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: kernel_size=3, stride=1, padding=1; followed by ReLU activation and BatchNorm1d
ResNetBlocks 1D	Residual Block: Two 1D Convolutions (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 flatten vector.
GRU	GRU Layer: input hidden units, input number of layers, batch_first=True, dropout for regularization
Dense Layers	GRU output (batch_size, hidden size) is flattened into a 1D vector (batch_size, hidden size). 2 Fully connected Layers: hidden size input features, hidden size 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	Description
Convolutions blocks	Conv2d hidden size input channel, hidden size output channels, kernel size 3x3, stride 1x1, padding 1x1, with ReLU activation and BatchNormalization
ResNetBlock2D	2 Conv2d layers, hidden size input/output channels, kernel size 3x3, stride 1x1, padding 1x1
	followed by ReLU activation and Batch Normalization 2D
MaxPool2d	Identity layer (preserve the input) followed by MaxPool 2D layer 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 (top 10 models):

Model	Batch Size	Hidden units GRU	Hidden units CNN	Number of Layers	Number of blocks	Dropout Rate	Factor (ReduceLROnPlateau)
1DCNNGRU	[16, 32, 48, 60]	[32, 64, 128, 256, 512]	[32, 64, 128, 256, 512]	[2, 3, 4]	[1 - 5]	[0.2, 0.3, 0.4]	[0.1 - 0.4]
2DCNN	[16, 32, 48]		[62 - 256]	[2, 3, 4]	[1 - 7]	[0.1 - 0.4]	[0.1 - 0.4]

Best configurations for the two model (30 epochs):

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

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 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 test other important factors such as batch size, optimizer choice and scheduler types as parameter of the configuration. Some parameters suche optimizer and sheduler are not included in the table above because default parameters from manual testing performed better. The models have been trained with 30 and later to 40 epochs with early stopping. Longer training did not improve accuracy on both on the smote and oversampled dataset.

RESULTS AND REFLECTIONS

Hypothesis: The initial hypothesis was 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 to come to this conclusion, and also on the dataset. On the other end, the hythesis that 2D models would overfit or be to complex for the timeserie dataset is not true. The 2D CNN is surprisingly the best performing model, is flexible enough to handle both a From the start the 1D models performance was hindered by the dataset's imbalance. Despite attempts to address this through changes in 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. More compex models were overfitting less because of their more complex architecture, but they did suffer from weight decay. This brought me to explore more models than need but was nonetheless a useful exploration. Key discoveries:

Balancing the Dataset: Blancing the dataset should have been the first step of the exploration. Oversampling techniques increase the risk of overfittig, 1D simple models struggled. In fact, they underperformed relative to 2D CNN and Transformer models, which were surprisingly better.

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.

Models

- GRU models can be slow and computationally expensive to train. Due to their recurrent nature they train in sequence however, they can perform very well when used in hybrid architectures, such as combining CNNs and GRUs. In these hybrid models, the CNN layers help efficiently extract features from the input data (e.g., images or time-series), and the GRU layers then capture the temporal dependencies or sequential patterns within the extracted features.
- GRU and unbalanced datasets: the GRU was at first underperforming, recurrent neural networks (RNNs), may not be the best choice for imbalanced datasets. Because of their capacity to "remember", they discard te minority or less represented class as 'noise' but they are great with timeseries balanced datasets in hybrid architectures.
- 1D CNN: it 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.
- 2D CNN: The CNN architecture emerged as the top performer for this dataset even using a balanced dataset with syntethic or oversampled data. It handled the semi-imbalanced data well and was quick to train. The 2D CNN 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. 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 seemed to be a promising architecture however it was too complex for this task. The downside that the training takes much longer and more computations.

Additional blocks

- Resnet blocks: CNNs and Transformers work really great with Resnet blocks. ResNet blocks improved performance by helping the model learn more complex features. 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 generalize.
- The Squeeze-and-Excitation (SE) block added significant value to the Transformer architectures enhancing its ability to focus on the most important features, but was not a significant addition to CNNs. In CNNs, the convolutional layers already focus on local patterns (e.g., edges, textures) in images or sequences. These patterns are often simple and don't necessarily need the same level of global feature recalibration that SE blocks provide.
- Transformers are less adaptable to syntethic data such as in a Smote dataset, they seem to detect somehow that the data

Useful finetuning options

- Dilated Convolutions: parameter dilation in sequence data helps to capture larger temporal context, increases the receptive field of the convolutional layer (by enlarging the kernel). The kernel is spaced out, effectively considering a larger portion of the input while keeping the kernel fixed. This allows the model to capture larger patterns or long-range dependencies in the data.
- MaxPooling vs Global Average Pooling (GAP):
- pack padded sequence: transform a padded batch of sequences into a packed sequence format to help RNNs process variable-length sequences efficiently by ignoring the padded values in the hidden state updates.

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?