# 

# **ENHANCED DIAGNOSIS OF ALZEHEIMER’S DISEASE THROUGH PREDICTIVE PROGRESSION ANALYSIS OF NEURO IMAGING SEQUENCES**

**A PROJECT REPORT**

#### Submitted by

**V. R. RISHENDRA [RA2111026010221]**

**RANGINENI YUKTHAMUKHI [RA2111026010262]**

#### Under the Guidance of

## **Dr. M. Karpagam**

(Assistant Professor, Department of Computational Intelligence)

### *in partial fulfillment of the requirements* *for the degree of*

## BACHELOR OF TECHNOLOGY

## in

## COMPUTER SCIENCE ENGINEERING

## with specialization in Artificial Intelligence and Machine Learning



## DEPARTMENT OF COMPUTATIONAL INTELLIGENCE COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

## KATTANKULATHUR- 603 203

### NOVEMBER 2024

Department of Computational Intelligence

##### SRM Institute of Science & Technology

##### Own Work Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

##### Degree/ Course : B.Tech computer science and engineering with specialization in

##### Artificial Intelligence and Machine Learning

**Student Name : V.R.Rishendra, Rangineni Yukthamukhi**

##### Registration Number :RA2111026010221, RA211026010262

**Title of Work : Enhanced Diagnosis of Alzheimer's Disease through**

**Predictive Progression Analysis of Neuro Imaging Sequences**

I / We hereby certify that this assessment compiles with the University’s Rules and Regulations relating to Academic misconduct and plagiarism\*\*, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

* Clearly referenced / listed all sources as appropriate
* Referenced and put in inverted commas all quoted text (from books, web, etc)
* Given the sources of all pictures, data etc. that are not my own
* Not made any use of the report(s) or essay(s) of any other student(s) either past or present
* Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
* Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

|  |
| --- |
| **DECLARATION:** |
| I am aware of and understand the University’s policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.  RA2111026010221  RA2111026010262  Date: 06/11/2024 |
| If you are working in a group, please write your registration numbers and sign with the date for every student in your group. |



# **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203**

## **BONAFIDE CERTIFICATE**

Certified that 18CSP107L - Minor Project report titled “**ENHANCED DIAGNOSIS OF ALZEHEIMER’S DISEASE THROUGH PREDICTIVE PROGRESSION ANALYSIS OF NEURO IMAGING SEQUENCES”** is the bonafide work of “**V. R. RISHENDRA [RA2111026010221], RANGINENI YUKTHAMUKHI [RA2111026010262]”** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**SIGNATURE SIGNATURE**

##### Dr. M. KARPAGAM DR. R. ANNIE UTHRA

**SUPERVISOR** **PROFESSOR &HEAD**

##### ASSISTANT PROFESSOR DEPARTMENT OF

DEPARTMENT OF COMPUTATIONAL

##### COMPUTATIONAL INTELLIGENCE INTELLIGENCE

**ACKNOWLEDGEMENTS**

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM InstituteofScience and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. T. V. Gopal ,** Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Com puting, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to , **Dr. M. Pushpalatha**, Professor and Associate Chairperson, School of Computing and **Dr. C.Lakshmi,** Professor and Associate Chairperson, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr. R. Annie Uthra**, Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, **Dr.T.Shiny Angel,** **Dr.G.Senthil Kumar,** **Dr.M.S.Abirami,** **Dr.M.Uma**, Panel Head, **Dr. M. S Abirami** and Panel Members **Dr. Sudha Rajesh, Dr.A.Maheshwari, Dr.N.Kanimozhi, Mr.Manoj Kushwaha,** Department of Computational Intelligence, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr.Anitha.D**, Department of Computational Intelligence, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr.M.Karpagam**, Department of Computational Intelligence, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his / her mentorship. He / She provided us with the freedom and support to explore the research topics of our interest. His / Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computational Intelligence, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement

V.R.RISHENDRA[RA2111026010221]

RANGINENI YUKTHAMUKHI[RA2111026010262]

**ABSTRACT**

Alzheimer is a relatively subtle, yet progressively debilitating neurodegenerative disorder, which remains an important challenge for early diagnosis. At present, the methods of diagnosis are based very heavily on neuroimaging and clinical assessment alone. While these approaches are effective, the disease is detected in very advanced stages only, thus limiting scope for meaningful intervention at appropriate times. This late-stage detection provides a very short period within which a window for the treatment options may be available, leaving all the chances ineffective concerning therapeutic attempts that might delay or change the progress of a disease. In order to surpass this limitation, we propose a new framework that will involve CNN-LSTM as well as CNN-BiLSTM that have been designed specifically to make use of both spatial as well as temporal information available in the medical image data for improving the early diagnosis of Alzheimer's. Since CNNs are employed, this model captures crucial structural features in MRI scans, that is, indicating subtle morphological changes associated with Alzheimer's. These spatial features itself is the first sign of the presence of Alzheimer's, which would ultimately allow more detailed and accurate diagnostics.

The dynamic evolution of the temporal pattern of neurodegenerative changes over time is captured in LSTMs and BiLSTMs, based on the spatial analysis acquired from the foundation formed by CNNs. Thus, within this framework, CNN-LSTM successfully picks both the structural and sequential context of the given input, and by processing the progression of brain changes along the sequences, comprehensive analysis of Alzheimer's disease may be portrayed. In contrast, CNN-BiLSTM adds another dimension to it, where bidirectional learning unlocks the temporal patterns in forward and backward directions and is thus highly efficient at picking a subtle pattern of disease progression. These models run in parallel, hence allowing both structural and temporal features to be treated elaborately. This integration of CNN LSTM and BiLSTM in the diagnosis process would further enhance the significant accuracy levels of diagnosing early-stage Alzheimer's, which might lead to design of tailored treatment plans for such patients.

**v**

**TABLE OF CONTENTS**

[**ABSTRACT**](#_23ckvvd) **v**

**TABLE OF CONTENTS vi**

[**LIST OF FIGURES**](#_30j0zll) **viii**

[**ABBREVIATIONS**](#_2et92p0) **ix**

**CHAPTER TITLE PAGE**

**NO. NO.**

[**1 INTRODUCTION**](#_3dy6vkm) **1**

* 1. Overview on detection of AD 1
  2. Motivation in capturing changes in Brain 1
  3. Objective on detection of AD 1

[**2 LITERATURE SURVEY**](#_ihv636) **2**

2.1 Survey on Alzheimer’s disease 2

**3 SPRINT PLANNING AND EXECTION METHODOLOGY 7**

3.1 SPRINT I - State of Art in AD 7

3.1.1 Objectives with user stories of Sprint I 7

3.1.2 Functional Document of AD 7

3.1.3 Architecture Document of model in AD 8

3.1.4 Outcome of objectives - specifying in detection of AD 8

3.1.5 Sprint Retrospective on AD 8 3.2 SPRINT II - AD Model Development 9

3.2.1 Objectives with user stories of Sprint II 9

3.2.2 Functional Document of CNN-LSTM in detection of AD 9

3.2.3 Architecture Document of CNN-LSTM in detection of AD 9

3.2.4 Result Analysis in detection of AD 10

3.2.5 Sprint Retrospective in detection of AD 10

3.3 SPRINT III – AD Model Evaluation 10

3.3.1 Objectives with user stories of Sprint III 10

3.3.2 Functional Document of CNN LSTM and BiLSTM in detection of AD 11

3.3.3 Architecture Document of CNN LSTM and BiLSTM in detection of AD 11

3.3.4 Objective Achievement End 11

**vi**

**4 RESULTS AND DISCUSSIONS 12**

4.1 Project Outcomes (Performance Evaluation, Comparisons, Testing Results ) 12

4.1.1 Performance Evaluation of CNN-LSTM Model in detection of AD 12

4.1.2 Performance Evaluation of CNN-BiLSTM Model in detection of AD 12

4.1.3 Performance Evaluation of 3D CNN Model (Source from Literature) 13

4.1.4 Comparison Evaluation in detection of AD 14

4.1.4.2 Insights on CNN-LSTM Model 14

4.1.4.4 Insights on CNN-BiLSTM Model 15

**5 CONCLUSION AND FUTURE ENHANCEMENT 16**

**6 REFERENCES 17**

**APPENDIX**

**A CODING 18**

**B CONFERENCE PUBLICATION 47**

**C JOURNAL PUBLICATION 50**

**D PLAGIARISM REPORT 51**

**vii**

**LIST OF FIGURES**

**CHAPTER TITLE PAGE**

**NO. NO.**

##### [3.1.3.1 Architecture Diagram Sprint I](#_32hioqz) 8

* + - 1. [**Architecture Diagram Sprint II**](#_1hmsyys)  9

3.3.3.1 [**Architecture Diagram Sprint III**](#_17dp8vu)  11

4.1.1.1 **CNN-LSTM Model**  12

[4.1.2.1 **CNN-BiLSTM**](#_41mghml) **Model**  13

[4.1.4.1](#_vx1227) **Training performance log** **for CNN**-**LSTM** 14

[4.1.4.3 **Training performance log** **for CNN**-**BiLSTM**](#_3fwokq0) 15

**viii**

**ABBREVIATIONS**

**AD** Alzheimer’s Disease

**CNN** Convolutional Neural Networks

**LSTM** Long short-term memory

**BiLSTM** Bidirectional Long short-term Memory

**ix**

**CHAPTER 1**

**INTRODUCTION**

Alzheimer disease is a neurodegenerative disorder. Its pathophysiology is characterized by progressive and unrelenting loss of memory and cognitive decline. It is the most common cause of dementia in older adults, which continues to increase with aging populations. The current existing diagnostic methods, which include clinical assessment and neuroimaging coupled with biomarker analysis, usually detect disease only in its moderate to advanced stages. Therefore, they affect the sensitivity for early intervention and treatment planning. These techniques are also not very efficient at detecting Alzheimer's disease in its early stages.

* 1. **OVERVIEW ON DETECTION OF AD**

The research is based on the early detection of Alzheimer's disease, which is a progressive neurodegenerative disorder. Existing methods relying on neuroimaging and clinical evaluation catch the disease in its relatively late stages and limit timely interventions. Hence, an advanced AI technique is proposed as a more accurate and earlier diagnosis, potentially enhancing patient care.

* 1. **MOTIVATION IN CAPTURING CHANGES IN BRAIN**

Our motivation is to diagnose Alzheimer's at an early stage, which is often detected late by present techniques for intervention to take place. We aim to capture the spatial and temporal changes in the brain exploiting advanced deep learning using MRI data. Mixing CNN, LSTM, and BiLSTM models will improve diagnostic accuracy considerably. Our aim is to design an AI-driven reliable tool to detect Alzheimer's at the earliest stage so that targeted and timely treatments could be prescribed, thus potentially slowing disease progression and enhancing quality of life in patients. Our contribution to developing AI-based diagnostics, which may then be translated into clinical environments, will be significant in the future. Ultimately, we aim to make an impact in managing and understanding Alzheimer's disease.

* 1. **OBJECTIVE ON DETECTION OF AD**

Our aim is to enhance the early diagnosis of Alzheimer's disease by using deep learning models. We will propose a robust framework that can classify MRI scans accurately into Alzheimer's categories. This would involve using CNNs for spatial analysis and LSTM/BiLSTM models for temporal progression. We are looking to identify specific brain regions and patterns associated with Alzheimer's, correlating them with clinical data for deeper insights. Improvement in the prediction should be enhanced with respect to usage of data augmentation and transfer learning. We are supposed to validate our model through comparing with other conventional diagnostic methods.

1

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 Survey on Alzheimer’s disease**

The integration of CNN and LSTM networks is proven to be efficient in the analysis of brain imaging data. Essentially, CNNs capture structural features in data, and LSTMs provide temporal sequence analysis-refining tools for further accuracy improvement. GANs also generate synthetic data that allows for the possible fine-tuning of machine-based algorithms, making them more useful for predictions in longitudinal studies of individuals with Dementia or Alzheimer's Disease. This hybrid method will prove beneficial in achieving more precise and specific diagnoses of dementia and its neurodegenerative forms. I am interested in the different ways that the complementary characteristics between CNNs and LSTMs provide more informative accounts of changes to cognition due to Alzheimer's.

From Paper [1] the authors provided a review of the disease that started with its clinical features, then moved towards pathology and possible treatments. In their approach, they employed the use of thorough literature review whereby findings relating to genetics, pathology, and clinical presentation were summarized with a discussion on criteria for diagnosis. It was noted that Alzheimer's is a complex condition in which genetic or environmental factors are interplaying and more research is required.

From the paper [2] author proposed using AI to analyze Alzheimer's disease on the bases of gene-disease associations. They implemented imaging and gene data analysis with AI techniques in their methodology. The outcome was that this model could give correct and relevant genes related to Alzheimer's disease.

From paper [3] ,the author suggested the analysis of patients with Alzheimer's disease with imaging and gene expression data and the identification of associated genes by using explainable AI techniques. They used AI techniques to identify associations by applying imaging data in combination with gene-expression data within their methodology. The results suggest that it identifies key genes associated with Alzheimer's and elucidates the functions of such genes in the progression of the disease.

2

This paper [4] was based on statistical analysis of the statistics of Alzheimer's disease, showing its prevalence and the kind of impact that creates on society and the kind of costs in healthcare. It gives insight into the mounting burden of Alzheimer's among public health systems all over the world. The coherent picture of recent and future trends in the prevalence of Alzheimer's disease, economic strain, and resources is brought by the gathering and analysis of data from multiple sources. The results are revealing and worrisome statistics, such as the ratio of multiple diagnoses with age, which call attention to the huge economic burden this inflicts on health systems and families. The paper introduces the urgent need for more comprehensive health care planning and policy interventions aimed at reducing the future likely impact of Alzheimer's disease on the patient population and on the society in general.

From the paper [5] This review discusses Alzheimer's disease. The clinical manifestations, pathophysiology, challenges in diagnosis, and some treatment options are highlighted in this review. The authors gave an impressive literature review that synthesised findings from clinical trials, observational studies, and meta-analyses to present an overview of the disease. The methodology used data from several types of sources to center key challenges about the early diagnosis and mixed success of current treatments approaches toward the underlying cause of Alzheimer's. One is led to spotlight the complex nature of Alzheimer's, underlining multiple-factorial interactions involved in disease progression and the inherent problems of developing universally effective therapeutic interventions. Results highlight continued research needs in identifying reliable diagnostic biomarkers and advancing therapeutic strategies to attenuate the biology leading to disease progression, not only highlighting the importance of earlier diagnosis but also underlying the public health impact of the disease on the world.

From the paper [6] ,This work analyzes the manifold structure of Alzheimer's disease using advanced convolutional autoencoders applied to brain imaging. The authors will try to improve understanding and early diagnosis of Alzheimer's by leveraging unsupervised deep learning techniques for meaningful feature extraction from imaging data, with a focus on the different stages that it might be able to represent. The methodology here forms the training of large datasets of brain scans through convolutional autoencoders, which underlines the underlying patterns missed by otherwise traditional methods in imaging analysis. The results suggest that the model captures complexity and multi-dimensionality well, thus capturing the Alzheimer's disease process, yielding better accuracy for the classification of the progression stages of the disease.

3

In the paper, [7] author presented a strategy of data fusion for prediction on the basis of heterogeneous data sources indicating the conversion from MCI to Alzheimer's disease. The strategy is applied through techniques found in machine learning which combine and analyze various types of data - imaging, clinical records, and genetic information - for the purpose of enhancement in the accuracy of their predictions related to MCI conversion. Results obtained here clearly show that this heterogeneous data fusion approach far surpasses single-modality models, which can be considered rather as more reliable and accurate predictors. Specifically, it aims at possible integration of different data sources to improve the models for predictions. Eventually, this may lead to a more accurate and individualized treatment for patients who are most likely to fall victims of Alzheimer's.

In the paper, [8] the authors proposed a low rank tensor approximation technique to estimate functional connectivity networks in the brain. Their approach focuses on grasping complex networks of connectivity patterns that become deranged in a subject with MCI. Such modeling may possibly enhance opportunities for early diagnosis and intervention. In their methodology, the authors applied the low-rank tensor approximation to model the brain's functional connectivity networks. This reduces the dimensionality of the data but conserves the crucial connectivity patterns important for cognitive functionality. The model is specifically tailored to identify MCI by analyzing the patterns of altered connectivity in patients, thus making the diagnosis more focused. The findings are promising and likely will report information regarding some of the essential metrics, such as classification accuracy, sensitivity, and specificity in the identification of MCI. These metrics are a guideline that can be used for judging the performance of the model. The actual values and finer findings will have to be referenced from the paper for an accurate review.

