Short-Term Electricity Demand Forecasting for Hungary Using Hybrid Deep Learning Models

Emre Aydoğmuş
AI & Data Engineering
Istanbul Technical University
Email: aydogmuse22@itu.edu.tr

Baturalp Taha Yılmaz
AI & Data Engineering
Istanbul Technical University
Email: yilmazbat22@itu.edu.tr

***Abstract*—This paper presents our implementation and comparison of various deep learning models for short-term electricity load forecasting using Hungarian grid data. We collected hourly load data from the ENTSO-E Transparency Platform and combined it with weather observations and calendar features. Our experiments include a baseline MLP, a 1D CNN, two literature-based hybrid architectures (TimesNet–TCN and Attention–CNN–LSTM), and a proposed CNN–TCN–GRU model whose hyperparameters we tuned using a Genetic Algorithm. We trained all models on 2016–2017 data, validated on 2018, and tested on 2019 (pre-COVID period). The GA-optimized model achieved the best test MAPE of 1.80%, outperforming both baselines and the literature-inspired models. We also discuss challenges we encountered with peak demand prediction and the limitations of our approach.**

I. Introduction

Electricity demand forecasting is crucial for grid operators since it helps with scheduling power plants, managing reserves, and planning maintenance. We got interested in this topic after reading about how machine learning is being used in the energy sector. For our deep learning project, we decided to focus on Hungary because it's a relatively small country with a stable grid, which makes it easier to analyze.

Short-term load forecasting (STLF) typically means predicting demand for the next few hours to a week. In our case, we're doing one-step-ahead forecasting with a 24-hour lookback window. We picked this setup after experimenting with different window sizes—24 hours seemed like a good balance between capturing daily patterns and keeping the model manageable.

The main contributions of our work are:

- A complete data pipeline that fetches load data from ENTSO-E, weather data from Open-Meteo, and generates calendar features for Hungary.
- Implementation of five different model architectures, from a simple MLP to hybrid deep learning models.
- A Genetic Algorithm-based hyperparameter search for our proposed CNN–TCN–GRU architecture.

II. Related Work

We looked at several papers while designing our models. Zuo et al. [1] proposed combining TimesNet blocks with Temporal Convolutional Networks (TCN), which inspired our TimesNet–TCN implementation. Their idea was to use the periodic structure-capturing ability of TimesNet together with TCN's dilated convolutions for longer-range dependencies.

Quansah and Tenkorang [2] used an attention-augmented CNN-LSTM architecture optimized with Particle Swarm Optimization (PSO). We liked their use of attention mechanisms but decided to use GA instead of PSO for hyperparameter tuning. GA felt more natural for the discrete choices we had (like layer sizes and kernel widths), though we can't claim one is definitively better than the other.

We also looked at standard ARIMA and regression approaches, but these seemed too limited for capturing the nonlinear patterns in electricity demand. That's why we focused entirely on neural network-based methods.

III. Dataset and Feature Engineering

A. Data Collection

We built our own data collection pipeline since we couldn't find a ready-made dataset that had everything we needed. The main components are:

Load Data: We used the ENTSO-E Transparency Platform API to download electricity consumption data for Hungary. The API returns 15-minute resolution data, which we aggregated to hourly. The raw dataset covers 2015–2024 with about 350,000 data points at 15-minute intervals. We had to handle some missing timestamps using linear interpolation.

Weather Data: We fetched hourly temperature data from the Open-Meteo Archive API for Hungary's five largest cities (Budapest, Debrecen, Szeged, Miskolc, Pécs). We then computed a population-weighted average temperature since Budapest dominates with about 1.75 million people. From this, we calculated Heating Degree Days (HDD) and Cooling Degree Days (CDD) using thresholds of 15°C and 22°C respectively.

Calendar Features: We generated various time-based features including:

- Cyclical encodings using sine/cosine for hour, day of week, month, and day of year
- Binary flags for weekends, Hungarian holidays, school holidays, work hours, peak hours, etc.
- One-hot encodings for day type (weekday, weekend, holiday) and season

We used the `holidays` Python library to get Hungarian public holidays, though we had to manually add school holiday periods based on typical Hungarian school calendars.

B. Feature Engineering

For time series features, we created:

- Lag features at 48h, 72h, 96h, 120h, 144h, and 168h (capturing weekly patterns)
- Rolling statistics (mean, std, min, max) for the same window sizes

We were careful to avoid data leakage by only using past values. The rolling statistics use a shift of 1 before computing, so the current value isn't included.

After merging everything, we ended up with 66 features per timestamp. Table I shows the breakdown.

