Deep Neural Networks for Reservoir Production Forecasting

Leveraging AI for Enhanced Petroleum Engineering

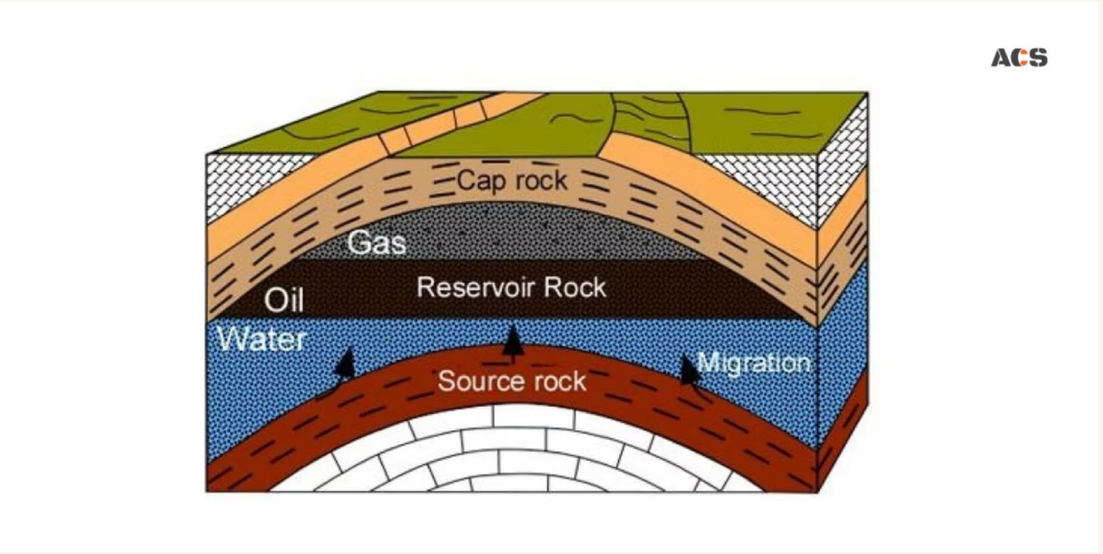
Presented by Novin Nekuee & Soroosh Danesh

Under the guidance of Dr. Emami & Eng. Nasiri

Introduction to Reservoir Simulation

Reservoir simulation predicts hydrocarbon reservoir behavior using mathematical models of fluid flow through porous rock. It helps engineers understand production rates and optimize recovery strategies.

Traditional simulations involve complex partial differential equations, considering rock and fluid properties, but are computationally expensive.



Proxy models offer a simplified, computationally efficient alternative, enabling rapid forecasting and scenario evaluation.

Project Overview: Deep Learning Proxy Model

This project develops a deep learning proxy model for reservoir production forecasting, leveraging deep neural networks and image processing.

1

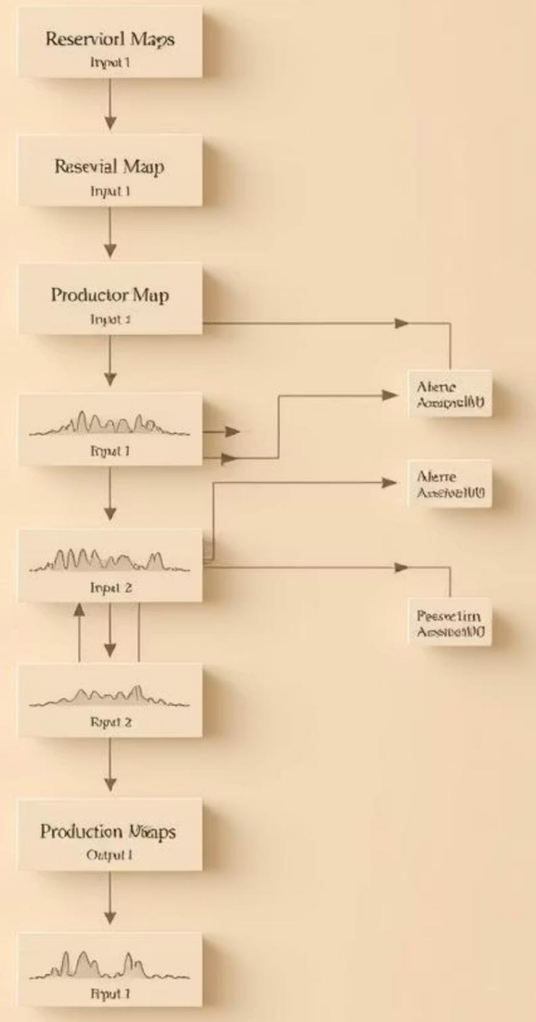
Objective

Accurately predict oil production rates and cumulative production from reservoir property maps.

2

Benefit

Provide near-instantaneous forecasts, significantly reducing computational time compared to traditional methods.



Data & Methodology

Dataset

- 1,050 reservoir model samples.
- Permeability and porosity maps (64x64 TIFF grids).
- Excel file with oil rates, cumulative production (6 time periods), and initial water saturation (Sw).

Inputs & Targets

- **Inputs:** Permeability map, Porosity map, Initial Sw.
- **Targets:** Oil production rates and cumulative oil production on the 1st day of months 1, 3, 5, 7, 9, 11.

Process

Data Preprocessing

Extract and process TIFF maps, apply normalization, and handle tabular data. Split into training, validation, and testing sets.

Model Evaluation & Optimization

Evaluate performance using regression metrics. Iterate and refine for practical utility.

Documentation & Presentation

Thoroughly document all steps, architecture, and results. Showcase technical implementation and practical value.

Learning Objectives & Impact

This project integrates deep learning, image processing, and petroleum engineering principles.

Hands-on Experience

Gain practical experience with proxy modeling techniques.

Multi-modal Data

Handle image and scalar inputs in neural networks.

Practical Challenges

Understand ML solution development for engineering.

Problem Solving

Leverage cutting-edge techniques for real-world problems.

Successful completion bridges advanced machine learning with practical engineering applications.

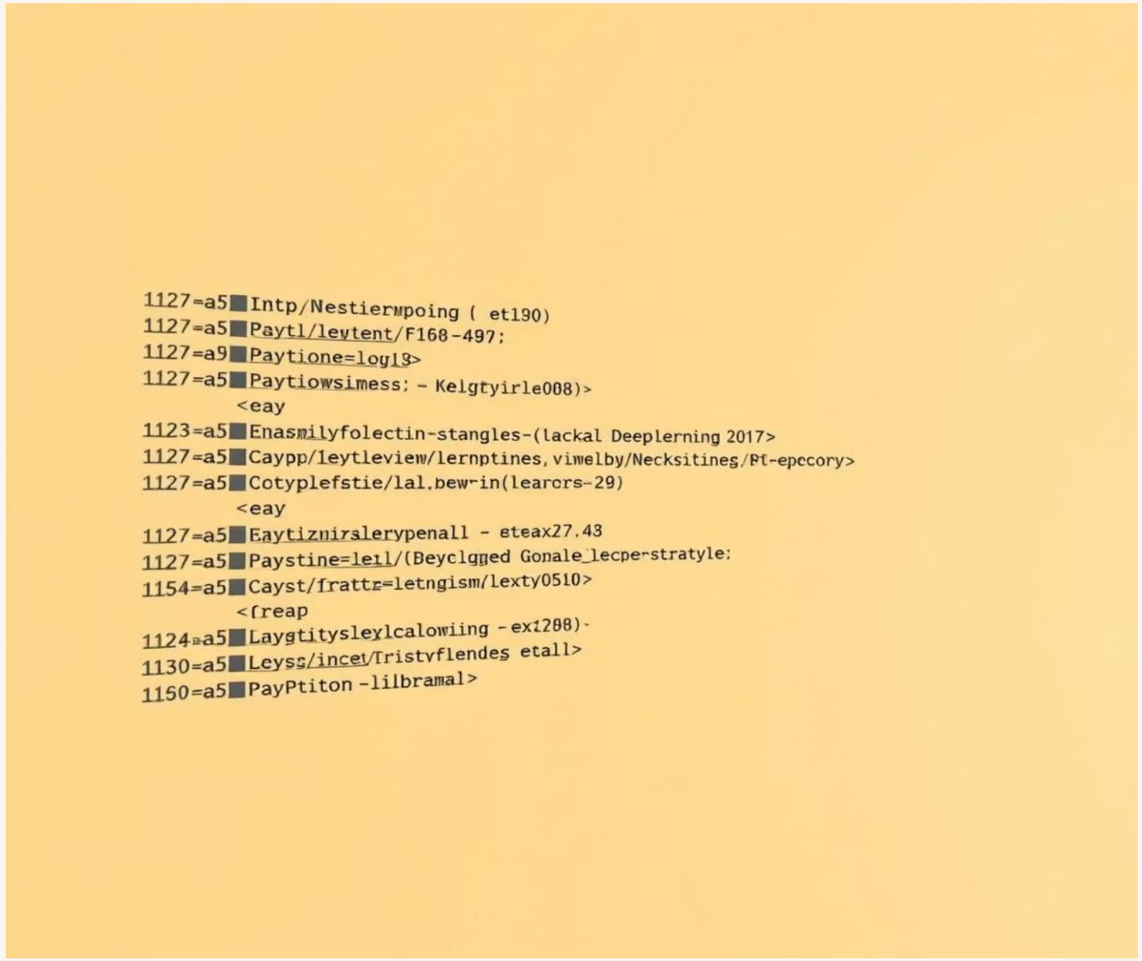
Process

1. Importing Essential Libraries

At the first part of this project we must import Python libraries for various stages of deep learning model development, from data handling to hyperparameter tuning and visualization. Each library plays a critical role in building a robust reservoir production forecasting system.

Core Components

- **Data & Numerical Ops:** `pandas` for DataFrames, `numpy` for array computations, and `os` for file management.
- **Deep Learning:** `tensorflow` and `keras` API for NN building, training, and evaluation (e.g., `Conv2D`, `Dense`, `Dropout` layers, `EarlyStopping`).
- **Data Preprocessing:** `scikit-learn` for `train_test_split` and `MinMaxScaler` for data normalization.



The efficient management of these libraries ensures seamless workflow from data ingestion to predictive modeling, crucial for accurate reservoir production forecasting.

Core Components

- **Visualization:** `matplotlib.pyplot` and `seaborn` for plotting data and model results.
- **Image Processing:** `PIL` (Pillow) for handling TIFF image files (e.g., seismic data, well logs).
- **Hyperparameter Tuning:** `optuna` for automated optimization.
- **Metrics:** `sklearn.metrics` for calculating performance metrics like MAE, MSE, and R2 score.

2. Loading the Dataset

The brief tabular data from `data.xlsx` is loaded into a pandas DataFrame named `data`. This provides a structured view of our production metrics.

```
data = pd.read_excel('./assets/data.xlsx')data.head()
```

sample number	Month (2026)	Initial Sw	Oil Rate (m3/day)	Cumulative Oil (M m3)
1	1	0.25	980.04	0.000681
1	3	0.25	410.07	25.471000
1	5	0.25	397.78	49.735000
1	7	0.25	388.08	73.408000
1	9	0.25	379.36	96.928000

3. Data Exploration and Validation

An initial Exploratory Data Analysis (EDA) is performed to understand the dataset's structure, quality, and statistical properties. This helps identify missing values and check data types.

- `.isna().sum()`: Counts missing values per column, crucial for identifying data quality issues.

```
data.isna().sum()
```

```
sample number 0
Month (2026) 0
Initial Sw 18
Oil Rate (m3/day) 18
Cumulative Oil (M m3) 19
dtype: int64
```

- `.describe()`: Provides a statistical summary of numerical columns, offering insights into distribution and scale.

```
data.describe()
```

	sample number	Month (2026)	Initial Sw	Oil Rate (m3/day)	Cumulative Oil (M m3)
count	6300.000000	6300.000000	6282.000000	6282.000000	6281.000000
mean	525.500000	6.000000	0.214938	1423.594093	197.630083
std	303.132813	3.415921	0.028712	2813.436016	501.371992
min	1.000000	1.000000	0.170000	-145.740000	0.000000
25%	263.000000	3.000000	0.190000	345.830000	16.166000
50%	525.500000	6.000000	0.210000	809.145000	85.372000
75%	788.000000	9.000000	0.240000	1423.175000	214.100000
max	1050.000000	11.000000	0.260000	25000.000000	6256.200000

- `.info()`: Summarizes DataFrame structure, including column data types and non-null counts.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6300 entries, 0 to 6299
Data columns (total 5 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   sample number          6300 non-null   int64
 1   Month (2026)           6300 non-null   int64
 2   Initial Sw             6282 non-null   float64
 3   Oil Rate (m3/day)      6282 non-null   float64
 4   Cumulative Oil (M m3)  6281 non-null   float64
dtypes: float64(3), int64(2)
memory usage: 246.2 KB
```

4. Data Preprocessing and Reshaping (For Numerical Data)

To ensure consistency with available porosity image maps, the initial DataFrame is filtered to include only the first 756 samples. The data, originally in a long format with six monthly entries per sample, is then reshaped into a wide format using `pivot_table`. This creates a single row per sample, with 12 new columns for Oil Rate and Cumulative Oil across six months. To align with our image data requirements and ensure model compatibility, we rigorously preprocess the tabular reservoir production data. This involves filtering, reshaping, and imputing missing values to create a clean, consistent dataset.

The dataset is truncated to include only the first 756 samples, matching the availability of porosity image maps. This ensures a one-to-one correspondence between tabular and image data.

```
num_available_samples = 756
df_filtered = data[data['sample number'] <= num_available_samples].copy()
```

Brief of processed tabular data

sample number	Cumulative Oil (M m3)_1	Cumulative Oil (M m3)_3	Cumulative Oil (M m3)_5	Cumulative Oil (M m3)_7	Cumulative Oil (M m3)_9	Cumulative Oil (M m3)_11	Oil Rate (m3/day)_1	Oil Rate (m3/day)_3	Oil Rate (m3/day)_5	Oil Rate (m3/day)_7	Oil Rate (m3/day)_9	Oil Rate (m3/day)_11	Initial Sw
1	0.000681	25.471	49.735	73.408	96.928	119.58	980.04	410.07	397.78	388.08	379.36	371.3	0.25
2	0.003307	118.83	218.46	302.95	378.49	444.91	4762.3	1828	1578.2	1385	1218.4	1088.8	0.23
3	0.001104	45.529	87.461	127.69	167.11	204.59	1590	725.02	687.42	659.53	635.76	614.49	0.21
4	0.003663	138.67	259.88	369.82	470.76	558.22	5274.3	2199.3	1987.1	1802.2	1628.1	1433.7	0.25
5	0.000214	9.7412	19.257	28.656	38.117	47.345	307.56	159.06	156	154.09	152.59	151.28	0.22

Reshaping to Wide Format

Original data, structured in a "long" format with six monthly entries per sample, is reshaped using `pivot_table`. This transformation yields a "wide" format, where each sample corresponds to a single row with 12 distinct columns for Oil Rate and Cumulative Oil across six-time steps.

```
data_pivot = df_filtered.pivot_table(
index='sample number’,
  columns='Month (2026)’,
  values=['Oil Rate (m3/day)', 'Cumulative
Oil (M m3)'])
```

Handling Missing Values

Post-pivot, column names are simplified, and any remaining missing values are addressed via mean imputation. While more complex methods exist, mean imputation was chosen for its simplicity and to avoid pre-normalization. A final check confirms data cleanliness.

