Electrocardiogram Time-series Classifications using Deep Learning

Oct 2023

Presentation Outline

- Context
- Pipeline Architecture
- Data
- Results
- Conclusions
- References
- Appendix

Context

Introduction: Significance of Arrhythmia Detection from ECGs

- **Foundation of Cardiology**: Electrocardiograms (ECGs) is a crucial diagnostic tool with an estimated 300 million recordings annually.
- **Core Challenge**: Detecting irregular heart rhythms, or arrhythmias, vital for diagnosing diseases like Myocardial Infarction, AV Block, Ventricular Tachycardia, and Atrial Fibrillation.
- **Pain Points**:
  - Computer predictions for non-sinus rhythms historically only about 50% accurate.
  - High variability in wave morphology across patients.
  - Imbalanced data.
- **Need for Advanced Solutions**: Given the life-altering implications of misdiagnoses, there is an acute need for sophisticated automated detection methods. Deep Learning (DL) offers a promising solution to address these challenges.

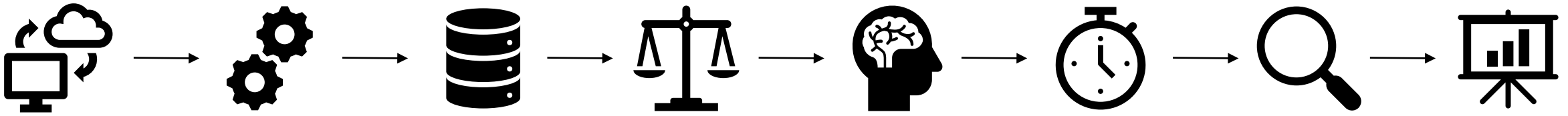
Deep Learning in ECG Analysis: Enhancements and Methodology

- **Literature Insights**:
  - Comprehensive studies on ECG and DL have showcased improvements in ECG diagnoses.
- **Advantages of DL in ECG Analysis**:
  - Deep Learning algorithms can automatically extract vital information from the variable morphologies present in ECG signals.
- **Our Approach**:
  - In this project, we will investigate three distinct oversampling techniques to address data imbalances in ECG datasets.

Pipeline Architecture

High-level Overview of the Deep Learning (DL) pipeline

- **Purpose**: Give a 10,000-foot view of the pipeline.



- The pipeline begins with data ingestion, ensuring data is collected and stored effectively.
- This data is then loaded, pre-processed, and resampled to address imbalances.
- Then, it enters the machine learning phases of building, training, and evaluating models.
- Finally, the results are analysed and visualised to draw actionable insights.

Module 1: Arguments & Configuration

- **Purpose**: The module handles argument parsing and configuration for machine learning processes.

- **Key Imports**: Various standard libraries like `argparse`, `glob`, and `os` are used. Constants and default values are imported from a module named `modules.default` which is commonly imported across the remaining modules.

- **Main Class** - ArgumentManager:
  - This class manages and constructs essential file paths.
  - It handles the parsing of command-line arguments, scans for datasets, creates subdirectories for models, and offers an organised dictionary structure.

Module 2: Data Manager

- **Purpose**: The module manages datasets for loading, padding, splitting, and provides access to train, validation, and test sets.

- **Key Imports**: `numpy`, `tensorflow`, `train_test_split` (from `sklearn.model_selection`).

- **Main Class** - DatasetManager:
  - Manages datasets for tasks like loading, padding, and splitting.
  - Provides properties for accessing padded training, validation, and test data.
  - Implements a lazy-loading mechanism for data, where datasets (training, validation, and test) are loaded and processed only when accessed.
  - To optimise memory usage, the class provides methods to clear out loaded data once they are no longer in use.

Module 3: Resampling

- **Purpose**: Provides functionality to oversample multi-class datasets using various techniques. It is akin to giving a microphone to underrepresented voices, ensuring that each class is adequately represented.

- **Key Imports**: `numpy`, `tensorflow`, `resample` (from `sklearn.utils`), `SMOTE`, `ADASYN` (from `imblearn.over_sampling`).

- **Main Class** - MultiClassOversampler:
  - Enables oversampling of multi-class datasets.
  - Provides methods for simple oversampling, orchestrating various oversampling methods, and unloading the resampled data.
  - Clears the resampled training data and labels. This is particularly useful when the datasets are large or the instance is no longer needed.

Module 4: Model Builder

- **Purpose**: Constructs and manages two model architectures primarily for classification tasks, incorporating individual class-wise F1 score evaluations.

- **Key Imports**: `tensorflow`, `backend` (from `tensorflow.python.keras`), `os`, and `typing` (`Callable`).

- **Main Class** - ModelArchitectureBuilder:
  - **Baseline Neural Network**: A simple feed-forward neural network with dense layers *(please refer to Appendix)*.
  - **Deep CNN with Residual Blocks**:
    - A convolutional neural network (CNN) with an initial block, followed by a series of residual blocks (both type 1 and type 2), and a final classification block. *(please refer to Appendix)*
    - This model's architecture is inspired by the work of Rajpurkar et al. (2017) [1] on arrhythmia detection with CNNs.
  - *F1 scores*: The class provides mechanisms to compute individual F1 scores for each class. This feature is crucial for multi-class scenarios, offering a detailed performance understanding for each class.

Module 5: Model Trainer

- **Purpose**: Train the models with various oversampling techniques using TensorFlow/Keras and visualise performance metrics.
- **Key Imports**: `tensorflow`, `numpy`, `matplotlib`, and `os`.
- **Main Class** - ModelTrainer:
  - Callbacks: Implement early stopping, adaptive learning rate, model checkpointing, and logging.
  - Training: Reproducibly train models with consistent initialisations across different oversampling techniques.
  - Plot Metrics: Visualise training and validation metrics.

Module 6: Model Evaluator

- **Purpose**: Evaluate pre-trained models on test data, generate insightful metrics, and visualise performance indicators for multi-class classification problems.
- **Key Imports**: `tensorflow`, `scikit-learn`, `pandas`, `matplotlib`, `seaborn`, and `os`.
- **Main Class** - ModelTrainer:
  - Facilitates Evaluation: Orchestrates the entire evaluation process for trained models on test data.
  - Comprehensive Reporting: Produces detailed classification reports, highlighting model precision, recall, and F1-score metrics for each class.
  - Visual Insights: Crafts confusion matrices and metric plots, offering a visual gauge of model performance.
  - Tailored to Multi-Class: Designed to specifically cater to multi-class classification challenges, ensuring each class's performance is dissected and displayed.

Data

Data Overview

- **Source**: The data was downloaded from Kaggle [2].
- **Dataset Origin**: The Physionet's MIT-BIH Arrhythmia Dataset is commonly used for research in ECG classification and contains annotated beat-by-beat classifications.
- **Arrhythmia Dataset**:
  - Number of Samples: 109446
  - Total Columns: 188
  - Time-series ECG data: Columns 0 to 186
  - Class labels: Column 187
  - Number of Classes (Categories): 5
  - Classes: ['N': 0, 'S': 1, 'V': 2, 'F': 3, 'Q': 4]

Data Overview

- Each row in the dataset represents a segmented ECG signal, with the sequential data points of the ECG signal being the values in the first 187 columns of that row.
- The signal values range from 0 to 1, suggesting that they are normalised signals.
- The last value in the row (the 188th column) represents the label or category of that particular ECG signal segment, indicating whether it's a normal heartbeat or some type of arrhythmia.

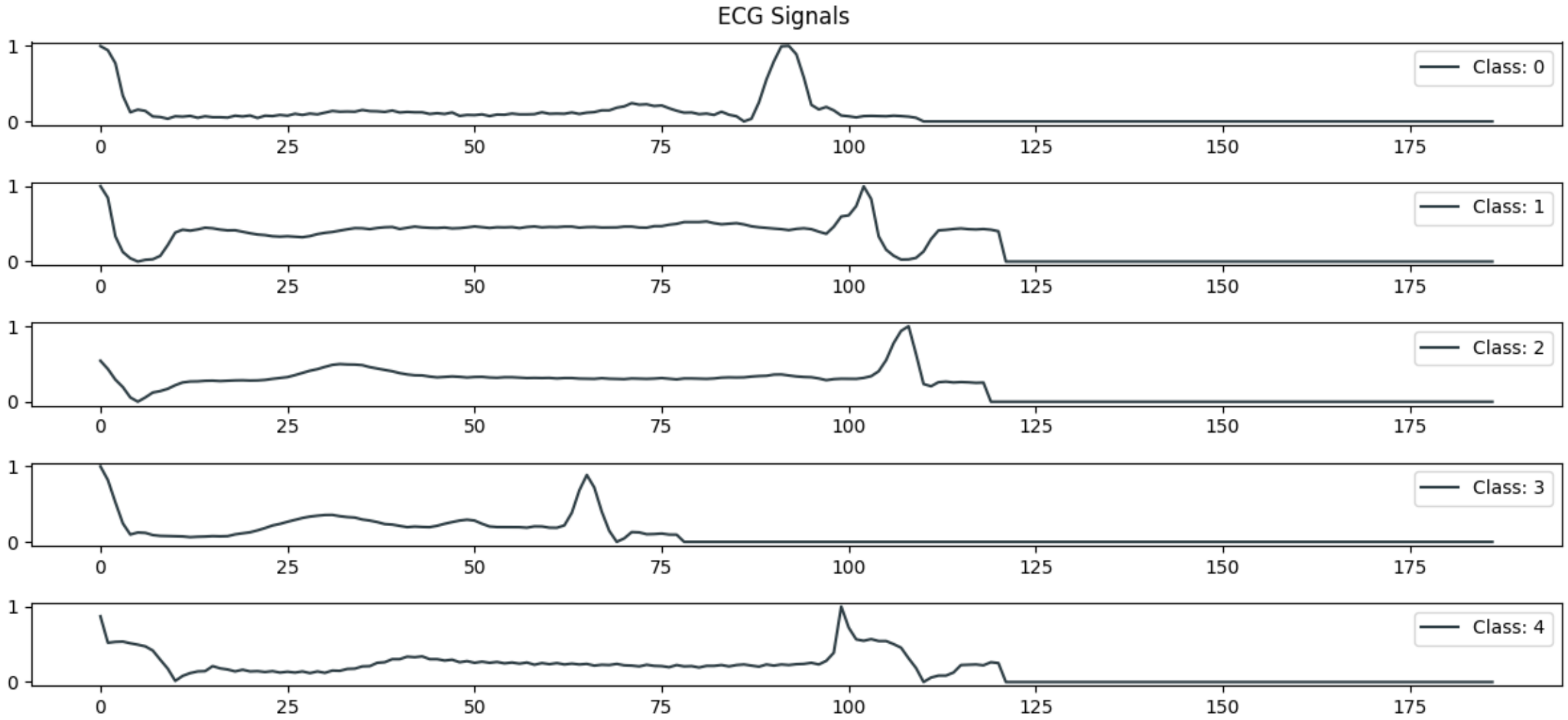
ECG signal				Class
signal1_timestep1	signal1_timestep2	…	signal1_timestep187	label1
signal2_timestep1	signal2_timestep2	…	signal2_timestep187	label2
signal3_timestep1	signal3_timestep2	…	signal3_timestep187	label3
…		…		…

Data Overview

- Annotations in this dataset were used to create five different beat categories in accordance with Association for the Advancement of Medical Instrumentation (AAMI) EC57 standard [3].
- The table below represents a summary of mappings between beat annotations in each category [3].

