Climate Emulation Using Deep Learning

CSE 151B - Spring 2025

Team Introduction - Group 23

Ethan Flores
(etflores@ucsd.edu)
3rd Year
Data Science Major

Adrian Apsay
(adapsay@ucsd.edu)
3rd Year
Data Science Major

Ryan Izadshenas
(rizadshenas@ucsd.edu)
3rd Year
Computer Science Major

Andrew Yin
(anyin@ucsd.edu)
3rd Year
Data Science Major

Summary

Our model uses a U-Net architecture with CoordConv input and separate decoder heads to predict global temperature and precipitation under future emissions scenarios. We tested several architectural variants, loss functions, and data transformations, ultimately selecting a configuration that balanced stability and accuracy. Our final model achieved a final RMSE of 0.9800 on the private dataset on Kaggle.

- How did you solve the problem
    - Developed and trained a U-Net with CoordConv inputs and dual output decoders
- What have you learned
    - Architectural changes like CoordConv and separate decoder heads led to better performance and generalization than deeper models or attention-based enhancements (at least for our case)
    - Prioritizing training stability and task-specific design choices was more effective than increasing model complexity
    - Hyperparameter tuning and choosing the right loss function was especially important (something we first overlooked)

Encoder Decoder

Convolutional Neural Network, U-Nets, LSTMs

Training Stability

Dual Decoders

Methodology

Data Processing

Preprocessing Pipeline:

- Performed input normalization via Z-score scaling (seperate for inputs and outputs)
    - Experimented with other data transformation techniques (more on this later):
        - i.e. log-transformation on *pr* due to skewness
        - T (time) input frames to provide model architecture temporal context (ConvLSTM)
- Non-spatial inputs were broadcast to spatial grids
- Nan filtering was applied before converting to pytorch tensors

Dataset Splitting:

- Training: All months from ssp126, ssp585, and first 901 months of ssp370
- Validation: Final 120 months of ssp370
- Test: Final 360 months of ssp245

Goal: Ensure stable model training, fair evaluation, and compatibility with leaderboard metrics all while preserving scientific validity in climate forecasting across SSPs.

Deep Learning Model

Modeling Framework:

- **Input**: 5-channel climate variable grid (48 x 72) per month
- **Output**: 2-channel prediction grid for tas and pr
- **Model Type**: CNN with spatial structure preservations → further experiments (shown later) specifically entail the different models we’ve tried

Why CNNs?

- Climate data is spatially structured and CNNs can exploit this locality
- U-Net specifically enables both global context and fine spatial detail via skip connections
- Encoder-decoder format well-suited for translating input grids to output grids
- Specific rationale to model design explained in experiments

Goals:

- How to best capture spatial dependencies while reducing overfitting

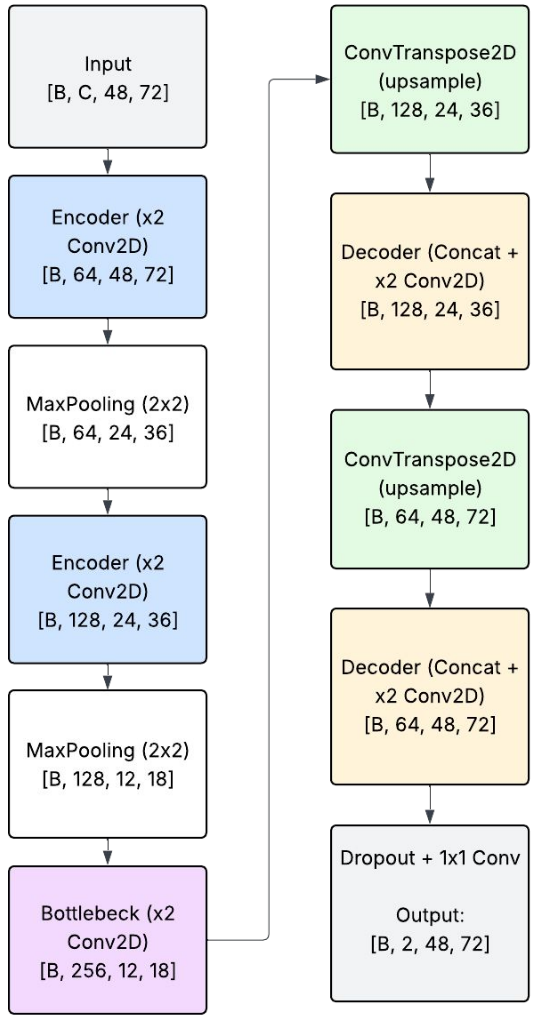
Experiments

Experiment 1 - Original UNet CNN

Model: U-NetCNN (Basic Version)