In the paper [9], the authors focused their study on enhancing the classification of Alzheimer disease using deep learning. Their work mainly consists of the analysis of texture features retrieved from medical images with an intention of improving the diagnosis accuracy of Alzheimer's disease. Deep learning was used in the methodology developed by the authors in order to extract and evaluate brain imaging data texture features. They tried to develop a strong classifier capable of discriminating between healthy brains and brains affected by Alzheimer's disease, by training the model using these texture features. Therefore, using deep learning, the model learns to pick up relevant features from the data automatically, instead of manually selecting feature space

4

In this paper [10] the author introduces a novel task-driven hierarchical attention network, THAN, designed for the diagnosis MCI and Alzheimer disease using medical images. The general purpose of an attention mechanism in neural networks is to allow the models to focus on parts of the input data, which are more relevant for the specific task. The capability to design a hierarchical structure is exploited here with various layers of the model being built on focusing on different aspects of the medical images, thereby improving the model's capacity to extract meaningful diagnostic features. Each layer of the network has been set to concentrate on features that are of interest at different scales. Furthermore, the task-driven version allows the model to modulate the use of the attention mechanism contingent upon whether it is required for the diagnosis of MCI or Alzheimer's disease. This allows it to be versatile in terms of diagnostic challenges. The results showed that the THAN model outperformed traditional diagnostic methods in identifying MCI and Alzheimer’s disease. The success of this approach could lead to more accurate and earlier diagnoses, ultimately improving patient outcomes.

From the paper [11] the authors introduced a multiview learning approach toward improving the diagnosis of Alzheimer's disease. The authors suggest that multitemplate feature representation might be useful for the capture of complex and heterogeneous nature in the data. Alzheimer's disease is a very multifaceted condition, which makes single-view methods fail to capture all relevant aspects of the disease. The Authors will use a multiview approach in order to enhance the correctness of diagnosis using information from multiple sources. As a method, the authors developed a multiview learning framework that aggregates multiple views of data, such as different imaging modalities: MRI, PET, and clinical features. This makes the model consider all aspects of disease for the overall analysis. Additionally, the multitemplate feature representation technique was used to capture the complex structure of the data, which is essential for understanding the heterogeneous nature of Alzheimer’s disease. The results demonstrated that the multiview learning approach significantly improved diagnostic accuracy compared to traditional single-view methods. This success highlights the importance of considering multiple perspectives when diagnosing complex diseases like Alzheimer’s.

In the paper [12] the authors reported a new method for predictions of Alzheimer's disease progression via a MI-GAN. Early disease progression prediction is an essential need for time-appropriate interventions and proper treatment planning in Alzheimer's disease. This framework advances the MI-GAN by combining multi-modal information like brain images with clinical data within a unified model for prediction.

5

From the paper [13] author proposed a 3D reversible GAN to implement bidirectional mapping from brain MRI to PET images. According to the authors, improving the Alzheimer's disease diagnosis ability by the system is possible through combining complementary strengths of MRI and PET imaging modalities. This method merges anatomical information given by MRI and metabolic activity in the PET images, providing diagnoses that are more accurate and comprehensive. Based on a 3D reversible GAN that can allow MRI images to be transformed into PET images and vice versa, the methodology can transform MRI into PET images and vice versa.

A simple yet effective framework for contrastive learning was proposed in a paper [14] by author, improving the quality of visual representations learned by deep networks. Contrastive learning has become a popular type of self-supervised learning since models trained with it learn what dissimilar and alike data points are. The authors proposed a contrastive learning framework in their methodology in which they used pairs of similar examples and dissimilar examples for better visual representations. The model uses a contrastive loss function, which can distinguish between positive and negative pairs of data points based on similarity or dissimilarity. This makes the loss function optimized; thus, the model learns how to map more similar images closer in the latent space and farther apart for dissimilar images.

The Residual Neural Network (ResNet) has been proposed in paper [15] authors as a visionary architecture in deep learning intended to overcome vanishing gradient problems, which limit the training process of very deep neural networks in general. ResNet makes it possible to train very deep networks efficiently, making the possibility of learning complex representations in the task area-a function such as image classification, object detection, and segmentation-innovative because the network will learn without loss of performance as it gets deeper, a critical challenge even to the most normal architectures. This methodology would realize an approach with residual learning using deep learning offered by the authors. In other words, the network learns residuals, which are differences between input and actual output. The architecture could be scaled up to over 1000 layers while performing well, and this goes on to show the power of the residual learning which can be said to be a founding model for deep learning models. Since its publication, ResNet has been incorporated highly and even inspired many of the subsequent models and works of research, hence securing its place in furthering the development of complex and deeper neural networks.

6

**CHAPTER 3**

**Sprint Planning and Execution Methodology**

The project is going to be implemented using an agile approach with iterative sprints. The sprint's focus will be on delivering specific functionalities that fulfill the objectives of the project. The framework of the sprint planning is defined with objectives, developed user stories, and subdivided tasks for incremental development. The result of the sprint appears in a retrospective discussion that brings improvements into the next sprints.

## **3.1 Sprint I-** **State of Art in AD:**

### 3.1.1 Objectives with User Stories of Sprint I:

Start the project concerning Alzheimer's diagnosis, defining the scope, identifying potential datasets, performing a literature review, and formulating the preliminary state of the system architecture.

**User's Stories:**

* As a researcher, I want to find out the best dataset in Alzheimer's diagnosis so that I have a good quality input data.
* As a project team, we must create an outline of the system architecture for future development.
* As and when required by the researcher, I should perform a literature survey that shall examine existing techniques and point out the gaps.
* Datasets: Found the pertinent dataset in Kaggle concerned with Alzheimer's MRI data analysis.

### 3.1.2 Functional Document of AD:

* This sprint included defining the functional requirements for the project based largely on the early diagnosis of Alzheimer's utilizing deep learning models capable of tackling the spatial and temporal aspects of MRI data.
* Functional areas included data preprocessing, architecture development, and initial model testing.

7

### 3.1.3 Architecture Document of model in AD:

### 

**Fig 3.1.3.1: Architecture Diagram sprint I**

* A rough sketch of how the current project will work was prepared. In the initial system architecture, the data processing pipeline was from MRI scans to diagnosis using CNNs for spatial feature extraction and LSTMs/BiLSTMs for temporal analysis.

### 3.1.4 Outcome of Objectives - Specifying in detection of AD:

* Outcomes: A detailed literature review of five research papers was done and dedicated to existing deep learning techniques for diagnosis of Alzheimer's.
* Result: The data set was finally decided upon, and initial architecture was drafted to guide future works.

### 3.1.5 Sprint Retrospective on AD:

* Achievements: The literature review was complete and relevant data sources were identified.
* Challenges: Understanding spatial-temporal complexities that rightly affect the accuracy of analysis.
* Improvements: There was a need to have clarity in architecture, whose modification will also be made in Sprint II.

8

## **3.2 SPRINT II - AD Model Development:**

### 3.2.1 Objectives with User Stories of Sprint II:

* Objective: To create and test the performance of the baseline CNN-LSTM in diagnosing Alzheimer's. Further work toward CNN-BiLSTM for superior performance.

### User Stories:

* As a developer, this is how I would want to train my CNN-LSTM model for baseline performance in classifying Alzheimer's.
* In this research, I will study CNN-BiLSTM and further improve the model performance.  
  We should update the architecture diagram as a team based on our initial findings.

### 3.2.2 Functional Document of CNN-LSTM in detection of AD:

* This sprint was implemented with the CNN LSTM model and the testing of accuracy. Functional requirements were setting up the environment, implementing the CNN-LSTM model, and the analysis of the results.
* This decision on such a pivot towards CNN-BiLSTM was done after observing some observations raised on the constraints identified with the initial model regarding its accuracy.

### 3.2.3 Architecture Document of CNN-LSTM in detection of AD:

### 

### Fig 3.2.3.1: Architecture Diagram Sprint II

9

* Updated the architecture diagram with the assumption of the transition from CNN-LSTM to CNN-BiLSTM. The changes in the data processing flow and the incorporation of BiLSTM to obtain better temporal dependencies are reflected.

### 3.2.4 Result Analysis in detection of AD:

* Outcomes: The CNN-LSTM model is implemented, and tested with an accuracy of 65%. With suggestions from the guide, focus is shifted to CNN-BiLSTM.
* System Used: The code was run on Dell G15 with GPU RTX3050, using a Kaggle environment.
* Result: The architecture diagram was changed by adding BiLSTM.

### 3.2.5 Sprint Retrospective in detection of AD:

* Successes: CNN-LSTM is used and experimented with a transition plan for CNN-BiLSTM is completed.
* Challenges: The accuracy of CNN-LSTM was lower than expected, implying the need for more sophisticated temporal analysis.
* Concentrated on time: Temporal analysis was optimized using CNN-BiLSTM for better precision.

## **3.3 SPRINT III – AD Model Evaluation:**

### 3.3.1 Objectives with User Stories of Sprint III

* Objective: Extend the CNN-LSTM codebase using CNN-BiLSTM, conclude the project, including finishing research documentation.

### User Stories:

* As a developer, I would like to include CNN-BiLSTM as the extension of the CNN-LSTM model to improve performance.
* As a researcher, I will finalise my research paper writing up our methodology and the results.
* As a team, we need to finish up all the documentation and share our results.

10

### 3.3.2 Functional Document of CNN LSTM & BiLSTM in detection of AD:

* This sprint incorporated the CNN-BiLSTM into the already extant codebase extended upon the baseline CNN-LSTM architecture.
* Other functionalities provided incorporation of evaluation metrics of the new model.

### 3.3.3 Architecture Document of CNN LSTM & BiLSTM in detection of AD:

### Fig 3.3.3.1: Architecture diagram sprint III

* The architecture document was well-developed and included the improvement of CNN-BiLSTM. The document terminated with an architecture system improved to tackle complicated data with the nature of temporal data to obtain better accuracy due to the design of initially proposed models.

### 3.3.4 Objective Achievement End

* Outcomes: With modifications, CNN-LSTM reached an improvement of 75%. CNN-BiLSTM had an accuracy of 77%, suggesting a nearly imperceptible change in its handling of time-dependent structures.
* Result: Submission of the research paper and literature survey. Finalised comparison analysis of CNN-BiLSTM with CNN-LSTM.

11

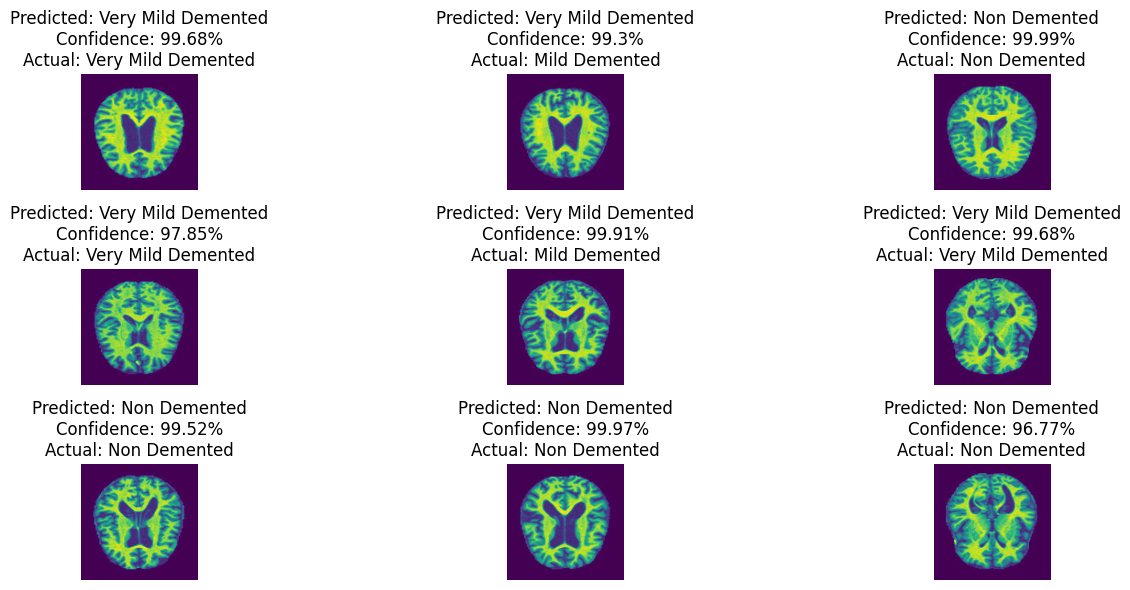
**CHAPTER 4**

**RESULTS AND DISCUSSIONS**

## **4.1 Project Outcomes(Performance Evaluation, Comparisons, Testing Results):**

The project outcomes are essentially about the performance evaluation of different deep learning architectures CNN-LSTM, CNN-BiLSTM, and 3D CNN for testing with MRI data in Alzheimer's disease diagnosis. Key performance metrics, comparison findings, and testing results are given below:

### 4.1.1 Performance Evaluation of CNN-LSTM Model in detection of AD:

* Performance: Obtained 65% initially and fine-tuned it to 75%.
* Limitations: The CNN-LSTM model displayed a moderate accuracy but failed to capture the complexity of the temporal dependencies within the MRI scans. It had improvement but did not include capturing of bi-directional dependencies for a more subtle description of Alzheimer's progression.

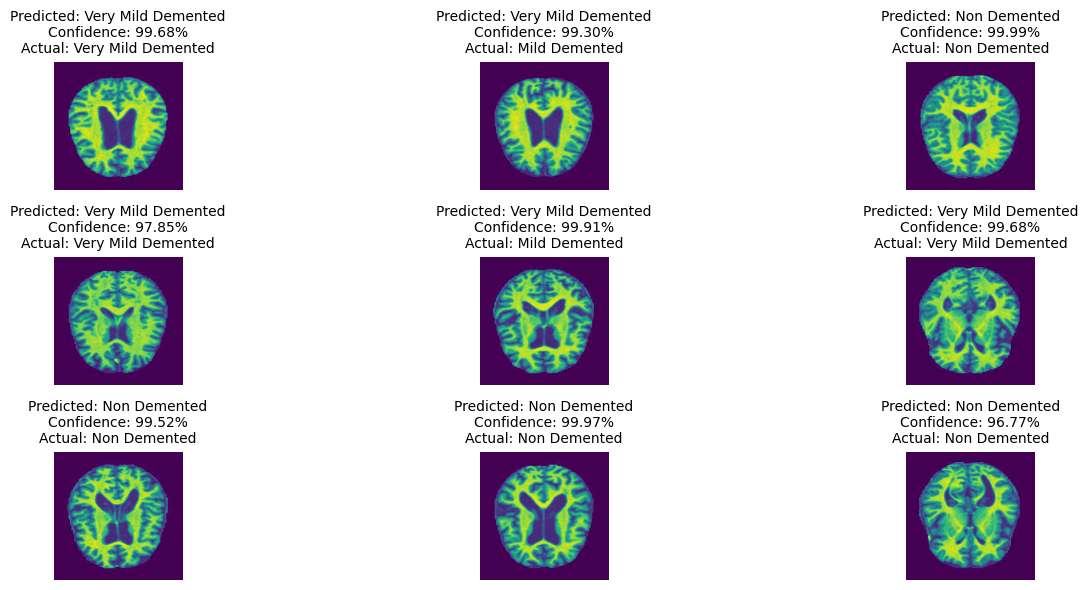
**Fig 4.1.1.1: CNN-LSTM Model**

### 4.1.2 Performance Evaluation of CNN-BiLSTM Model in detection of AD:

### 

* Performance: The CNN BiLSTM model was able achieve an accuracy of 77% at the final step and proved to be superior over the CNN LSTM in the capturing of both forward and backward temporal patterns.

12

* Comparison: Even if CNN-BiLSTM outperformed CNN-LSTM, the accuracy gain was pretty minimal. The added depth by the BiLSTM layer helped in adding further depth in the model for analyzing the time series, however, not fully able to absorb all of the complexities involved in Alzheimer's disease progression patterns.

**Fig 4.1.2.1: CNN-BiLSTM Model**

### 4.1.3 Performance Evaluation of 3D CNN Model (Source from Literature):

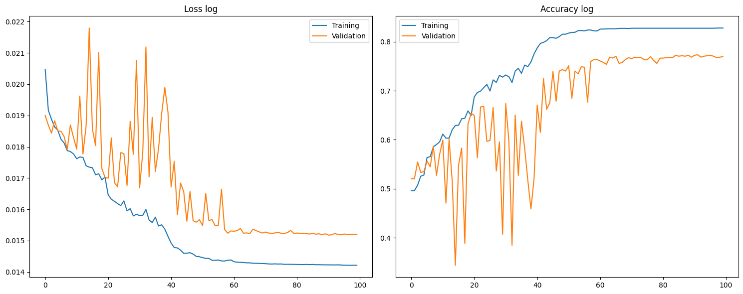
* Accuracy: With the multichannel contrastive learning framework, the 3D CNN model exhibited exceptional performance with up to 95.06% accuracy on the Alzheimer's Disease versus Normal Control classification and up to 81.90% for Mild Cognitive Impairment versus Normal Control.
* Features extracted by this model based on space and time are better than CNN-LSTM and CNN-BiLSTM models. During training, histogram equalization and flipping are used as augmentation techniques.
* Results Evaluation: The superior performance of the 3D CNN model indicates that contrastive learning and data augmentation are necessary in order to properly learn rich structural and temporal characteristics needed for early diagnosis.