```
for col in data_pivot.columns:
if data_pivot[col].isnull().any():
mean_val = data_pivot[col].mean()
data_pivot[col].fillna(mean_val,
inplace=True)
```

This meticulous preprocessing step ensures our tabular data is precisely aligned and ready for integration with the image-based features, forming a robust foundation for our predictive model.

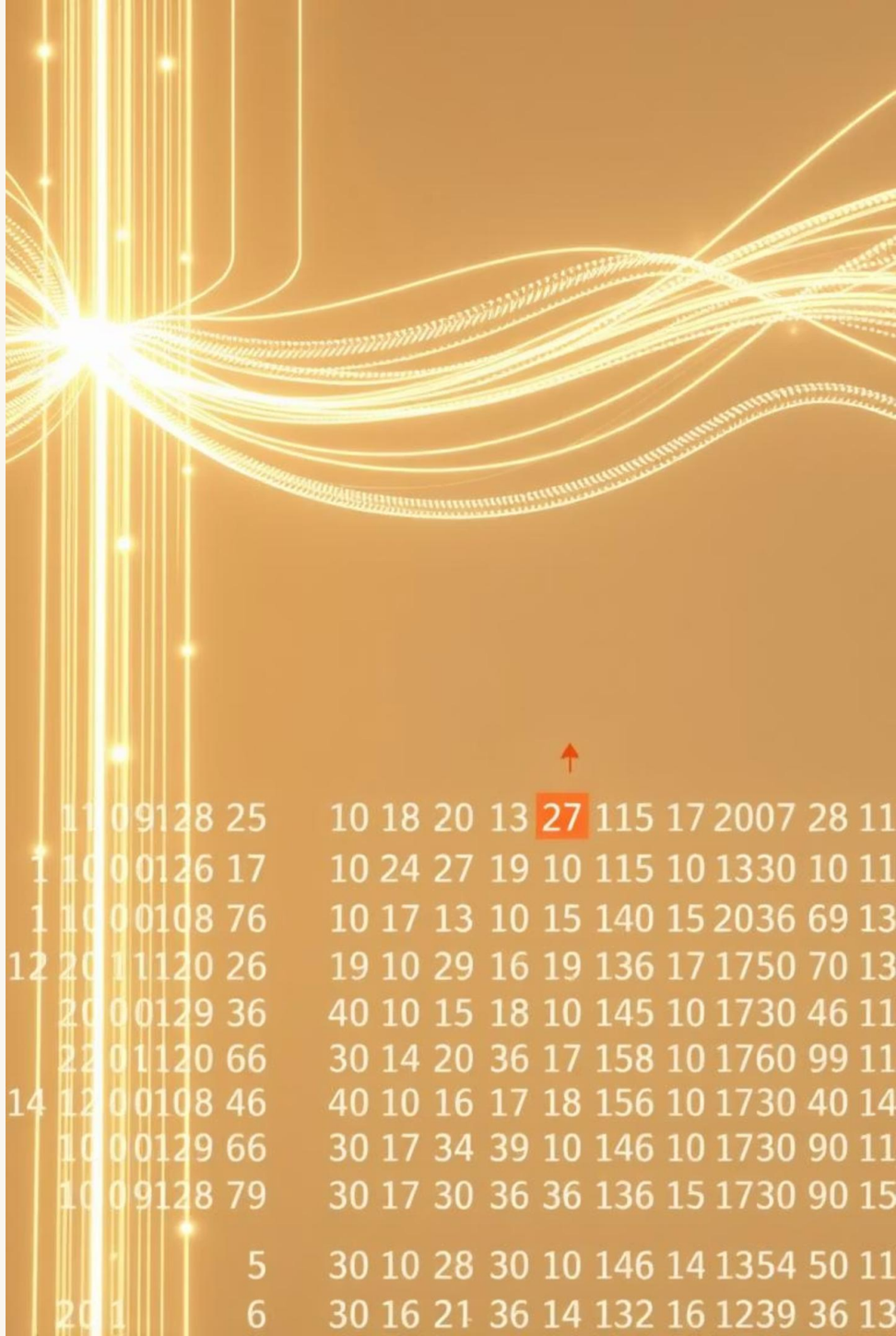
Data Validation & Insights

After preprocessing, it’s critical to validate the dataset's integrity. We verify dimensions and confirm the absence of missing values, revealing key insights into our data completeness.

- **Dataset Dimensions:** The final DataFrame comprises 753 rows and 14 columns. The reduction from 756 initial samples confirms that 3 samples were entirely absent from the original tabular data.
- **Image Alignment Requirement:** The observed 3 missing tabular samples necessitate a corresponding removal of their image data to maintain perfect alignment between the two modalities. This cross-validation is critical for model accuracy.
- **Cleanliness Confirmation:** A final check using `.isna().sum()` verifies that the pivoted and imputed data is entirely free of missing values, making it ready for the next phase of modeling.

> "Garbage in, garbage out" - Ensuring data quality at every step is paramount for reliable reservoir performance predictions.

This validation ensures that our prepared tabular data forms a reliable and consistent foundation, ready for integration with the image features and subsequent machine learning tasks.



```
data_pivot.shape
```

(753, 14)

Finally, another check for missing values confirms that the pivoted and imputed data is clean and ready for the next stage.

```
data_pivot.isna().sum()
```

```
sample number                0
Cumulative Oil (M m3)_1      0
Cumulative Oil (M m3)_3      0
Cumulative Oil (M m3)_5      0
Cumulative Oil (M m3)_7      0
Cumulative Oil (M m3)_9      0
Cumulative Oil (M m3)_11     0
Oil Rate (m3/day)_1          0
Oil Rate (m3/day)_3          0
Oil Rate (m3/day)_5          0
Oil Rate (m3/day)_7          0
Oil Rate (m3/day)_9          0
Oil Rate (m3/day)_11         0
Initial Sw                   0
dtype: int64
```

Two Approaches to Outlier Management

Post-Training Outlier Handling (main project)

Train on full dataset, then implement outlier detection/handling during inference.

- Model sees complete data distribution during training
- Can apply confidence thresholds or ensemble techniques to flag predictions on outlier inputs
- Preserves information about data boundaries and edge cases
- Risk: Model may be biased by extreme values during training

Pre-Training Outlier Removal (side poroject)

Remove anomalous data points from the training dataset before model development.

- Ensures model learns from cleaner, more representative patterns
- Often employs statistical methods (z-score, IQR) or density-based approaches (DBSCAN, isolation forests)
- Reduces noise in gradient calculations during optimization
- Risk: May discard valuable edge cases that represent real-world scenarios

Both approaches require careful validation with domain experts to distinguish between true anomalies and rare but valid data points. The optimal strategy often depends on your specific dataset characteristics and business requirements.

Understanding Outliers in Our Dataset

*This step has just used for side project

Why Outlier Detection Matters

Outliers can significantly skew model training, especially when working with:

- Time-series data like our 12 target variables
- Rate-based measurements (Oil Rate)
- Cumulative calculations that amplify anomalies

Our experimental approach helps determine whether removing statistical outliers improves model performance on this particular dataset.

Key Stats

Original dataset: 753 samples

Detected outliers: 45 samples (6%)

Cleaned dataset: 708 samples

We'll examine the boxplot visualization technique, apply the IQR method programmatically, then verify our results with a second visualization.

*Each step which has a sign of prime (" ' "), it's for side project on the other hand, it's for main project

Understanding Outliers in Our Dataset

1. Initial Visualization: Identifying Outliers with Boxplots

Visualization Process

Generated boxplots for all 12 target variables using seaborn's boxplot function with a figure size of 18×8 to accommodate all variables.

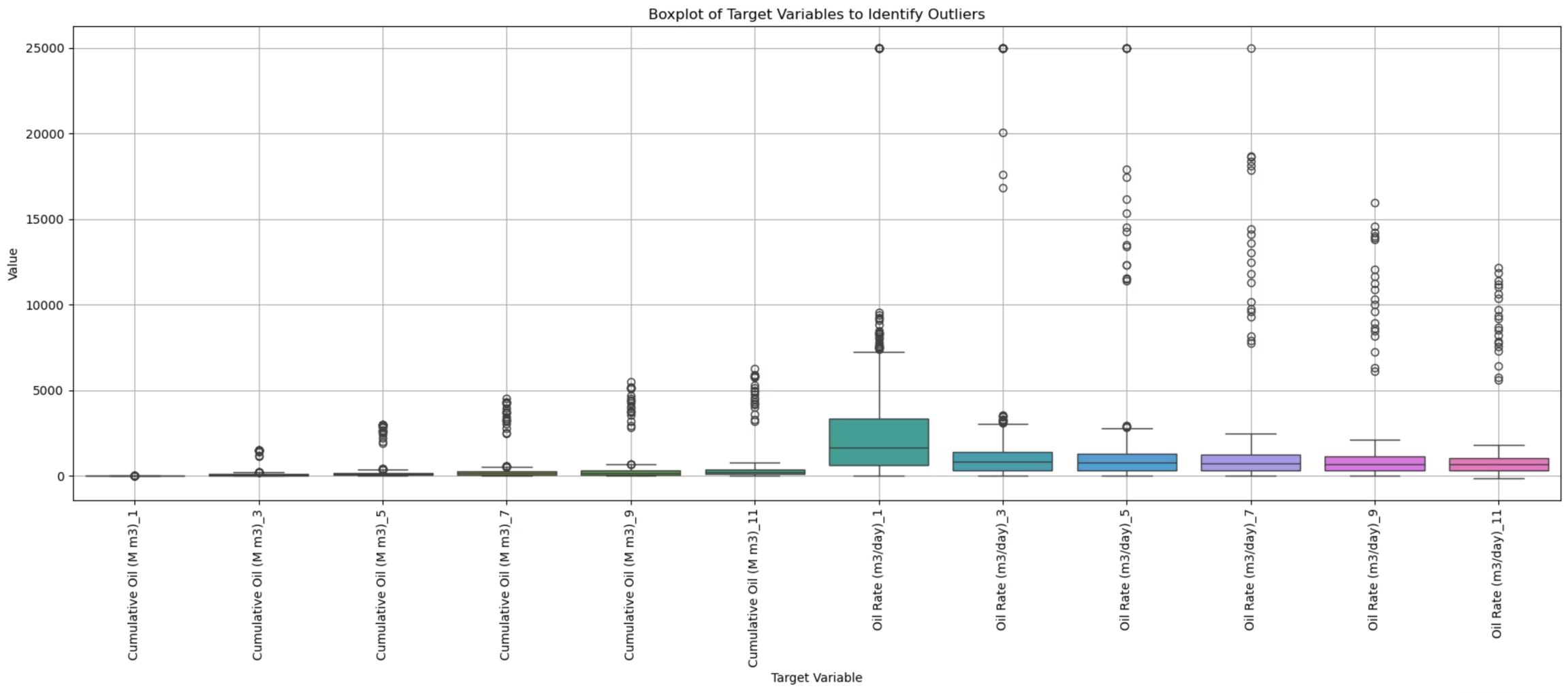
```
plt.figure(figsize=(18, 8))
sns.boxplot(data=df_targets_for_plot)
```

Observations

- Points outside the whiskers represent statistical outliers
- Oil Rate variables showed significant outliers
- Cumulative Oil in later months contained extreme values
- Y-axis scale was dominated by these extreme points

Understanding Outliers in Our Dataset

Before outliers removal



Understanding Outliers in Our Dataset

2. Programmatic Outlier Removal using the IQR Method

Calculate Quartiles & IQR

For each target variable:

```
Q1 = data_pivot[col].quantile(0.25) Q3 = data_pivot[col].quantile(0.75) IQR = Q3 - Q1
```

Define Boundaries

Set upper and lower bounds using the standard 1.5×IQR rule:

```
lower_bound = Q1 - 1.5 * IQR upper_bound = Q3 + 1.5 * IQR
```

Identify Outliers

Flag samples outside boundaries in any target variable:

```
column_outliers = data_pivot[ (data_pivot[col] < lower_bound) | (data_pivot[col] > upper_bound) ].index
```

Remove & Verify

Drop all identified outliers and confirm the new dataset size:

45 outliers removed (6% of data)

Original: 753 samples → Cleaned: 708 samples

Understanding Outliers in Our Dataset

3. Verifying Outlier Removal

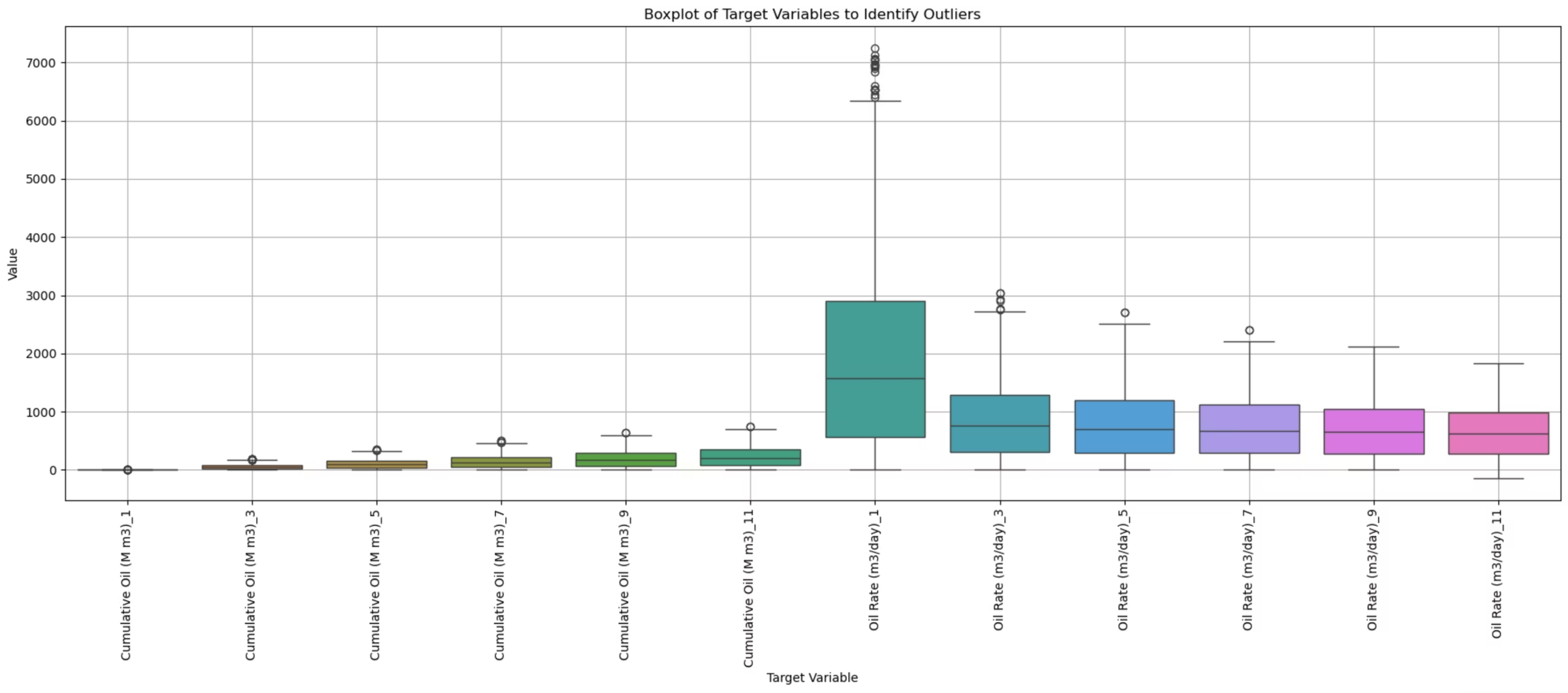
Analysis of the Cleaned Dataset

- Extreme values no longer present in the visualization
- Y-axis rescaled to show finer distribution details
- More concentrated data distribution visible in all 12 variables
- Cleaner dataset ready for experimental model training

This cleaned dataset (n=708) can now be compared against the original dataset (n=753) in model training to evaluate the impact of outlier removal on prediction accuracy.

Understanding Outliers in Our Dataset

After outliers removal



5,5′. Loading and Stacking Image Data (Image Data)

Our dataset consists of paired permeability and porosity TIFF maps for 756 reservoir samples. The processing pipeline:

Load Raw Images

Import permeability and porosity TIFF files from their respective directories

Convert to Arrays

Transform image files into NumPy arrays with float32 precision

Stack Channels

Combine paired maps into single (64, 64, 2) arrays using np.stack

Create Final Dataset

Combine all samples into a single 4D array of shape (756, 64, 64, 2)

6. Aligning Tabular and Image Datasets

Data Mismatch Challenge

Identified discrepancy: 756 image samples vs. 753 tabular records in the Excel file

Synchronization Solution

Extract valid sample numbers from tabular data and filter image array to remove orphaned samples

Final Dataset Structure

- X_image: filtered reservoir maps (753, 64, 64, 2)
- X_numerical: 'Initial Sw' values (753, 1)
- y: 12 target production variables (753, 12)

The alignment process ensures all samples have corresponding image and tabular data, preventing errors during model training and evaluation.

6′. Aligning Tabular and Image Datasets

Data Mismatch Challenge

Identified discrepancy: 756 image samples vs. 753 tabular records in the Excel file

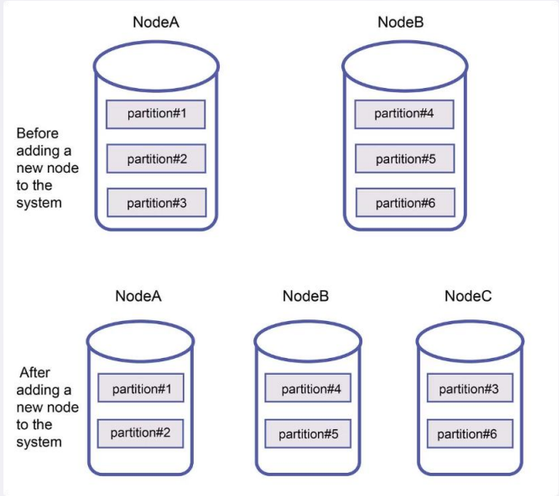
Synchronization Solution

Extract valid sample numbers from tabular data and filter image array to remove orphaned samples

Final Dataset Structure

- X_image: filtered reservoir maps (708, 64, 64, 2)
- X_numerical: 'Initial Sw' values (708, 1)
- y: 12 target production variables (708, 12)

The alignment process ensures all samples have corresponding image and tabular data, preventing errors during model training and evaluation.



7. Data Partitioning Strategy

We employ a two-stage splitting approach to create properly balanced datasets:

1

Initial Split

Separate 15% of data as test set using train_test_split with random_state=42

2

Secondary Split

Divide remaining 85% into training (70.6%) and validation (14.4%) sets

The validation size is calculated as 0.1765 of the remaining data (0.15/0.85)

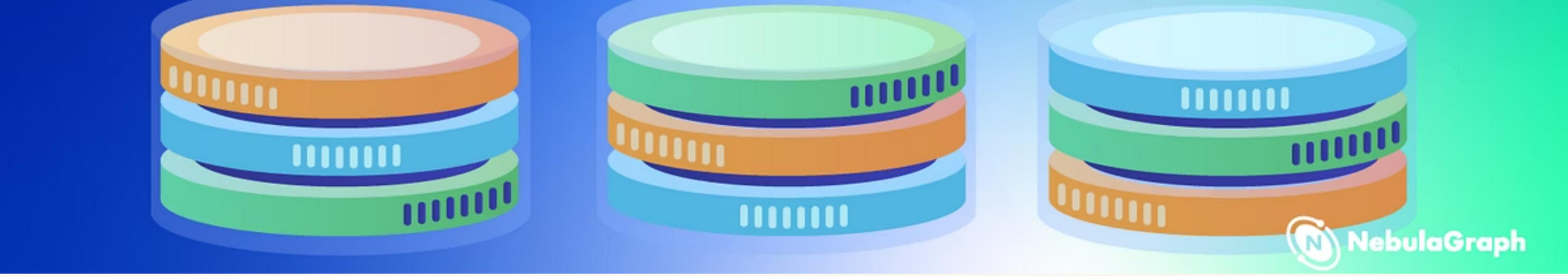
3

Final Distribution

Training: 527 samples (70%)

Validation: 113 samples (15%)

Testing: 113 samples (15%)



7′. Data Partitioning Strategy

We employ a two-stage splitting approach to create properly balanced datasets:

1

Initial Split

Separate 15% of data as test set using train_test_split with random_state=42

2

Secondary Split

Divide remaining 85% into training (70.6%) and validation (14.4%) sets

The validation size is calculated as 0.1765 of the remaining data (0.15/0.85)

3

Final Distribution

Training: 494 samples (70%)

Validation: 107 samples (15%)

Testing: 107 samples (15%)

8,8′. Data Range Analysis and Normalizing

Normalization Methodology

We implement proper normalization to scale all values to [0,1] range while preventing data leakage:

Tabular Data Scaling

Applied MinMaxScaler to target variables (y) and numerical input (Initial Sw)

- Fit scalers on training data only
- Apply the same fitted scalers to transform validation and test sets

Image Data Scaling

Implemented channel-wise normalization with manual min-max scaling:

```
X_scaled = (X - X_train_min) / (X_train_max - X_train_min)
```

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

This preserves the unique statistical distribution of each physical property

NaN Handling

Applied np.nan_to_num to replace NaN values with zeros

This ensures numerical stability during model training

This approach maintains the integrity of our test sets as truly unseen data while ensuring consistent scaling across all datasets.

8. Data Range Analysis and Normalizing

Before normalization, we assessed the value ranges across all datasets to understand scaling requirements:

Image Data

Training: nan to nan (permeability)

Validation: 0 to 11547.098

Test: 0 to 13316.762

Numerical Data

Initial Sw ranges consistently across all sets:

Min: 0.17

Max: 0.26

Target Values

Training: ~0 to 25000.0

Validation: -145.74 to 25000.0

Test: ~0 to 25000.0

The significant range differences between features confirmed the need for normalization to ensure all inputs contribute effectively during model training.

8′. Data Range Analysis and Normalizing

Before normalization, we assessed the value ranges across all datasets to understand scaling requirements:

Image Data

Training: 0 to inf (permeability)

Validation: nan to nan

Test: 0 to 277.761

Numerical Data

Initial Sw ranges consistently across all sets:

Min: 0.17

Max: 0.26

Target Values

Training: -145.74 to 7249.2

Validation: ~0 to 6900.7

Test: ~0 to 7044.4

The significant range differences between features confirmed the need for normalization to ensure all inputs contribute effectively during model training.

8. Data Range Analysis and Normalizing Normalization Results

Image Data (Post-Scaling)

- Training set: [0.0, 1.0]
- Test set: [0.0, 0.9875]
- Validation set: [0.0, 0.8562]

Numerical Data (Post-Scaling)

- All sets successfully scaled to [0.0, 1.0] range

Target Variables (Post-Scaling)

- All sets successfully scaled to [~0, 1.0] range

Note the slight range variations in test/validation sets. This is expected behavior when applying training-derived scaling parameters to unseen data distributions.

8′. Data Range Analysis and Normalizing Normalization Results

Image Data (Post-Scaling)

- Training set: [0.0, 1.0]
- Test set: [0.0, 1.0001]
- Validation set: [0.0, 0.9998]

Numerical Data (Post-Scaling)

- All sets successfully scaled to [0.0, 1.0] range

Target Variables (Post-Scaling)

- Training set: [0.0, 1.0]
- Test set: [0.0004, 1.0178]
- Validation set: [-0.0001, 0.9519]

Note the slight range variations in test/validation sets. This is expected behavior when applying training-derived scaling parameters to unseen data distributions.

9,9′.Feature Correlation Analysis

We analyzed Pearson correlations between all features to understand relationships between inputs and targets:

1 Correlation Visualization

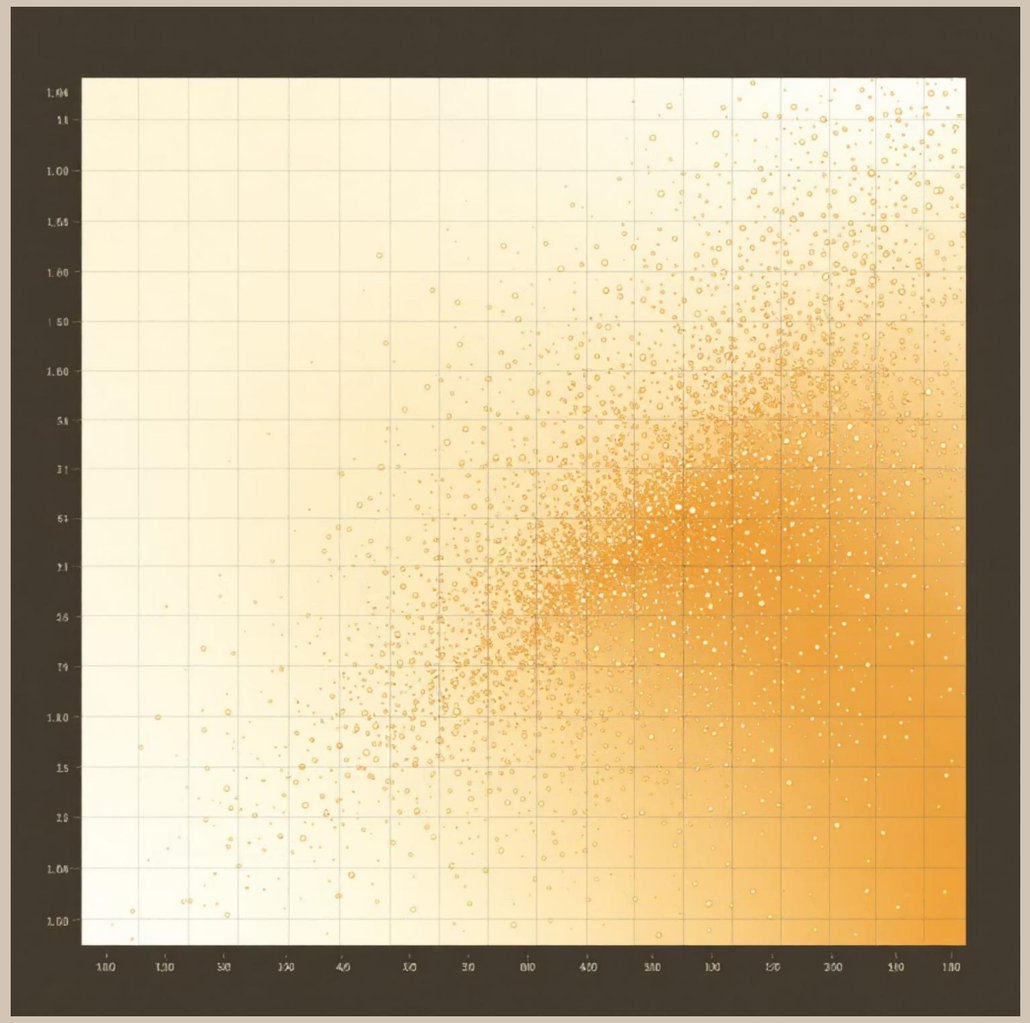
Created a heatmap to visualize the strength and direction of relationships between all variables

Used 'coolwarm' color mapping with red indicating positive correlation and blue indicating negative correlation

2 Initial Sw Relationships

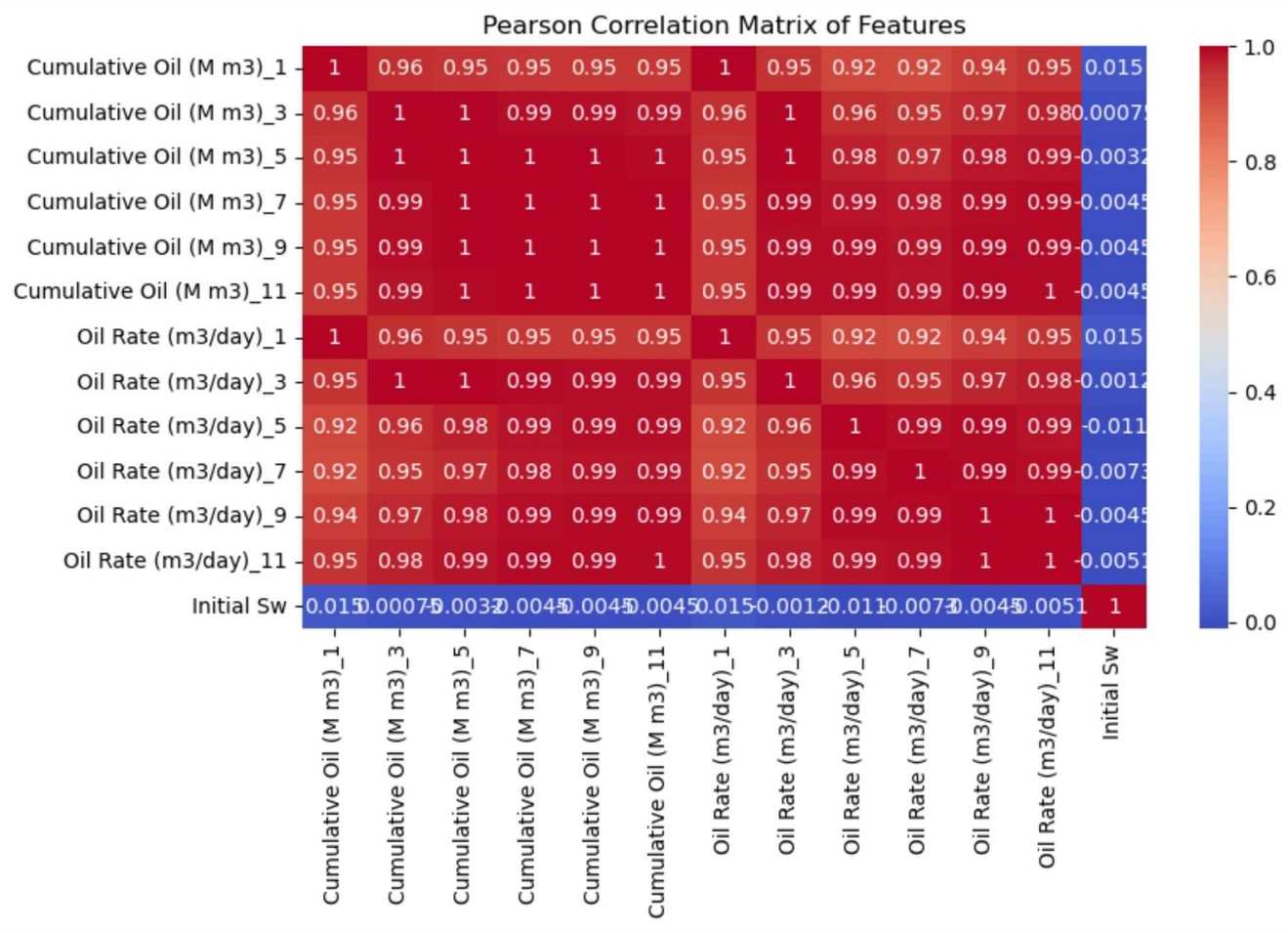
Extracted specific correlation values between our single numerical input feature (Initial Sw) and all target variables

This reveals which production metrics are most strongly influenced by initial water saturation



This analysis provides valuable insights into feature relationships but doesn't impact our feature selection strategy as we're using all available features in the model.

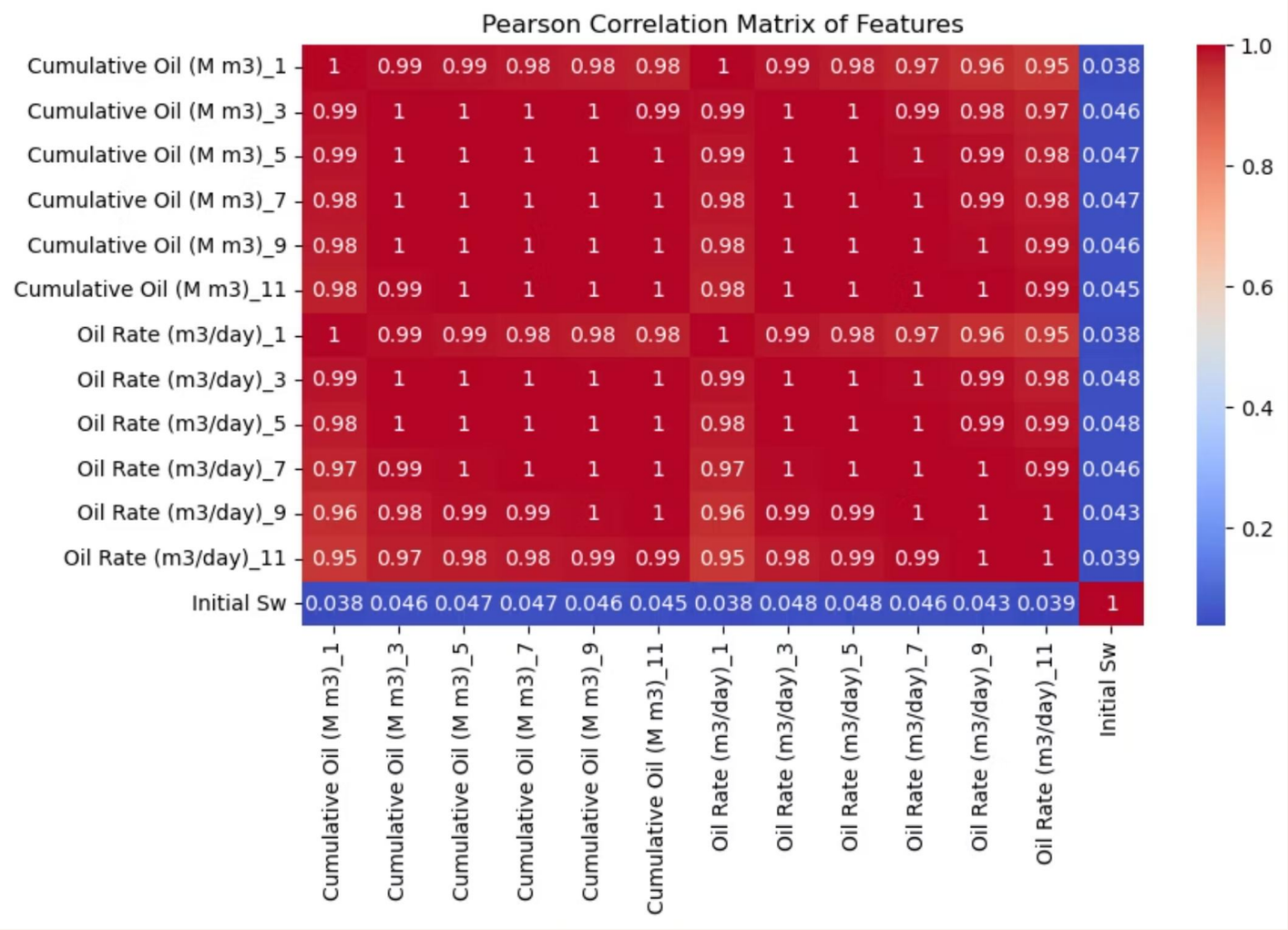
9.Visualizing Feature Correlations results



Correlation of 'Initial Sw' with Production Data

	Initial Sw
Initial Sw	1.000000
Oil Rate (m3/day)_1	0.015051
Cumulative Oil (M m3)_1	0.015051
Cumulative Oil (M m3)_3	0.000748
Oil Rate (m3/day)_3	-0.001189
Cumulative Oil (M m3)_5	-0.003161
Cumulative Oil (M m3)_9	-0.004512
Cumulative Oil (M m3)_7	-0.004521
Oil Rate (m3/day)_9	-0.004528
Cumulative Oil (M m3)_11	-0.004532
Oil Rate (m3/day)_11	-0.005080
Oil Rate (m3/day)_7	-0.007303
Oil Rate (m3/day)_5	-0.011496

9'.Visualizing Feature Correlations results



Correlation of 'Initial Sw' with Production Data

	Initial Sw
Initial Sw	1.000000
Oil Rate (m3/day)_5	0.048151
Oil Rate (m3/day)_3	0.047877
Cumulative Oil (M m3)_7	0.046707
Cumulative Oil (M m3)_5	0.046706
Oil Rate (m3/day)_7	0.046189
Cumulative Oil (M m3)_9	0.046135
Cumulative Oil (M m3)_3	0.045793
Cumulative Oil (M m3)_11	0.045343
Oil Rate (m3/day)_9	0.042900
Oil Rate (m3/day)_11	0.039098
Oil Rate (m3/day)_1	0.038003
Cumulative Oil (M m3)_1	0.038002

Key Takeaways

Multi-Channel Reservoir Imaging

Successfully combined permeability and porosity maps into a dual-channel 4D tensor (753, 64, 64, 2) for side project and (708, 64, 64, 2) for side project

Data Synchronization

Resolved dataset misalignment issues to ensure perfect correspondence between image and tabular data

Strategic Data Partitioning

Implemented a balanced 70/15/15 split for training, validation, and testing

Proper Normalization

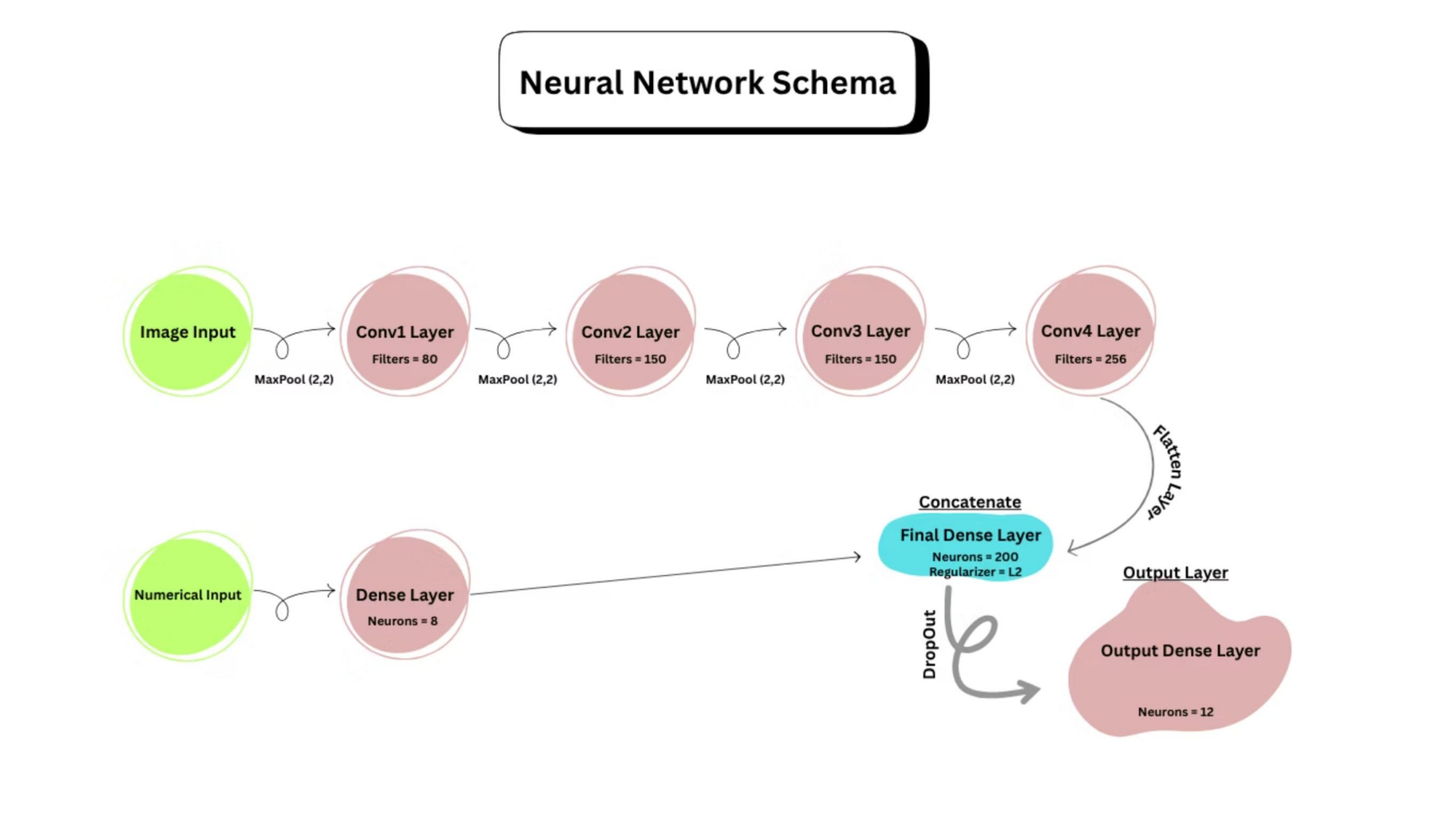
Applied best-practice scaling techniques to prevent data leakage while ensuring consistent [0,1] ranges

This preprocessing pipeline provides a solid foundation for building advanced deep learning models that can effectively analyze reservoir characteristics and predict production outcomes.

Next steps: Implement CNN architecture to extract spatial features from reservoir maps and combine with Initial Sw data to predict the 12 target variables.

10,10′. Building the Final Optimized Model

Model Overview



10,10′. Building the Final Optimized Model

Model Overview

Architecture

Multi-branch CNN with convolutional layers for spatial features and dense layers for numerical processing

Inputs

2-channel image data (64×64) and 1 numerical feature

Output

12 regression targets predicted simultaneously

Our challenge was to build a model that could efficiently extract features from both image and numerical data, combining them to make accurate predictions across multiple targets.

10. Building the Final Optimized Model

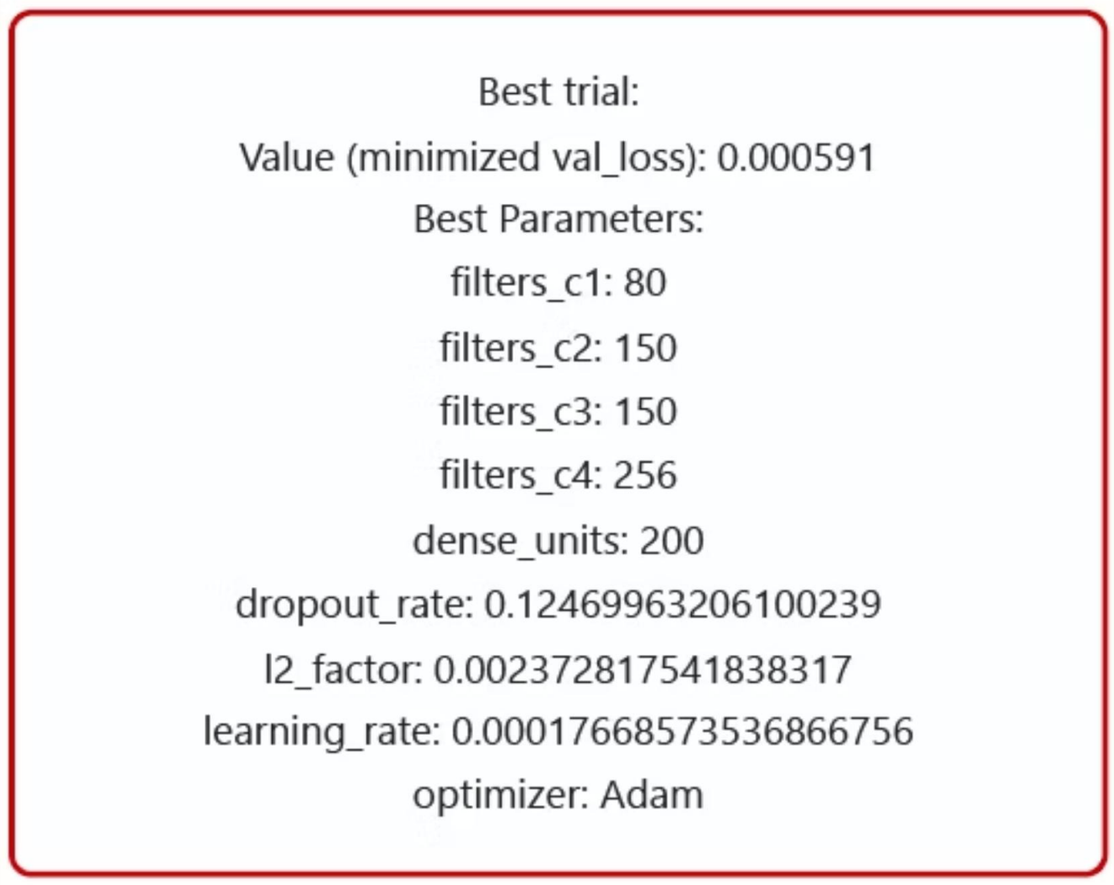
Hyperparameter Optimization Strategy

We leveraged Optuna for systematic hyperparameter tuning, searching through a complex parameter space to minimize validation loss.

After 100 trials, trial #31 emerged as the best performer with a validation loss of 0.00059.

Search Space Included:

- Filter counts across 4 Conv2D layers
- Dense layer unit counts
- Regularization parameters (dropout, L2)
- Learning rate and optimizer selection



10′. Building the Final Optimized Model

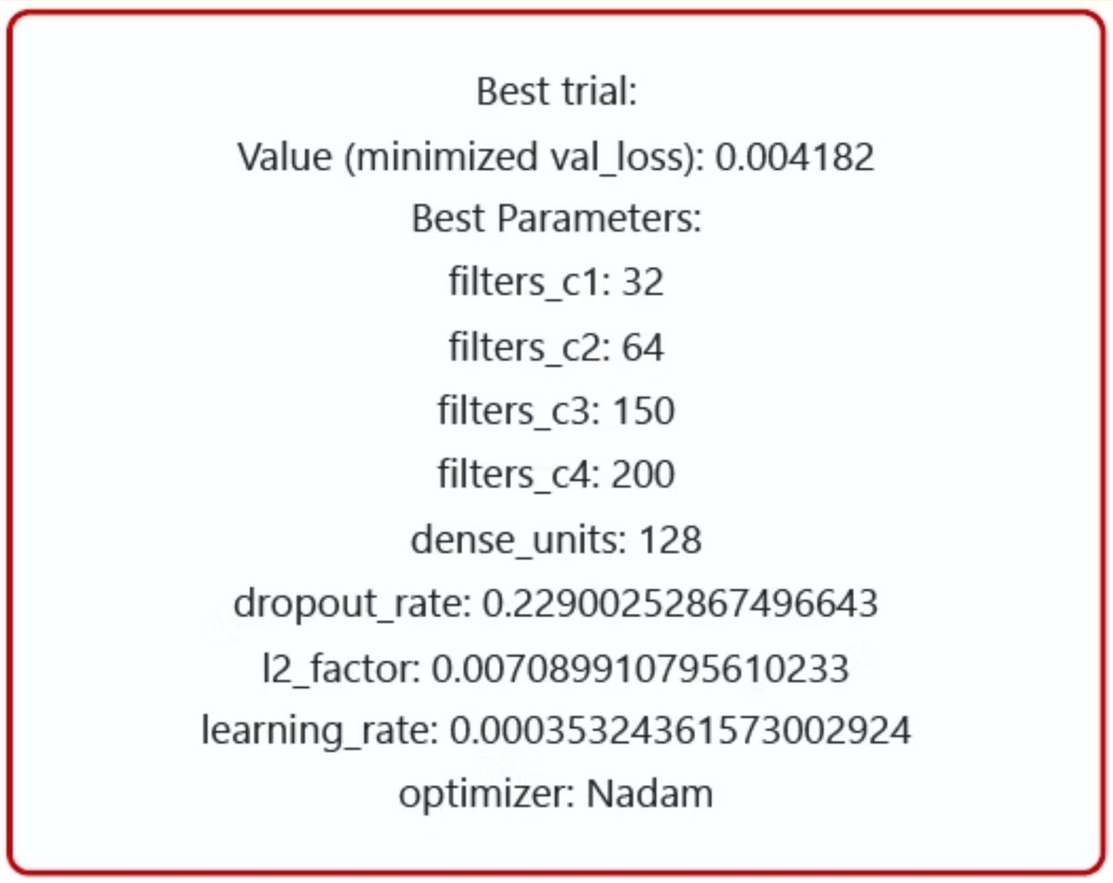
Hyperparameter Optimization Strategy

We leveraged Optuna for systematic hyperparameter tuning, searching through a complex parameter space to minimize validation loss.

After 100 trials, trial #47 emerged as the best performer with a validation loss of 0.00418.

Search Space Included:

- Filter counts across 4 Conv2D layers
- Dense layer unit counts
- Regularization parameters (dropout, L2)
- Learning rate and optimizer selection