	Category	Annotation
Class 0	**N** *Normal and variants of normal rhythms*	Normal, Left/Right bundle branch block, Atrial escape, Nodal escape
Class 1	**S** *Supra-ventricular premature or ectopic beats*	Atrial premature, Aberrant atrial premature, Nodal premature, Supra-ventricular premature
Class 2	**V** *Ventricular rhythms*	Premature ventricular contraction, Ventricular escape
Class 3	**F** *Fusion beats*	Fusion of ventricular and normal
Class 4	**Q** *Paced rhythms and related beats*	Paced, Fusion of paced and normal, Unclassifiable

Random Signals for each Category



Training set

- The Training set contains 87,553 samples. The distribution of classes is as follows:

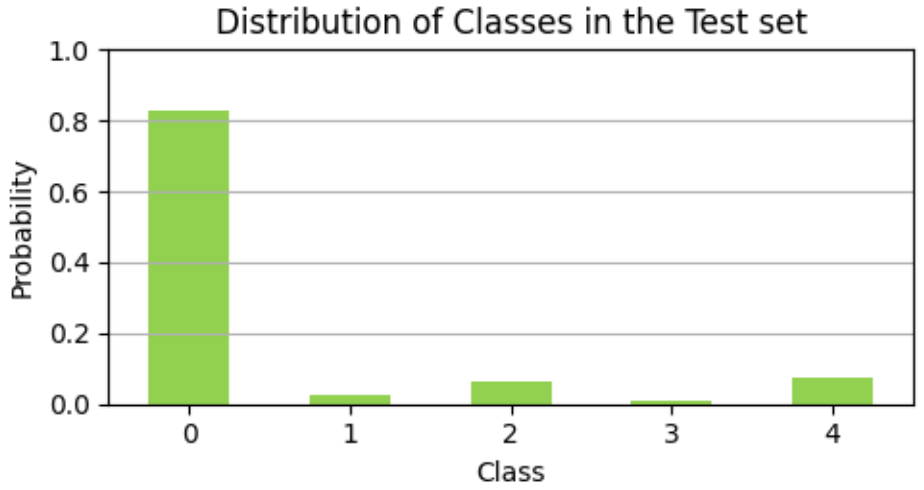
Label	# Samples
Class 0 (N)	72,470 (82.77%)
Class 1 (S)	2,223 (2.54%)
Class 2 (V)	5,788 (6.61%)
Class 3 (F)	641 (0.73%)
Class 4 (Q)	6,431 (7.35%)

- **Imbalance**: The most prominent observation is the class imbalance. Class 0 overwhelmingly dominates the dataset, holding about 82.77% of the samples. Such imbalances can be challenging for many machine learning algorithms, as they might be biased towards predicting the majority class.

Test set

- The Test set contains 21,892 samples. The distribution of classes is as follows:

Label	# Samples
Class 0 (N)	18,118 (82.76%)
Class 1 (S)	556 (2.54%)
Class 2 (V)	1,448 (6.61%)
Class 3 (F)	162 (0.74%)
Class 4 (Q)	1,608 (7.35%)



- **Consistency with Training Set**: The class distribution in the test set closely mirrors that of the training set. This is desirable, as it ensures that the evaluation of a model trained on the training set would be done in a setting that reflects its training environment.
- **Imbalance Continues**: Just like in the training set, there is a significant class imbalance in the test set, with Class 0 being the majority.

Addressing Class Imbalances: The Need & Solution

Why Resampling?

- **Class Imbalance**: Highlights that the original dataset had a dominant majority class and underrepresented minority classes.
- **Implications**: Models trained on imbalanced data can be biased towards the majority class, potentially leading to suboptimal performance.
- **Objective**: A balanced class distribution to improve model sensitivity across all classes.

Addressing Class Imbalances: The Need & Solution

Solution

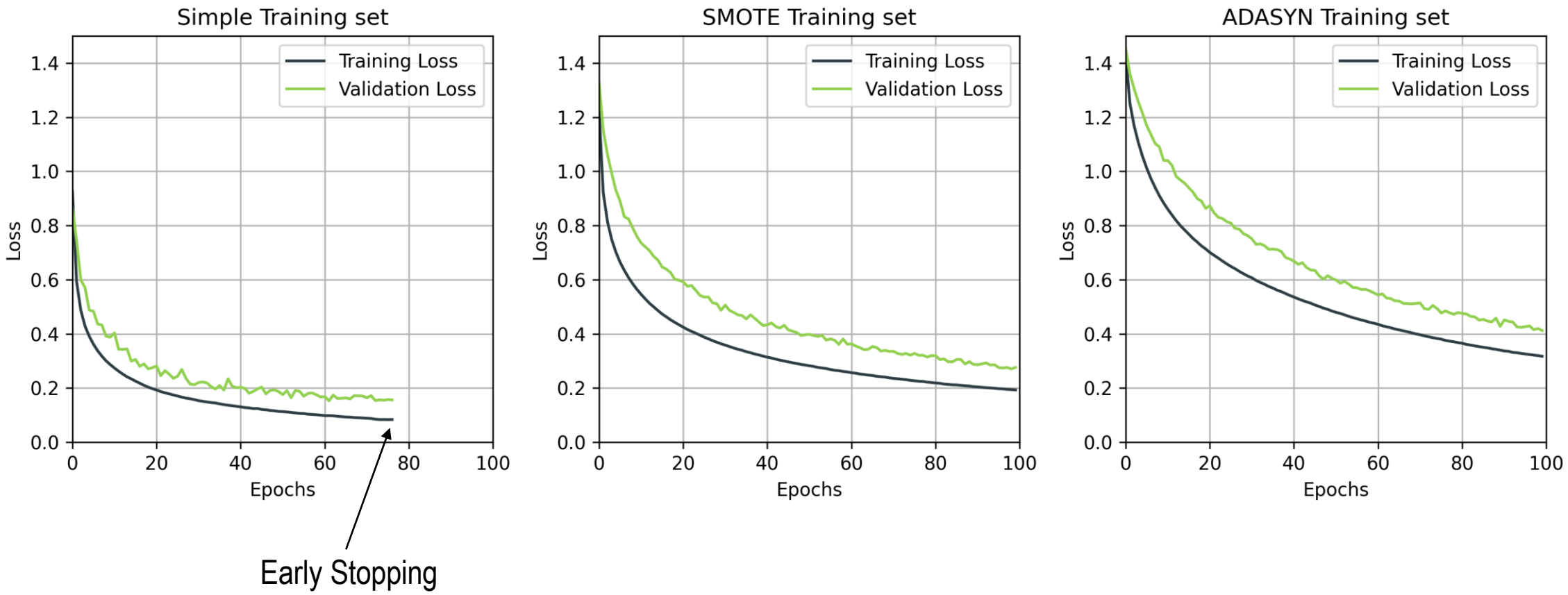
- Three Oversampling techniques are investigated:
  - **Simple Oversampling**: Duplicates random samples from the minority classes until they match the number of samples in the majority class.
  - **SMOTE** (Synthetic Minority Over-sampling Technique): Generates synthetic samples by interpolating between existing minority samples. This is done by choosing two or more similar instances and producing a new, synthetic instance at a point between them.
  - **ADASYN** (Adaptive Synthetic Sampling): Similar to SMOTE but adds slight random noise. It focuses on generating samples next to the original samples which are wrongly classified using a k-Nearest Neighbours classifier.

Results

Training Loss Curves | Baseline Neural Network

Baseline Neural Network

- Maximum number of Epochs = 100
- Batch size = 2048



Training Loss Curves | Baseline Neural Network

- For all three training sets (Simple, SMOTE, and ADASYN), the training and validation loss decrease over epochs, indicating the models are learning. However, there are variations in their patterns:
  - **Simple**: The gap between training and validation loss slightly widens, hinting at potential overfitting, though not overly pronounced.
  - **SMOTE**: Validation loss experiences more fluctuation than the Simple training set. A relatively wider gap between training and validation loss points to some overfitting.
  - **ADASYN**: The validation loss is the most variable, and the distinction between training and validation loss is the most pronounced among the three, implying the highest degree of overfitting.

Training Loss Curves | Baseline Neural Network

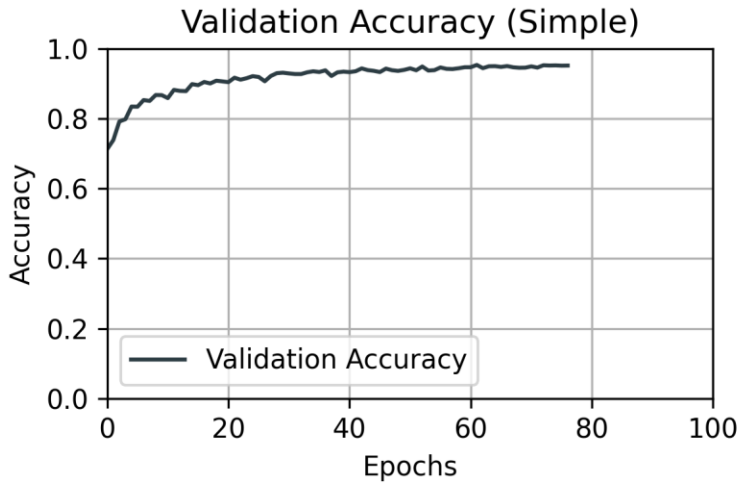
- **Performance**: The Simple training set's loss curves show a more consistent convergence pattern. Both the training and validation loss decrease steadily over epochs. The smoother decline suggests that the model trained on the Simple training set has achieved a better fit to the data compared to the models trained on the SMOTE and ADASYN training sets.