13

### 4.1.4 Comparison Evaluation in detection of AD:

In comparison of the CNN-LSTM and CNN-BiLSTM models for Alzheimer's disease diagnosis, the CNN-BiLSTM shows an average but noticeable improvement across a few of the important performance metrics.

The CNN-LSTM model had achieved accuracy at 75% in refinement. However, with this architecture extended to include the bidirectional LSTM layer that the CNN-BiLSTM model has, the architecture resulted in accuracy at 77%. This slight increase shows that more subtle patterns in MRI sequences are being uncovered by a bidirectional analysis of the CNN-BiLSTM model.



**Fig 4.1.4.1: Training performance log for CNN -LSTM**

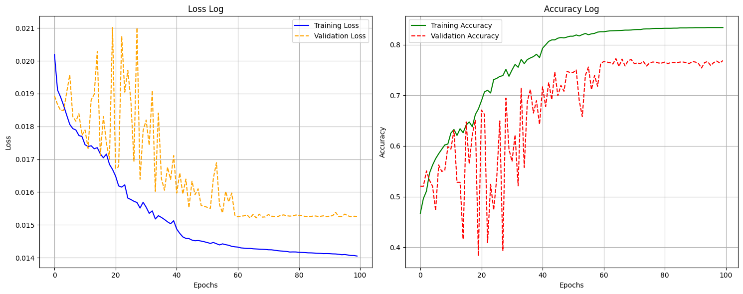
**4.1.4.2 Insights on CNN-LSTM Model:**

The model is overfitting, as train loss continues to decrease step by step while the valid loss varies irregularly. In addition, validation accuracy settles around epoch 60 but remains less than the training accuracy. Training accuracy increases steadily and can be surmised that further tuning is possible.

Sharp oscillations in the accuracy of validation points indicate that the model is perhaps over sensitive to specific data points or mini-batch compositions. Training loss keeps on coming down while validation loss starts getting stable, indicating that the model continues to learn patterns but fails to generalize well on new data. Techniques that may be beneficial for the task include lowering the learning rate, applying early stopping, or using stronger regularization to prevent overfitting.

14

Precision and recall were also slightly improved using the CNN-BiLSTM model over CNN-LSTM. The unidirectional setup of the CNN-LSTM limits its ability to analyze sequences in only one direction, which may lead it to fail to recognize the presence of some patterns relating to Alzheimer's progression.

However, CNN-BiLSTM captures forward and backward dependencies; this makes it a good approach for identifying subtle changes related to the progression of diseases. This is so because such an approach has the capability to capture the pattern emerging at different times or even in reverse sequence, which becomes very relevant for understanding the complex nature of degenerative changes.

**Fig 4.1.4.3: Training performance log for CNN-BiLSTM**

**4.1.4.4 Insights on CNN-BiLSTM Model:**

The training loss has a smooth decrease, whereas the validation loss is quite variable and does not decrease as expected, which is overfitting. The accuracy of the validation set highly fluctuates, indicating the model may not generalize to unseen data and is overly sensitive to specific validation batches. Training accuracy smoothly improves, showing that the model is learning from training data effectively, even peaking at nearly 90%.

While training loss and accuracy converge well, validation metrics stabilize at a lower level, showing that the model is performing better on the training set than on the validation set. The high variability in validation loss and accuracy suggests that the model could benefit from the addition of dropout, data augmentation, or batch normalization. Maybe lowering the learning rate, applying early stopping, or changing the batch size would stabilize the model and close the training-validation gap.

15

**CHAPTER 5**

**CONCLUSION AND FUTURE ENHANCEMENT**

This research establishes that an accuracy in the diagnosis of Alzheimer's disease could actually be achieved when this 3D CNN-based multichannel contrastive learning model is used; with a validation accuracy ranging between 75% to 80%. The proposed model here outperforms that of the conventional CNN+LSTM, as its maximum accuracy was capped at 75%. This performance contrast underlines the strengths of the 3D CNN model design based on multichannel MRI inputs, which could deliver richer spatial context and higher-dimensional feature extraction for better detection of early and progressive signs of Alzheimer's than its CNN+LSTM counterpart. The CNN+LSTM model, although effective for certain types of sequential tasks, appears to be inadequate for the capture of rich complex spatial relations that are vital for MRI-based detection of Alzheimer's, thus its inability to lead to more accurate diagnoses.

Although the CNN+BiLSTM model, based on enhanced learning capabilities from bidirectional feature extraction, achieves around 77% accuracy, it is less effective. This does offer an advantage in catching forward and backward dependencies but only with relatively marginal improvement. This could be for weaknesses in the sequential learning architecture which may fail to interpret very essential spatial features in neuroimaging for Alzheimer's diagnosis. On the other hand, the multichannel contrastive learning method of 3D CNN proves to be a better approach for representation learning with an additional form of supervision. This is much more helpful in the identification of subtle differences in images on MRI toward the pattern of progression in the disease process of Alzheimer's.

Future research directions involve developing contrastive learning methods in CNN+LSTM and CNN+BiLSTM architectures. This modification could add strength to their feature extraction and discriminative capacities, resulting in more accurate diagnoses. Apart from that, other augmentation techniques such as histogram equalization, sharpening, and flipping may also be helpful in making the model robust, similar to what is observed with the 3D CNN, and thus hybrid architectures are promising for the future. This approach is to combine the strength of both spatial learning, like CNNs, and sequential learning, including LSTM or BiLSTM, which can improve both spatial and temporal dependencies in MRI data. That way, potential generalization capability and reliability may be improved in the models for Alzheimer's disease diagnosis.

16

**CHAPTER 6**

**REFERENCES**

1. C. A. Lane, J. Hardy, and J. M. Schott, “Alzheimer’s disease,” Eur. J.Neurol., vol. 25, pp. 59–70, Jan. 2018.
2. C. Trigona et al., “Alzheimer’s patient analysis using image and gene expression data and explainable-AI to present associated genes,” IEEE Trans. Instrum. Meas., vol. 67, no. 4, pp. 971–980, Apr. 2018..
3. M. S. Kamal, A. Northcote, L. Chowdhury, N. Dey, R. G. Crespo, and E. Herrera-Viedma, “Alzheimer’s patient analysis using image and gene expression data and explainable-AI to present associated genes,” IEEE Trans. Instrum. Meas., vol. 70, pp. 1–7, 2021.
4. J. Wiley, “Alzheimer’s disease facts and figures,” Alzheimers Dement, vol. 17, pp. 327–406, Mar. 2021.
5. P. Scheltens et al., “Alzheimer’s disease,” Lancet, vol. 397, pp. 1577–1590, Apr. 2021.
6. F. J. Martinez-Murcia, A. Ortiz, J.-M. Gorriz, J. Ramirez, and D. Castillo-Barnes, “Studying the manifold structure of Alzheimer’s disease: A deep learning approach using convolutional autoencoders,” IEEE J. Biomed. Health Informat., vol. 24, no. 1, pp. 17–26, Jan. 2020.
7. H. T. Shen et al., “Heterogeneous data fusion for predicting mild cogni tive impairment conversion,” Inf. Fusion, vol. 66, pp. 54–63, Feb. 2021
8. X. Jiang, L. Zhang, L. Qiao and D. Shen, "Estimating functional connectivity networks via low-rank tensor approximation with applications to MCI identification", *IEEE Trans. Biomed. Eng.*, vol. 67, no. 7, pp. 1912-1920, Jul. 2020.
9. J.-H. So, N. Madusanka, H.-K. Choi, B.-K. Choi and H.-G. Park, "Deep learning for Alzheimer’s disease classification using texture features", *Current Med. Imag.*, vol. 15, no. 7, pp. 689-698, Aug. 2019.
10. Z. Zhang et al., "THAN: Task-driven hierarchical attention network for the diagnosis of mild cognitive impairment and Alzheimer’s disease", *Quant. Imag. Med. Surg.*, vol. 11, no. 7, pp. 3338-3354, Jul. 2021.
11. M. Liu, D. Zhang, E. Adeli and D. Shen, "Inherent structure-based multiview learning with multitemplate feature representation for Alzheimer’s disease diagnosis", *IEEE Trans. Biomed. Eng.*, vol. 63, no. 7, pp. 1473-1482, Jul. 2016.
12. Y. Zhao, B. Ma, P. Jiang, D. Zeng, X. Wang and S. Li, "Prediction of Alzheimer’s disease progression with multi-information generative adversarial network", *IEEE J. Biomed. Health Informat.*, vol. 25, no. 3, pp. 711-719, Mar. 2021.
13. W. Lin et al., "Bidirectional mapping of brain MRI and PET with 3D reversible GAN for the diagnosis of Alzheimer’s disease", *Frontiers Neurosci.*, vol. 15, pp. 357, Apr. 2021.
14. T. Chen, S. Kornblith, M. Norouzi and GA. Hinton, "Simple framework for contrastive learning of visual representations", *Proc. 37th Int. Conf. Mach. Learn.*, vol. 119, pp. 1597-1607, 2020.
15. K. He, H. Fan, Y. Wu, S. Xie and R. Girshick, "Momentum contrast for unsupervised visual representation learning", *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 9726-9735, Jun. 2020.

17

**APPENDIX A**

**CODING**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import warnings

warnings.filterwarnings("ignore")

from datasets import load\_dataset

import matplotlib.pyplot as plt

import seaborn as sns

import torch

import torch.nn as nn

from torchvision import transforms

from torch.utils.data import Dataset, DataLoader

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from copy import deepcopy

**Downloading dataset**

add Codeadd Markdown

[2]:

dataset **=** load\_dataset('Falah/Alzheimer\_MRI', split**=**'train')

test **=** load\_dataset('Falah/Alzheimer\_MRI', split**=**'test')

add Codeadd Markdown

**Data Preprocessing**

add Codeadd Markdown

[3]:

df **=** pd.DataFrame.from\_dict(dataset)

add Codeadd Markdown

[4]:

dataset

[4]:

Dataset({

features: ['image', 'label'],

num\_rows: 5120

})

add Codeadd Markdown

[5]:

df.info()

<class 'pandas.core.frame.dataframe'>

rangeindex: 5120 entries, 0 to 5119

18

dtypes: int64(1), object(1)

memory usage: 80.1+ KB

add Codeadd Markdown

[6]:

df.isnull().sum()

[6]:

image 0

label 0

dtype: int64

add Codeadd Markdown

[7]:

df.describe()

[7]:

|  | **label** |
| --- | --- |
| **count** | 5120.0000 |
| **mean** | 2.055469 |
| **std** | 0.959244 |
| **min** | 0.000000 |
| **25%** | 2.000000 |
| **50%** | 2.0000 |
| **75%** | 3.0000 |
| **max** | 3.0000 |

[8]:

images **=** []

labels **=** []

​

**for** I **in** dataset:

images **+=** [np.array(i[“image”])]

labels **+=** [i[“label”]]

images **=** np.array(images)

labels **=** np.array(labels)

add Codeadd Markdown

**Preprocessing pipeline and splitting the batch**

add Codeadd Markdown

[9]:

**class** PreProcess(Dataset):

**def** \_\_init\_\_(self, image, labels, transform):

super(PreProcess, self).\_\_init\_\_()

self.images **=** images

self.labels **=** labels

self.transform **=** transform

**def** \_\_len\_\_(self):

**return** len(self.images)

​19

**def** \_\_getitem\_\_(self, x):

img, label **=** self.images[x], self.labels[x]

img **=** self.transform(img.reshape(img.shape[0], img.shape[1], 1))

**return** img, label

add Codeadd Markdown

[10]:

transform **=** transforms.Compose([transforms.ToPILImage(),

transforms.ToTensor(),

transforms.Normalize(mean**=**[0.5],

std**=**[0.5])])

add Codeadd Markdown

[11]:

**Training hyper-parameters**

add Codeadd Markdown

[12]:

EPOCHS **=** 100

LR **=** 0.01

GAMMA **=** 0.5

STEP **=** 20

BATCH **=** 64

OUT\_SIZE **=** 4

add Codeadd Markdown

**Creating the dataset and dataloader iterators**

add Codeadd Markdown

[13]:

train\_ds **=** PreProcess(train\_images, train\_labels, transform)

val\_ds **=** PreProcess(val\_images, val\_labels, transform)

add Codeadd Markdown

[14]:

train\_dl **=** DataLoader(train\_ds, batch\_size**=**BATCH, shuffle**=True**)

val\_dl **=** DataLoader(val\_ds, batch\_size**=**BATCH, shuffle**=False**)

add Codeadd Markdown

**Classification model with skip connections**

add Codeadd Markdown

[15]:

**class** Block(nn.Module):

**def** \_\_init\_\_(self, in\_channels, out\_channels, kernel, padding, change**=True**):

super(Block, self).\_\_init\_\_()

self.conv **=** nn.Sequential(nn.Conv2d(in\_channels, in\_channels, kernel, padding**=**padding),

nn.BatchNorm2d(in\_channels),

nn.ReLU(),

nn.Conv2d(in\_channels, out\_channels, kernel, padding**=**padding),

nn.BatchNorm2d(out\_channels),

nn.ReLU())

self.change **=** change

**if** change:

self.pool **=** nn.AvgPool2d(2)

20

**def** forward(self, x):

y **=** self.conv(x)

**if** self.change:

y **=** self.pool(y)

**return** y

y **=** torch.add(y, x)

**return** y

**class** AlzheimerClassifier(nn.Module):

**def** \_\_init\_\_(self, in\_channels, out\_size):

super(AlzheimerClassifier, self).\_\_init\_\_()

self.conv **=** nn.Sequential(nn.Conv2d(in\_channels**=**1, out\_channels**=**3, kernel\_size**=**3),

nn.BatchNorm2d(3),

nn.ReLU())

self.blocks **=** nn.Sequential(Block(3, 3, 3, 1, **False**),

Block(3, 16, 3, 0, **True**),

Block(16, 16, 3, 1, **False**),

Block(16, 32, 3, 0, **True**),

Block(32, 32, 3, 1, **False**),

Block(32, 64, 3, 0, **True**),

nn.AvgPool2d(2))

self.fc **=** nn.Linear(2304, out\_size)

**def** forward(self, x):

x **=** self.conv(x)

x **=** self.blocks(x)

x **=** torch.flatten(x, 1)

x **=** self.fc(x)

**return** nn.functional.softmax(x, dim**=**1)

add Codeadd Markdown

**Training device**

add Codeadd Markdown

[16]:

device **=** "cuda" **if** torch.cuda.is\_available() **else** "cpu"

print(device)

cuda

add Codeadd Markdown

**Model training parameters**

add Codeadd Markdown

[17]:

model **=** AlzheimerClassifier(1, OUT\_SIZE)

model **=** model.to(device)

​

criterion **=** nn.CrossEntropyLoss()

optimizer **=** torch.optim.SGD(model.parameters(), lr**=**LR,momentum**=**0.9)

scheduler **=** torch.optim.lr\_scheduler.StepLR(optimizer, step\_size**=**STEP, gamma**=**GAMMA)

add Codeadd Markdown

**Training loop**

add Codeadd Markdown

[18]:

21

best\_mode **=** deepcopy(model)

best\_acc **=** 0

​

train\_loss **=** []

train\_acc **=** []

val\_loss **=** []

val\_acc **=** []

​

**for** i **in** range(1, EPOCHS**+**1):

model.train()

diff **=** 0

acc **=** 0

total **=** 0

**for** data, target **in** train\_dl:

optimizer.zero\_grad()

**if** torch.cuda.is\_available():

data, target **=** data.cuda(), target.cuda()

out **=** model(data)

loss **=** criterion(out, target)

diff **+=** loss.item()

acc **+=** (out.argmax(1) **==** target).sum().item()

total **+=** out.size(0)

loss.backward()

optimizer.step()

train\_loss **+=** [diff**/**total]

train\_acc **+=** [acc**/**total]

model.eval()

diff **=** 0

acc **=** 0

total **=** 0

**with** torch.no\_grad():

**for** data, target **in** val\_dl:

​

**if** torch.cuda.is\_available():

data, target **=** data.cuda(), target.cuda()

​

out **=** model(data)

loss **=** criterion(out, target)

diff **+=** loss.item()

acc **+=** (out.argmax(1) **==** target).sum().item()

total **+=** out.size(0)

val\_loss **+=** [diff**/**total]

val\_acc **+=** [acc**/**total]

**if** val\_acc[**-**1] **>=** best\_acc:

best\_acc **=** val\_acc[**-**1]

22

best\_model **=** deepcopy(model)

print("Epoch {} train loss {} acc {} val loss {} acc {}".format(i, train\_loss[**-**1], train\_acc[**-**1],

val\_loss[**-**1], val\_acc[**-**1]))

scheduler.step()