Rationale:

- **Encoder-Decoder Architecture [1]**
  - Used a two-level encoder-decoder structure (2 downsampling + 2 upsampling blocks) with skip connections and a bottleneck to extract multi-scale spatial features
    - Pooling = Downsample → Compress feature info
    - Transposed Conv = Upsample → Restore spatial resolution for dense prediction
- **Shared Output Head**
  - Produced both tas and pr predictions from a single output layer using a final 1×1 convolution
  - Final output is [B, 2, 48, 72] vector → predicting *tas* and *pr* jointly as two output channels over 48x72 spatial grid
- **Dropout Regularization**
  - Applied dropout before the final projection layer to reduce overfitting and improve generalization during training



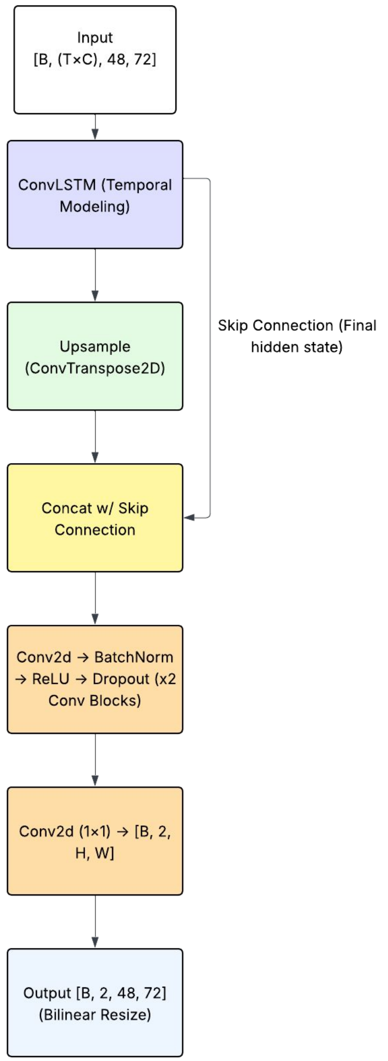
Experiment 2 - ConvLSTMNet

Model: ConvLSTM [2] combining ConvLSTM layers for temporal modeling and U-Net-style decoding for spatial refinement.

Rationale:

- **Spatiotemporal Architecture**
  - 2-layer ConvLSTM module to capture both spatial and temporal dependencies across input sequences
    - Pooling-free recurrent structure → learns temporal patterns across T input frames, represented as months (reshaped as [B, T, C, H, W])
    - Final hidden state from ConvLSTM is decoded with one U-Net-style upsampling block and skip connection
- **Temporal Encoding**: ConvLSTM hidden states evolve by integrating history over time
- **Skip Connections**: Final ConvLSTM hidden state is combined with temporal skip connection before decoding
- **Shared Output Head:** Predicts *tas* and *pr* jointly from decoded spatial features using a 1x1 convolution
  - Final output is [B, 2, 48, 72] using bilinear interpolation to match original spatial resolution → predicting *tas* and *pr* over 48x72 spatial grid

Findings: Excelled on SSP370 (validation set) by capturing sharp temporal trends but overfit to these, performing poorly on the smoother SSP245 scenario (leaderboard set). Scenario-aware conditioning or temporal attention could improve generalization.