10. Building the Final Optimized Model

Model summary

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
image_input (InputLayer)	(None, 64, 64, 2)	0	-
conv2d (Conv2D)	(None, 62, 62, 80)	1,520	image_input[0][0]
max_pooling2d (MaxPooling2D)	(None, 31, 31, 80)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 29, 29, 150)	108,150	max_pooling2d[0]…
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 150)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 12, 12, 150)	202,650	max_pooling2d_1[…
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 150)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 4, 4, 256)	345,856	max_pooling2d_2[…
numerical_input (InputLayer)	(None, 1)	0	-
flatten (Flatten)	(None, 4096)	0	conv2d_3[0][0]
dense (Dense)	(None, 8)	16	numerical_input[…
concatenate (Concatenate)	(None, 4104)	0	flatten[0][0], dense[0][0]
dense_1 (Dense)	(None, 200)	821,000	concatenate[0][0]
dropout (Dropout)	(None, 200)	0	dense_1[0][0]
output (Dense)	(None, 12)	2,412	dropout[0][0]

Total params: 1,481,604 (5.65 MB)

Trainable params: 1,481,604 (5.65 MB)

Non-trainable params: 0 (0.00 B)

10′. Building the Final Optimized Model

Model summary

Model: "functional"

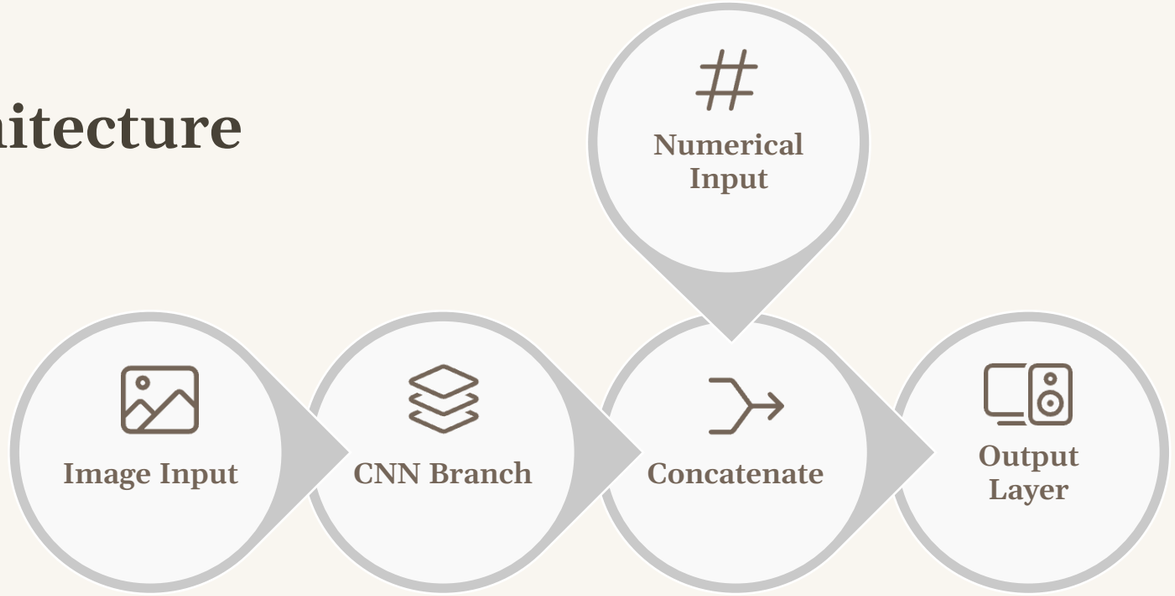
Layer (type)	Output Shape	Param #	Connected to
image_input (InputLayer)	(None, 64, 64, 2)	0	-
conv2d (Conv2D)	(None, 62, 62, 32)	608	image_input[0][0]
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 29, 29, 64)	18,496	max_pooling2d[0]…
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 12, 12, 150)	86,550	max_pooling2d_1[…
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 150)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 4, 4, 200)	270,200	max_pooling2d_2[…
numerical_input (InputLayer)	(None, 1)	0	-
flatten (Flatten)	(None, 3200)	0	conv2d_3[0][0]
dense (Dense)	(None, 8)	16	numerical_input[…
concatenate (Concatenate)	(None, 3208)	0	flatten[0][0], dense[0][0]
dense_1 (Dense)	(None, 128)	410,752	concatenate[0][0]
dropout (Dropout)	(None, 128)	0	dense_1[0][0]
output (Dense)	(None, 12)	1,548	dropout[0][0]

Total params: 788,170 (3.01 MB)

Trainable params: 788,170 (3.01 MB)

Non-trainable params: 0 (0.00 B)

Optimal Model Architecture



1 CNN Branch

Four Conv2D layers (80→150→150→256) filters for main project and Four Conv2D layers (32→64→150→200) filters for side project with SELU activation and MaxPooling, extracting increasingly complex spatial features

2 Numerical Branch

Small dense layer (8 units) to process the single numerical input feature

3 Feature Fusion

Concatenation layer merges flattened CNN features with numerical branch output

4 Final Processing

- Dense layer (200 units) with L2 regularization (0.0024) and dropout (0.125), feeding into linear output layer with 12 units (for main project)
- Dense layer (128 units) with L2 regularization (0.0071) and dropout (0.229), feeding into linear output layer with 12 units (for side project)

Best Hyperparameters from Optuna (for main project)

80

Filters C1

First Conv2D layer

150

Filters C2

Second Conv2D layer

150

Filters C3

Third Conv2D layer

256

Filters C4

Fourth Conv2D layer

200

Dense Units

Final dense layer

12.5%

Dropout Rate

For regularization

0.0024

L2 Factor

Weight regularization

0.00017

Learning Rate

For Aadam optimizer

Best Hyperparameters from Optuna (for side project)

32

Filters C1

First Conv2D layer

64

Filters C2

Second Conv2D layer

150

Filters C3

Third Conv2D layer

200

Filters C4

Fourth Conv2D layer

128

Dense Units

Final dense layer

22.9%

Dropout Rate

For regularization

0.0071

L2 Factor

Weight regularization

0.00035

Learning Rate

For Nadam optimizer

11,11′. Defining Callbacks and Compiling the Model

Training Strategy & Callbacks

Advanced Callbacks

Early Stopping

Monitored val_loss with patience=15

Restored best weights automatically

For main project training helted have not recognized, best weights from epoch 200

For side project training halted at epoch 189, best weights from epoch 174

Reduce LR On Plateau

Monitored val_loss with patience=10

Reduction factor: 0.1

Minimum learning rate: 1e-5

12,12′. Model Training

Training Results & Convergence

Training Configuration

- Batch size: 32
- epochs: 200
- Optimizer:
  - Aadam with optimal learning rate (0.00017) (main project)
  - Nadam with optimal learning rate (0.00035) (side project)
- Loss function: Mean Squared Error
- Metrics: MAE, RMSE

13. Final Model Evaluation on the Test Set

Main project model Performance on Test Set

174.89

MAE

Mean Absolute Error

241,136

MSE

Mean Squared Error

491.06

RMSE

Root Mean Squared Error

0.979

R^2

Coefficient of Determination

Our model achieved impressive performance on the completely unseen test set, explaining 97.9% of the variance in the target variables (R^2=0.979).

To generate these metrics, we:

1. Made predictions on scaled test inputs
2. Applied inverse scaling to convert predictions back to original units
3. Calculated standard regression metrics against ground truth

*These metrics has calculated after reverse scaling

13′. Final Model Evaluation on the Test Set

Side project model Performance on Test Set

82.10

MAE

Mean Absolute Error

35,819

MSE

Mean Squared Error

189.26

RMSE

Root Mean Squared Error

0.933

R^2

Coefficient of Determination

Our model achieved impressive performance on the completely unseen test set, explaining 93.3% of the variance in the target variables (R^2=0.933).

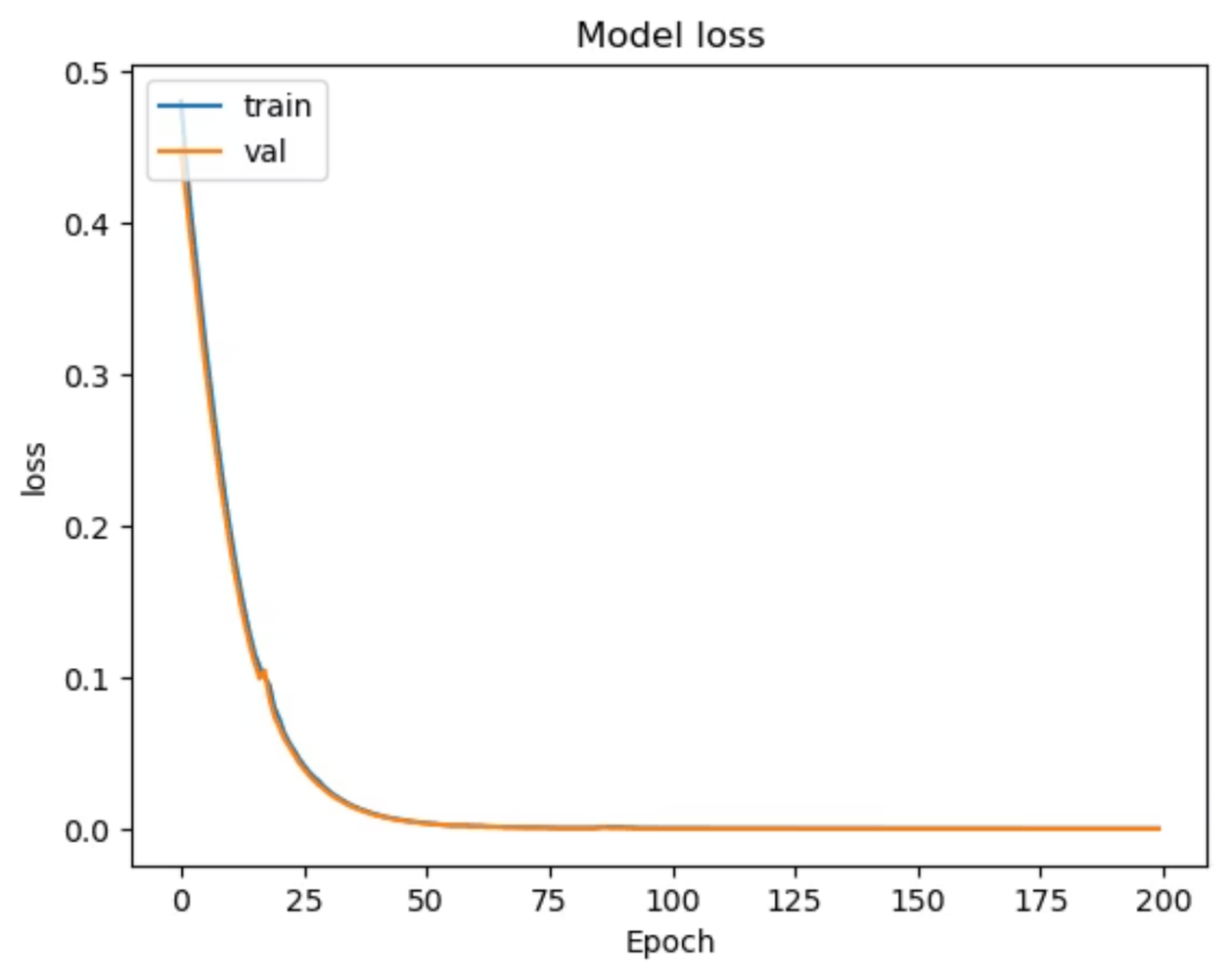
To generate these metrics, we:

1. Made predictions on scaled test inputs
2. Applied inverse scaling to convert predictions back to original units
3. Calculated standard regression metrics against ground truth

*These metrics has calculated after reverse scaling

14. Visualizing Training History

Loss function of main project



The training process exhibited excellent convergence characteristics:

Smooth Learning Curve

Both training and validation loss decreased sharply and smoothly, eventually converging to very low error values

No Overfitting

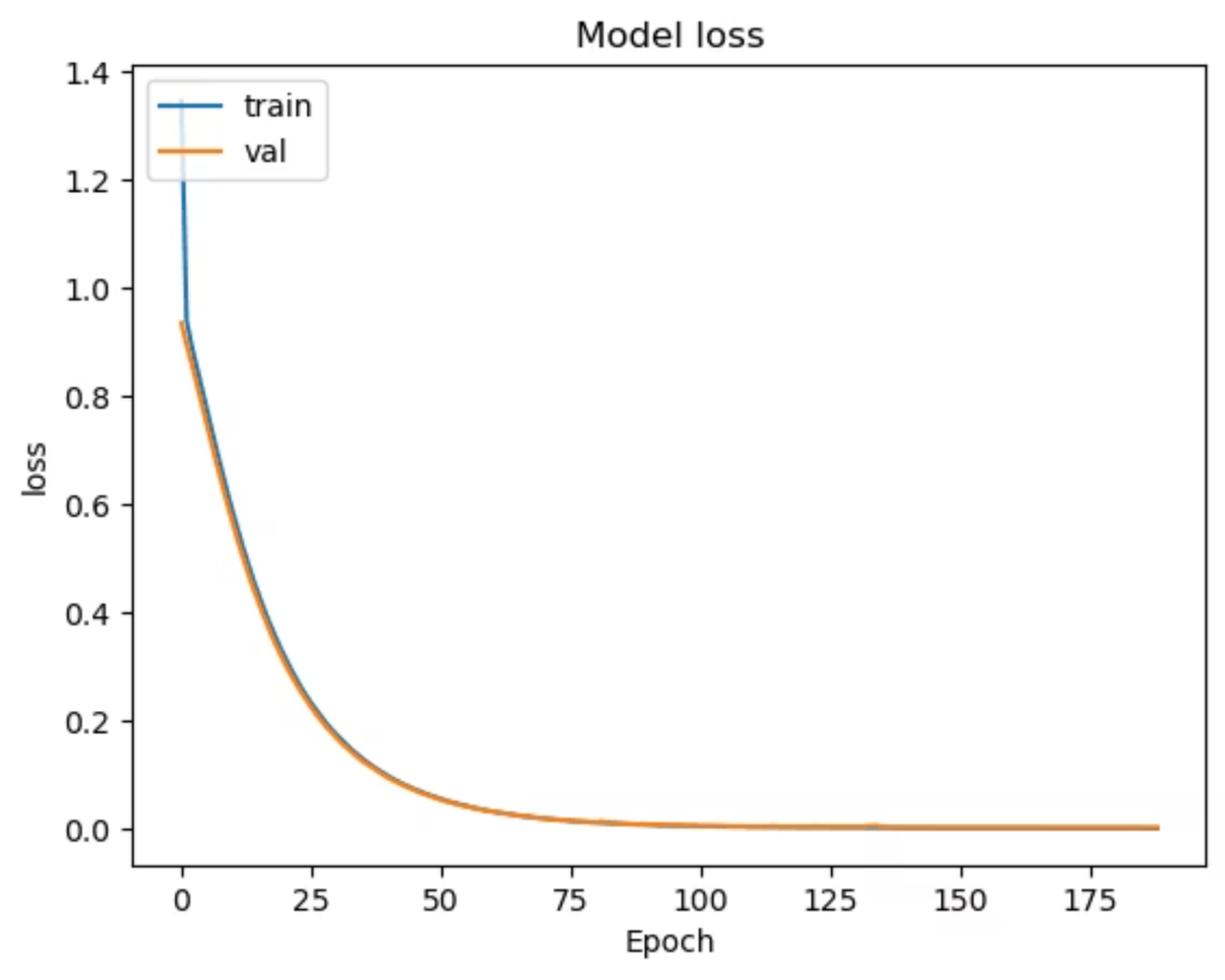
The minimal and stable gap between training and validation loss curves indicates the model generalized well to unseen data

Effective Regularization

The combination of L2 regularization and dropout successfully prevented memorization of training data

14′. Visualizing Training History

Loss function of side project



The training process exhibited excellent convergence characteristics:

Smooth Learning Curve

Both training and validation loss decreased sharply and smoothly, eventually converging to very low error values

No Overfitting

The minimal and stable gap between training and validation loss curves indicates the model generalized well to unseen data

Effective Regularization

The combination of L2 regularization and dropout successfully prevented memorization of training data

15. Model Validation and Visualization on unseen data

15.1 Actual vs. Predicted Analysis

Initial visual inspection and correlation assessment

15.2 Residual Analysis

Systematic patterns and outlier impact

15.3 Single Sample Case Study