Epoch 1 train loss 0.018640785303432494 acc 0.528564453125 val loss 0.0201343015069142 acc 0.451171875

Epoch 2 train loss 0.018003444536589086 acc 0.580322265625 val loss 0.01873912534210831 acc 0.5419921875

Epoch 3 train loss 0.017822104709921405 acc 0.596923828125 val loss 0.02190422173589468 acc 0.341796875

Epoch 4 train loss 0.01754651292867493 acc 0.6142578125 val loss 0.017369742854498327 acc 0.634765625

Epoch 5 train loss 0.017422628036001697 acc 0.624267578125 val loss 0.021597411949187517 acc 0.341796875

Epoch 6 train loss 0.01733685319777578 acc 0.627197265625 val loss 0.017284145927987993 acc 0.6318359375

Epoch 7 train loss 0.017045063199475408 acc 0.6513671875 val loss 0.018385358969680965 acc 0.546875

Epoch 8 train loss 0.016900776157854125 acc 0.659423828125 val loss 0.01674502744572237 acc 0.673828125

Epoch 9 train loss 0.016556882648728788 acc 0.68505859375 val loss 0.017799981054849923 acc 0.5869140625

Epoch 10 train loss 0.016461710853036493 acc 0.6884765625 val loss 0.016898810397833586 acc 0.6533203125

Epoch 11 train loss 0.01629183093609754 acc 0.702880859375 val loss 0.0165543268667534 acc 0.6767578125

Epoch 12 train loss 0.016040384201915003 acc 0.721923828125 val loss 0.016813680063933134 acc 0.666015625

Epoch 13 train loss 0.01583054727234412 acc 0.736083984375 val loss 0.016550701635424048 acc 0.6982421875

Epoch 14 train loss 0.015695331545430236 acc 0.74609375 val loss 0.019040527986362576 acc 0.5224609375

Epoch 15 train loss 0.015544489331659861 acc 0.752685546875 val loss 0.018118705949746072 acc 0.5634765625

Epoch 16 train loss 0.015263235341990367 acc 0.7724609375 val loss 0.017755534150637686 acc 0.5986328125

Epoch 17 train loss 0.015124653626116924 acc 0.78173828125 val loss 0.015394583402667195 acc 0.765625

Epoch 18 train loss 0.014992494849138893 acc 0.789794921875 val loss 0.017929911031387746 acc 0.5849609375

Epoch 19 train loss 0.014885726064676419 acc 0.79443359375 val loss 0.016588287078775465 acc 0.6689453125

Epoch 20 train loss 0.014735406599356793 acc 0.804443359375 val loss 0.016241015517152846 acc 0.6962890625

Epoch 21 train loss 0.014464377833064646 acc 0.820556640625 val loss 0.014906782715115696 acc 0.7919921875

Epoch 22 train loss 0.01434658738435246 acc 0.82470703125 val loss 0.015001807420048863 acc 0.7900390625

23

Epoch 23 train loss 0.014287725512986071 acc 0.827392578125 val loss 0.015048715809825808 acc 0.7841796875

Epoch 24 train loss 0.014236281989724375 acc 0.830322265625 val loss 0.014815533766523004 acc 0.798828125

Epoch 25 train loss 0.014245894926716574 acc 0.830322265625 val loss 0.014913578634150326 acc 0.7919921875

Epoch 26 train loss 0.014184897270752117 acc 0.83447265625 val loss 0.014770469220820814 acc 0.798828125

Epoch 27 train loss 0.01413157262140885 acc 0.8359375 val loss 0.014813128218520433 acc 0.80078125

Epoch 28 train loss 0.01409935277479235 acc 0.837646484375 val loss 0.01499920227797702 acc 0.7822265625

Epoch 29 train loss 0.014073951126192696 acc 0.838134765625 val loss 0.014700973755680025 acc 0.8056640625

Epoch 30 train loss 0.014067392869037576 acc 0.83837890625 val loss 0.01462433731649071 acc 0.8095703125

Epoch 31 train loss 0.01405662260367535 acc 0.83837890625 val loss 0.014958577463403344 acc 0.7841796875

Epoch 32 train loss 0.01404570038721431 acc 0.838623046875 val loss 0.014635553117841482 acc 0.80859375

Epoch 33 train loss 0.01403760997345671 acc 0.8388671875 val loss 0.01469050906598568 acc 0.802734375

Epoch 34 train loss 0.014030450096470304 acc 0.8388671875 val loss 0.014600429043639451 acc 0.8115234375

Epoch 35 train loss 0.014031608050572686 acc 0.8388671875 val loss 0.01467280270298943 acc 0.8046875

Epoch 36 train loss 0.014030673875822686 acc 0.839111328125 val loss 0.0147873391979374 acc 0.8037109375

Epoch 37 train loss 0.014019270849530585 acc 0.839111328125 val loss 0.014704901725053787 acc 0.8046875

Epoch 38 train loss 0.014018579211551696 acc 0.839599609375 val loss 0.014615612046327442 acc 0.8095703125

Epoch 39 train loss 0.014005908305989578 acc 0.839599609375 val loss 0.01459334115497768 acc 0.8134765625

Epoch 40 train loss 0.013998083464684896 acc 0.839599609375 val loss 0.014668849762529135 acc 0.8076171875

Epoch 41 train loss 0.013782424299279228 acc 0.861083984375 val loss 0.014884834177792072 acc 0.8134765625

Epoch 42 train loss 0.013222332956502214 acc 0.931396484375 val loss 0.014583743584807962 acc 0.8408203125

Epoch 43 train loss 0.012961535132490098 acc 0.947509765625 val loss 0.016549687716178596 acc 0.677734375

Epoch 44 train loss 0.012819188617868349 acc 0.9560546875 val loss 0.015577749174553901 acc 0.7421875

Epoch 45 train loss 0.012658090272452682 acc 0.962158203125 val loss 0.0145669796038419 acc 0.81640625

Epoch 46 train loss 0.012526313556008972 acc 0.96923828125 val loss 0.014247319602873176 acc 0.8486328125

Epoch 47 train loss 0.012443357991287485 acc 0.971435546875 val loss 0.014565469522494823 acc 0.822265625

Epoch 48 train loss 0.012318025779677555 acc 0.975830078125 val loss 0.014300409355200827 acc 0.8349609375

24

Epoch 49 train loss 0.012272200066945516 acc 0.976318359375 val loss 0.014724898326676339 acc 0.80859375

Epoch 50 train loss 0.012203568156110123 acc 0.977294921875 val loss 0.014487667009234428 acc 0.8154296875

Epoch 51 train loss 0.01214741355215665 acc 0.979736328125 val loss 0.014151538314763457 acc 0.83984375

Epoch 52 train loss 0.012089080366422422 acc 0.981689453125 val loss 0.01405511045595631 acc 0.849609375

Epoch 53 train loss 0.012052255638991483 acc 0.982421875 val loss 0.014032692939508706 acc 0.857421875

Epoch 54 train loss 0.012021450384054333 acc 0.982666015625 val loss 0.014327841810882092 acc 0.8310546875

Epoch 55 train loss 0.012011423517833464 acc 0.98291015625 val loss 0.014087573857977986 acc 0.8505859375

Epoch 56 train loss 0.011991930077783763 acc 0.983154296875 val loss 0.01384707388933748 acc 0.869140625

Epoch 57 train loss 0.011969141458394006 acc 0.983154296875 val loss 0.013961298449430615 acc 0.8525390625

Epoch 58 train loss 0.011949017411097884 acc 0.98388671875 val loss 0.014033934334293008 acc 0.8486328125

Epoch 59 train loss 0.011957598049775697 acc 0.984375 val loss 0.014158423931803554 acc 0.84375

Epoch 60 train loss 0.011920142598683015 acc 0.985107421875 val loss 0.01393834693590179 acc 0.8525390625

Epoch 61 train loss 0.011903370366781019 acc 0.985107421875 val loss 0.013895356794819236 acc 0.8583984375

Epoch 62 train loss 0.011897806150955148 acc 0.985107421875 val loss 0.013766885153017938 acc 0.86328125

Epoch 63 train loss 0.011895325340447016 acc 0.985107421875 val loss 0.01393004140118137 acc 0.853515625

Epoch 64 train loss 0.01189192448509857 acc 0.985107421875 val loss 0.013803646375890821 acc 0.8720703125

Epoch 65 train loss 0.011892598777194507 acc 0.985107421875 val loss 0.013831537100486457 acc 0.8603515625

Epoch 66 train loss 0.011890510577359237 acc 0.985107421875 val loss 0.013809944328386337 acc 0.8642578125

Epoch 67 train loss 0.011885935455211438 acc 0.985107421875 val loss 0.013846232905052602 acc 0.8603515625

Epoch 68 train loss 0.01188496476970613 acc 0.9853515625 val loss 0.013747755554504693 acc 0.876953125

Epoch 69 train loss 0.011881308833835647 acc 0.9853515625 val loss 0.014062746078707278 acc 0.8486328125

Epoch 70 train loss 0.011877965764142573 acc 0.985595703125 val loss 0.013879091944545507 acc 0.8564453125

Epoch 71 train loss 0.011873112787725404 acc 0.985595703125 val loss 0.01372492010705173 acc 0.87109375

Epoch 72 train loss 0.011872323462739587 acc 0.985595703125 val loss 0.013791271892841905 acc 0.8671875

Epoch 73 train loss 0.011871440437971614 acc 0.985595703125 val loss 0.014006314740981907 acc 0.8544921875

Epoch 74 train loss 0.011869779453263618 acc 0.985595703125 val loss 0.013760568690486252 acc 0.876953125

25

Epoch 75 train loss 0.011869388734339736 acc 0.985595703125 val loss 0.01376438478473574 acc 0.8671875

Epoch 76 train loss 0.01186742150457576 acc 0.985595703125 val loss 0.01378651795675978 acc 0.869140625

Epoch 77 train loss 0.011864889951539226 acc 0.985595703125 val loss 0.01371666748309508 acc 0.8740234375

Epoch 78 train loss 0.011864782311022282 acc 0.985595703125 val loss 0.013723181968089193 acc 0.876953125

Epoch 79 train loss 0.011866034983540885 acc 0.985595703125 val loss 0.013761494599748403 acc 0.8662109375

Epoch 80 train loss 0.011863965832162648 acc 0.985595703125 val loss 0.01369768101722002 acc 0.8759765625

Epoch 81 train loss 0.011861714490805753 acc 0.985595703125 val loss 0.013737686502281576 acc 0.8701171875

Epoch 82 train loss 0.011859266393003054 acc 0.98583984375 val loss 0.013751616585068405 acc 0.869140625

Epoch 83 train loss 0.01185831002658233 acc 0.98583984375 val loss 0.01368404267122969 acc 0.8720703125

Epoch 84 train loss 0.011859116202685982 acc 0.986083984375 val loss 0.013784742273855954 acc 0.8662109375

Epoch 85 train loss 0.011855583565193228 acc 0.986083984375 val loss 0.013736917870119214 acc 0.8701171875

Epoch 86 train loss 0.011856233948492445 acc 0.986083984375 val loss 0.013706412340980023 acc 0.8759765625

Epoch 87 train loss 0.011855660239234567 acc 0.986083984375 val loss 0.013803379726596177 acc 0.8642578125

Epoch 88 train loss 0.011855380987981334 acc 0.986083984375 val loss 0.013759193127043545 acc 0.8720703125

Epoch 89 train loss 0.011854944124934264 acc 0.986083984375 val loss 0.013698334922082722 acc 0.8779296875

Epoch 90 train loss 0.011856033146614209 acc 0.986083984375 val loss 0.013711671985220164 acc 0.876953125

Epoch 91 train loss 0.011854785800096579 acc 0.986083984375 val loss 0.013724449439905584 acc 0.873046875

Epoch 92 train loss 0.011853920586872846 acc 0.986083984375 val loss 0.013732400024309754 acc 0.8681640625

Epoch 93 train loss 0.011852535491925664 acc 0.986083984375 val loss 0.01370806380873546 acc 0.876953125

Epoch 94 train loss 0.011852849565912038 acc 0.986083984375 val loss 0.013674025249201804 acc 0.87890625

Epoch 95 train loss 0.011851826071506366 acc 0.986083984375 val loss 0.013703800039365888 acc 0.87109375

Epoch 96 train loss 0.01185262757644523 acc 0.986083984375 val loss 0.013731345359701663 acc 0.8681640625

Epoch 97 train loss 0.011851152186864056 acc 0.986083984375 val loss 0.013699008035473526 acc 0.873046875

Epoch 98 train loss 0.011851007657242008 acc 0.986083984375 val loss 0.01368651824304834 acc 0.875

Epoch 99 train loss 0.011850922674057074 acc 0.986083984375 val loss 0.013678877206984907 acc 0.873046875

Epoch 100 train loss 0.011852143114083447 acc 0.986083984375 val loss 0.013684857403859496 acc 0.8701171875

26

**Training performance log**

add Codeadd Markdown

[19]:

fig, axes **=** plt.subplots(ncols**=**2, figsize**=**(15, 6))

index **=** 0

axes[index].plot(train\_loss, label**=**"Training")

axes[index].plot(val\_loss, label**=**"Validation")

axes[index].legend()

axes[index].set\_title("Loss log")

​

index **+=** 1

​

axes[index].plot(train\_acc, label**=**"Training")

axes[index].plot(val\_acc, label**=**"Validation")

axes[index].legend()

axes[index].set\_title("Accuracy log")

​

plt.tight\_layout()

plt.show()

add Codeadd Markdown

[20]:

**def** predict(img):

img **=** transform(img).view(1, 1, 128, 128)

best\_model.eval()

**with** torch.no\_grad():

**if** torch.cuda.is\_available():

img **=** img.cuda()

out **=** model(img)

index **=** out.argmax(1).item()

**return** index, out.cpu().detach().numpy()[0][index]

add Codeadd Markdown

**Mapping Pipeline**

add Codeadd Markdown

[21]:

index\_label **=** {0: "Mild Demented",

1: "Moderate Demented",

2: "Non Demented",

3: "Very Mild Demented"}

add Codeadd Markdown

**Predict TestValues**

add Codeadd Markdown

[22]:

pred **=** []

proba **=** []

truth **=** []

​**for** i **in** test:

index, conf **=** predict(np.array(i["image"]))

pred **+=** [index]

proba **+=** [conf]

truth **+=** [i["label"]]

add Codeadd Markdown

27

**Test values evaluation**

add Codeadd Markdown

[23]:

score **=** accuracy\_score(pred, truth)

cm **=** confusion\_matrix(pred, truth)

report **=** classification\_report(pred, truth)

print(report)

sns.heatmap(cm, annot**=True**, fmt**=**'d')

plt.title("Score: {}%".format(round(score**\***100, 2)))

plt.show()

precision recall f1-score support

0 0.77 0.65 0.71 205

1 0.00 0.00 0.00 0

2 0.92 0.89 0.91 659

3 0.82 0.91 0.86 416

accuracy 0.86 1280

macro avg 0.63 0.61 0.62 1280

weighted avg 0.87 0.86 0.86 1280

add Codeadd Markdown

**Visual inspection of results with confidence scores**

add Codeadd Markdown

[24]:

fig, axes **=** plt.subplots(nrows**=**3, ncols**=**3, figsize**=**(15, 6))

index **=** 0

​**for** i **in** range(3):

**for** j **in** range(3):

axes[i][j].imshow(np.array(test[index]["image"]))

axes[i][j].set\_title("Predicted: {}\nConfidence: {}%\nActual: {}".format(index\_label[pred[index]],

round(proba[index]**\***100, 2),

index\_label[truth[index]]))

index **+=** 1

plt.tight\_layout()

plt.show()

add Codeadd Markdown

**TRAINING LOOP CNN+LSTM**

add Codeadd Markdown

[25]:

**class** ConvBlock(nn.Module):

**def** \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size, padding, use\_pool**=True**):

super(ConvBlock, self).\_\_init\_\_()

self.conv\_layer **=** nn.Sequential(nn.Conv2d(in\_channels, in\_channels, kernel\_size, padding**=**padding),

nn.BatchNorm2d(in\_channels),

nn.ReLU(),

28

nn.Conv2d(in\_channels, out\_channels, kernel\_size, padding**=**padding),

nn.BatchNorm2d(out\_channels),

nn.ReLU())

self.use\_pool **=** use\_pool

**if** use\_pool:

self.pool\_layer **=** nn.AvgPool2d(2)

**def** forward(self, input\_tensor):

output\_tensor **=** self.conv\_layer(input\_tensor)

**if** self.use\_pool:

output\_tensor **=** self.pool\_layer(output\_tensor)

**return** output\_tensor

output\_tensor **=** torch.add(output\_tensor, input\_tensor)