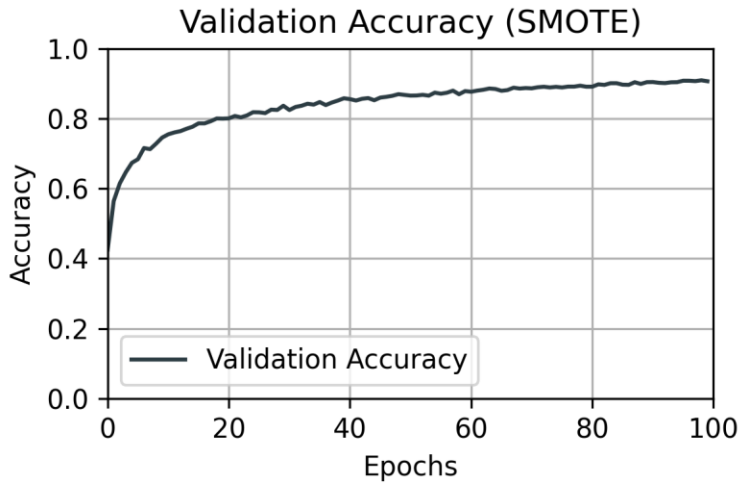
- **Early Stopping**: is a regularisation technique where training is halted once the model's performance on the validation set starts to degrade (or stops improving). It is an effective way to prevent overfitting by not allowing the model to continue training once it begins to memorise the training data rather than generalising from it.

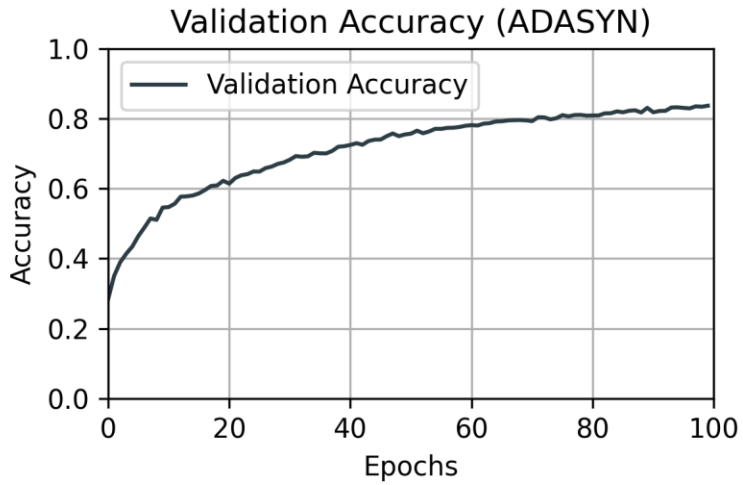
    - Early stopping is particularly interesting because it might suggest that the model trained on the **Simple training set** reached a satisfactory level of performance earlier, thereby requiring fewer epochs. This could be an indication of a better generalisation capability, at least within the context of this specific training setup.

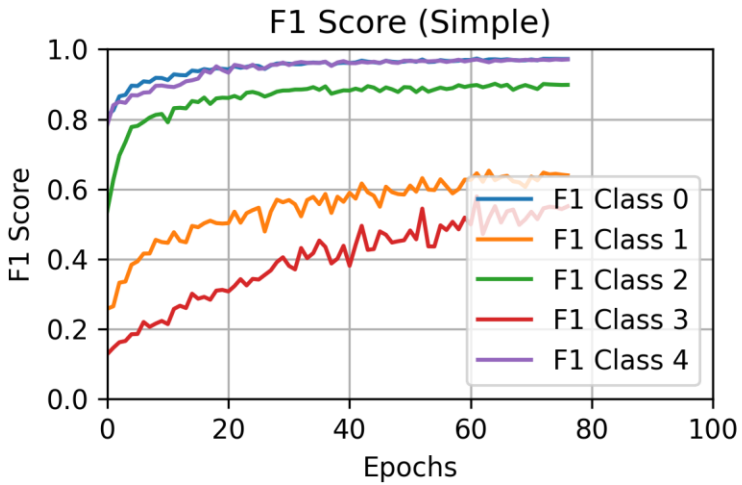
Validation Metrics | Baseline Neural Network

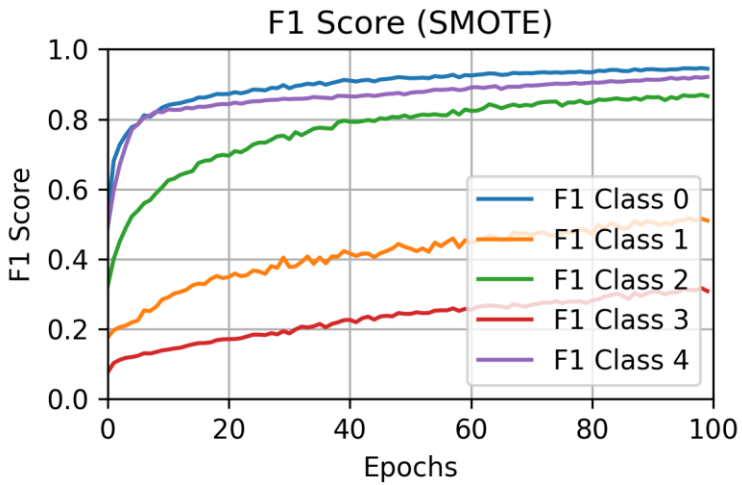
- The plots give insights into the efficacy of the training process across different training sets and classes.

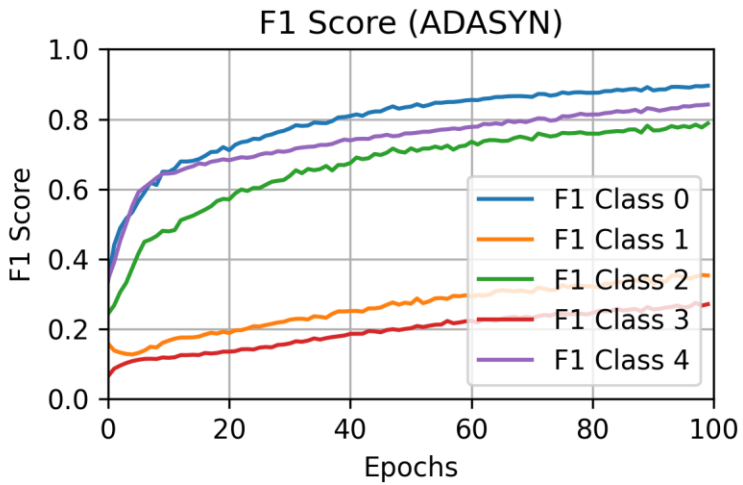












Validation Metrics | Baseline Neural Network

- **Simple training set**:
  - The validation accuracy increases over epochs, suggesting an improvement in model performance.
  - The F1 scores for all classes tend to improve over epochs. There's noticeable variability in the F1 scores for different classes.
- **SMOTE training set** :
  - The validation accuracy also rises over epochs, but there's more variability in the latter epochs.
  - F1 scores improve over epochs, with some classes showing more fluctuation than others.
- **ADASYN training set**:
  - The validation accuracy increases over epochs, showing some fluctuation in the latter epochs.
  - F1 scores improve over epochs. Again, there's noticeable variability among different classes.

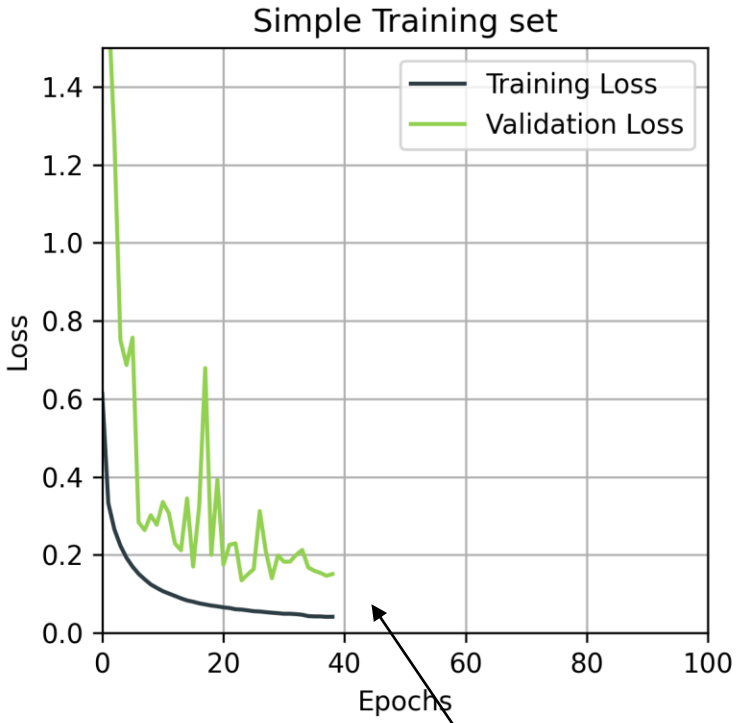
Validation Metrics | Baseline Neural Network

- The **Simple training set**, despite the class imbalances, provided higher F1 scores for minority classes 1 and 3 compared to the SMOTE and ADASYN training sets. This suggests a few possibilities:
  - **Quality over Quantity**: Synthetic data generation techniques like SMOTE and ADASYN aim to balance the dataset by creating synthetic samples. However, these synthetic samples might not necessarily capture the underlying complexities and patterns of the real data. This can sometimes lead to a degradation in performance, even if the dataset becomes more balanced in terms of class distribution.
  - **Model Fit**: The model might have achieved a better fit on the original data (Simple training set) without the need for synthetic augmentation. This could be due to inherent characteristics of the dataset or the model's ability to capture patterns more effectively in the absence of synthetic samples.
  - **Overfitting on Synthetic Samples**: While synthetic samples increase the number of minority class instances, they can sometimes lead the model to overfit on the synthetic patterns, reducing its ability to generalise well on real, unseen data.
  - **Generalisation**: The simple dataset's performance suggests that, for this specific problem and model architecture, the raw dataset without any synthetic augmentation provides better generalisation, at least for the minority classes in question.