Granular analysis of individual predictions

15.4 Outlier Removal Analysis

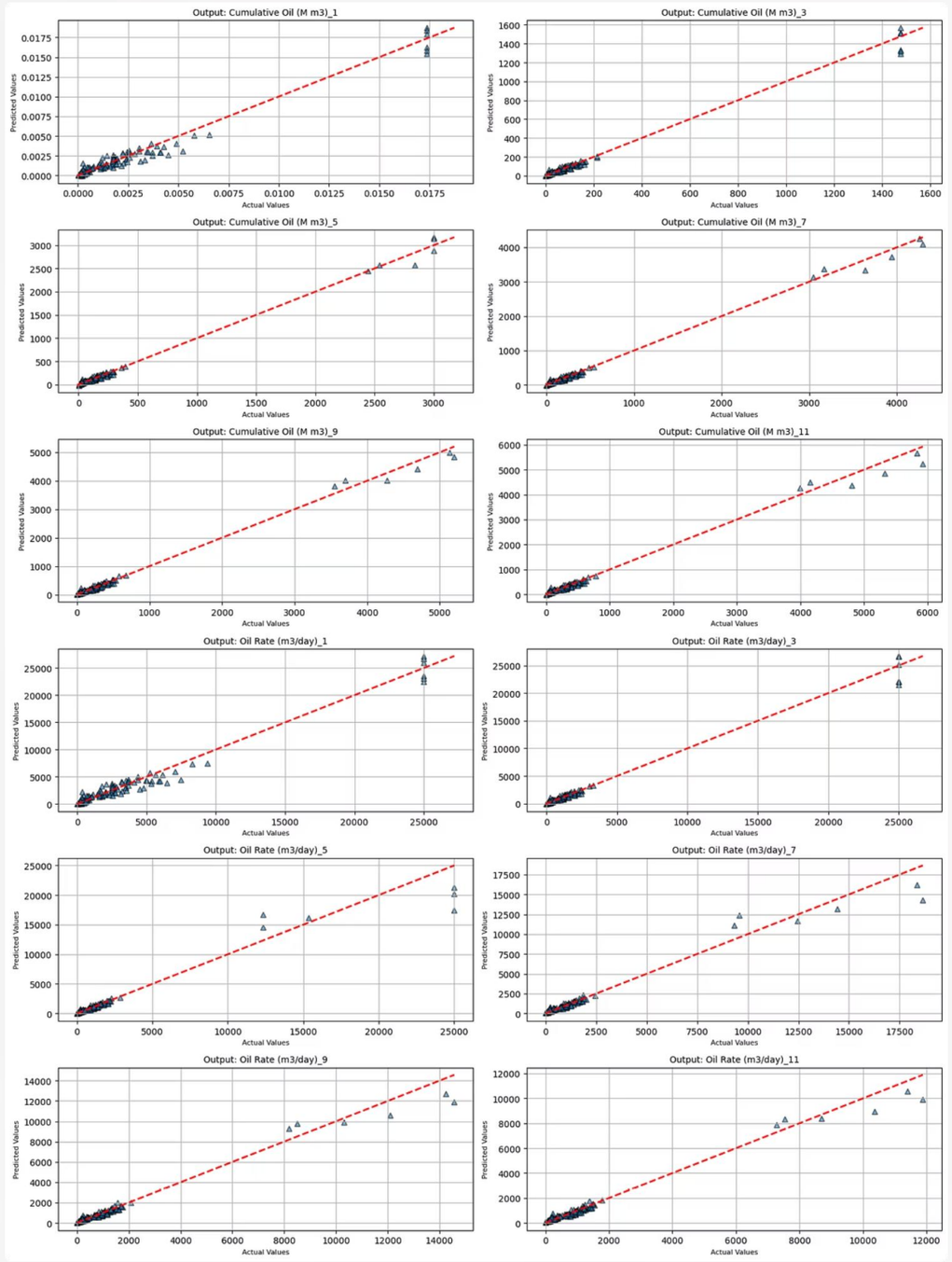
Performance assessment with and without outliers

15.5 Visualizing the Outlier Threshold

The threshold line effectively separates these few outlier predictions

15.6 Final Diagnostics

Validating model reliability and calibration



15.1 Actual vs. Predicted: Initial Analysis

Key Observations

- Strong linear correlation for all outputs
- Most data points clustered tightly around the perfect prediction line
- Several distinct outlier points visible with noticeably larger errors

Implications

While overall metrics (like R^2) were very high, this visual inspection revealed that a few specific points might be disproportionately influencing the results, potentially masking the true performance on the majority of the data.

15.2 Residual Analysis: Understanding Error Patterns

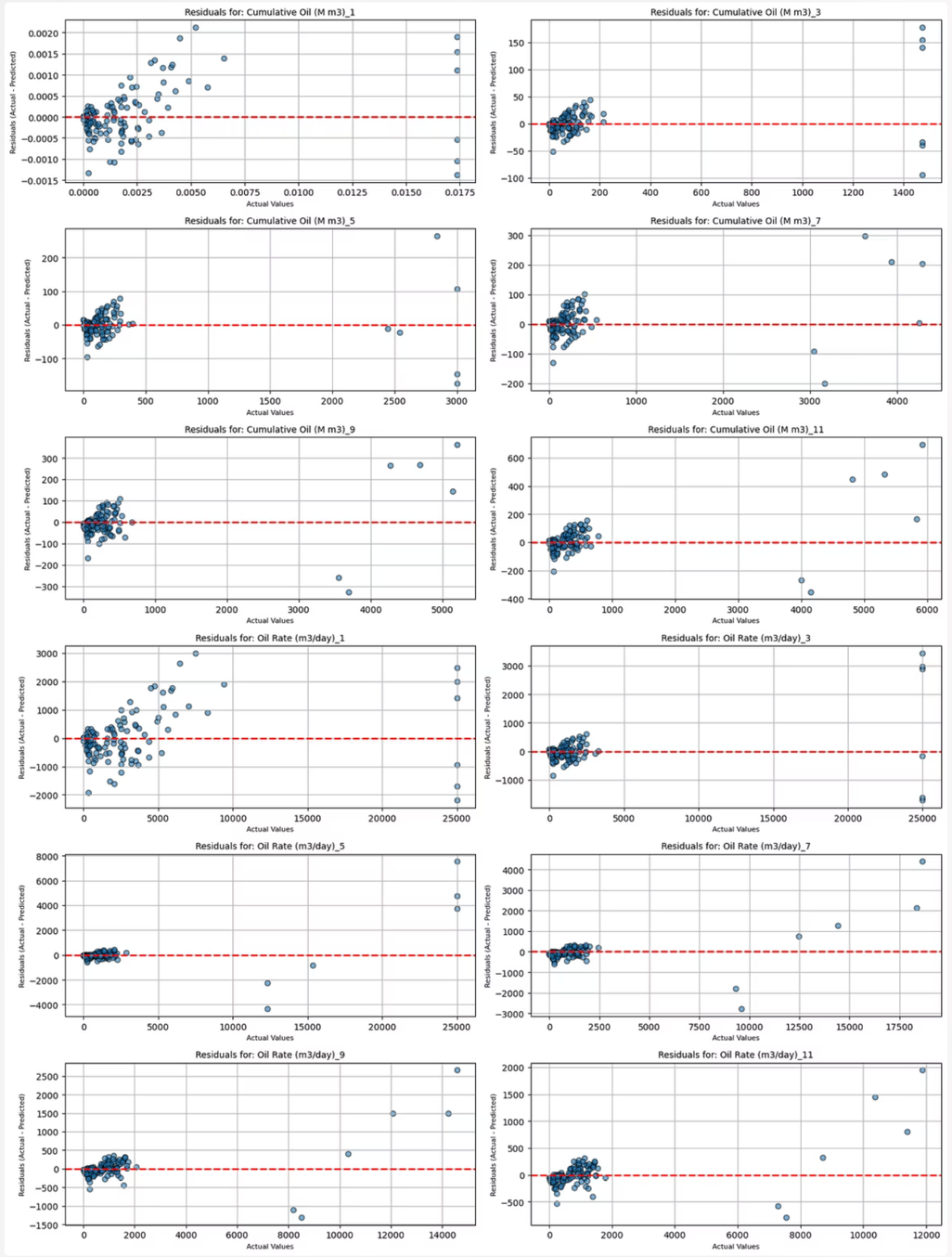
Ideal Residual Pattern

In an ideal model, residuals should be randomly scattered in a horizontal band around the zero-error line, showing no discernible patterns.

What We Found

Two distinct patterns emerged across all 12 subplots:

1. The vast majority of residuals clustered in a very tight band near y=0
2. A small number of points scattered far above or below the main cluster

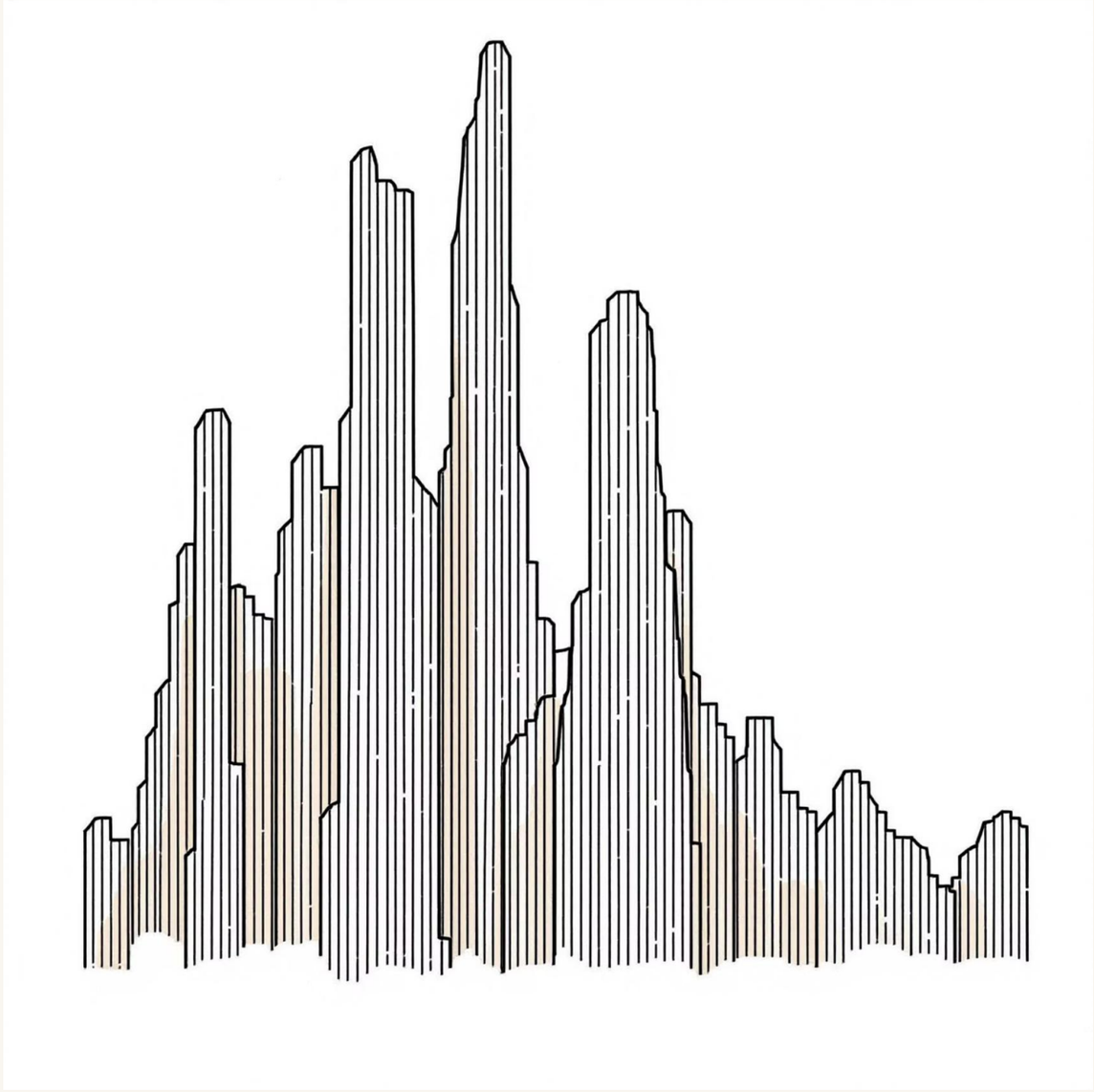


The Impact of Performance Outliers

The few outlier points identified have a disproportionately large effect on overall performance metrics.

> Because metrics like Mean Squared Error (MSE) square the error term, large residuals heavily penalize the model and inflate the final error scores.

This can mask the model's excellent performance on the overwhelming majority of the data points.



This visual analysis clearly justifies the decision to re-calculate performance

15.3 Case Study: Single Sample Prediction Analysis

While aggregate metrics provide a high-level view, examining individual predictions offers concrete insights into model behavior.

Output Variable	Actual	Predicted
Cumulative Oil (M m3)_1	0.02	0.02
Cumulative Oil (M m3)_3	1475.0	1515.0
Cumulative Oil (M m3)_5	3000.0	3174.4
Cumulative Oil (M m3)_7	4257.0	4254.3

Case study

Output Variable	Actual	Predicted
Oil Rate (m3/day)_1	25000.0	23575.6
Oil Rate (m3/day)_3	25000.0	26716.9
Oil Rate (m3/day)_5	25000.0	21247.7
Oil Rate (m3/day)_7	18353.0	16218.2
Oil Rate (m3/day)_9	14233.0	12736.2
Oil Rate (m3/day)_11	11398.0	10596.7

This sample (index=35) shows the model's prediction accuracy in its original, physical units—providing a granular perspective beyond summary statistics.

15.4 Performance Without Outliers

93.31	234,35	153.08	98.2%
MAE	**MSE**	**RMSE**	**R^2**
Mean Absolute Error on cleaned data	Mean Squared Error	Root Mean Squared Error after removing outliers	Coefficient of determination on typical data points

These metrics were calculated after removing the top 5% of prediction errors (defined as those above the 95th percentile threshold). The significant improvement confirms the model is exceptionally accurate on the vast majority of data.

15.5 Visualizing the Outlier Threshold

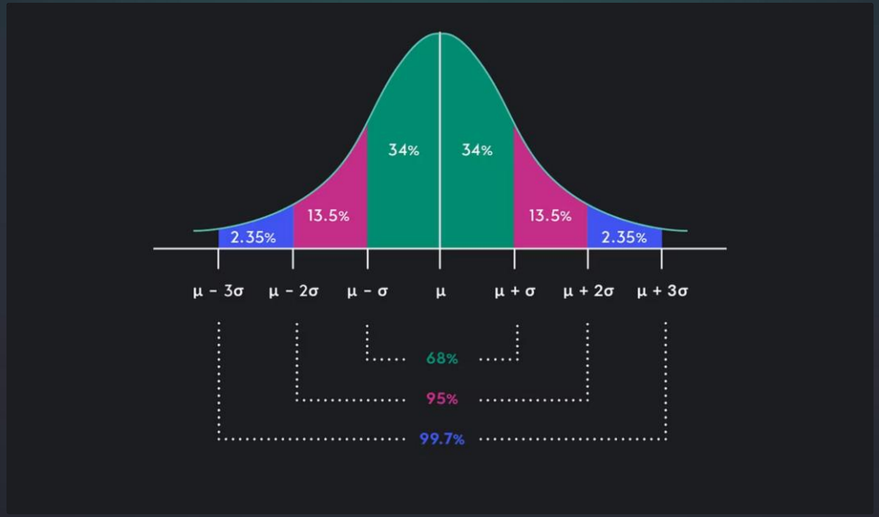
Process:

1. Calculate absolute error for every prediction
2. Define threshold as 95th percentile of errors
3. Flag samples with errors above this threshold
4. Recalculate metrics on remaining 95% of data

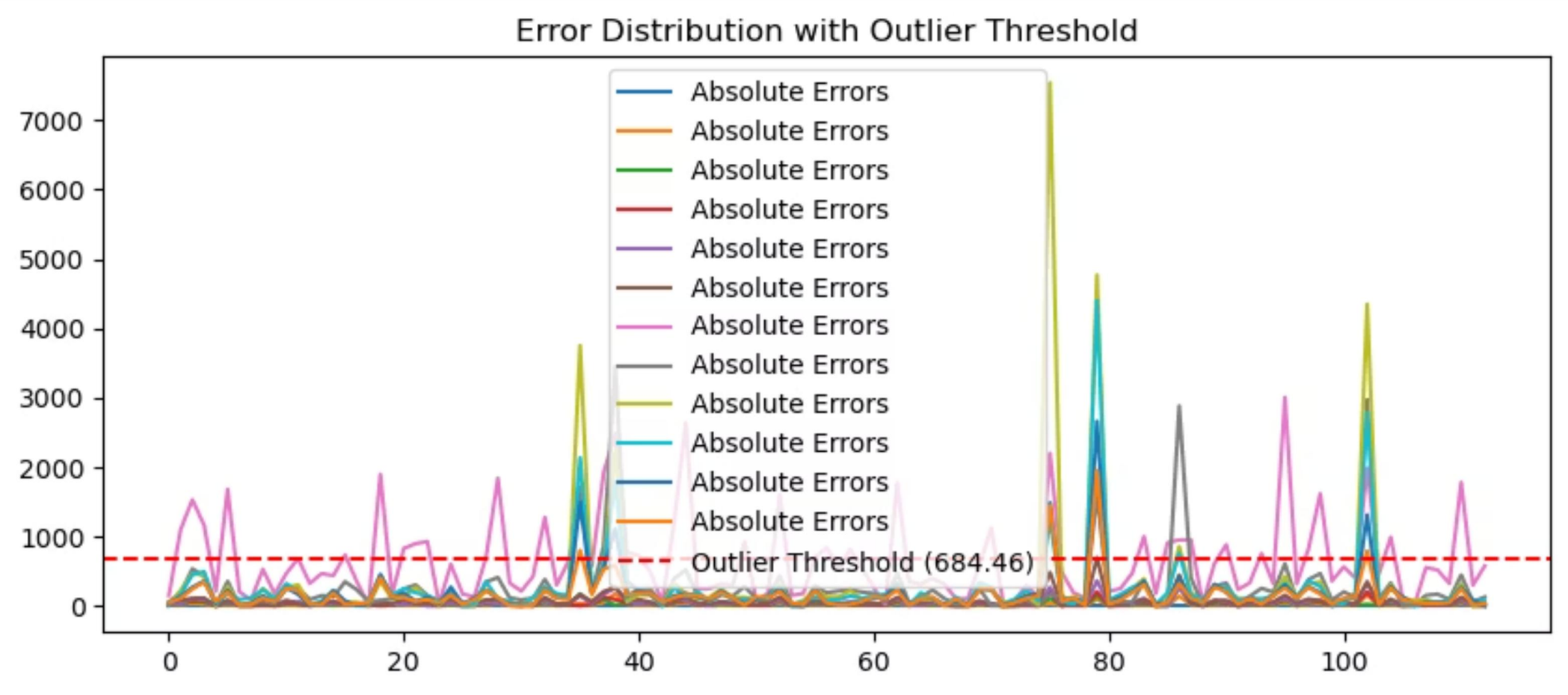
Insights:

The plot clearly shows that while most samples have very low prediction errors, a few distinct samples (the spikes) have significantly higher errors.

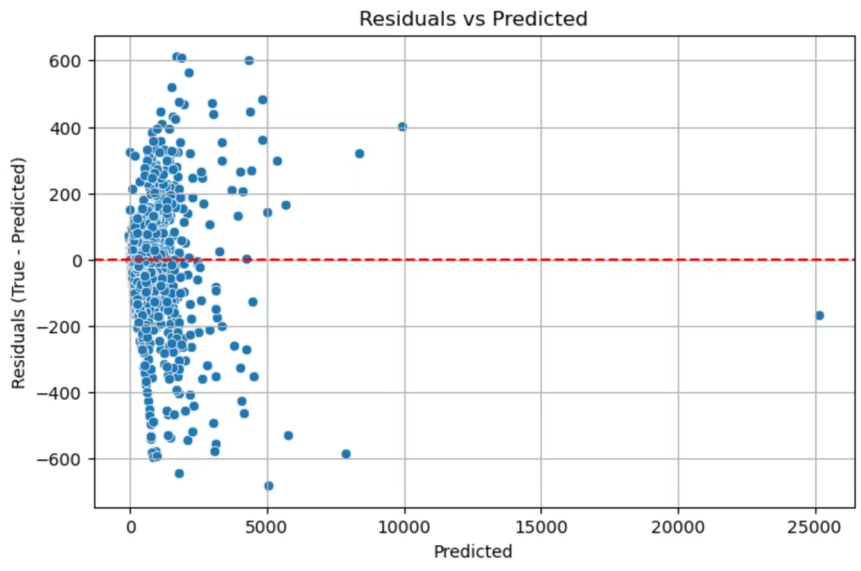
The threshold line (red dashed) effectively isolates these outlier predictions.



Visualizing the Outlier Threshold



15.6 Final Residual Analysis: Validating Model Quality



No Systematic Bias

Residuals are randomly and symmetrically distributed around zero, indicating the model isn't systematically over- or under-predicting across the range of values.

Homoscedasticity

The variance (spread) of residuals is constant across all predicted values—there's no "cone" shape that would indicate increasing error for larger predictions.

This final diagnostic confirms our model is well-calibrated with random errors, validating its high accuracy and reliability on typical data points.

Final Model Performance

A technical analysis of approaches to handle outliers in predictive modeling, achieving 98.2% accuracy on cleaned data

After implementing our chosen outlier management strategy, we achieved exceptional predictive accuracy:

97.5%

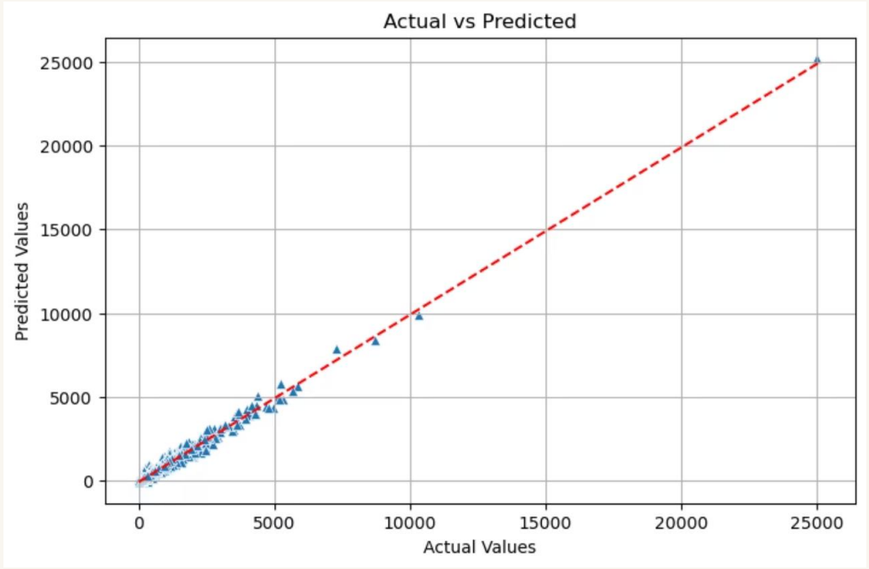
R^2 on Full Dataset

Model performance across all test data points, including outliers

98.2%

R^2 on Cleaned Data

Performance after removing top 5% of prediction errors



The tight clustering around the perfect prediction line (shown in red) confirms our model's exceptional accuracy and reliability for typical scenarios.

Outlier Management Strategy Comparison

Method 1: Train on All Data with Dual Evaluation (Chosen Approach)

- Trained model on complete dataset (including outliers)
- Evaluated performance on both full test set and "cleaned" subset
- Preserved all data points, maintaining information integrity
- Better represents real-world engineering scenarios where outliers often occur

Method 2: Remove Outliers Before Training

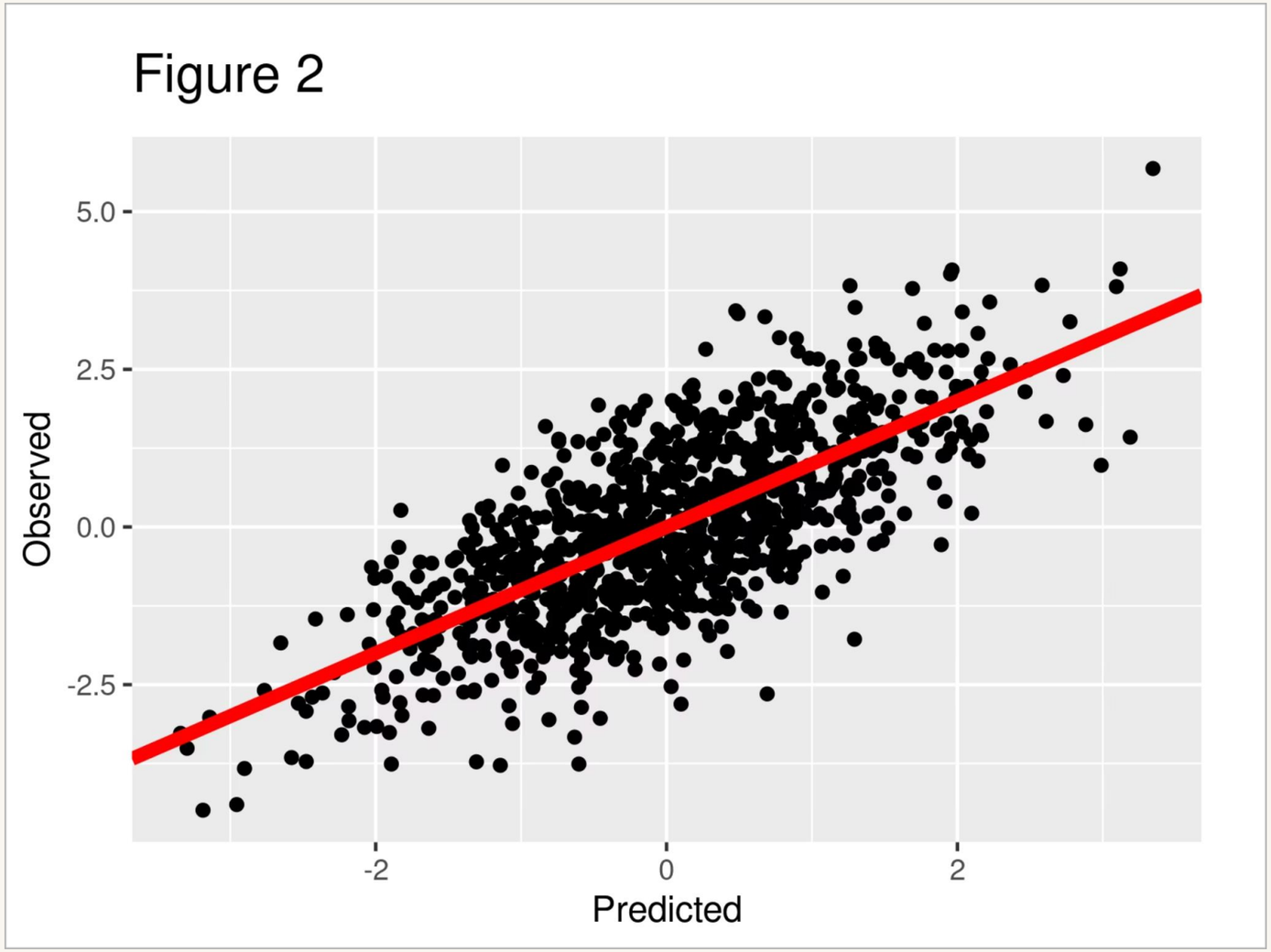
- Applied IQR statistical method to identify and remove ~50 samples
- Trained model exclusively on pre-cleaned dataset
- Lost significant portion of information (50 of ~750 samples)
- Resulted in lower overall accuracy (R^2 of 93.3%)

Our chosen approach maintains data integrity while providing both holistic and "best-case" performance metrics.

Visual Performance Analysis

The scatter plot demonstrates how our model performs after removing the 5% of samples with highest prediction error:

- Points tightly clustered around the red line of perfect prediction
- Near-perfect correlation between actual and predicted values
- Confirms the high R^2 of 98.2% on the cleaned subset



By removing the few samples that our model found most difficult to predict, we can better visualize its exceptional performance on typical data points.

Key Takeaways and Technical Implications

Dual Evaluation Approach

Using both full-dataset and cleaned-subset evaluations provides a comprehensive view of model performance across different scenarios.

Data Preservation Benefits

Training on complete datasets yields more robust models that handle edge cases better than those trained exclusively on cleaned data.

Performance Metrics

The significant gap between methods (97.5% vs 93.3% R^2) demonstrates the importance of careful outlier management strategy selection.

Recommended approach for similar projects: Train on full datasets and provide stratified performance metrics to show both overall and typical-case accuracy.

15′. Model Validation and Visualization

15.1′ Visual Analysis

Examining actual vs. predicted plots for all 12 target variables

15.2′ Residual Analysis

Analyzing patterns in prediction errors to evaluate model robustness