**return** output\_tensor

​

**class** AlzheimerLSTMClassifier(nn.Module):

**def** \_\_init\_\_(self, input\_channels, num\_classes):

super(AlzheimerLSTMClassifier, self).\_\_init\_\_()

self.initial\_conv **=** nn.Sequential(nn.Conv2d(in\_channels**=**1, out\_channels**=**3, kernel\_size**=**3),

nn.BatchNorm2d(3),

nn.ReLU())

self.conv\_blocks **=** nn.Sequential(ConvBlock(3, 3, 3, 1, **False**),

ConvBlock(3, 16, 3, 0, **True**),

ConvBlock(16, 16, 3, 1, **False**),

ConvBlock(16, 32, 3, 0, **True**),

ConvBlock(32, 32, 3, 1, **False**),

ConvBlock(32, 64, 3, 0, **True**),

nn.AvgPool2d(2))

*# Set LSTM parameters*

self.lstm\_input\_dim **=** 64 *# Adjust according to the flattened output shape of your CNN*

self.lstm\_hidden\_dim **=** 128

self.lstm\_layers **=** 1

self.lstm\_layer **=** nn.LSTM(input\_size**=**self.lstm\_input\_dim, hidden\_size**=**self.lstm\_hidden\_dim,

num\_layers**=**self.lstm\_layers, batch\_first**=True**)

self.fc\_layer **=** nn.Linear(self.lstm\_hidden\_dim, num\_classes)

**def** forward(self, x):

x **=** self.initial\_conv(x)

x **=** self.conv\_blocks(x)

*# Flattening to sequence*

batch\_size **=** x.size(0)

x **=** torch.flatten(x, 2) *# shape: (batch\_size, channels, sequence\_length)*

x **=** x.permute(0, 2, 1) *# shape: (batch\_size, sequence\_length, channels)*

29

*# Passing through LSTM*

lstm\_out, (hidden\_state, cell\_state) **=** self.lstm\_layer(x)

lstm\_out **=** lstm\_out[:, **-**1, :] *# Take the output from the last time step*

x **=** self.fc\_layer(lstm\_out)

**return** nn.functional.softmax(x, dim**=**1)

​

add Codeadd Markdown

[26]:

*# Define the model using AlzheimerLSTMClassifier instead of AlzheimerClassifier*

model **=** AlzheimerLSTMClassifier(1, OUT\_SIZE)

​

*# Move the model to the available device (CUDA or CPU)*

model **=** model.to(device)

​

*# Loss function: Cross Entropy is still suitable for classification tasks*

criterion **=** nn.CrossEntropyLoss()

​

*# Optimizer: Using Stochastic Gradient Descent (SGD)*

optimizer **=** torch.optim.SGD(model.parameters(), lr**=**LR,momentum**=**0.9)

​

*# Learning Rate Scheduler: StepLR with specified step size and gamma*

scheduler **=** torch.optim.lr\_scheduler.StepLR(optimizer, step\_size**=**STEP, gamma**=**GAMMA)

​

add Codeadd Markdown

[27]:

**import** torch

**from** copy **import** deepcopy

​

*# Assuming 'model' is an instance of AlzheimerLSTMClassifier*

best\_model **=** deepcopy(model)

best\_acc **=** 0

​

train\_loss **=** []

train\_acc **=** []

val\_loss **=** []

val\_acc **=** []

​

**for** epoch **in** range(1, EPOCHS **+** 1):

model.train()

epoch\_train\_loss **=** 0

epoch\_train\_acc **=** 0

total\_train\_samples **=** 0

**for** batch\_data, batch\_target **in** train\_dl:

optimizer.zero\_grad()

**if** torch.cuda.is\_available():

batch\_data, batch\_target **=** batch\_data.cuda(), batch\_target.cuda()

30

*# Forward pass*

output **=** model(batch\_data)

loss **=** criterion(output, batch\_target)

*# Accumulate loss and accuracy*

epoch\_train\_loss **+=** loss.item()

epoch\_train\_acc **+=** (output.argmax(1) **==** batch\_target).sum().item()

total\_train\_samples **+=** output.size(0)

*# Backward pass and optimization*

loss.backward()

optimizer.step()

train\_loss.append(epoch\_train\_loss **/** total\_train\_samples)

train\_acc.append(epoch\_train\_acc **/** total\_train\_samples)

*# Validation phase*

model.eval()

epoch\_val\_loss **=** 0

epoch\_val\_acc **=** 0

total\_val\_samples **=** 0

**with** torch.no\_grad():

**for** batch\_data, batch\_target **in** val\_dl:

​

**if** torch.cuda.is\_available():

batch\_data, batch\_target **=** batch\_data.cuda(), batch\_target.cuda()

​

*# Forward pass*

output **=** model(batch\_data)

loss **=** criterion(output, batch\_target)

*# Accumulate validation loss and accuracy*

epoch\_val\_loss **+=** loss.item()

epoch\_val\_acc **+=** (output.argmax(1) **==** batch\_target).sum().item()

total\_val\_samples **+=** output.size(0)

val\_loss.append(epoch\_val\_loss **/** total\_val\_samples)

val\_acc.append(epoch\_val\_acc **/** total\_val\_samples)

*# Check if current model has the best accuracy*

**if** val\_acc[**-**1] **>=** best\_acc:

best\_acc **=** val\_acc[**-**1]

best\_model **=** deepcopy(model)

*# Print epoch statistics*

print(f"Epoch {epoch} | Train Loss: {train\_loss[**-**1]:.4f} | Train Acc: {train\_acc[**-**1]:.4f} | "

f"Val Loss: {val\_loss[**-**1]:.4f} | Val Acc: {val\_acc[**-**1]:.4f}")

31

*# Step the learning rate scheduler*

scheduler.step()

​

Epoch 1 | Train Loss: 0.0205 | Train Acc: 0.4963 | Val Loss: 0.0190 | Val Acc: 0.5205

Epoch 2 | Train Loss: 0.0192 | Train Acc: 0.4963 | Val Loss: 0.0187 | Val Acc: 0.5205

Epoch 3 | Train Loss: 0.0189 | Train Acc: 0.5073 | Val Loss: 0.0184 | Val Acc: 0.5547

Epoch 4 | Train Loss: 0.0186 | Train Acc: 0.5256 | Val Loss: 0.0188 | Val Acc: 0.5332

Epoch 5 | Train Loss: 0.0185 | Train Acc: 0.5281 | Val Loss: 0.0185 | Val Acc: 0.5352

Epoch 6 | Train Loss: 0.0182 | Train Acc: 0.5637 | Val Loss: 0.0185 | Val Acc: 0.5566

Epoch 7 | Train Loss: 0.0181 | Train Acc: 0.5654 | Val Loss: 0.0183 | Val Acc: 0.5449

Epoch 8 | Train Loss: 0.0179 | Train Acc: 0.5854 | Val Loss: 0.0179 | Val Acc: 0.5869

Epoch 9 | Train Loss: 0.0178 | Train Acc: 0.5903 | Val Loss: 0.0187 | Val Acc: 0.5273

Epoch 10 | Train Loss: 0.0178 | Train Acc: 0.5957 | Val Loss: 0.0183 | Val Acc: 0.5703

Epoch 11 | Train Loss: 0.0176 | Train Acc: 0.6116 | Val Loss: 0.0179 | Val Acc: 0.5996

Epoch 12 | Train Loss: 0.0177 | Train Acc: 0.6033 | Val Loss: 0.0196 | Val Acc: 0.4707

Epoch 13 | Train Loss: 0.0177 | Train Acc: 0.6035 | Val Loss: 0.0178 | Val Acc: 0.6016

Epoch 14 | Train Loss: 0.0174 | Train Acc: 0.6204 | Val Loss: 0.0187 | Val Acc: 0.5156

Epoch 15 | Train Loss: 0.0173 | Train Acc: 0.6292 | Val Loss: 0.0218 | Val Acc: 0.3438

Epoch 16 | Train Loss: 0.0173 | Train Acc: 0.6296 | Val Loss: 0.0186 | Val Acc: 0.5479

Epoch 17 | Train Loss: 0.0171 | Train Acc: 0.6428 | Val Loss: 0.0180 | Val Acc: 0.5830

Epoch 18 | Train Loss: 0.0171 | Train Acc: 0.6438 | Val Loss: 0.0210 | Val Acc: 0.3887

Epoch 19 | Train Loss: 0.0169 | Train Acc: 0.6589 | Val Loss: 0.0173 | Val Acc: 0.6318

Epoch 20 | Train Loss: 0.0170 | Train Acc: 0.6497 | Val Loss: 0.0170 | Val Acc: 0.6543

Epoch 21 | Train Loss: 0.0165 | Train Acc: 0.6875 | Val Loss: 0.0170 | Val Acc: 0.6494

Epoch 22 | Train Loss: 0.0163 | Train Acc: 0.6965 | Val Loss: 0.0183 | Val Acc: 0.5635

Epoch 23 | Train Loss: 0.0163 | Train Acc: 0.6992 | Val Loss: 0.0169 | Val Acc: 0.6670

Epoch 24 | Train Loss: 0.0162 | Train Acc: 0.7063 | Val Loss: 0.0167 | Val Acc: 0.6680

Epoch 25 | Train Loss: 0.0161 | Train Acc: 0.7129 | Val Loss: 0.0178 | Val Acc: 0.5967

Epoch 26 | Train Loss: 0.0163 | Train Acc: 0.6995 | Val Loss: 0.0178 | Val Acc: 0.5986

Epoch 27 | Train Loss: 0.0159 | Train Acc: 0.7222 | Val Loss: 0.0168 | Val Acc: 0.6660

Epoch 28 | Train Loss: 0.0160 | Train Acc: 0.7170 | Val Loss: 0.0188 | Val Acc: 0.5361

Epoch 29 | Train Loss: 0.0158 | Train Acc: 0.7314 | Val Loss: 0.0178 | Val Acc: 0.5957

Epoch 30 | Train Loss: 0.0158 | Train Acc: 0.7278 | Val Loss: 0.0208 | Val Acc: 0.4072

Epoch 31 | Train Loss: 0.0158 | Train Acc: 0.7322 | Val Loss: 0.0167 | Val Acc: 0.6738

Epoch 32 | Train Loss: 0.0158 | Train Acc: 0.7288 | Val Loss: 0.0178 | Val Acc: 0.5977

Epoch 33 | Train Loss: 0.0160 | Train Acc: 0.7168 | Val Loss: 0.0212 | Val Acc: 0.3848

Epoch 34 | Train Loss: 0.0157 | Train Acc: 0.7402 | Val Loss: 0.0170 | Val Acc: 0.6504

Epoch 35 | Train Loss: 0.0156 | Train Acc: 0.7458 | Val Loss: 0.0189 | Val Acc: 0.5273

Epoch 36 | Train Loss: 0.0157 | Train Acc: 0.7356 | Val Loss: 0.0172 | Val Acc: 0.6377

Epoch 37 | Train Loss: 0.0155 | Train Acc: 0.7524 | Val Loss: 0.0179 | Val Acc: 0.5840

Epoch 38 | Train Loss: 0.0155 | Train Acc: 0.7493 | Val Loss: 0.0191 | Val Acc: 0.5186

Epoch 39 | Train Loss: 0.0154 | Train Acc: 0.7585 | Val Loss: 0.0199 | Val Acc: 0.4590

Epoch 40 | Train Loss: 0.0151 | Train Acc: 0.7751 | Val Loss: 0.0191 | Val Acc: 0.5205

Epoch 41 | Train Loss: 0.0149 | Train Acc: 0.7874 | Val Loss: 0.0167 | Val Acc: 0.6709

Epoch 42 | Train Loss: 0.0148 | Train Acc: 0.7966 | Val Loss: 0.0175 | Val Acc: 0.6152

Epoch 43 | Train Loss: 0.0148 | Train Acc: 0.7988 | Val Loss: 0.0158 | Val Acc: 0.7246

Epoch 44 | Train Loss: 0.0147 | Train Acc: 0.8022 | Val Loss: 0.0168 | Val Acc: 0.6621

Epoch 45 | Train Loss: 0.0146 | Train Acc: 0.8086 | Val Loss: 0.0166 | Val Acc: 0.6758

Epoch 46 | Train Loss: 0.0146 | Train Acc: 0.8083 | Val Loss: 0.0156 | Val Acc: 0.7393

Epoch 47 | Train Loss: 0.0146 | Train Acc: 0.8071 | Val Loss: 0.0166 | Val Acc: 0.6787

Epoch 48 | Train Loss: 0.0146 | Train Acc: 0.8105 | Val Loss: 0.0156 | Val Acc: 0.7402

Epoch 49 | Train Loss: 0.0145 | Train Acc: 0.8154 | Val Loss: 0.0156 | Val Acc: 0.7432

32

Epoch 50 | Train Loss: 0.0145 | Train Acc: 0.8154 | Val Loss: 0.0157 | Val Acc: 0.7402

Epoch 51 | Train Loss: 0.0145 | Train Acc: 0.8176 | Val Loss: 0.0155 | Val Acc: 0.7510

Epoch 52 | Train Loss: 0.0144 | Train Acc: 0.8191 | Val Loss: 0.0165 | Val Acc: 0.6846

Epoch 53 | Train Loss: 0.0144 | Train Acc: 0.8193 | Val Loss: 0.0156 | Val Acc: 0.7402

Epoch 54 | Train Loss: 0.0144 | Train Acc: 0.8228 | Val Loss: 0.0157 | Val Acc: 0.7344

Epoch 55 | Train Loss: 0.0144 | Train Acc: 0.8228 | Val Loss: 0.0155 | Val Acc: 0.7490

Epoch 56 | Train Loss: 0.0144 | Train Acc: 0.8223 | Val Loss: 0.0155 | Val Acc: 0.7480

Epoch 57 | Train Loss: 0.0143 | Train Acc: 0.8240 | Val Loss: 0.0166 | Val Acc: 0.6768

Epoch 58 | Train Loss: 0.0143 | Train Acc: 0.8240 | Val Loss: 0.0154 | Val Acc: 0.7598

Epoch 59 | Train Loss: 0.0144 | Train Acc: 0.8223 | Val Loss: 0.0152 | Val Acc: 0.7637

Epoch 60 | Train Loss: 0.0144 | Train Acc: 0.8220 | Val Loss: 0.0153 | Val Acc: 0.7637

Epoch 61 | Train Loss: 0.0143 | Train Acc: 0.8259 | Val Loss: 0.0153 | Val Acc: 0.7607

Epoch 62 | Train Loss: 0.0143 | Train Acc: 0.8259 | Val Loss: 0.0153 | Val Acc: 0.7578

Epoch 63 | Train Loss: 0.0143 | Train Acc: 0.8262 | Val Loss: 0.0154 | Val Acc: 0.7539

Epoch 64 | Train Loss: 0.0143 | Train Acc: 0.8264 | Val Loss: 0.0152 | Val Acc: 0.7686

Epoch 65 | Train Loss: 0.0143 | Train Acc: 0.8264 | Val Loss: 0.0152 | Val Acc: 0.7666

Epoch 66 | Train Loss: 0.0143 | Train Acc: 0.8264 | Val Loss: 0.0152 | Val Acc: 0.7705

Epoch 67 | Train Loss: 0.0143 | Train Acc: 0.8274 | Val Loss: 0.0154 | Val Acc: 0.7559

Epoch 68 | Train Loss: 0.0143 | Train Acc: 0.8274 | Val Loss: 0.0153 | Val Acc: 0.7578

Epoch 69 | Train Loss: 0.0143 | Train Acc: 0.8274 | Val Loss: 0.0153 | Val Acc: 0.7637

Epoch 70 | Train Loss: 0.0143 | Train Acc: 0.8271 | Val Loss: 0.0152 | Val Acc: 0.7676

Epoch 71 | Train Loss: 0.0143 | Train Acc: 0.8276 | Val Loss: 0.0153 | Val Acc: 0.7656

Epoch 72 | Train Loss: 0.0143 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7686

Epoch 73 | Train Loss: 0.0143 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7676

Epoch 74 | Train Loss: 0.0143 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7676

Epoch 75 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0153 | Val Acc: 0.7637

Epoch 76 | Train Loss: 0.0143 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7637

Epoch 77 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7695

Epoch 78 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0153 | Val Acc: 0.7617

Epoch 79 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0153 | Val Acc: 0.7559

Epoch 80 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7666

Epoch 81 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7666

Epoch 82 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7676

Epoch 83 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7676

Epoch 84 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7676

Epoch 85 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7725

Epoch 86 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7705

Epoch 87 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7715

Epoch 88 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7705

Epoch 89 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7725

Epoch 90 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7686

Epoch 91 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7725

Epoch 92 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7734

Epoch 93 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7686

Epoch 94 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7705

Epoch 95 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7715

Epoch 96 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7725

Epoch 97 | Train Loss: 0.0142 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7705