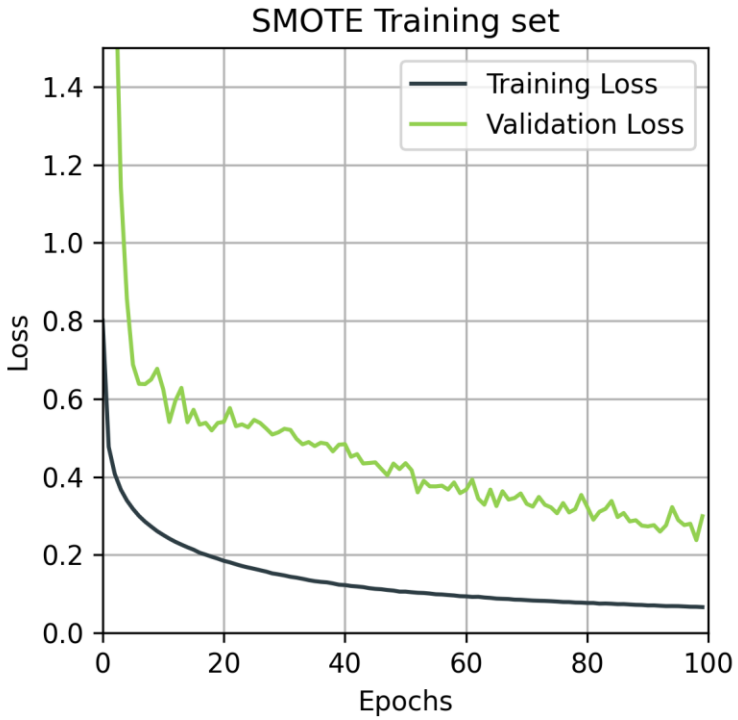
Training Loss Curves | Deep CNN with Residual Blocks

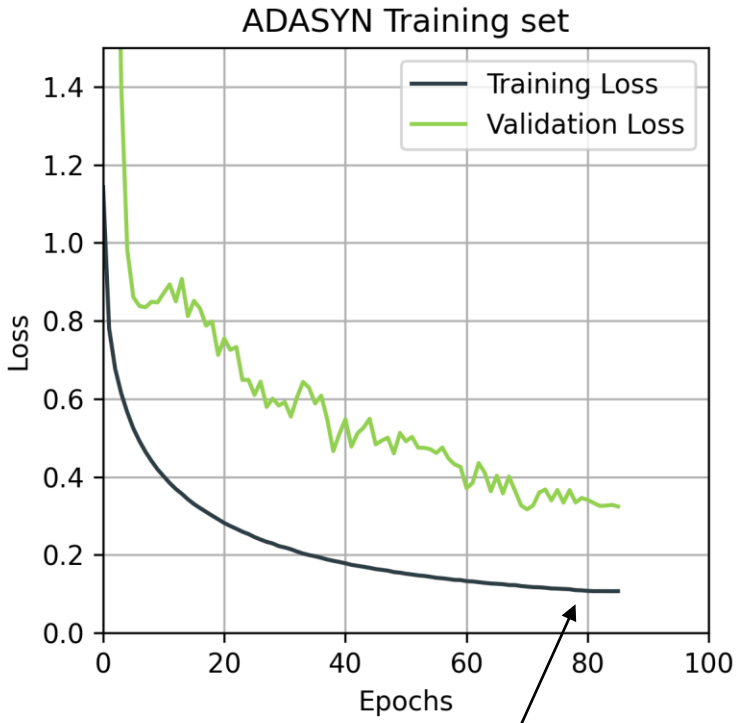
Deep CNN with Residual Blocks

- Maximum number of Epochs = 100
- Batch size = 2048



Early Stopping





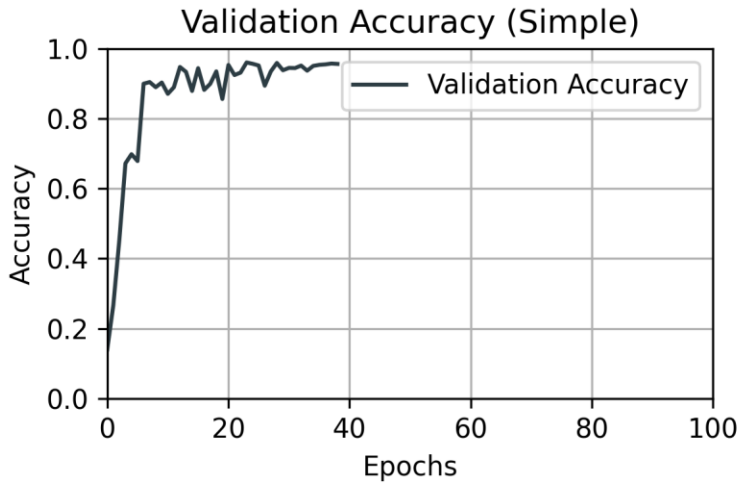
Early Stopping

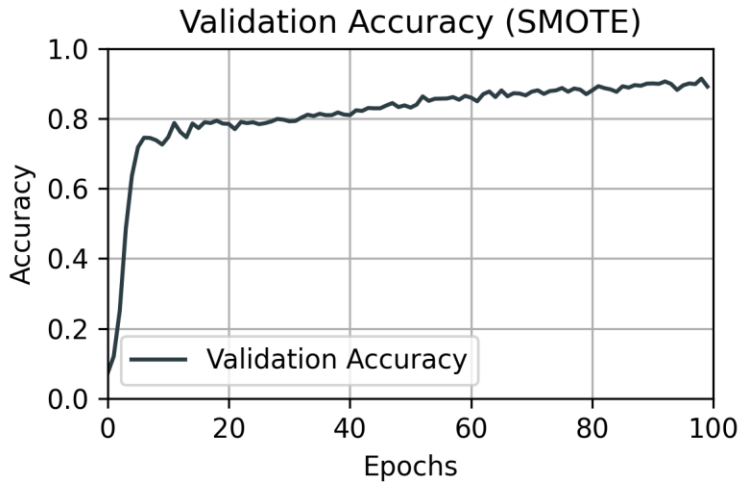
Training Loss Curves | Deep CNN with Residual Blocks

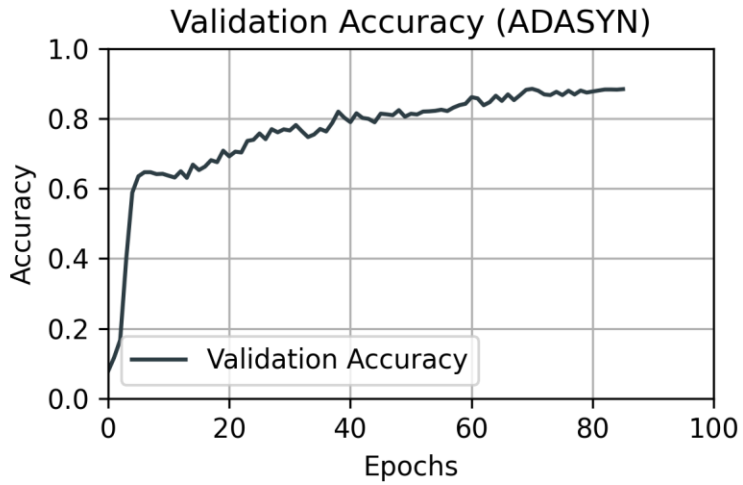
- For all three approaches, the training loss consistently decreases over epochs, which is expected as the model learns from the training data.
- The validation loss for the **Simple** approach starts high but decreases rapidly, suggesting a good fit to the validation data.
- The validation loss for the **SMOTE** and **ADASYN** approaches starts much higher and decreases more gradually.
- The **ADASYN** approach's validation loss decreases but remains relatively high, suggesting potential underfitting or less effective data augmentation compared to the other methods.

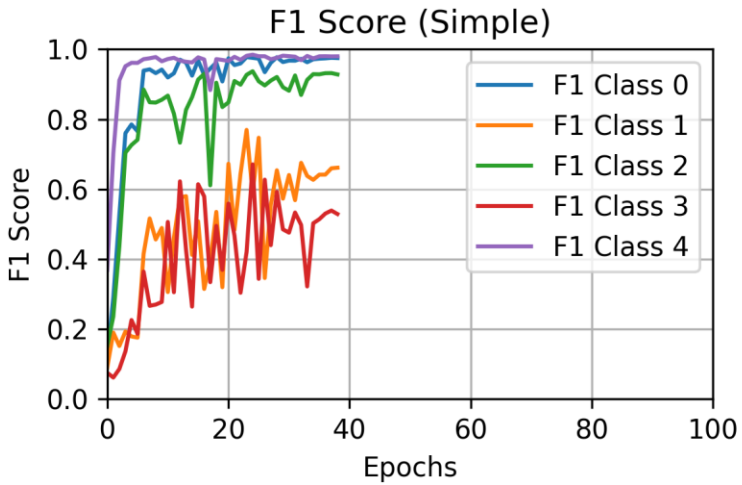
Validation Metrics | Deep CNN with Residual Blocks

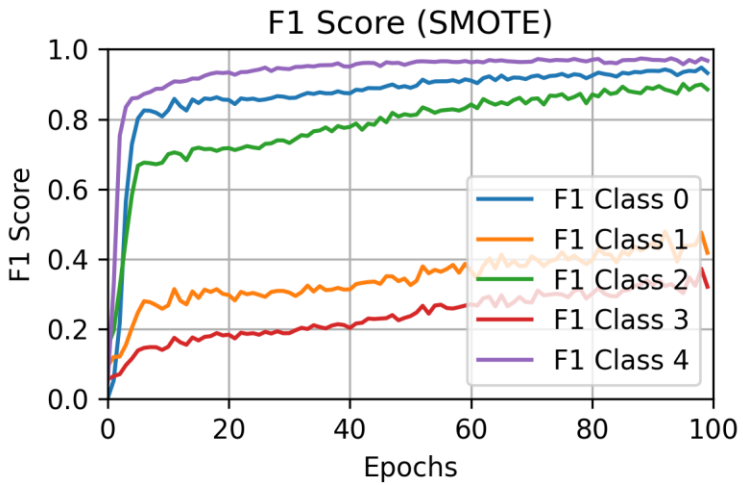
- The plots give insights into the efficacy of the training process across different training sets and classes.