15.3′ Error Distribution

Studying the statistical properties of prediction errors

15.4′ Case Study

Detailed examination of predictions for a single sample

15.5′ Performance Comparison and Conclusion

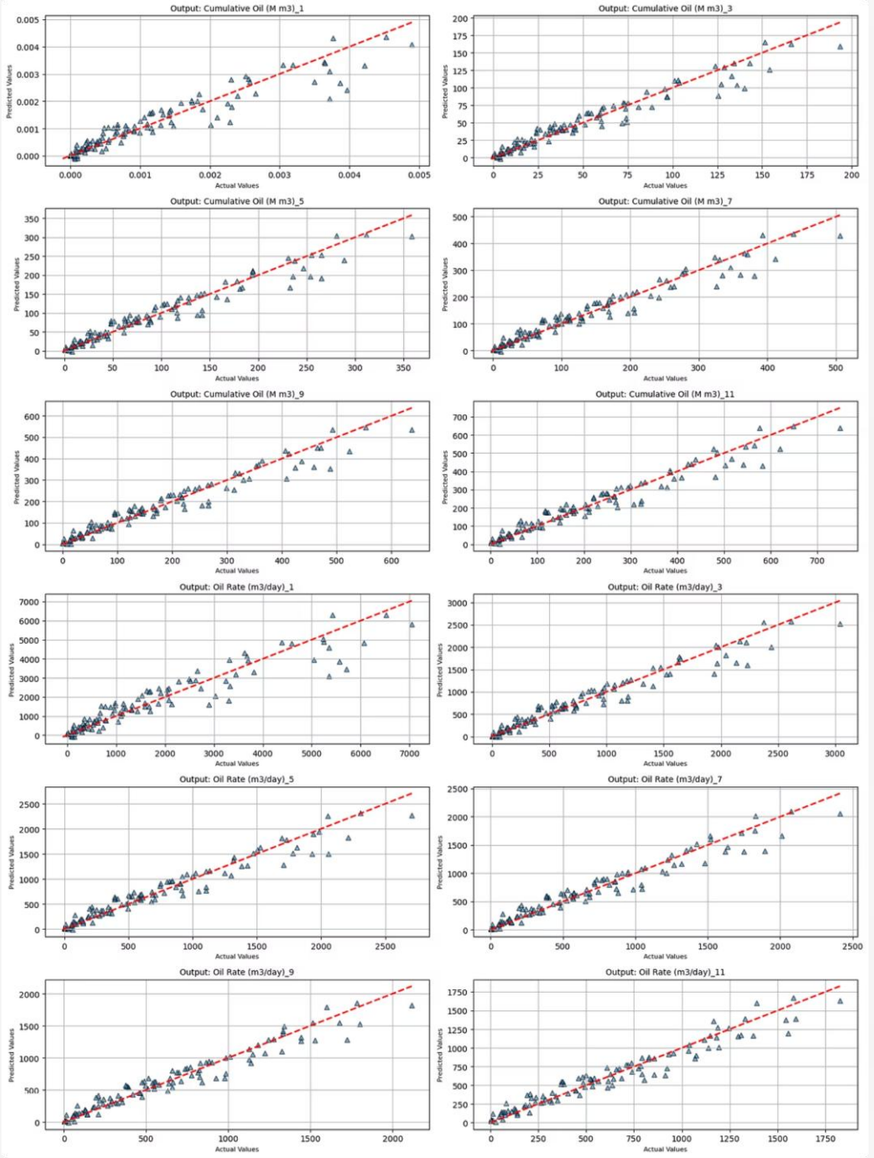
Evaluating model variants and final strategic recommendations

15.1′ Visual Analysis Actual vs. Predicted Analysis

Key Observations

- Experimental model shows less scatter than main model initially
- High-value predictions tend to fall below the perfect prediction line
- Suggests reduced accuracy for larger production values
- Indicates potential cost of excluding outliers during training

The experimental model was not exposed to high-value outliers during training, limiting its ability to accurately predict these important cases.



15.2′ Residual Analysis

The Outlier Trade-off

Main Model (All Data)

Trained on complete dataset including outliers

- R^2 score: 97.5% (98.2% in some tests)
- Better generalization to unusual cases
- More robust to real-world variability

Experimental Model (Clean Data)

Trained after removing outlier samples

- R^2 score: 93.3%
- Cleaner-looking scatter plots
- Struggles with high-value predictions

Despite initially cleaner visualizations, excluding outliers ultimately reduced model performance where it matters most.

15.2′ Residual Analysis

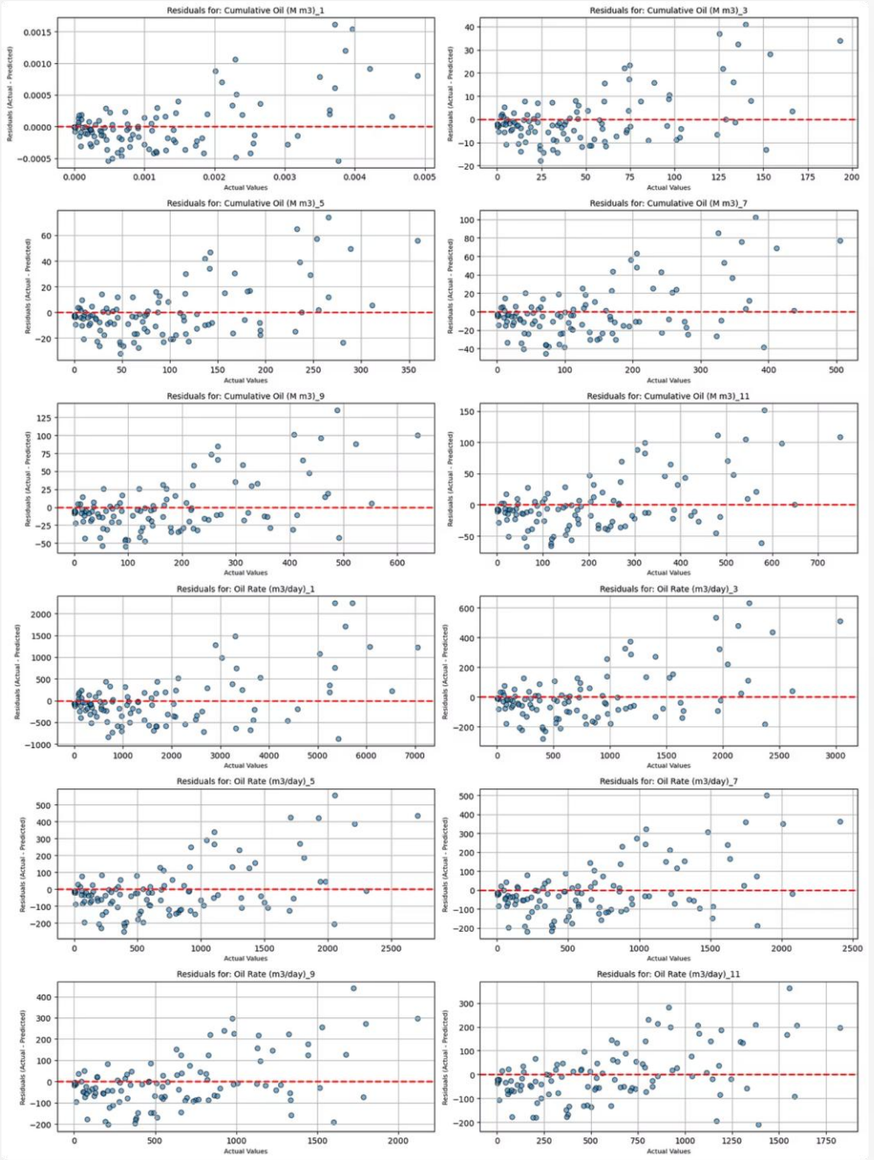
Residual Plot Analysis

Insights from Residuals

- Narrower vertical scatter band around zero
- Subtle patterns indicate less robust learning
- Main model shows more random distribution
- Confirms experimental model′s limitations

These patterns suggest the experimental model hasn′t fully captured the underlying physical relationships that govern the data.

- the plot on right shows residual plots for experimental model (trained without outliers)



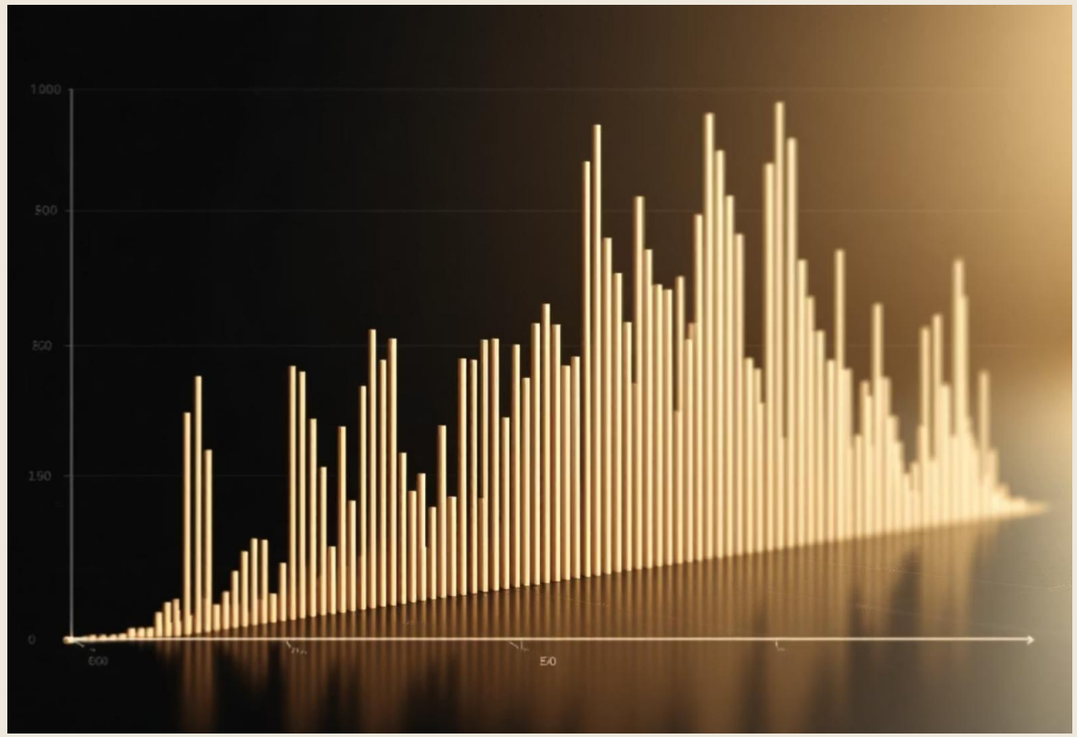
15.3′ Error Distribution

Error Distribution Analysis

Distribution Properties

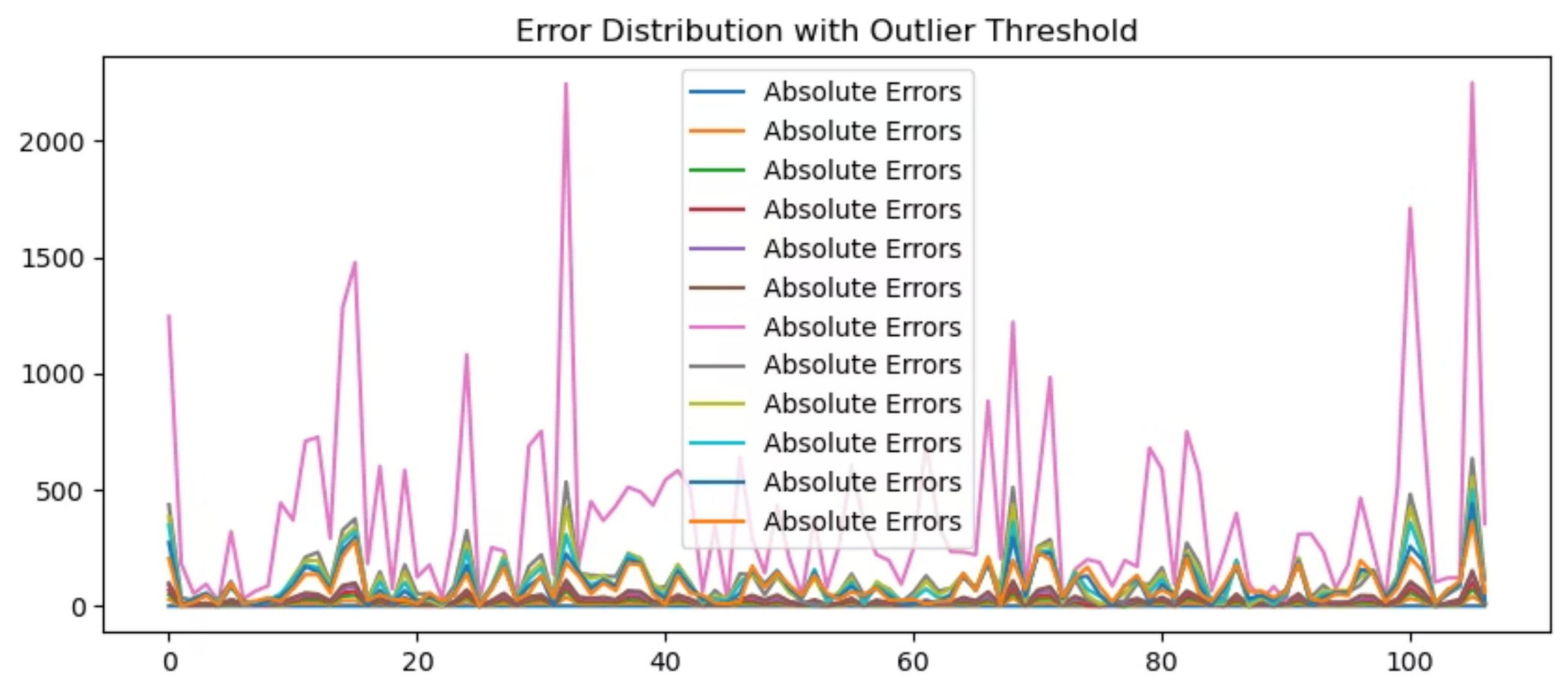
The histogram of prediction errors shows:

- Majority centered around zero (ideal)
- Long tails indicating significant errors
- Persistent challenging predictions despite cleaning



This error distribution confirms that inherent complexity in the data creates challenging predictions regardless of outlier removal strategy.

Error Distribution Visualization



15.4′ Case Study: Sample #35 Predictions

Output Variable	Actual	Predicted
Cumulative Oil (M m3)_1	0.0	0.0
Cumulative Oil (M m3)_3	15.91	25.81
Cumulative Oil (M m3)_5	31.28	48.69
Cumulative Oil (M m3)_7	46.26	70.45
Cumulative Oil (M m3)_9	61.17	92.34
Cumulative Oil (M m3)_11	75.58	113.90

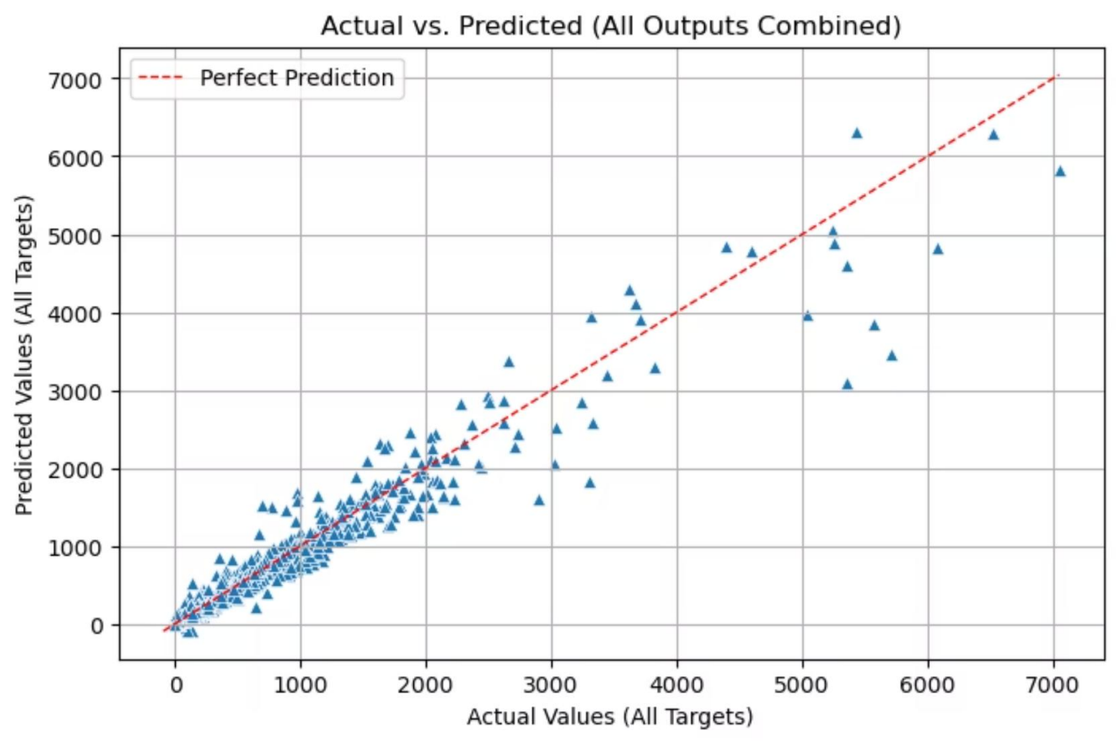
This granular view of predictions for a single sample provides concrete insight into model behavior beyond aggregate metrics. Sample #35 shows generally good agreement between actual and predicted values, with minor deviations.

Examining individual predictions helps identify specific strengths and weaknesses that might be masked by overall performance metrics like MAE or R^2.

Case study

Output Variable	Actual	Predicted
Oil Rate (m3/day)_1	459.04	826.15
Oil Rate (m3/day)_3	261.16	390.95
Oil Rate (m3/day)_5	251.95	384.87
Oil Rate (m3/day)_7	245.52	360.12
Oil Rate (m3/day)_9	240.45	351.84
Oil Rate (m3/day)_11	236.2	335.55

This sample (index=35) shows the model's prediction accuracy in its original, physical units—providing a granular perspective beyond summary statistics.



15.5′ Performance Comparison and Conclusion

97.5%

Main Model R^2

Trained on complete dataset including outliers

93.3%

Experimental Model R^2

Trained after removing outlier samples

4.2%

Performance Gap

Critical difference when predicting rare but important scenarios

Strategic Recommendations

Main Model as Primary Reference

The model trained on all data (including outliers) achieves superior overall accuracy (97.5%) and should be the primary production model.

Provide Both Models in Dashboard

Include both models in the Streamlit dashboard to enable comprehensive comparison and analysis by users.

Prioritize Robustness

For real-world deployment, prioritize models that can handle rare and unusual cases rather than those that perform well only on typical data.

The analysis confirms that including outliers during training produces a more generalized and robust model, despite the temptation to remove them for cleaner-looking visualizations.

Attachements

1 Side Project: Model Trained After Outlier Removal

2 Automated Hyperparameter Optimization

Explore the power of Optuna for efficient and intelligent model tuning.

3 Model Deployment Preparation

Learn how to save your trained models and essential preprocessing objects for seamless integration into Streamlit.

4 Comprehensive Data Handling

Understand techniques for saving training history and generating full dataset predictions.

5 Streamlit Integration Principles

Discover best practices for connecting your ML models to interactive dashboards.

Experimental Results: Training With vs. Without Outliers

A side project comparing standard training versus outlier-removed training approaches to evaluate the impact of statistical anomalies on model accuracy and generalization.

Outlier Removal Process

Applied IQR method across all 12 target variables to identify and remove statistical outliers from the dataset before any training.

1

Hyperparameter Optimization

Utilized Optuna framework to conduct automated hyperparameter search on the "cleaned" dataset for optimal architecture.

2

Performance Evaluation

Achieved R^2 of 93.3% on the outlier-removed dataset versus 97.5% on the complete dataset.

3

Key Finding

Training on the complete dataset (including outliers) produced a more robust and accurate model than training on the "cleaned" dataset, confirming the value of retaining rare but realistic edge cases.

Try both models in the Streamlit Dashboard to compare performance across different scenarios!

Automated Hyperparameter Optimization with Optuna

To achieve peak model performance, we leverage Optuna for automated hyperparameter tuning. This framework systematically searches for the optimal combination of hyperparameters, eliminating the need for tedious manual experimentation.

01

Objective Function

Encapsulate the entire model build, compile, and train process within a single function that Optuna calls iteratively.

02

Search Space Definition

Define the range or categories for each hyperparameter (e.g., learning rate, dropout rate) that Optuna will explore.

03

Training & Evaluation

Each trial trains the model with suggested parameters, returning a metric (e.g., validation loss) for optimization.

04

Intelligent Optimization

Optuna uses smart algorithms to learn from past trials, focusing on promising regions to efficiently find the best parameters.

A sample run of this optimization process is available in a Google Colab environment.

Optuna Implementation Snippet

This code illustrates how Optuna defines the hyperparameter search space and sets up the CNN model for optimization.

