616 for the custom model, 230 376 for the Optuna model, and 11 182 668 for the pretrained network.

Based on these findings, further work should should focus on better leveraging pretrained models. These models are trained on large scale datasets to learn robust feature extraction, which is critical when working with small datasets like AgroPest-12. Optuna hyperparameter optimization can still play a role in fine-tuning learning rates, the number of neurons and hidden layers in fully connected layers, activation functions, and optimizers. Additionally, experimenting with different pretrained architectures and weight initialization strategies is recommended.

Across all experiments, we observed variability in per-class precision, likely due to class imbalance. The model is fitted towards the sum of training examples, and the decision rule is likely to be biased towards the majority class. Class 11 has 960 and class 8 has 797 images, a difference of 20% with precision per class of 100% and 74%. To address this, further work should implement methods for imbalanced datasets, such as applying higher penalties for misclassifications in minority classes, upsampling minority classes, or downsampling majority classes [14].

V. CODE AVAILABILITY

The code for this project is available at `https://github.com/geomort/project3/tree/main`.

[1] Imagenet. `https://image-net.org/index.php`. Accessed: 2025-11-28.

[2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[3] S. V. Ault, S. N. Liao, and L. Musolino. *Principles of Data Science*. OpenStax, Houston, Texas, 2025. Accessed from OpenStax on Jan 24, 2025.

[4] N. O. Directorate. Factpages: Wellbore 1/9-7 — well logs. `https://factpages.sodir.no/en/wellbore/PageView/With/WellLogs/4652`, 2025. Accessed: 2025-12-07.

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Feedforward Networks*. MIT Press, 2016.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[7] C. Herrera, S. A. Minor, L. J. Anna, and T. C. C. Jr. Porosity-depth trends and regional uplift calculated from sonic logs, uinta basin, utah. Technical Report Scientific Investigations Report 2005-5051, U.S. Geological Survey, 2005.

[8] F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 754–762, Bejing, China, 22–24 Jun 2014. PMLR.

[9] Lightning AI. Torchmetrics documentation, 2025. Accessed: 2025-12-15.

[10] R. Majumdar. Crop pests dataset. `https://www.kaggle.com/datasets/rupankarmajumdar/crop-pests-dataset/data/code`, 2023. Accessed: 2025-11-21.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, 2019.

[12] PyTorch. Reducelronplateau. `https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html`, 2025. Accessed: 2025-12-01.

[13] PyTorch Contributors. *torchvision.models.resnet18 — Torchvision Models Documentation.* PyTorch Project, 2025. Includes definitions for ‘ResNet18$_Weights‘andinferencetransforms.$

[14] S. Raschka, Y. Liu, and V. Mirjalili. *Machine Learning with PyTorch and Scikit-Learn.* Packt Publishing, https://sebastianraschka.com/blog/2022/ml-pytorch-book.html, 2022.

[15] S. Raschka, Y. Liu, and V. Mirjalili. *Machine Learning with PyTorch and Scikit-Learn.* Packt Publishing, https://sebastianraschka.com/blog/2022/ml-pytorch-book.html, 2022.

[16] S. Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2025.

[17] T. Yep. Torch-summary, 3 2020.