Top Architectures Overview

2. 1D CNN with GRU and Resnetblock (1DCNNGRU)

Architecture: A hybrid model architecture with a 1 dimentional convolutional network with resnet blocks and a GRU

Layer Type	Description
ConvBlocks 1D	1D Convolution: 128 filters, kernel_size=3, stride=1, padding=1; followed by ReLU activation and BatchNorm1d with 128 channels.
ResNetBlocks 1D	Residual Block: Two 1D Convolutions (128 filters, kernel_size=3, stride=1, padding=1), followed by ReLU activation and BatchNorm1d. Includes a skip connection.
MaxPool1d	Max Pooling: kernel_size=2, stride=2. Reduces the sequence length by half.
**Linear **	Linear Layer: Flattens the output of convolutions, transforming the features into a 128-dimensional vector.
GRU	GRU Layer: 128 hidden units, 5 layers, batch_first=True, dropout=0.2 for regularization, capturing sequential dependencies.
Dense Layers	GRU output (batch_size, 128) is flattened into a 1D vector (batch_size, 128).
	2 Fully connected Layers: 128 input features, 128 output features with ReLu Activation, 5 output features (final classification).

2. 2D Convolutional Neural Network (CNN) with ResNet block

The model is a 2 dimentional convolutional network with a Resnet block **Architecture**:

Layer Type	Details
Convolutions blocks	Conv2d 1 input channel, 128 output channels, kernel size 3x3, stride 1x1, padding 1x1, with ReLU activation and BatchNormalization
ResNetBlock2D	Conv2d, 128 input/output channels, kernel size 3x3, stride 1x1, padding 1x1
	BatchNorm2d 128 channels, followed by ReLU activation and Batch Normalization 2D
Projection	Identity layer
MaxPool2d	Kernel size 2x2, stride 2x2
Dense Layers	Flatten input tensor starting from the first dimension
	2 Linear layers with input flattened features, hidden size output/input followed by ReLU activation with final 5 output classes

HYPERTUNING SEARCHSPACE

The hyperparameters of the model were optimized using the following search space:

Hyperparameter CNNGRU	Search Space	Search Method
Hidden units GRU	[32, 64, 128, 256, 512]	Random Search
Batch Size	[16, 32, 48, 60]	Random Search
Number of Layers	[2, 3, 4]	Random Search
Number of blocks	[1 - 5]	Random Search
Dropout Rate	[0.2, 0.3, 0.4]	Random Search
Optimizer	['Adam', 'AdamW']	Random Search
Dataset	['smote', 'oversampled']	Random Search
Factor (ReduceLROnPlateau)	[0.1 - 0.4]	Random Search

Hyperparameter 2D CNN	Search Space	Search Method
Batch Size	[16, 32, 48]	Random Search
Hidden units	[62 - 256]	Random Search
Number of Layers	[2, 3, 4]	Random Search
Number of blocks	[1 - 7]	Random Search
Dropout Rate	[0.1 - 0.4]	Random Search
Factor (ReduceLROnPlateau)	[0.1 - 0.4]	Random Search

Best configurations for the two model (30 epochs):

	iterations	accuracy	recallmacro	batch	hidden	dropout	num_layers	num_blocks	factor	gru_hidden	trainfile

	iterations	accuracy	recallmacro	batch	hidden	dropout	num_layers	num_blocks	factor	gru_hidden	trainfile
1DCNNGRU	28	0.9858	0.9593	32	64	0.4	2	5	0.2	256	oversampled
2DCNN	30	0.9888	0.9744	16	128	0.3	3	1	0.2	--	oversampled

Hyperparameter tuning began with a manual exploration of parameter ranges, such the ranges of number of blocks and layers and **Bayesian Optimization** function in Ray HyperOptSearch from Ray Tune (Bayesian optimization) for hyperparameter optimization. HyperOptSearch searches space probabilistically and uses the observed performance of previous trials to make predictions about which hyperparameters are most likely to yield good results.

In the second stage of tuning, the process expanded to include not only the architectural parameters (e.g., number of layers, hidden sizes, and blocks) but also other important factors such as batch size, optimizer choice and scheduler types as parameter of the configuration. This holistic approach enabled more precise optimization, leading to improved training dynamics and model performance. The models have been trained with 30 and later to 40 epochs with early stopping en both on the smote and oversampled dataset.