Epoch 98 | Train Loss: 0.0142 | Train Acc: 0.8279 | Val Loss: 0.0152 | Val Acc: 0.7676

Epoch 99 | Train Loss: 0.0142 | Train Acc: 0.8281 | Val Loss: 0.0152 | Val Acc: 0.7686

Epoch 100 | Train Loss: 0.0142 | Train Acc: 0.8279 | Val Loss: 0.0152 | Val Acc: 0.7695

33

add Codeadd Markdown

**Training performance log**

add Codeadd Markdown

[28]:

fig, axes **=** plt.subplots(ncols**=**2, figsize**=**(15, 6))

​

index **=** 0

​axes[index].plot(train\_loss, label**=**"Training")

axes[index].plot(val\_loss, label**=**"Validation")

axes[index].legend()

axes[index].set\_title("Loss log")

​

index **+=** 1

​

axes[index].plot(train\_acc, label**=**"Training")

axes[index].plot(val\_acc, label**=**"Validation")

axes[index].legend()

axes[index].set\_title("Accuracy log")

​

plt.tight\_layout()

plt.show()

add Codeadd Markdown

**Predict TestValues**

add Codeadd Markdown

[29]:

**def** predict(img):

*# Transform the image and reshape it for the model*

img **=** transform(img).view(1, 1, 128, 128) *# 1 batch, 1 channel, 128x128 size*

best\_model.eval() *# Ensure the model is in evaluation mode*

**with** torch.no\_grad():

*# Move the image to GPU if available*

**if** torch.cuda.is\_available():

img **=** img.cuda()

​

*# Forward pass through the best\_model (using LSTM + CNN)*

out **=** best\_model(img)

*# Get the index of the highest score (predicted class) and its confidence score*

predicted\_class **=** out.argmax(1).item()

confidence\_score **=** out.cpu().detach().numpy()[0][predicted\_class]

**return** predicted\_class, confidence\_score

add Codeadd Markdown

[30]:

pred **=** []

proba **=** []

truth **=** []

​34

**for** i **in** test:

index, conf **=** predict(np.array(i["image"]))

pred **+=** [index]

proba **+=** [conf]

truth **+=** [i["label"]]

add Codeadd Markdown

**Test values evaluation**

add Codeadd Markdown

[31]:

score **=** accuracy\_score(pred, truth)

cm **=** confusion\_matrix(pred, truth)

report **=** classification\_report(pred, truth)

print(report)

sns.heatmap(cm, annot**=True**, fmt**=**'d')

plt.title("Score: {}%".format(round(score**\***100, 2)))

plt.show()

add Codeadd Markdown

**Visual inspection of results with confidence scores\***

add Codeadd Markdown

[32]:

*# Import necessary libraries*

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

​

*# Create a 3x3 grid of subplots*

fig, axes **=** plt.subplots(nrows**=**3, ncols**=**3, figsize**=**(15, 6))

index **=** 0

​

*# Loop through the grid to plot images with predictions, confidence, and true labels*

**for** i **in** range(3):

**for** j **in** range(3):

*# Display the image in color*

axes[i][j].imshow(np.array(test[index]["image"])) *# No cmap, so it shows color if available*

*# Set title with predicted label, confidence score, and actual label*

axes[i][j].set\_title(

"Predicted: {}\nConfidence: {}%\nActual: {}".format(

index\_label[pred[index]],

round(proba[index] **\*** 100, 2),

index\_label[truth[index]]

)

*# Hide axis*

axes[i][j].axis('off')

index **+=** 1

​

35

*# Adjust layout for better spacing*

plt.tight\_layout()

plt.show()

​

add Codeadd Markdown

**TRAINING LOOP CNN+BiLSTM**

add Codeadd Markdown

[33]:

**import** torch

**import** torch.nn **as** nn

​

**class** ConvBlock(nn.Module):

**def** \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size, padding, use\_pool**=True**):

super(ConvBlock, self).\_\_init\_\_()

self.conv\_layer **=** nn.Sequential(

nn.Conv2d(in\_channels, in\_channels, kernel\_size, padding**=**padding),

nn.BatchNorm2d(in\_channels),

nn.ReLU(),

nn.Conv2d(in\_channels, out\_channels, kernel\_size, padding**=**padding),

nn.BatchNorm2d(out\_channels),

nn.ReLU()

)

self.use\_pool **=** use\_pool

**if** use\_pool:

self.pool\_layer **=** nn.AvgPool2d(2)

**def** forward(self, input\_tensor):

output\_tensor **=** self.conv\_layer(input\_tensor)

**if** self.use\_pool:

output\_tensor **=** self.pool\_layer(output\_tensor)

**else**:

output\_tensor **=** torch.add(output\_tensor, input\_tensor) *# Residual connection for non-pooling layers*

**return** output\_tensor

​

**class** AlzheimerBiLSTMClassifier(nn.Module):

**def** \_\_init\_\_(self, input\_channels, num\_classes):

super(AlzheimerBiLSTMClassifier, self).\_\_init\_\_()

*# Initial Convolution block*

self.initial\_conv **=** nn.Sequential(

nn.Conv2d(in\_channels**=**input\_channels, out\_channels**=**3, kernel\_size**=**3),

nn.BatchNorm2d(3),

nn.ReLU()

)

36

*# Stack of convolutional blocks*

self.conv\_blocks **=** nn.Sequential(

ConvBlock(3, 3, 3, 1, **False**), *# Residual block*

ConvBlock(3, 16, 3, 0, **True**), *# Pooling block*

ConvBlock(16, 16, 3, 1, **False**), *# Residual block*

ConvBlock(16, 32, 3, 0, **True**), *# Pooling block*

ConvBlock(32, 32, 3, 1, **False**), *# Residual block*

ConvBlock(32, 64, 3, 0, **True**), *# Pooling block*

nn.AvgPool2d(2)

)

*# BiLSTM parameters*

self.lstm\_input\_dim **=** 64 *# Adjust to flattened output from CNN blocks*

self.lstm\_hidden\_dim **=** 128

self.lstm\_layers **=** 1

self.bidirectional **=** **True**

self.num\_directions **=** 2 **if** self.bidirectional **else** 1

*# BiLSTM layer*

self.lstm\_layer **=** nn.LSTM(

input\_size**=**self.lstm\_input\_dim,

hidden\_size**=**self.lstm\_hidden\_dim,

num\_layers**=**self.lstm\_layers,

batch\_first**=True**,

bidirectional**=**self.bidirectional

)

*# Fully connected layer for classification*

self.fc\_layer **=** nn.Linear(self.lstm\_hidden\_dim **\*** self.num\_directions, num\_classes)

**def** forward(self, x):

*# CNN Feature extraction*

x **=** self.initial\_conv(x)

x **=** self.conv\_blocks(x)

*# Flatten to a sequence for LSTM input*

batch\_size **=** x.size(0)

x **=** torch.flatten(x, 2) *# Flatten height and width to a single dimension*

x **=** x.permute(0, 2, 1) *# Reshape to (batch\_size, sequence\_length, features)*

*# Pass through BiLSTM*

lstm\_out, (hidden\_state, cell\_state) **=** self.lstm\_layer(x)

*# Take the output from the last time step*

lstm\_out **=** lstm\_out[:, **-**1, :] *# Take output of the final time step*

*# Pass through fully connected layer*

x **=** self.fc\_layer(lstm\_out)

**return** nn.functional.softmax(x, dim**=**1)

​

37

​

add Codeadd Markdown

[34]:

*# Define the model using AlzheimerBiLSTMClassifier instead of AlzheimerLSTMClassifier*

model **=** AlzheimerBiLSTMClassifier(1, OUT\_SIZE)

​

*# Move the model to the available device (CUDA or CPU)*

model **=** model.to(device)

​

*# Loss function: Cross Entropy is still suitable for classification tasks*

criterion **=** nn.CrossEntropyLoss()

​

*# Optimizer: Using Stochastic Gradient Descent (SGD)*

optimizer **=** torch.optim.SGD(model.parameters(), lr**=**LR,momentum**=**0.9)

​

*# Learning Rate Scheduler: StepLR with specified step size and gamma*

scheduler **=** torch.optim.lr\_scheduler.StepLR(optimizer, step\_size**=**STEP, gamma**=**GAMMA)

​

add Codeadd Markdown

[35]:

**import** torch

**from** copy **import** deepcopy

​

*# Assuming 'model' is an instance of AlzheimerBiLSTMClassifier*

best\_model **=** deepcopy(model)

best\_acc **=** 0

​

train\_loss **=** []

train\_acc **=** []

val\_loss **=** []

val\_acc **=** []

​

**for** epoch **in** range(1, EPOCHS **+** 1):

model.train() *# Set the model to training mode*

epoch\_train\_loss **=** 0

epoch\_train\_acc **=** 0

total\_train\_samples **=** 0

**for** batch\_data, batch\_target **in** train\_dl:

optimizer.zero\_grad() *# Zero the gradients*

**if** torch.cuda.is\_available():

batch\_data, batch\_target **=** batch\_data.cuda(), batch\_target.cuda()

*# Forward pass*

output **=** model(batch\_data)

loss **=** criterion(output, batch\_target)

*# Accumulate loss and accuracy*

epoch\_train\_loss **+=** loss.item()

epoch\_train\_acc **+=** (output.argmax(1) **==** batch\_target).sum().item()

38

total\_train\_samples **+=** output.size(0)

*# Backward pass and optimization*

loss.backward()

optimizer.step()

*# Average training loss and accuracy*

train\_loss.append(epoch\_train\_loss **/** total\_train\_samples)

train\_acc.append(epoch\_train\_acc **/** total\_train\_samples)

*# Validation phase*

model.eval() *# Set the model to evaluation mode*

epoch\_val\_loss **=** 0

epoch\_val\_acc **=** 0

total\_val\_samples **=** 0

**with** torch.no\_grad():

**for** batch\_data, batch\_target **in** val\_dl:

**if** torch.cuda.is\_available():

batch\_data, batch\_target **=** batch\_data.cuda(), batch\_target.cuda()

​

*# Forward pass*

output **=** model(batch\_data)

loss **=** criterion(output, batch\_target)

*# Accumulate validation loss and accuracy*

epoch\_val\_loss **+=** loss.item()

epoch\_val\_acc **+=** (output.argmax(1) **==** batch\_target).sum().item()

total\_val\_samples **+=** output.size(0)

*# Average validation loss and accuracy*

val\_loss.append(epoch\_val\_loss **/** total\_val\_samples)

val\_acc.append(epoch\_val\_acc **/** total\_val\_samples)

*# Check if current model has the best accuracy*

**if** val\_acc[**-**1] **>=** best\_acc:

best\_acc **=** val\_acc[**-**1]

best\_model **=** deepcopy(model) *# Keep a copy of the best model*

*# Print epoch statistics*

print(f"Epoch {epoch} | Train Loss: {train\_loss[**-**1]:.4f} | Train Acc: {train\_acc[**-**1]:.4f} | "

f"Val Loss: {val\_loss[**-**1]:.4f} | Val Acc: {val\_acc[**-**1]:.4f}")

39

*# Step the learning rate scheduler*

scheduler.step()

​Epoch 1 | Train Loss: 0.0202 | Train Acc: 0.4673 | Val Loss: 0.0189 | Val Acc: 0.5205

Epoch 2 | Train Loss: 0.0191 | Train Acc: 0.4963 | Val Loss: 0.0187 | Val Acc: 0.5205

Epoch 3 | Train Loss: 0.0189 | Train Acc: 0.5107 | Val Loss: 0.0185 | Val Acc: 0.5508

Epoch 4 | Train Loss: 0.0186 | Train Acc: 0.5464 | Val Loss: 0.0185 | Val Acc: 0.5322

Epoch 5 | Train Loss: 0.0183 | Train Acc: 0.5623 | Val Loss: 0.0190 | Val Acc: 0.5205

Epoch 6 | Train Loss: 0.0181 | Train Acc: 0.5750 | Val Loss: 0.0196 | Val Acc: 0.4727

Epoch 7 | Train Loss: 0.0179 | Train Acc: 0.5847 | Val Loss: 0.0183 | Val Acc: 0.5625

Epoch 8 | Train Loss: 0.0179 | Train Acc: 0.5935 | Val Loss: 0.0182 | Val Acc: 0.5498

Epoch 9 | Train Loss: 0.0177 | Train Acc: 0.6021 | Val Loss: 0.0184 | Val Acc: 0.5518

Epoch 10 | Train Loss: 0.0177 | Train Acc: 0.6038 | Val Loss: 0.0177 | Val Acc: 0.5986

Epoch 11 | Train Loss: 0.0174 | Train Acc: 0.6255 | Val Loss: 0.0179 | Val Acc: 0.5938

Epoch 12 | Train Loss: 0.0174 | Train Acc: 0.6326 | Val Loss: 0.0173 | Val Acc: 0.6309

Epoch 13 | Train Loss: 0.0174 | Train Acc: 0.6208 | Val Loss: 0.0188 | Val Acc: 0.5283

Epoch 14 | Train Loss: 0.0173 | Train Acc: 0.6340 | Val Loss: 0.0190 | Val Acc: 0.5283

Epoch 15 | Train Loss: 0.0174 | Train Acc: 0.6262 | Val Loss: 0.0203 | Val Acc: 0.4160

Epoch 16 | Train Loss: 0.0172 | Train Acc: 0.6414 | Val Loss: 0.0171 | Val Acc: 0.6475

Epoch 17 | Train Loss: 0.0170 | Train Acc: 0.6475 | Val Loss: 0.0183 | Val Acc: 0.5645

Epoch 18 | Train Loss: 0.0172 | Train Acc: 0.6379 | Val Loss: 0.0175 | Val Acc: 0.6221

Epoch 19 | Train Loss: 0.0168 | Train Acc: 0.6621 | Val Loss: 0.0170 | Val Acc: 0.6514

Epoch 20 | Train Loss: 0.0167 | Train Acc: 0.6731 | Val Loss: 0.0210 | Val Acc: 0.3828

Epoch 21 | Train Loss: 0.0165 | Train Acc: 0.6892 | Val Loss: 0.0167 | Val Acc: 0.6709

Epoch 22 | Train Loss: 0.0162 | Train Acc: 0.7068 | Val Loss: 0.0168 | Val Acc: 0.6621

Epoch 23 | Train Loss: 0.0161 | Train Acc: 0.7100 | Val Loss: 0.0207 | Val Acc: 0.4092

Epoch 24 | Train Loss: 0.0162 | Train Acc: 0.7046 | Val Loss: 0.0190 | Val Acc: 0.5244

Epoch 25 | Train Loss: 0.0158 | Train Acc: 0.7310 | Val Loss: 0.0197 | Val Acc: 0.4736

Epoch 26 | Train Loss: 0.0158 | Train Acc: 0.7334 | Val Loss: 0.0188 | Val Acc: 0.5371

Epoch 27 | Train Loss: 0.0157 | Train Acc: 0.7373 | Val Loss: 0.0169 | Val Acc: 0.6494

Epoch 28 | Train Loss: 0.0157 | Train Acc: 0.7388 | Val Loss: 0.0210 | Val Acc: 0.3936

Epoch 29 | Train Loss: 0.0155 | Train Acc: 0.7512 | Val Loss: 0.0164 | Val Acc: 0.6943

Epoch 30 | Train Loss: 0.0157 | Train Acc: 0.7375 | Val Loss: 0.0179 | Val Acc: 0.5938

Epoch 31 | Train Loss: 0.0155 | Train Acc: 0.7500 | Val Loss: 0.0182 | Val Acc: 0.5693

Epoch 32 | Train Loss: 0.0154 | Train Acc: 0.7610 | Val Loss: 0.0174 | Val Acc: 0.6221

Epoch 33 | Train Loss: 0.0154 | Train Acc: 0.7554 | Val Loss: 0.0191 | Val Acc: 0.5215

Epoch 34 | Train Loss: 0.0152 | Train Acc: 0.7710 | Val Loss: 0.0160 | Val Acc: 0.7148

Epoch 35 | Train Loss: 0.0153 | Train Acc: 0.7625 | Val Loss: 0.0184 | Val Acc: 0.5576

Epoch 36 | Train Loss: 0.0152 | Train Acc: 0.7705 | Val Loss: 0.0164 | Val Acc: 0.6875

Epoch 37 | Train Loss: 0.0152 | Train Acc: 0.7737 | Val Loss: 0.0161 | Val Acc: 0.7119

Epoch 38 | Train Loss: 0.0151 | Train Acc: 0.7764 | Val Loss: 0.0168 | Val Acc: 0.6650

Epoch 39 | Train Loss: 0.0150 | Train Acc: 0.7810 | Val Loss: 0.0164 | Val Acc: 0.6895

Epoch 40 | Train Loss: 0.0151 | Train Acc: 0.7744 | Val Loss: 0.0171 | Val Acc: 0.6426