TABLE I
FEATURE CATEGORIES

Category	Count
Lag features	6
Rolling statistics	24
Cyclical time encodings	10
Calendar binary flags	16
Calendar one-hot	8
Weather (temp, HDD, CDD)	3
Total	**67**

C. Data Split

We used a chronological split to prevent any look-ahead bias:

- **Training:** 2016–2017 (roughly 125,000 samples)
- **Validation:** 2018 (about 27,000 samples)
- **Test:** 2019 (about 27,000 samples)

We specifically excluded 2020 and later to avoid the demand pattern changes caused by COVID-19 lockdowns.

IV. Model Architectures

A. Baseline: Tabular MLP

Our simplest model is a feedforward neural network that takes the features at time t and predicts load at $t+1$. It doesn't use any sequential structure—just two hidden layers (64 and 32 units) with ELU activations, dropout (11.5%), and L1/L2 regularization. We used AdamW optimizer with a learning rate of 0.0001.

This model achieved a test MAPE of 2.41%, which is decent but clearly shows that there's room for improvement with sequence-aware models.

B. Baseline: 1D CNN

The 1D CNN takes a sliding window of 24 hours as input and applies convolutional layers along the time axis. We used two Conv1D layers with 64 and 128 filters, followed by global average pooling and a dense output layer. The key advantage is that it can capture local patterns like daily cycles.

C. TimesNet–TCN Hybrid

Based on Zuo et al.'s work, this model combines:

- An input convolution to project features to a hidden dimension
- Three residual TCN blocks with dilations of 1, 2, and 4 to capture different temporal scales
- Global average pooling and a feedforward output layer

Each TCN block uses two 1D convolutions with batch normalization and residual connections. We found that the dilated convolutions really help with capturing weekly patterns.

D. Attention–CNN–LSTM

Inspired by Quansah and Tenkorang, this architecture has:

- A 1D CNN frontend for local feature extraction
- An LSTM layer (64 hidden units) for sequential modeling
- A temporal attention layer that computes a weighted average of LSTM outputs

The attention mechanism learns to focus on the most relevant time steps, which seemed to help especially around irregular events like holidays.

E. Proposed: GA-Optimized CNN–TCN–GRU

Our main contribution is a hybrid model that combines:

- A CNN layer for initial feature extraction
- Stacked residual TCN blocks for multi-scale temporal patterns
- A GRU layer for sequential dependencies
- Multi-head temporal attention for aggregation
- A feature fusion gate that learns to modulate the final representation

Instead of manually tuning the architecture, we used a Genetic Algorithm to search over the hyperparameter space. This is the part we spent the most time on, so we describe it in detail in the next section.

V. Genetic Algorithm Optimization

A. Search Space

We defined a search space covering the main architectural choices:

TABLE II
GA SEARCH SPACE

Hyperparameter	Options
CNN channels	16, 32, 64
TCN channels	32, 64, 128
GRU hidden size	32, 64, 128
Attention heads	2, 4, 8
Dropout rate	0.1, 0.2, 0.3
Kernel size	3, 5
Layer normalization	True, False

One tricky thing was making sure the GRU hidden size is divisible by the number of attention heads (required for multi-head attention). We handled this by automatically adjusting invalid configurations to the nearest valid option.

B. GA Implementation

We implemented a basic GA with:

- Population size: 8
- Generations: 24
- Elite preservation: Top 2 individuals
- Tournament selection with $k = 3$
- Uniform crossover
- Mutation rate: 20% per gene

To keep the search tractable, we evaluated fitness using only the last 30% of the training data (about 38,000 samples) and trained each candidate for just 8 epochs with early stopping. The fitness metric was validation MAPE.

The whole GA search took about 50 minutes on a Tesla T4 GPU (we used Google Colab for all our experiments).

C. Best Configuration Found

After 24 generations, the best configuration was:

- CNN channels: 32
- TCN channels: 128
- GRU hidden: 128
- Attention heads: 8
- Dropout: 0.1
- Kernel size: 5
- Layer normalization: enabled

We then retrained this configuration on the full training set for 50 epochs with early stopping (patience=7).

VI. Experimental Setup

All models were implemented in PyTorch and trained on Google Colab with a Tesla T4 GPU. We used:

- Adam optimizer with learning rate 10^{-3}
- ReduceLROnPlateau scheduler (factor=0.5, patience=2-3)
- Early stopping based on validation MAE
- Gradient clipping at 1.0 for the hybrid models
- Batch size of 64
- StandardScaler normalization for both inputs and targets

For evaluation, we report:

- Mean Absolute Error (MAE) in MW
- Root Mean Squared Error (RMSE) in MW
- Mean Absolute Percentage Error (MAPE)

We always inverse-transform predictions before computing metrics to get results in actual MW units.

VII. Results

A. Model Comparison

Table III shows the test set performance for all models.