Input [B, (T×C), 48, 72]
ConvLSTM (Temporal Modeling)
Upsample (ConvTranspose2D)
Skip Connection (Final hidden state)
Concat w/ Skip Connection
Conv2d → BatchNorm → ReLU → Dropout (x2 Conv Blocks)
Conv2d (1×1) → [B, 2, H, W]
Output [B, 2, 48, 72] (Bilinear Resize)

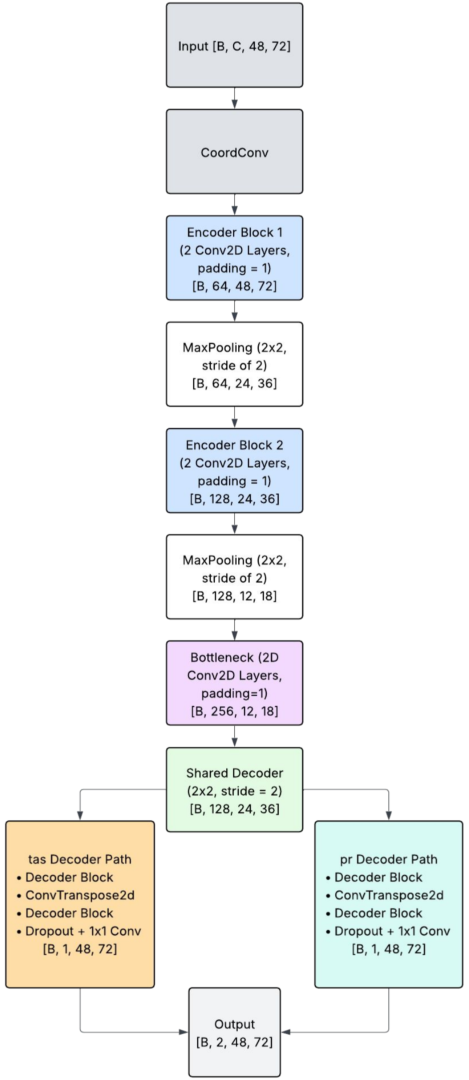
Experiment 3 - UNetCNN (Improved)

Model: Improved U-NetCNN with CoordConv and Dual Decoders

Rationale:

- **CoordConv Input [3]**
  - The input tensor is augmented with 2D coordinate channels (x, y) ranging from -1 to 1
  - Allows network to learn position-dependent patterns more effectively in geospatial climate data
- **Encoder–Decoder Backbone (U-Net)**
  - Standard U-Net-style architecture with downsampling (via max-pooling) and upsampling (via transposed convolutions)
  - Two levels of downsampling and two skip connections to preserve spatial detail
- **Dual Decoder Heads [4]**
  - Two decoder paths (for *tas* and *pr*) split after the bottleneck layer
  - Each path has its own set of decoder blocks and final 1×1 conv head
  - Prevents gradient interference between tasks; outputs concatenated along the channel dimension → [B, 2, 48, 72]

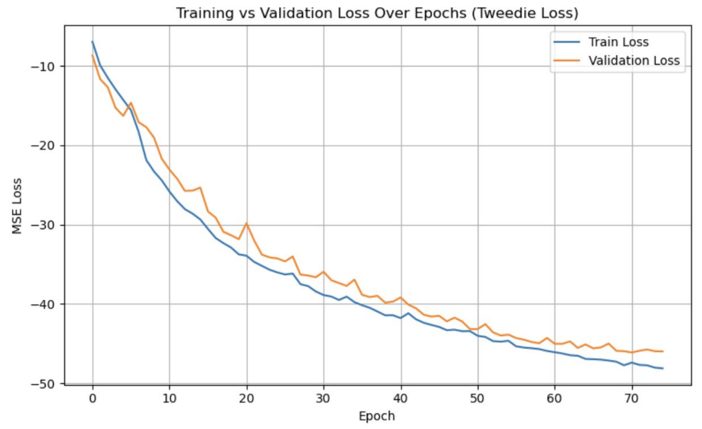
Findings: Improved performance on *pr* compared to shared-head models as it reduced task interference, more stable training, good generalization. Experimented with *pr* log-transform using this model, but led to worse performance. Experimented further with SE blocks and attention gates, but increased complexity → training instability and led to worse performance.