Epoch 41 | Train Loss: 0.0149 | Train Acc: 0.7930 | Val Loss: 0.0160 | Val Acc: 0.7168

Epoch 42 | Train Loss: 0.0147 | Train Acc: 0.7996 | Val Loss: 0.0166 | Val Acc: 0.6777

Epoch 43 | Train Loss: 0.0146 | Train Acc: 0.8064 | Val Loss: 0.0159 | Val Acc: 0.7256

Epoch 44 | Train Loss: 0.0146 | Train Acc: 0.8093 | Val Loss: 0.0164 | Val Acc: 0.6914

Epoch 45 | Train Loss: 0.0146 | Train Acc: 0.8093 | Val Loss: 0.0155 | Val Acc: 0.7461

Epoch 46 | Train Loss: 0.0145 | Train Acc: 0.8127 | Val Loss: 0.0163 | Val Acc: 0.6992

Epoch 47 | Train Loss: 0.0145 | Train Acc: 0.8140 | Val Loss: 0.0159 | Val Acc: 0.7197

Epoch 48 | Train Loss: 0.0145 | Train Acc: 0.8132 | Val Loss: 0.0161 | Val Acc: 0.7080

Epoch 49 | Train Loss: 0.0145 | Train Acc: 0.8149 | Val Loss: 0.0156 | Val Acc: 0.7471

Epoch 50 | Train Loss: 0.0145 | Train Acc: 0.8167 | Val Loss: 0.0156 | Val Acc: 0.7451

40

Epoch 51 | Train Loss: 0.0145 | Train Acc: 0.8167 | Val Loss: 0.0155 | Val Acc: 0.7451

Epoch 52 | Train Loss: 0.0144 | Train Acc: 0.8191 | Val Loss: 0.0155 | Val Acc: 0.7500

Epoch 53 | Train Loss: 0.0145 | Train Acc: 0.8171 | Val Loss: 0.0164 | Val Acc: 0.6895

Epoch 54 | Train Loss: 0.0144 | Train Acc: 0.8198 | Val Loss: 0.0169 | Val Acc: 0.6582

Epoch 55 | Train Loss: 0.0144 | Train Acc: 0.8220 | Val Loss: 0.0156 | Val Acc: 0.7393

Epoch 56 | Train Loss: 0.0144 | Train Acc: 0.8196 | Val Loss: 0.0154 | Val Acc: 0.7559

Epoch 57 | Train Loss: 0.0144 | Train Acc: 0.8213 | Val Loss: 0.0160 | Val Acc: 0.7109

Epoch 58 | Train Loss: 0.0144 | Train Acc: 0.8223 | Val Loss: 0.0157 | Val Acc: 0.7393

Epoch 59 | Train Loss: 0.0143 | Train Acc: 0.8247 | Val Loss: 0.0160 | Val Acc: 0.7178

Epoch 60 | Train Loss: 0.0143 | Train Acc: 0.8254 | Val Loss: 0.0153 | Val Acc: 0.7627

Epoch 61 | Train Loss: 0.0143 | Train Acc: 0.8254 | Val Loss: 0.0152 | Val Acc: 0.7666

Epoch 62 | Train Loss: 0.0143 | Train Acc: 0.8264 | Val Loss: 0.0153 | Val Acc: 0.7656

Epoch 63 | Train Loss: 0.0143 | Train Acc: 0.8271 | Val Loss: 0.0153 | Val Acc: 0.7646

Epoch 64 | Train Loss: 0.0143 | Train Acc: 0.8274 | Val Loss: 0.0153 | Val Acc: 0.7617

Epoch 65 | Train Loss: 0.0143 | Train Acc: 0.8276 | Val Loss: 0.0152 | Val Acc: 0.7725

Epoch 66 | Train Loss: 0.0143 | Train Acc: 0.8279 | Val Loss: 0.0153 | Val Acc: 0.7568

Epoch 67 | Train Loss: 0.0143 | Train Acc: 0.8281 | Val Loss: 0.0152 | Val Acc: 0.7715

Epoch 68 | Train Loss: 0.0143 | Train Acc: 0.8286 | Val Loss: 0.0153 | Val Acc: 0.7578

Epoch 69 | Train Loss: 0.0143 | Train Acc: 0.8286 | Val Loss: 0.0152 | Val Acc: 0.7686

Epoch 70 | Train Loss: 0.0142 | Train Acc: 0.8289 | Val Loss: 0.0152 | Val Acc: 0.7705

Epoch 71 | Train Loss: 0.0142 | Train Acc: 0.8296 | Val Loss: 0.0153 | Val Acc: 0.7627

Epoch 72 | Train Loss: 0.0142 | Train Acc: 0.8298 | Val Loss: 0.0153 | Val Acc: 0.7637

Epoch 73 | Train Loss: 0.0142 | Train Acc: 0.8298 | Val Loss: 0.0153 | Val Acc: 0.7627

Epoch 74 | Train Loss: 0.0142 | Train Acc: 0.8308 | Val Loss: 0.0153 | Val Acc: 0.7666

Epoch 75 | Train Loss: 0.0142 | Train Acc: 0.8311 | Val Loss: 0.0153 | Val Acc: 0.7578

Epoch 76 | Train Loss: 0.0142 | Train Acc: 0.8311 | Val Loss: 0.0153 | Val Acc: 0.7637

Epoch 77 | Train Loss: 0.0142 | Train Acc: 0.8318 | Val Loss: 0.0153 | Val Acc: 0.7656

Epoch 78 | Train Loss: 0.0142 | Train Acc: 0.8320 | Val Loss: 0.0153 | Val Acc: 0.7656

Epoch 79 | Train Loss: 0.0142 | Train Acc: 0.8320 | Val Loss: 0.0153 | Val Acc: 0.7637

Epoch 80 | Train Loss: 0.0142 | Train Acc: 0.8320 | Val Loss: 0.0153 | Val Acc: 0.7637

Epoch 81 | Train Loss: 0.0142 | Train Acc: 0.8323 | Val Loss: 0.0153 | Val Acc: 0.7656

Epoch 82 | Train Loss: 0.0142 | Train Acc: 0.8323 | Val Loss: 0.0153 | Val Acc: 0.7627

Epoch 83 | Train Loss: 0.0142 | Train Acc: 0.8323 | Val Loss: 0.0153 | Val Acc: 0.7646

Epoch 84 | Train Loss: 0.0141 | Train Acc: 0.8325 | Val Loss: 0.0153 | Val Acc: 0.7646

Epoch 85 | Train Loss: 0.0141 | Train Acc: 0.8325 | Val Loss: 0.0153 | Val Acc: 0.7646

Epoch 86 | Train Loss: 0.0141 | Train Acc: 0.8330 | Val Loss: 0.0153 | Val Acc: 0.7656

Epoch 87 | Train Loss: 0.0141 | Train Acc: 0.8330 | Val Loss: 0.0153 | Val Acc: 0.7656

Epoch 88 | Train Loss: 0.0141 | Train Acc: 0.8330 | Val Loss: 0.0153 | Val Acc: 0.7646

Epoch 89 | Train Loss: 0.0141 | Train Acc: 0.8333 | Val Loss: 0.0153 | Val Acc: 0.7627

Epoch 90 | Train Loss: 0.0141 | Train Acc: 0.8333 | Val Loss: 0.0152 | Val Acc: 0.7666

Epoch 91 | Train Loss: 0.0141 | Train Acc: 0.8335 | Val Loss: 0.0153 | Val Acc: 0.7656

Epoch 92 | Train Loss: 0.0141 | Train Acc: 0.8335 | Val Loss: 0.0153 | Val Acc: 0.7627

Epoch 93 | Train Loss: 0.0141 | Train Acc: 0.8335 | Val Loss: 0.0154 | Val Acc: 0.7539

Epoch 94 | Train Loss: 0.0141 | Train Acc: 0.8335 | Val Loss: 0.0153 | Val Acc: 0.7646

Epoch 95 | Train Loss: 0.0141 | Train Acc: 0.8337 | Val Loss: 0.0153 | Val Acc: 0.7666

Epoch 96 | Train Loss: 0.0141 | Train Acc: 0.8337 | Val Loss: 0.0153 | Val Acc: 0.7588

Epoch 97 | Train Loss: 0.0141 | Train Acc: 0.8337 | Val Loss: 0.0153 | Val Acc: 0.7646

Epoch 98 | Train Loss: 0.0141 | Train Acc: 0.8337 | Val Loss: 0.0152 | Val Acc: 0.7676

Epoch 99 | Train Loss: 0.0141 | Train Acc: 0.8337 | Val Loss: 0.0153 | Val Acc: 0.7637

Epoch 100 | Train Loss: 0.0140 | Train Acc: 0.8337 | Val Loss: 0.0153 | Val Acc: 0.7686

41

**Training performance log**

add Codeadd Markdown

[36]:

**import** matplotlib.pyplot **as** plt

​

*# Create subplots for loss and accuracy*

fig, axes **=** plt.subplots(ncols**=**2, figsize**=**(15, 6))

*# Plotting training and validation loss*

axes[0].plot(train\_loss, label**=**"Training Loss", color**=**'blue', linestyle**=**'-')

axes[0].plot(val\_loss, label**=**"Validation Loss", color**=**'orange', linestyle**=**'--')

axes[0].legend()

axes[0].set\_title("Loss Log")

axes[0].set\_xlabel("Epochs")

axes[0].set\_ylabel("Loss")

axes[0].grid()

​*# Plotting training and validation accuracy*

axes[1].plot(train\_acc, label**=**"Training Accuracy", color**=**'green', linestyle**=**'-')

axes[1].plot(val\_acc, label**=**"Validation Accuracy", color**=**'red', linestyle**=**'--')

axes[1].legend()

axes[1].set\_title("Accuracy Log")

axes[1].set\_xlabel("Epochs")

axes[1].set\_ylabel("Accuracy")

axes[1].grid()

​

plt.tight\_layout()

plt.show()

​

**Predict TestValues**

add Codeadd Markdown

[37]:

**def** predict(img):

*# Transform the image and reshape it for the model*

img **=** transform(img).view(1, 1, 128, 128) *# Assuming input is 128x128 and single channel*

img **=** img.repeat(1, 3, 1, 1) *# Repeat single channel to match 3-channel RGB*

​

*# Ensure model is in evaluation mode*

best\_model.eval()

**with** torch.no\_grad():

*# Move image to GPU if available*

**if** torch.cuda.is\_available():

img **=** img.cuda()

best\_model.cuda()

​

*# Forward pass through the best\_model*

out **=** best\_model(img)

​

*# Optionally, apply softmax if model output is logits*

*# out = torch.nn.functional.softmax(out, dim=1)*

42

*# Get the index of the highest score (predicted class) and its confidence score*

predicted\_class **=** out.argmax(1).item()

confidence\_score **=** torch.nn.functional.softmax(out, dim**=**1)[0][predicted\_class].item()

​

**return** predicted\_class, confidence\_score

​

add Codeadd Markdown

[38]:

**import** torch

**from** PIL **import** Image

**import** torchvision.transforms **as** transforms

​

*# Define the transform to convert images to grayscale and resize*

transform **=** transforms.Compose([

transforms.ToPILImage(), *# Convert NumPy array to PIL Image if necessary*

transforms.Grayscale(num\_output\_channels**=**1), *# Convert to grayscale*

transforms.Resize((128, 128)), *# Resize to the expected input size*

transforms.ToTensor() *# Convert to tensor*

])

**def** predict(img):

*# Transform the image and reshape it for the model*

img **=** transform(img).view(1, 1, 128, 128) *# 1 batch, 1 channel, 128x128 size*

best\_model.eval() *# Ensure the model is in evaluation mode*

**with** torch.no\_grad():

*# Move the image and model to GPU if available*

**if** torch.cuda.is\_available():

img **=** img.cuda()

best\_model.cuda()

​

*# Forward pass through the best\_model*

out **=** best\_model(img)

​

*# Apply softmax to get confidence scores as probabilities*

out **=** torch.nn.functional.softmax(out, dim**=**1)

*# Get the index of the highest score (predicted class) and its confidence score*

predicted\_class **=** out.argmax(1).item()

confidence\_score **=** out[0][predicted\_class].item()

​

**return** predicted\_class, confidence\_score

​

add Codeadd Markdown

**Test values evaluation**

add Codeadd Markdown

43

[45]:

**import** torch

**from** sklearn.metrics **import** confusion\_matrix, classification\_report, accuracy\_score

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

​

*# Function to evaluate the model, generate confusion matrix, classification report, and accuracy score*

**def** evaluate\_model(model, val\_dl, class\_names):

model.eval() *# Set model to evaluation mode*

all\_preds **=** []

all\_labels **=** []

**with** torch.no\_grad():

**for** batch\_data, batch\_labels **in** val\_dl:

*# Move data to GPU if available*

**if** torch.cuda.is\_available():

batch\_data, batch\_labels **=** batch\_data.cuda(), batch\_labels.cuda()

​

*# Forward pass*

outputs **=** model(batch\_data)

\_, preds **=** torch.max(outputs, 1) *# Get predicted class labels*

*# Store predictions and true labels*

all\_preds.extend(preds.cpu().numpy()) *# Move to CPU and convert to NumPy*

all\_labels.extend(batch\_labels.cpu().numpy())

​

*# Generate confusion matrix*

cm **=** confusion\_matrix(all\_labels, all\_preds)

​

*# Generate classification report*

report **=** classification\_report(all\_labels, all\_preds, target\_names**=**class\_names, zero\_division**=**0)

​

*# Calculate accuracy score*

accuracy **=** accuracy\_score(all\_labels, all\_preds)

​

**return** cm, report, accuracy

​

*# Plot the confusion matrix using seaborn's heatmap*

**def** plot\_confusion\_matrix(cm, class\_names, accuracy):

plt.figure(figsize**=**(8, 6))

sns.heatmap(cm, annot**=True**, fmt**=**'d', xticklabels**=**class\_names, yticklabels**=**class\_names)

plt.xlabel('Predicted Labels')\

plt.ylabel('True Labels')

plt.title("Score ({:.2f}%)".format(accuracy **\*** 100))

plt.show()

44

​

*# Example usage*

class\_names **=** ['Class0', 'Class1', 'Class2', 'Class3'] *# Replace with actual class names*

cm, report, accuracy **=** evaluate\_model(best\_model, val\_dl, class\_names)

​

*# Print the classification report*

print("Classification Report:")

print(report)

​

*# Plot the confusion matrix with accuracy*

plot\_confusion\_matrix(cm, class\_names, accuracy)

Classification Report:

precision recall f1-score support

Class0 0.00 0.00 0.00 136

Class1 0.00 0.00 0.00 5

Class2 0.80 0.95 0.87 533

Class3 0.73 0.81 0.77 350

accuracy 0.77 1024

macro avg 0.38 0.44 0.41 1024

weighted avg 0.67 0.77 0.71 1024

add Codeadd Markdown

**Visual inspection of results with confidence scores**

add Codeadd Markdown

[40]:

*# Import necessary libraries*

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

​

*# Create a 3x3 grid of subplots (for 9 images)*

fig, axes **=** plt.subplots(nrows**=**3, ncols**=**3, figsize**=**(15, 6))

index **=** 0 *# To keep track of the images*

*# Loop through the grid to plot images with predictions, confidence, and true labels*

**for** i **in** range(3):

**for** j **in** range(3):

*# Display the image in color*

axes[i][j].imshow(np.array(test[index]["image"])) *# Assumes 'image' is a key in test*

*# Set title with predicted label, confidence score, and actual label*

axes[i][j].set\_title(

"Predicted: {}\nConfidence: {:.2f}%\nActual: {}".format(

index\_label[pred[index]], *# Predicted label*

45

proba[index] **\*** 100, *# Confidence score in percentage*

index\_label[truth[index]] *# Actual label*

),

fontsize**=**10

)

*# Hide the axes for a cleaner look*

axes[i][j].axis('off')

*# Move to the next image in the dataset*

index **+=** 1

​

*# Adjust layout for better spacing between subplots*

plt.tight\_layout()