The sequence-based models consistently outperform the tabular MLP, confirming that capturing temporal dependencies matters for this task. The hybrid architectures (TimesNet–TCN and Attention–CNN–LSTM) perform similarly, both achieving around 1.9% MAPE.

Our GA-optimized model achieves the best results with 1.80% MAPE and 90.31 MW MAE. This is about a 25% improvement over the MLP baseline and a 5–6% improvement over the literature-inspired models.

TABLE III
Test Set Results (2019)

Model	MAE (MW)	RMSE (MW)	MAPE (%)
MLP Baseline	119.71	155.54	2.41
1D CNN	106.00	140.38	2.15
TimesNet–TCN	95.29	127.10	1.92
Attention–CNN–LSTM	95.52	131.53	1.90
Proposed (GA)	**90.31**	**122.04**	**1.80**

B. GA Convergence

The best validation MAPE started around 2.08% in generation 1 and steadily improved to about 1.88% by the final generation. We observed that the population mean also decreased over time, indicating that the GA was successfully exploring and exploiting good regions of the search space.

Interestingly, the final best architecture ended up being moderately sized—not the largest possible. The GA found that a larger TCN (128 channels) paired with moderate CNN (32 channels) worked better than maxing out all components.

C. Error Analysis

Looking at when our models make mistakes, we noticed a few patterns:

Holidays and Special Days: All models struggled more during holidays, especially around Christmas (December 24–26). The typical daily pattern gets disrupted, and even with calendar features, the models have trouble.

Extreme Peaks and Valleys: The models tend to smooth out extreme values. When actual demand has a sharp spike or drop, predictions are typically more moderate. This makes RMSE worse relative to MAE.

Weather Transitions: Sudden temperature changes (like cold fronts) cause prediction errors. The HDD/CDD features help, but they're computed daily and can't capture intra-day temperature swings.

The residual distribution is roughly centered at zero with a standard deviation around 100–120 MW, which seems reasonable given that Hungarian demand typically ranges from 4,000 to 7,000 MW.

VIII. Discussion

A. What Worked

- **Hybrid architectures:** Combining CNN, TCN, and recurrent components consistently outperformed single-approach models.
- **Attention mechanisms:** Both the temporal attention in the CNN–LSTM model and multi-head attention in our proposed model helped the networks focus on relevant parts of the input sequence.
- **GA optimization:** The automated search found a better configuration than we could through manual tuning, and it only took 50 minutes.
- **Rich feature engineering:** The lag and rolling features, especially at 168h (weekly), were important for capturing weekly patterns.

B. Limitations

- **Single country:** We only tested on Hungary. It would be interesting to see how these models transfer to other countries.
- **Pre-COVID data only:** We avoided 2020+ data, but eventually models need to handle pandemic-era patterns.
- **One-step-ahead only:** Multi-step forecasting might require different architectures.
- **Limited GA search:** With more compute, we could explore larger populations and more generations.

C. Challenges We Faced

The data collection part was harder than expected. The ENTSO-E API has rate limits and sometimes returns incomplete data, so we had to implement retry logic and gap-filling. Getting the timezones right (Hungary uses CET/CEST with daylight saving) was also tricky—we eventually decided to keep everything in UTC.

We initially tried different window sizes (12, 24, 48, 72 hours) but found that 24 hours gave a good trade-off between capturing daily cycles and keeping computation manageable. Longer windows didn't improve results much but significantly increased training time.

IX. Conclusion

We implemented and compared five neural network architectures for short-term electricity load forecasting on Hungarian data. Our main findings are:

1) Hybrid temporal architectures (combining CNN, TCN, and recurrent layers) outperform simpler baselines by about 20–25% in terms of MAPE.
2) Using a Genetic Algorithm for architecture search is practical and effective, finding a good configuration in under an hour on a free GPU.
3) Despite good overall performance (1.8% MAPE), all models struggle with extreme demand events and holidays.

For future work, we'd like to explore:

- Asymmetric loss functions that penalize under-prediction of peaks more heavily
- Multi-task learning for different time horizons
- Transfer learning across different countries/regions
- Incorporating real-time features like renewable generation forecasts

All our code and notebooks are available in zip file in the ninova.

Acknowledgment

During the development of the project, large language models were used as supportive tools when needed. The experiments were conducted using Google Colab's free GPU resources.

References

[1] C. Zuo, J. Wang, M. Liu, S. Deng, and Q. Wang, "An Ensemble Framework for Short-Term Load Forecasting Based on TimesNet and TCN," *Energies*, vol. 16, no. 14, 2023.

[2] P. K. Quansah and E. K. A. Tenkorang, "Short-Term Load Forecasting Using a Particle Swarm Optimized Multi-Head Attention-Augmented CNN-LSTM Model," *arXiv preprint arXiv:2309.03694*, 2023.