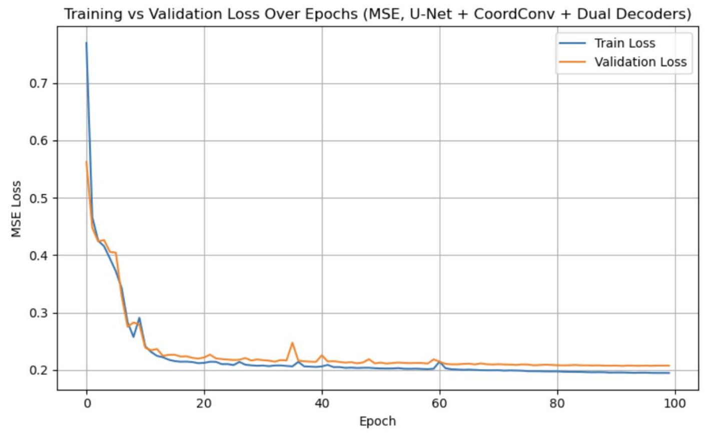


Input [B, C, 48, 72]
CoordConv
Encoder Block 1 (2 Conv2D Layers, padding = 1) [B, 64, 48, 72]
MaxPooling (2x2, stride of 2) [B, 64, 24, 36]
Encoder Block 2 (2 Conv2D Layers, padding = 1) [B, 128, 24, 36]
MaxPooling (2x2, stride of 2) [B, 128, 12, 18]
Bottleneck (2D Conv2D Layers, padding=1) [B, 256, 12, 18]
Shared Decoder (2x2, stride = 2) [B, 128, 24, 36]
tas Decoder Path • Decoder Block • ConvTranspose2d • Decoder Block • Dropout + 1x1 Conv [B, 1, 48, 72]
pr Decoder Path • Decoder Block • ConvTranspose2d • Decoder Block • Dropout + 1x1 Conv [B, 1, 48, 72]
Output [B, 2, 48, 72]

Training & Hyperparameter Tuning

- Epochs: Ranging from 50-200 epochs, usually on the lower end (with Early Stoppage)
- Batch Size: 64 to balance gradient stability and memory usage
- Learning Rate Scheduler: Used CosineAnnealingLR for smooth decay in the final phase
- EarlyStopping: Primarily monitored val/tas/rmse, with various patience
- Different Loss Functions, Data Transformations to handle *pr*
    - Log-Transform on *pr* due to skewness
    - Tweedie Loss, Smooth L1 Loss, Huber Loss to handle highly skewed data of *pr* as an alternative experiment
    - Utilized when architecture had dual decoder heads (assigning each variable its own output layer)
    - Plain MSE remained the most consistent
- Weighted Loss Tuning for Trade-Offs
    - weighted *pr* loss more over *tas* as that was the hardest task to minimize for our group during the project, but there weren't any improvements





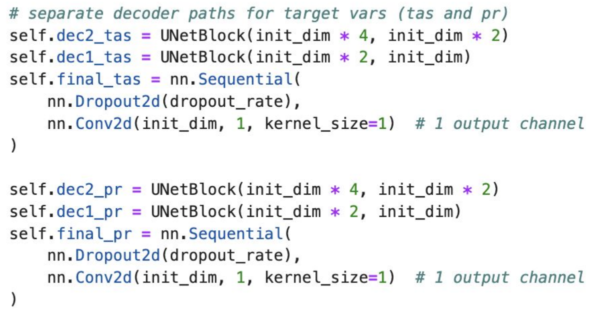
Discussion

Strength/Weakness/Improvements + Future Work

- **Strengths**
  - Using a separate decoder head for both tas and pr led to more better generalization and better training stability
  - Adding CoordConv input channels helped the model to improve spatial awareness
  - CosineAnnealing with early stopping maintained stable training and prevented overfitting onto the public Kaggle dataset
- **Weaknesses**
  - Key hyperparameters (e.g., depth, width, kernel size) were not systematically tuned
  - Our best model operates on single time steps, limiting temporal understanding
  - It lacks uncertainty quantification, which is valuable for volatile targets like precipitation
- **Improvements, Future Work**
  - Utilize / improve on temporal architectures like ConvLSTMs, Transformers, or other self-attention-based temporal architectures to capture patterns across sequential dependencies alongside spatial awareness
  - Experiment with entirely separate architectures + loss functions to predict *tas* and *pr*, respectively
  - Data augmentation with random shifts, noise, or scaling for improved generalization and robustness