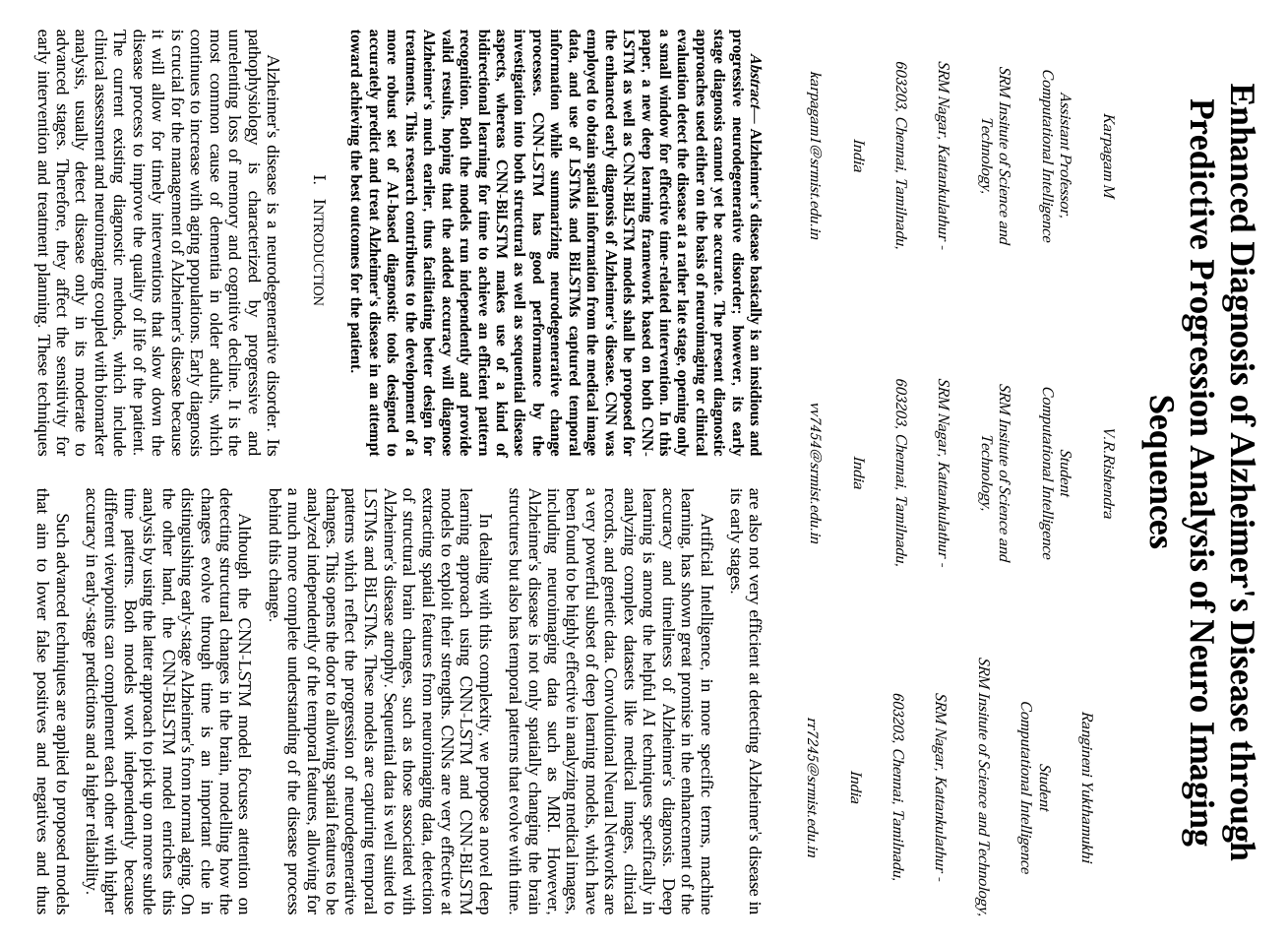
plt.show()

​

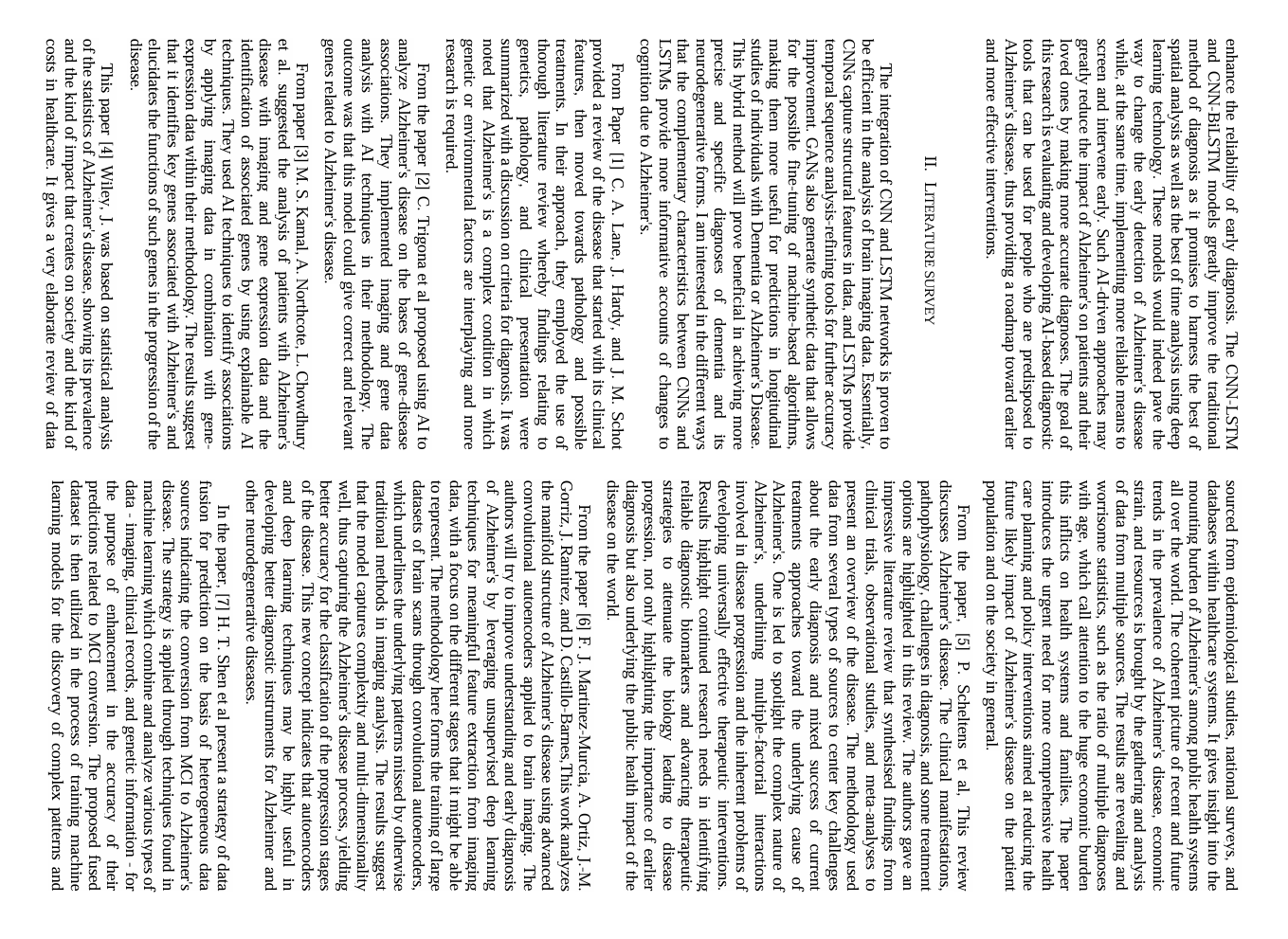
46

**APPENDIX B**

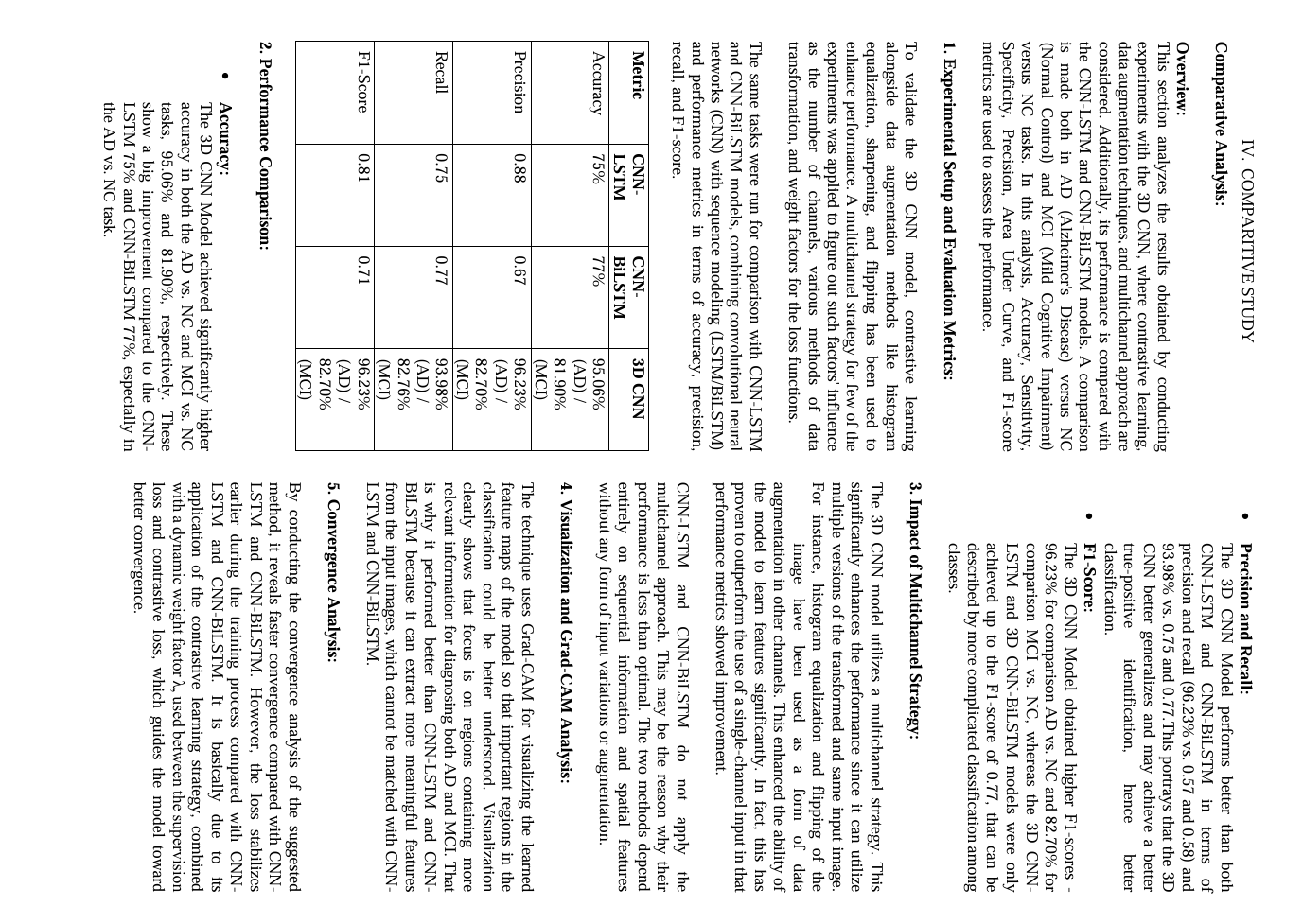
**CONFERENCE PUBLICATION**

****

47



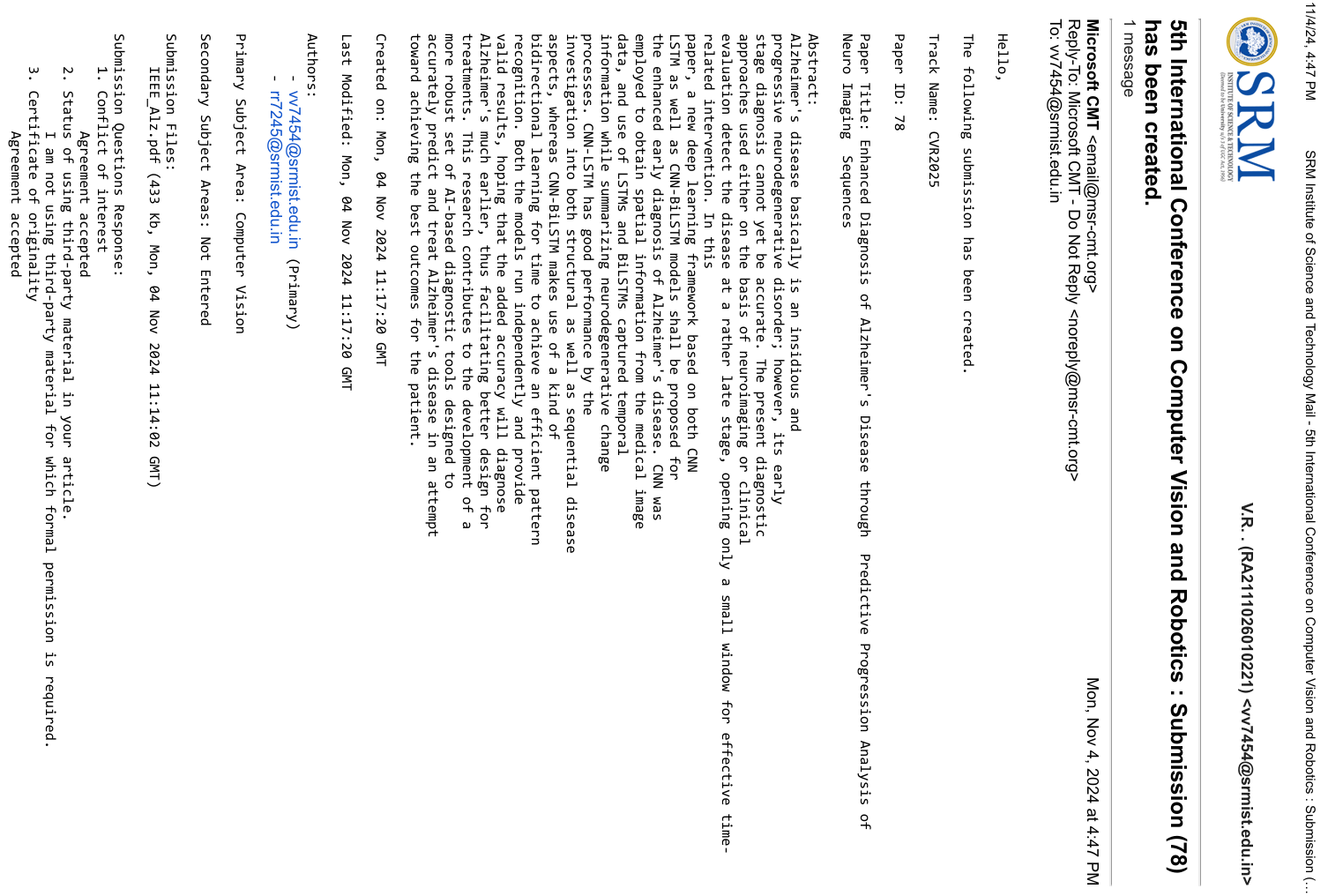
48



49

**APPENDIX C**

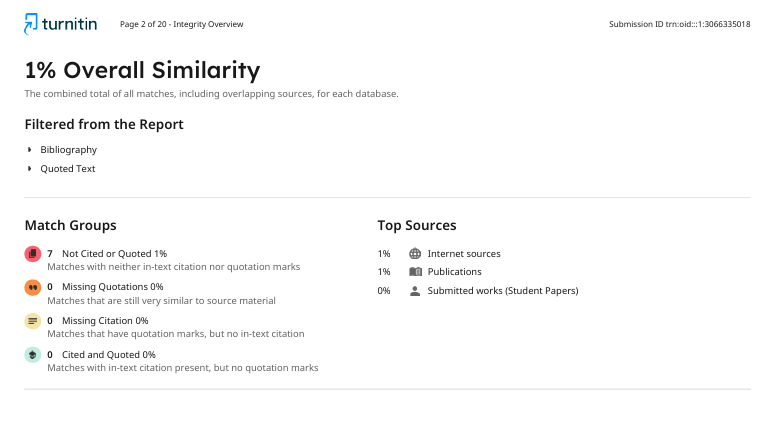
**JOURNAL PUBLICATION**

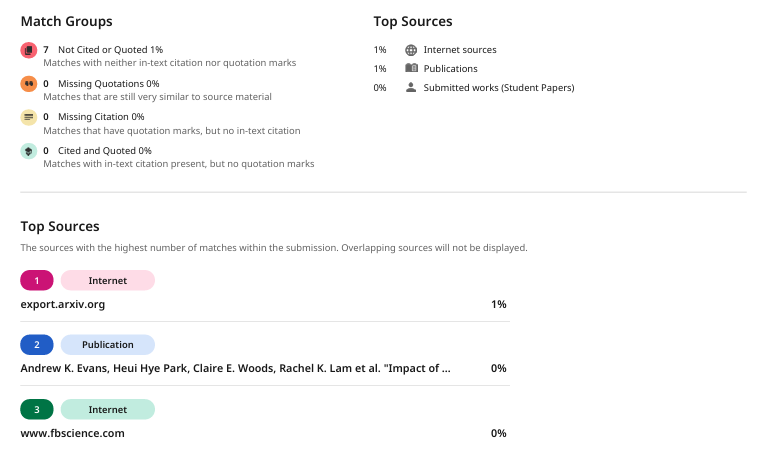
****

50

**APPENDIX D**

**PLAGARISM REPORT**

****

****

51