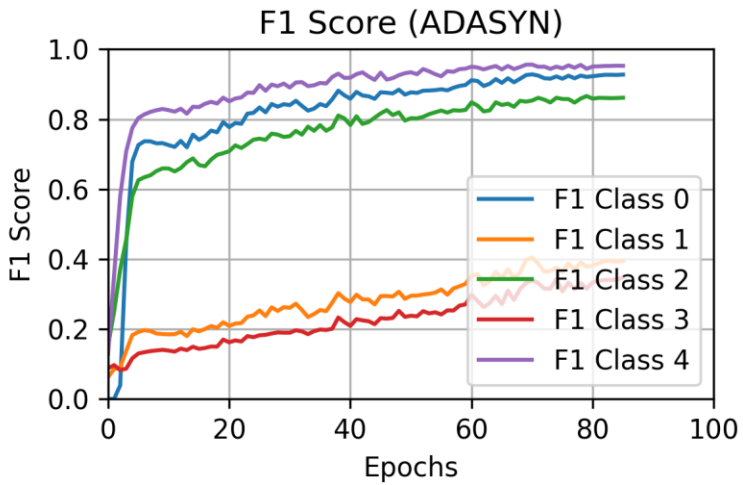












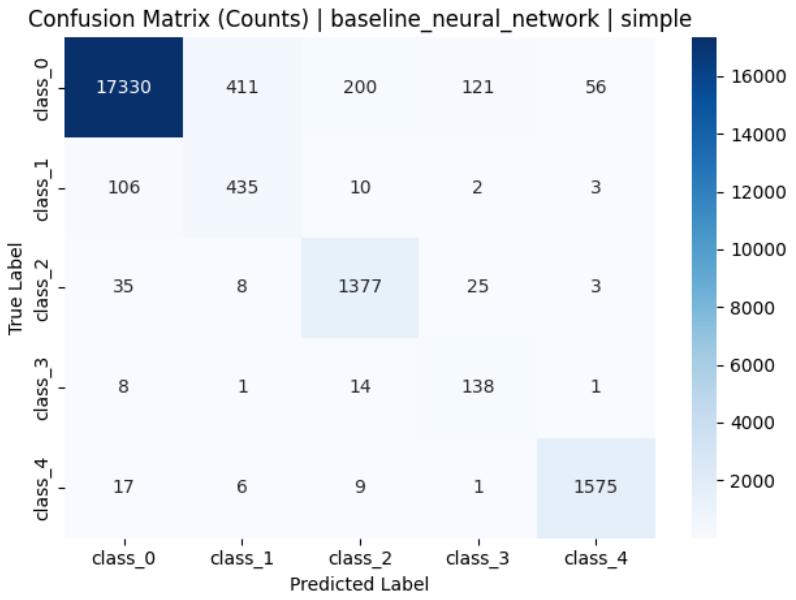
Validation Metrics | Deep CNN with Residual Blocks

- Class 0:
  - The **Simple** approach sees a rapid increase in F1-Score.
  - The **SMOTE** approach starts lower but surpasses the Simple approach in the latter epochs.
  - The **ADASYN** approach experiences a steady rise but remains below the other two.
- Class 1:
  - The **Simple** approach starts higher and generally maintains a lead.
  - The **SMOTE** and **ADASYN** approaches start lower and increase at a more gradual rate.
- Class 2:
  - All three approaches increase over time, with the Simple approach leading for most epochs.
- Class 3:
  - The **Simple** approach starts higher and remains above the other two throughout.
  - The **SMOTE** and **ADASYN** approaches start low but see a rise over the epochs.
- Class 4:
  - The **Simple** and **SMOTE** approaches start high and remain close to each other.
  - The **ADASYN** approach starts lower but gradually catches up.

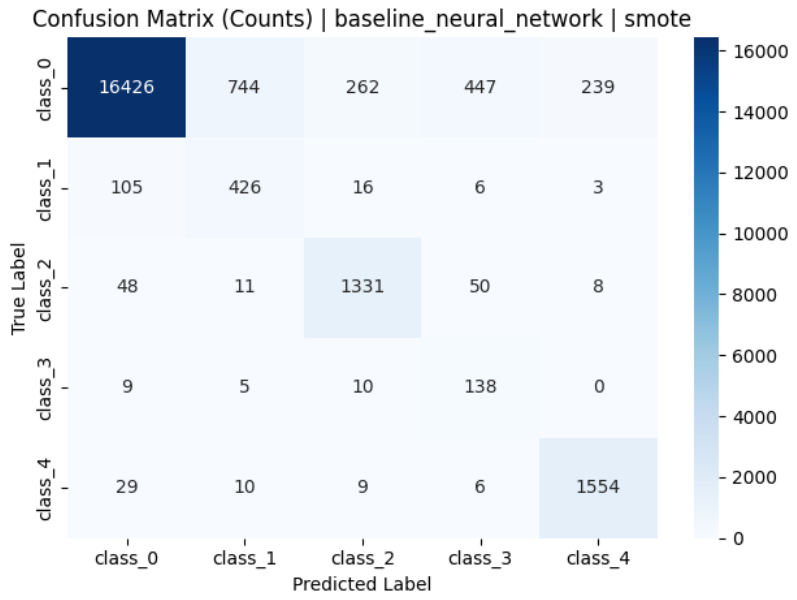
Confusion Matrix | Baseline Neural Network

Baseline Neural Network

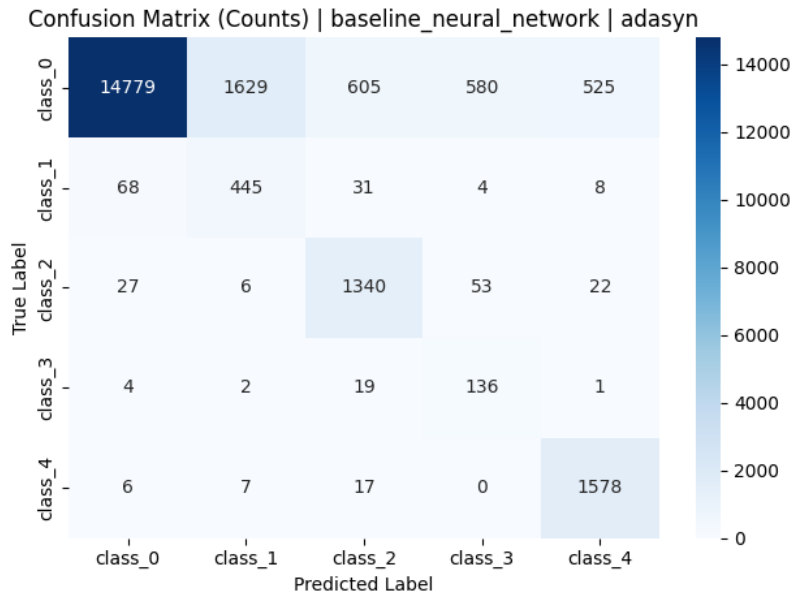
Simple



SMOTE



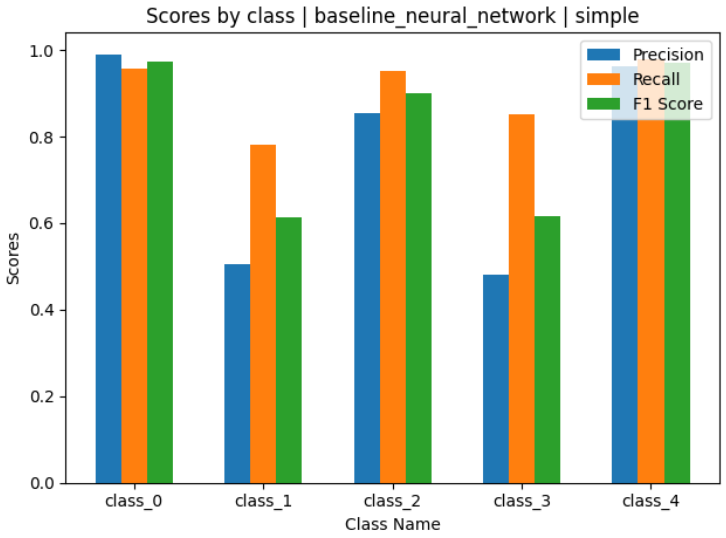
ADASYN



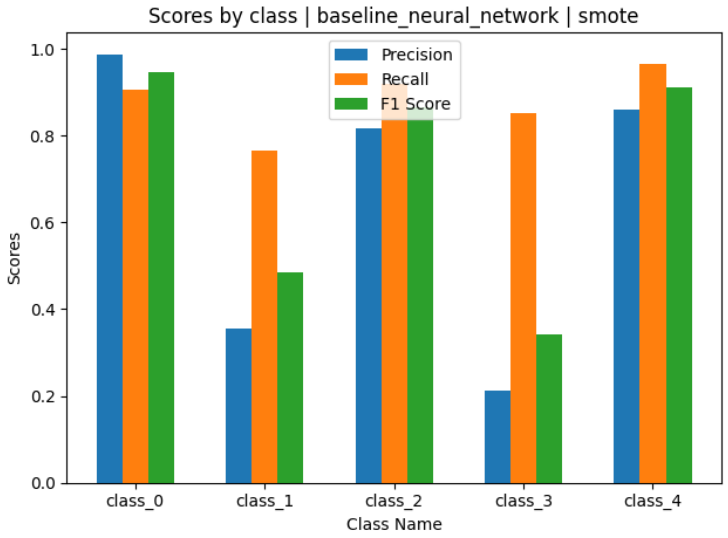
Precision, Recall, and F1-score per class | Baseline Neural Network