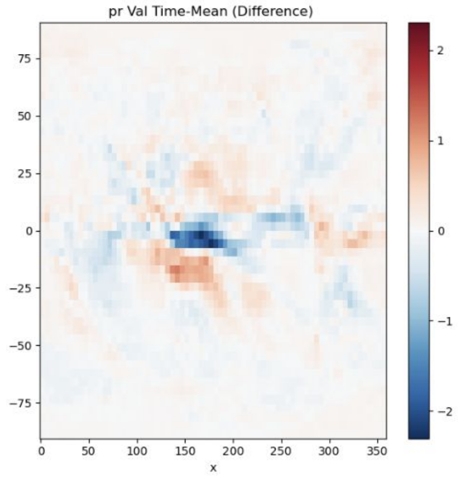
What We’ve Learned

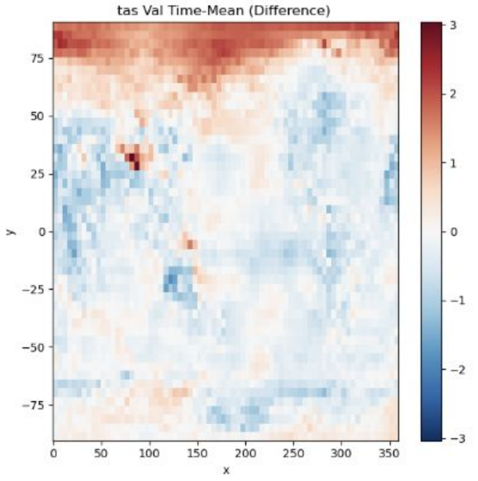
1. Simple architectures *could* work better than complex ones when well-matched to the task and well-tuned
2. Task separation significantly boosts stability → reduces gradient interference from the two tasks, allowing the model to independently learn from their patterns
3. Temporal trends are hard to generalize as it could overfit on training data focused on specific SSPs → scenario-aware embeddings or attention can be explored
4. Training stability and validation performance are often better goals than blindly increasing model depth or size
5. Hyperparameter Tuning and Loss-Function choice should NOT be overlooked by any means
6. Spatial and error visualization are *very* helpful



```
# separate decoder paths for target vars (tas and pr)
self.dec2_tas = UNetBlock(init_dim * 4, init_dim * 2)
self.dec1_tas = UNetBlock(init_dim * 2, init_dim)
self.final_tas = nn.Sequential(
    nn.Dropout2d(dropout_rate),
    nn.Conv2d(init_dim, 1, kernel_size=1)  # 1 output channel
)

self.dec2_pr = UNetBlock(init_dim * 4, init_dim * 2)
self.dec1_pr = UNetBlock(init_dim * 2, init_dim)
self.final_pr = nn.Sequential(
    nn.Dropout2d(dropout_rate),
    nn.Conv2d(init_dim, 1, kernel_size=1)  # 1 output channel
)
```





References

[1] B. Yu, H. Yin, and Z. Zhu, “ST-UNet: A spatio-temporal U-network for graph-structured time series modeling,” arXiv preprint arXiv:1903.05631, 2021. [Online]. Available: https://arxiv.org/abs/1903.05631

[2] D. Wang, Y. Yang and S. Ning, "DeepSTCL: A Deep Spatio-temporal ConvLSTM for Travel Demand Prediction," 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 2018, pp. 1-8, doi: 10.1109/IJCNN.2018.8489530.

[3] R. Liu, J. Lehman, P. Molino, F. Petroski Such, E. Frank, A. Sergeev, and J. Yosinski, “An intriguing failing of convolutional neural networks and the CoordConv solution,” in *Advances in Neural Information Processing Systems*, vol. 31, 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/60106888f8977b71e1f15db7bc9a88d1-Paper.pdf

[4] J. Lin and G. Michailidis, “A multi-task encoder–dual-decoder framework for mixed frequency data prediction,” *International Journal of Forecasting*, vol. 40, no. 3, pp. 942–957, 2024. [Online]. Available: https://doi.org/10.1016/j.ijforecast.2023.08.003