RESULTS AND REFLECTIONS

Initial Hypothesis: The initial hypothesis posited that 1D models, specifically GRU and 1D CNN, would better fit the training set due to the sequential nature of the data. At the end, the hypothesis was confirmed, 1D models did perform well but a lot of work was done on the dataset. From the start the 1D models performance was hindered by the dataset's imbalance. Despite attempts to address this through the model architecture, the 1D models overfit to the majority class and perform bad on the recall metric. Therefore, I explored the possibilities of 2D models which seemed handle better on the imbalaced dataset. This brought me to explore more models than need but was nonetheless a useful exploration. Key learnings:

- CNN 2D: Results: The 2D CNN architecture emerged surprisingly as the top performer for this dataset even using a balanced dataset. It handled the semi-imbalanced data well, especially with the class weights applied, and delivered strong performance overall. It was surprising because the dataset is a time series which most of the time fits better to 1D CNNs, 2D CNN seemed like over the top. Reasoning: CNNs are particularly suited for spatial data and can learn hierarchical features in images or sequences and with the implementation of extra residual blocks the CNN was able to retain more information and avoid overfitting. This model was quicker to train, and deliverd the top results.
- 2D Transformer: Implementing 2D convolutions within the transformer architecture was a good idea and gave better results than 1D convolution. The 2D convolutions capture more detailed spatial features, which complements the transformer’s self-attention mechanism. Conclusion: 2D convolutions combined with transformer blocks seem to be a promising architecture for this task, however the training takes much longer to train and hypertune due to the extra parameters. Also the
- 1D CNN: Results: Both architectures suffered from overfitting to the majority class, especially after applying upsampling. The models essentially memorized the majority class patterns and struggled to learn features for the minority classes. The GRU was at first underperforming. I discovered that recurrent neural networks (RNNs), such as GRUs, may not be the best choice for imbalanced datasets. Their capacity to "remember" the majority class can lead to poor generalization, especially when faced with minority classes, but they are great with timeseries balanced datasets.

Balancing the Dataset: Upon balancing the dataset using oversampling techniques, 1D models still struggled. In fact, they underperformed relative to 2D CNN and Transformer models, which were surprisingly better. To enhance the performance of these models, I introduced ResNet blocks. The ResNet blocks enabled the models to focus on local features, improving generalization and helping them maintain consistent performance throughout the training. This modification improved the CNN’s ability to capture essential features, leading to better generalization.

Performance with SMOTE (Synthetic Data): When I switched to a synthetic dataset generated by SMOTE (Synthetic Minority Over-sampling Technique), the results shifted drastically. Surprisingly, the 1D models began to perform much better. This was particularly evident with the GRU, which achieved the best performance seen throughout the experiment. On the other hand, the 2D models (especially the Transformer) seemed to struggle, underperforming compared to when the real dataset was used. This suggests that Transformers may have been treating the synthetic data as noise, likely due to its inability to distinguish between real and synthetic data effectively.

Why 1D Models Excelled with SMOTE: Interestingly, 1D models (specifically the GRU) thrived with the synthetic dataset. The GRU not only outperformed previous results but also reached new heights in terms of model accuracy. It was particularly sensitive to the synthetic data, which may have allowed it to better capture underlying temporal patterns that were less apparent in the original, imbalanced dataset. Given that GRU models are good at modeling sequential relationships, the synthetic data might have provided clearer signal patterns, boosting performance.

Layer Normalization for GRU: While the GRU was achieving great performance, it was training slower than expected. To speed up the training process, I applied Layer Normalization. This adjustment allowed the GRU to converge faster, as it stabilized the learning process by normalizing the activations. This likely helped mitigate issues like vanishing gradients, leading to smoother and quicker convergence.

Other learnings and insights: I tried hypertuning optimizers, patience, batch and schedulers and found out that default values sometimes are often a good choice. Futher I learned that without a seed in the configuration it is impossible to reproduce the exact model that has been trained.

Connect your results to your hypotheses. What did you discover? What are new insights? What hypotheses were confirmed, and which were rejected?