Baseline Neural Network

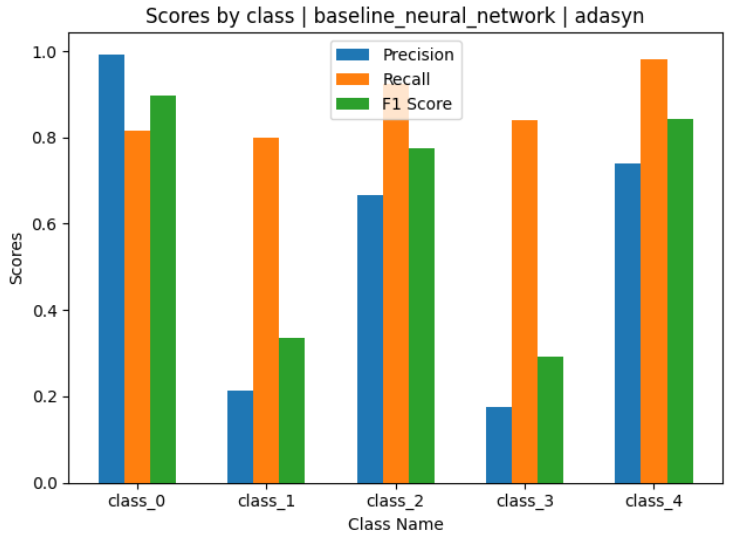
Simple



SMOTE



ADASYN



Precision, Recall, and F1-score per class | Baseline Neural Network

Simple

Baseline Neural Network

Model	Technique	Label	Precision	Recall	F1-score
Baseline	Simple	class_0	99.05%	95.65%	97.32%
		class_1	50.52%	78.24%	61.40%
		class_2	85.53%	95.10%	90.06%
		class_3	48.08%	85.19%	61.47%
		class_4	96.15%	97.95%	97.04%

Precision, Recall, and F1-score per class | Baseline Neural Network

SMOTE

Baseline Neural Network

Model	Technique	Label	Precision	Recall	F1-score
Baseline	SMOTE	class_0	98.85%	90.66%	94.58%
		class_1	35.62%	76.62%	48.63%
		class_2	81.76%	91.92%	86.54%
		class_3	21.33%	85.19%	34.12%
		class_4	86.14%	96.64%	91.09%

Precision, Recall, and F1-score per class | Baseline Neural Network

Baseline Neural Network

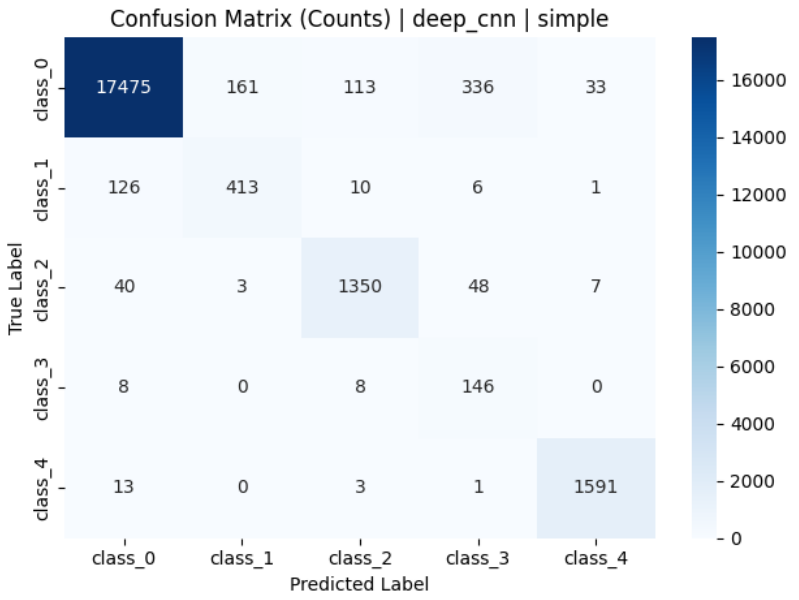
ADASYN

Model	Technique	Label	Precision	Recall	F1-score
Baseline	ADASYN	class_0	99.29%	81.57%	89.56%
		class_1	21.30%	80.04%	33.65%
		class_2	66.60%	92.54%	77.46%
		class_3	17.59%	83.95%	29.09%
		class_4	73.95%	98.13%	84.34%

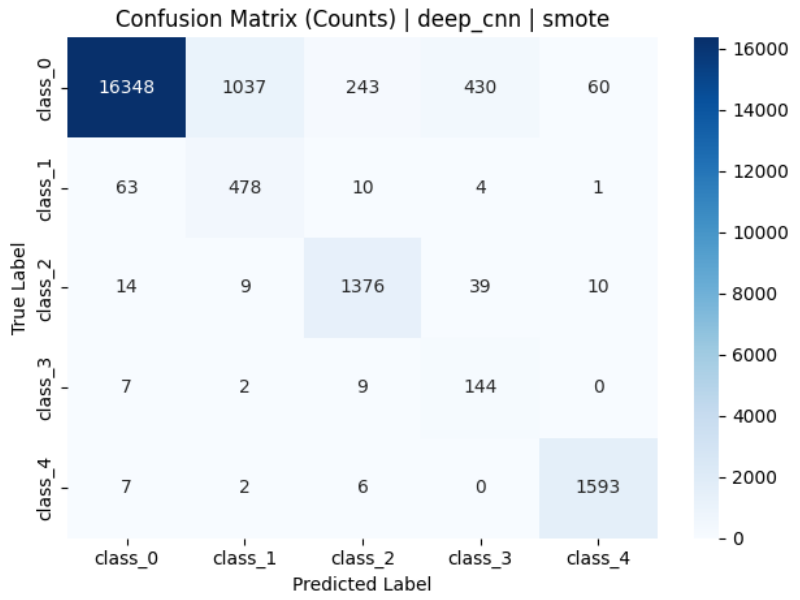
Confusion Matrix | Deep CNN with Residual Blocks

Deep CNN with Residual Blocks

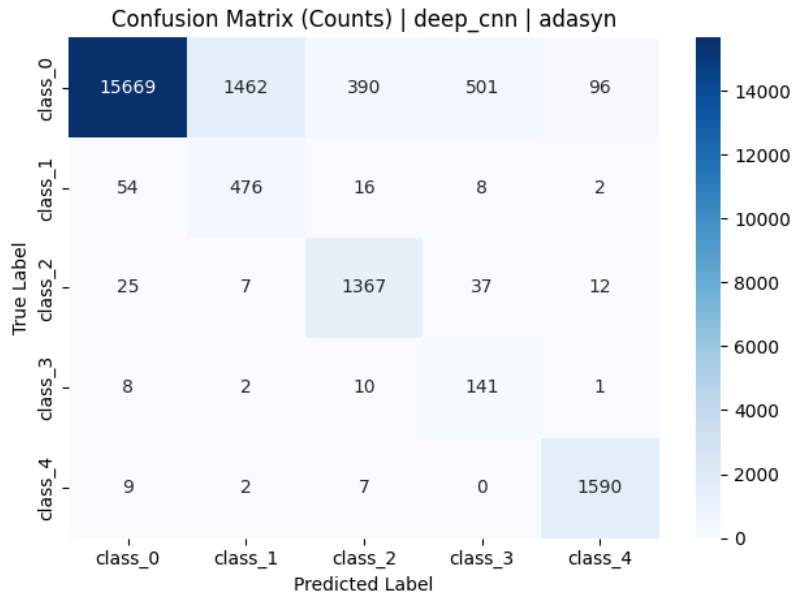
Simple



SMOTE



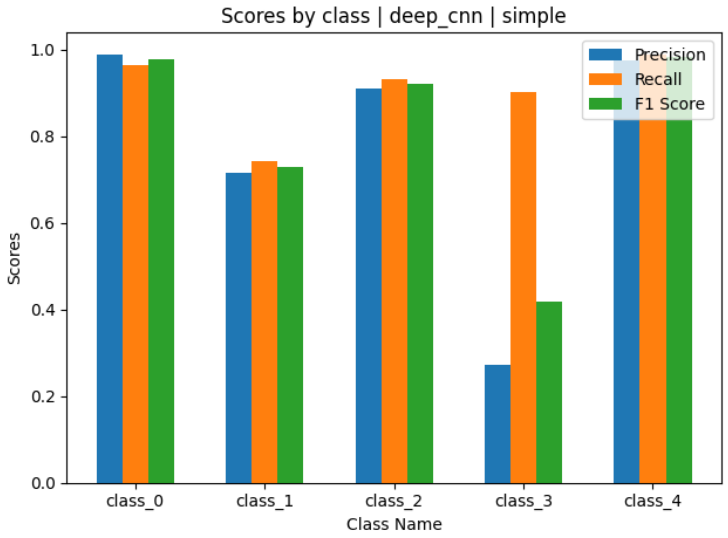
ADASYN



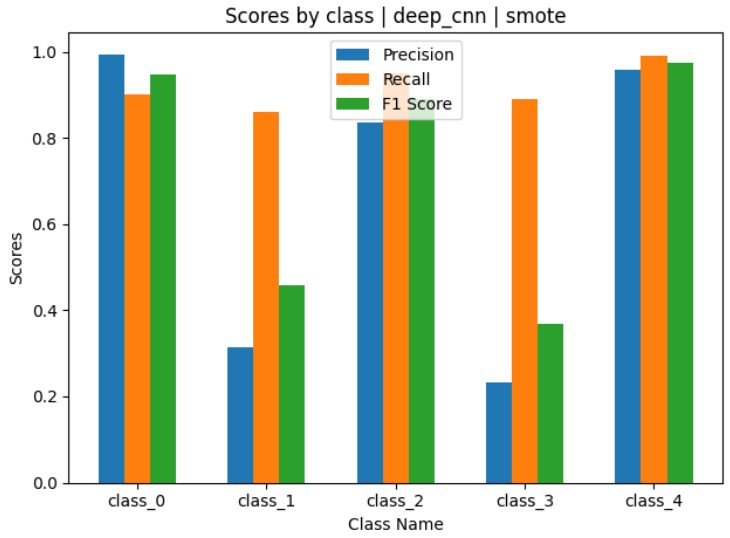
Precision, Recall, and F1-score per class | Deep CNN with Residual Blocks

Deep CNN with Residual Blocks

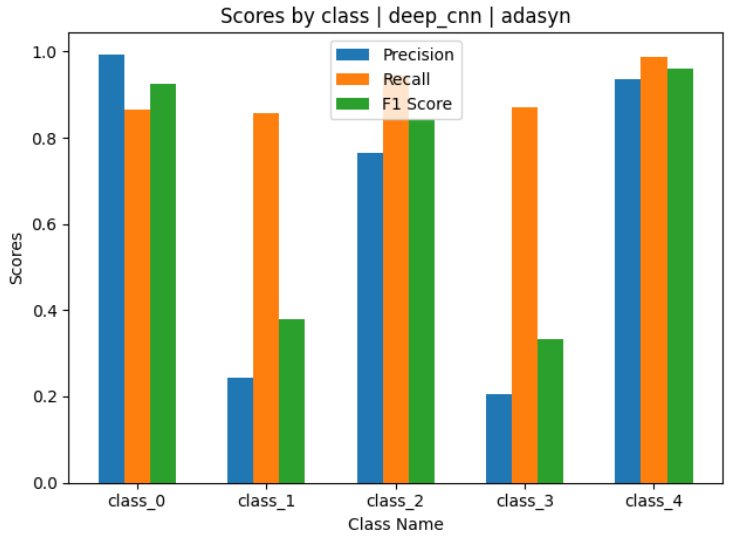
Simple



SMOTE



ADASYN



Precision, Recall, and F1-score per class | Deep CNN with Residual Blocks

Simple

Deep CNN with Residual Blocks

Model	Technique	Label	Precision	Recall	F1-score
Baseline	Simple	class_0	98.94%	96.45%	97.68%
		class_1	71.58%	74.28%	72.90%
		class_2	90.97%	93.23%	92.09%
		class_3	27.19%	90.12%	41.77%
		class_4	97.49%	98.94%	98.21%