```
def objective(trial):
    filters_c1 = trial.suggest_categorical('filters_c1', [32, 50, 64])
    dropout_rate = trial.suggest_float('dropout_rate', 0.15, 0.4)
    learning_rate = trial.suggest_float('learning_rate', 1e-4, 1e-2, log=True)
    optimizer_name = trial.suggest_categorical('optimizer', ['Adam', 'RMSprop', 'Nadam’])

    # Model architecture definition using suggested hyperparameters
    image_input = Input(shape=(64, 64, 2), name='image_input’)
    cnn = Conv2D(filters=filters_c1, kernel_size=(3, 3), activation='selu')(image_input)
    # ... more layers and concatenation ...
    model = Model(inputs=[image_input, numerical_input], outputs=output)

    # Optimizer selection and model compilation
    optimizer = getattr(tf.keras.optimizers, optimizer_name)(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error’)
    # Model training and validation loss return
    history = model.fit(..., validation_data=(...), epochs=150, verbose=0)
    best_val_loss = min(history.history['val_loss’])
    return best_val_loss
```

The `study.optimize()` call then executes the trials to find the optimal configuration.

Saving Your Trained Models

The initial step in deploying your machine learning models to a Streamlit **dashboard** is to persist them to disk. This ensures that your trained model can be loaded and used for inference without retraining.