Precision, Recall, and F1-score per class | Deep CNN with Residual Blocks

SMOTE

Deep CNN with Residual Blocks

Model	Technique	Label	Precision	Recall	F1-score
Baseline	SMOTE	class_0	99.45%	90.23%	94.61%
		class_1	31.28%	85.97%	45.87%
		class_2	83.70%	95.03%	89.00%
		class_3	23.34%	88.89%	36.97%
		class_4	95.73%	99.07%	97.37%

Precision, Recall, and F1-score per class | Deep CNN with Residual Blocks

ADASYN

Deep CNN with Residual Blocks

Model	Technique	Label	Precision	Recall	F1-score
Baseline	ADASYN	class_0	99.39%	86.48%	92.49%
		class_1	24.42%	85.61%	38.00%
		class_2	76.37%	94.41%	84.43%
		class_3	20.52%	87.04%	33.22%
		class_4	93.47%	98.88%	96.10%

Extracting insights

- **Model Performance**:
  - For **class_0**, the **Deep CNN** model trained with **Simple Oversampling** exhibits the highest F1 score of 97.68%, indicating it achieves a balanced trade-off between precision and recall for this class.
  - For **class_4**, the **Deep CNN** model again stands out with **SMOTE** achieving an F1 score of 97.37%.
  - Interestingly, **class_3** generally has lower F1 scores across all combinations compared to other classes, suggesting it might be a challenging class to predict.
- **Oversampling Techniques**:
  - **Simple Oversampling** seems to perform well for the **Deep CNN** model across most classes, especially for **class_0** and **class_4**.
  - **SMOTE** and **ADASYN** show varied results, but **SMOTE** appears to be beneficial for **class_4** in the **Deep CNN** model, giving it the second-highest F1 score across all combinations.

Extracting insights

- **Precision-Recall trade-off**:
  - Some classes, like **class_1** with the **Baseline** model and **Simple Oversampling**, have a notably lower precision but higher recall, indicating a larger number of false positives but good coverage of the actual positives.
  - Conversely, **class_1** with the **Deep CNN** model and **SMOTE** has a much higher precision but lower recall, indicating fewer false positives but potentially missing out on some actual positives.
- **Comparison between Models**:
  - The **Deep CNN** model generally achieves higher F1 scores across most classes and oversampling techniques compared to the **Baseline** model, suggesting that the added complexity and depth of the CNN might be capturing patterns in the data more effectively.

Extracting insights

- **Challenges with Specific Classes:**
  - **class_3** is consistently challenging across both models and all oversampling techniques. Even though the recall is often high, indicating that a large portion of actual **class_3** instances are correctly identified, the precision is low, leading to many false positives. This suggests that predictions for **class_3** often overlap with predictions for other classes.
- **Effectiveness of ADASYN**:
  - **ADASYN** tends to give relatively lower F1 scores for **class_0** compared to other oversampling techniques, especially with the **Baseline** model. This suggests that for this specific class, **ADASYN** might not be the most effective oversampling strategy.

In summary, the choice of model and oversampling technique can vary in effectiveness depending on the class in question. The **Deep CNN** model generally offers better performance, and while **Simple Oversampling** often provides good results, the choice between **SMOTE** and **ADASYN** might depend on the specific class and the desired trade-off between precision and recall.

High-level summary of the models' performance using macro-averaged metrics

- The results in the table were derived by computing the macro-average values for Precision, Recall, and F1-score across all classes from the classification reports

Model	Technique	Precision	Recall	F1-score
Baseline	Simple	75.87%	90.42%	81.46%
Baseline	SMOTE	64.74%	88.21%	70.99%
Baseline	ADASYN	55.75%	87.25%	62.82%
Deep CNN	Simple	77.23%	90.61%	80.53%
Deep CNN	SMOTE	66.70%	91.84%	72.77%
Deep CNN	ADASYN	62.84%	90.48%	68.85%

Extracting insights from macro-averaged metrics

- **Highest Macro-Averaged F1 Score**:
  - The **Baseline** model trained with **Simple Oversampling** achieved the highest macro-averaged F1 score of 81.46%. This indicates that, on average, across all classes, this combination provides the most balanced performance between precision and recall.
- **Model Performance**:
  - Across all oversampling techniques, the **Deep CNN** model consistently outperforms the **Baseline** model in terms of macro-averaged precision, recall, and F1 score. This indicates that the **Deep CNN** is generally more effective at making balanced predictions across all classes.

Extracting insights from macro-averaged metrics

- **Oversampling Techniques** :
  - **Simple Oversampling** consistently provides the highest macro-averaged precision, recall, and F1 scores for both models. This suggests that duplicating random samples from minority classes is an effective strategy to achieve balanced performance across classes.
  - **SMOTE** provides the second-best results in most cases, particularly shining with the **Deep CNN** model where it achieves a macro-averaged recall of 91.84%, the highest across all combinations.
  - **ADASYN**, on the other hand, tends to yield the lowest macro-averaged precision, recall, and F1 scores, indicating that it might not be as effective as the other techniques in balancing performance across classes.
- **Model Performance**:
  - Across all oversampling techniques, the **Deep CNN** model consistently outperforms the **Baseline** model in terms of macro-averaged precision, recall, and F1 score. This indicates that the **Deep CNN** is generally more effective at making balanced predictions across all classes.

Extracting insights from macro-averaged metrics

- **Precision-Recall trade-off**:
  - For the **Baseline** model, there's a noticeable drop in macro-averaged precision when transitioning from **Simple Oversampling** to **SMOTE** and then to **ADASYN**. However, the macro-averaged recall remains relatively stable, indicating a trade-off where increased false positives are being accepted to maintain coverage of the actual positives.
  - The **Deep CNN** model demonstrates a similar trend, although the drop in precision from **Simple Oversampling** to **SMOTE** is less pronounced.
- **Overall Effectiveness of Techniques**:
  - If the goal is to achieve a balance between precision and recall across all classes, **Simple Oversampling** appears to be the most effective technique. However, if maximising recall (at the potential expense of precision) is a priority, **SMOTE** with the **Deep CNN** model is a strong contender.

- In summary, the **Deep CNN** model generally offers better macro-averaged performance across all classes compared to the **Baseline** model. While **Simple Oversampling** is often the top performer, the choice between **SMOTE** and **ADASYN** might hinge on specific goals, such as maximising recall.

Conclusions

Conclusions

- **Significant Progress in ECG Analysis**:
  - Our approach using Deep Learning techniques, specifically the Deep CNN architecture, has demonstrated significant advancement in ECG time-series classification. This research underscores the potential of leveraging advanced machine learning methods to enhance diagnostic capabilities in cardiology.
- **Oversampling's Impact**:
  - Through our exploration of various oversampling techniques, we have identified that Simple Oversampling often provides a balanced trade-off between precision and recall. While synthetic data augmentation methods like SMOTE and ADASYN can be valuable, they may not always be the optimal choice for every dataset or problem.
- **Comparison with Established Research**:
  - The results achieved in our study are in line with those presented by Rajpurkar, P., et al., 2017 [1]. This is significant as it demonstrates that our methods and findings are not only valid but also in line with cutting-edge research in the field.

Conclusions

- **Deep CNN's Superiority**:
  - The Deep CNN model consistently outperformed the Baseline Neural Network across various metrics. This suggests that for ECG data, models with greater depth and complexity can capture intricate patterns more effectively, leading to improved classification accuracy.
- **Class-Specific Challenges**:
  - Our results indicate that certain arrhythmia classes, such as class_3, remain challenging to predict. This highlights the importance of continuous research to refine models and techniques for such underrepresented or complex classes.
- **Holistic Approach**:
  - This research not only delved into model architectures but also addressed critical aspects like data imbalance, which is often a bottleneck in medical datasets. By addressing these challenges holistically, we have laid a foundation for more robust and reliable ECG analysis systems in the future.

References

1. Rajpurkar, P., Hannun, A. Y., Haghpanahi, M., Bourn, C., & Ng, A. Y. (2017). Cardiologist-level arrhythmia detection with convolutional neural networks. arXiv preprint arXiv:1707.01836.
2. Fazeli, Shayan. ECG Heartbeat Categorization Dataset. Kaggle, [19th September 2023], https://www.kaggle.com/datasets/shayanfazeli/heartbeat?datasetId=29414&language=Python.
3. Kachuee, M., Fazeli, S., & Sarrafzadeh, M. (2018). ECG Heartbeat Classification: A Deep Transferable Representation. arXiv preprint arXiv:1805.00794.

Appendix

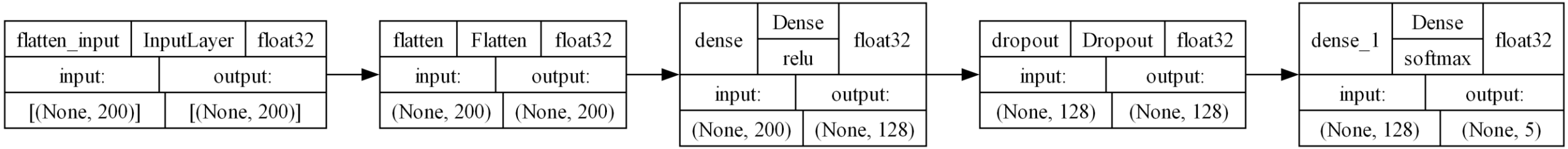
Baseline Neural Network [1 / 2]

- **Input Layer**:
  - The network begins with an input layer that accepts data of shape `input_size=200`.
- **Flatten Layer**:
  - This layer reshapes the input data, ensuring it is in a format suitable for the subsequent dense layers.
- **Dense Layer**:
  - This is a fully connected layer comprising `UNITS_DENSE=128` neurons.
  - Activation Function: `ReLU` (Rectified Linear Activation) is used, which introduces non-linearity and aids in capturing patterns and relations in the data.
- **Dropout Layer**:
  - This layer is introduced for regularisation purposes to prevent overfitting.

Baseline Neural Network [2 / 2]

- **Dropout Layer**:
  - Dropout Rate: During training, a fraction (equal to `DROPOUT_RATE=0.2`) of the neurons in the preceding layer are randomly dropped out or turned off, which helps in ensuring that the network doesn't overly rely on any particular neuron and generalises better.
- **Output Dense Layer**:
  - This is the final fully connected layer comprising `num_classes=5` neurons, corresponding to the number of target classes in the classification task.
  - Activation Function: `Softmax`. The softmax activation function ensures that the output values are in the range (0, 1) and sum up to 1, making them interpretable as probabilities for each class.

Baseline Neural Network | Plot



Deep CNN with Residual Blocks

- **Input Layer**:
  - The network starts with an input layer that accepts data of shape (`input_size=200, 1`). The single channel typically represents univariate time-series data (sequences).
- **Initial Block**:
  - Convolutional Layer:
    - Uses `INITIAL_NUM_FILTERS=16` filters and a kernel size specified by `KERNEL_SIZE=4`.
    - Padding is set to 'same' to ensure the output has the same width and height as the input.
  - Batch Normalisation: Normalises the activations of the neurons in this layer.
  - ReLU Activation: Introduces non-linearity into the model.
- **Residual Blocks**:
  - These are the core of the deep CNN, providing the ability to train deeper models without the network suffering from vanishing or exploding gradients.

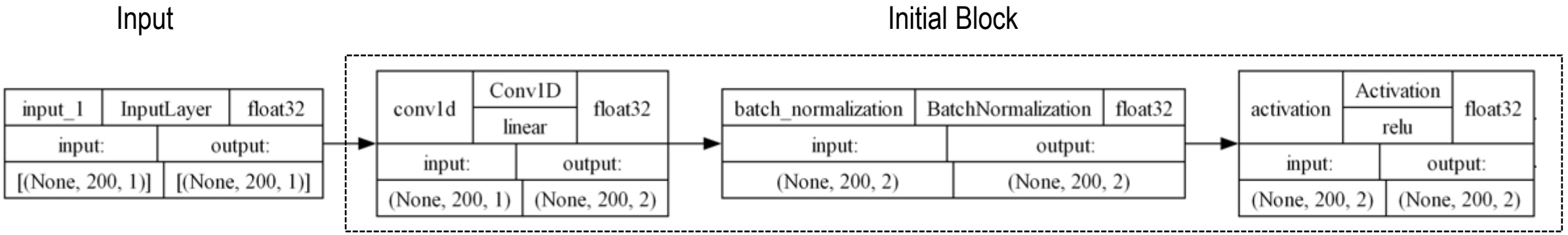
Deep CNN with Residual Blocks

- **Residual Blocks**:
  - The architecture dynamically constructs these blocks in a loop.
  - Iterative construction:
    - The number of filters in the convolutional layers starts with `INITIAL_NUM_FILTERS=16` and scales by a factor, k, which begins as 1 and is incremented every 4th residual block.
    - The network subsamples its inputs in every alternate residual block to reduce dimensionality and increase the receptive field.
    - Whenever the scaling factor k is increased, the channels in the shortcut connection are padded to match the increased number of channels in the main path.
  - Type 1 Residual Block:
    - Used as the first block. This is the simpler form and consists of two convolutional layers, each followed by batch normalisation, ReLU activation, and dropout. The output from this block is added to its input, creating the residual connection.

Deep CNN with Residual Blocks

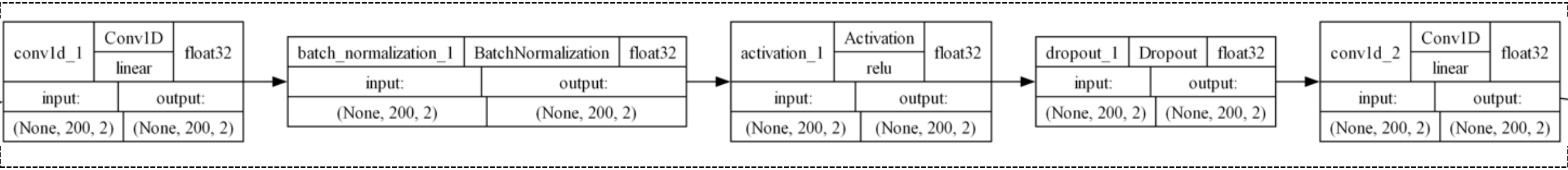
- **Residual Blocks**:
  - Type 2 Residual Block:
    - A more complex form used from the second block onwards. It has two sets of batch normalisation, ReLU activation, dropout, and convolutional layers. Depending on its position in the model and the parameters, it might also include subsampling (using max pooling) and channel padding (using a 1x1 convolution).
- **Final Block**:
  - Batch Normalisation: Normalises the activations.
  - ReLU Activation: Introduces non-linearity.
  - Flatten Layer: Reshapes the data to make it suitable for the final dense layer.
  - Dense Layer: A fully connected layer with `num_classes=5` neurons.
  - Softmax Activation: Ensures the outputs are probabilities summing up to 1.

Deep CNN with Residual Blocks | Plot

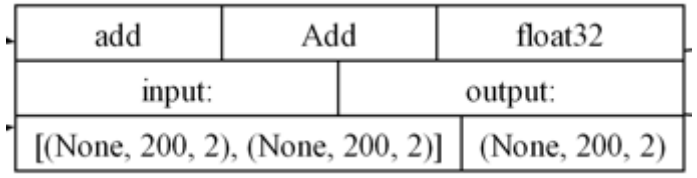


Deep CNN with Residual Blocks | Plot

First Residual Block

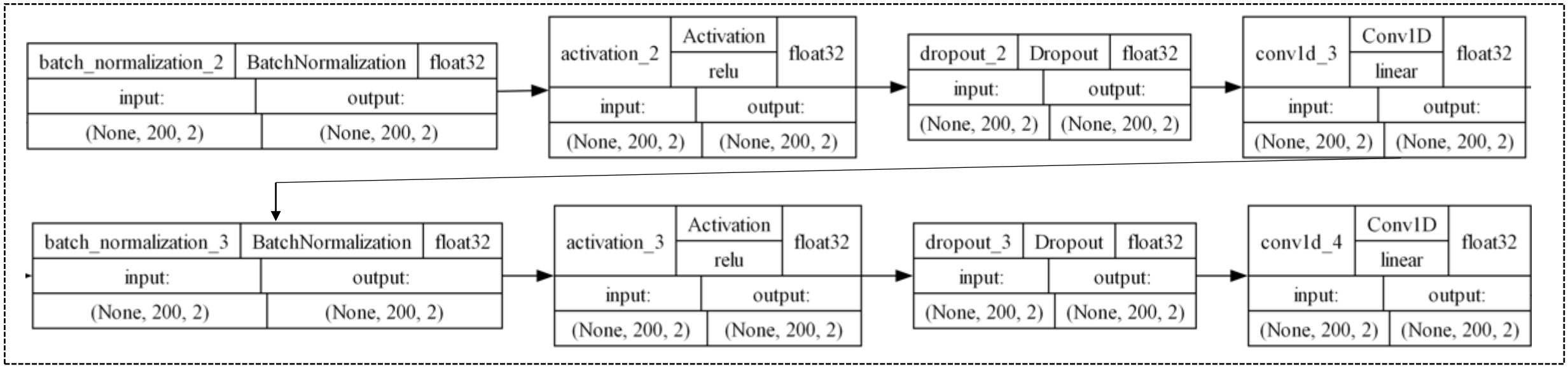


First Residual connection

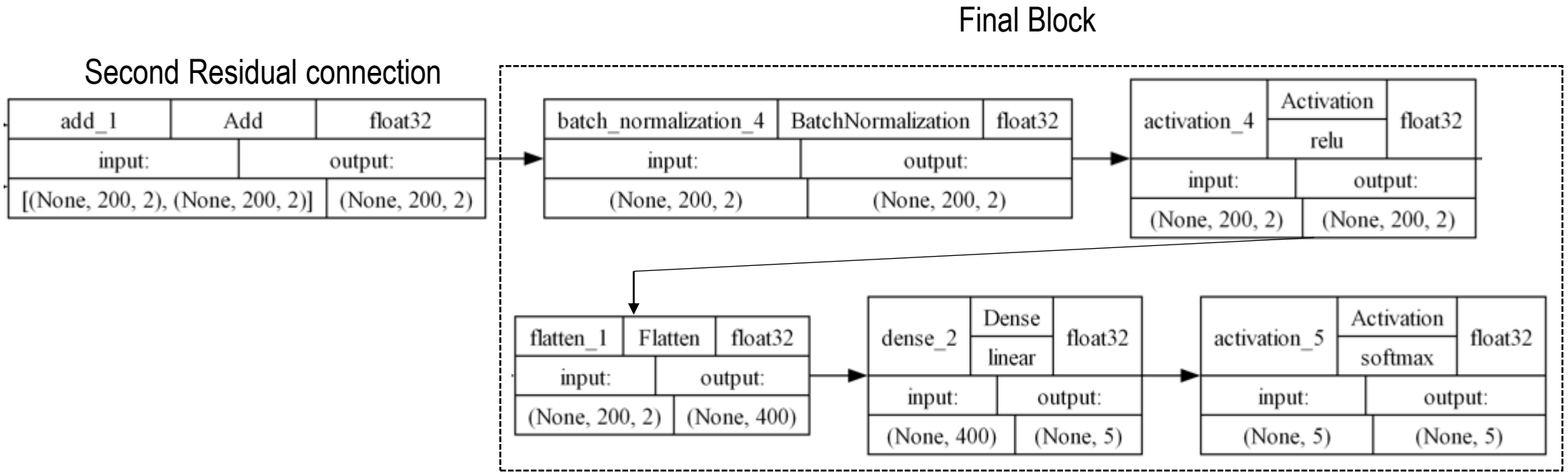


Deep CNN with Residual Blocks | Plot

Second Residual Block



Deep CNN with Residual Blocks | Plot



Deep CNN with Residual Blocks | Plot