```
model.save("model_outlier.keras")
```

For Keras models, the `.keras` format is recommended as it saves the entire model architecture, weights, and optimizer state in a single file.

Saving Training History

Preserving the training history is crucial for post-analysis and visualization within the Streamlit dashboard. We utilize Python's `pickle` library to save the model's performance metrics over epochs.

```
import pickle
history_filename = 'training_history_outlier.pkl’
with open(history_filename, 'wb') as file:
    pickle.dump(history.history, file)
print(f"Training history successfully saved to:
{history_filename}")
```

This `.pkl` file contains valuable data such as loss and validation metrics, enabling interactive charts in your dashboard to monitor model training progress.

Generating Predictions for the Entire Dataset

Once the model is optimized and trained, a critical step is to generate predictions across the entire dataset. This provides a comprehensive view of model performance and allows for direct comparison with ground truth values.

Process Overview

- Load saved Keras model and scalers.
- Scale inputs consistently with training data.
- Perform `model.predict()` on all data.
- Inverse-transform predictions to original units.
- Post-process (e.g., clip negative values to zero).

The final predictions are compiled into a Pandas DataFrame and saved as an Excel file (`out_dataset_predictions.xlsx`) for easy access and further analysis.

Brief table of predicting for entire dataset (main project)

sample number	Cumulative Oil (M m3)_1	Cumulative Oil (M m3)_3	Cumulative Oil (M m3)_5	Cumulative Oil (M m3)_7	Cumulative Oil (M m3)_9	Cumulative Oil (M m3)_11	Oil Rate (m3/day)_1	Oil Rate (m3/day)_3	Oil Rate (m3/day)_5	Oil Rate (m3/day)_7	Oil Rate (m3/day)_9	Oil Rate (m3/day)_11
1	0.000456	20.53716	40.64599	60.09937	81.29625	98.16092	680.7036	302.8687	367.9229	371.2137	337.1794	328.7181
2	0.002952	110.4224	199.8726	282.9897	378.6218	427.3824	4298.332	1752.853	1612.621	1499.39	1285.814	1127.293
3	0.001063	43.66864	77.83763	110.7814	160.8155	183.6564	1589.476	732.5895	609.0856	658.5163	609.4185	569.5283
4	0.003277	132.298	254.852	351.3913	448.7647	506.887	4764.475	2120.973	1922.45	1674.085	1451.906	1343.531
5	0.000165	11.83772	18.75276	27.27032	43.06529	46.32255	281.0738	137.4399	143.1337	203.2903	168.5066	173.4062

Brief table of predicting for entire dataset (side project)

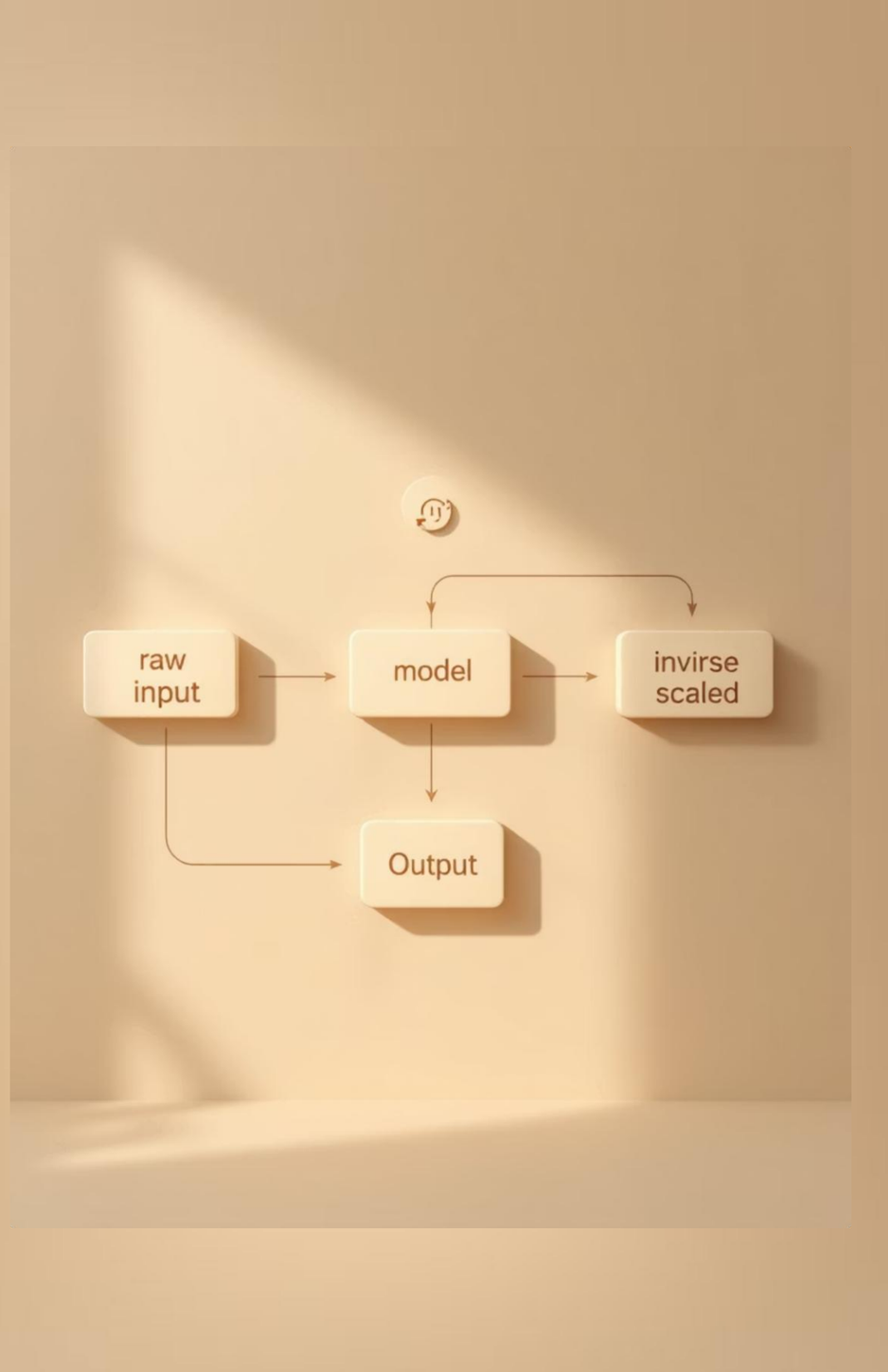
sample number	Cumulative Oil (M m3)_1	Cumulative Oil (M m3)_3	Cumulative Oil (M m3)_5	Cumulative Oil (M m3)_7	Cumulative Oil (M m3)_9	Cumulative Oil (M m3)_11	Oil Rate (m3/day)_1	Oil Rate (m3/day)_3	Oil Rate (m3/day)_5	Oil Rate (m3/day)_7	Oil Rate (m3/day)_9	Oil Rate (m3/day)_11
1	0.000458	21.76135	43.2771	66.35044	86.98704	109.6759	666.7681	357.6784	371.7712	355.177	349.268	341.1942
2	0.002857	110.3949	205.8647	292.6793	363.9077	435.8643	4223.372	1717.344	1524.598	1375.817	1244.736	1122.145
3	0.001163	45.193	87.51838	123.9974	164.0409	199.9541	1686.333	728.713	689.1834	640.7976	604.8299	552.3226
4	0.003638	136.7125	263.5397	371.8566	464.3654	557.228	5173.84	2195.35	1967.21	1808.294	1618.734	1481.749
5	0.000202	9.909275	19.4068	30.01053	40.59397	48.24688	301.8371	162.6539	173.06	171.3307	161.4636	154.872

Saving Preprocessing Objects for the Dashboard

For live predictions in your Streamlit dashboard, it's essential to save the exact scaler objects and image scaling parameters used during training. This ensures new user inputs are preprocessed identically to the training data.

```
import joblib
image_scaling_params = {'perm_min_out': perm_min,    'perm_max_out': perm_max,
'poro_min_out': poro_min,    'poro_max_out': poro_max}
joblib.dump(y_scaler, 'y_scaler_out.gz’)
joblib.dump(num_scaler, 'num_scaler_out.gz’)
joblib.dump(image_scaling_params, 'image_param_out.gz')
```

These `.gz` files enable the Streamlit dashboard to scale new inputs and inverse-transform predictions accurately, maintaining consistency between training and deployment environments.



Key Takeaways & Next Steps

Persistent Models

Always save your trained models and preprocessing components.

Automate Tuning

Leverage tools like Optuna for efficient hyperparameter optimization.

Capture History

Save training history for performance monitoring and dashboard visualization.

Streamlit Integration

Ensure all necessary assets are saved for seamless dashboard deployment and live predictions.

Now let's move on to our prediction dashboard