Detecting Alzheimer's Severity in MRI Scans

Aryana Far, Ashok Sundararaman, Pranav Viswanathan, Jun Park

Problem Statement

Alzheimer's disease is a neurodegenerative disease that affects 1/9 elderly, and 6 million Americans today. The disease is characterized by impaired memory and brain atrophy, which can be visualized through brain imaging. Early, automated, and accurate ML-driven detection could be key in encouraging preventative measures, allowing family planning, and bolstering further scientific research.

Objective

We developed and optimized 21 different ML models, prioritizing accuracy as our main success metric while also minimizing false negatives due to the sensitive medical context of this task.

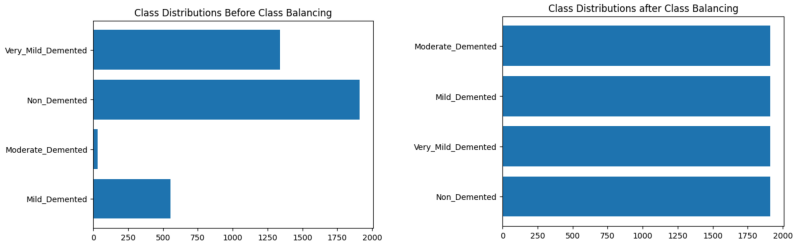
Datasets

To ensure robustness in experimentation, we used two datasets. The first dataset contained 6.4k preprocessed 128x128 pixel brain scan images, grouped into the following 4 Alzheimer's severity classes: non-demented, very mildly demented, mildly demented, and moderately demented. Each image was an independent axial brain scan taken from an arbitrary height but with the same orientation.
The second dataset contained about 86k unprocessed 244x488 brain scan images, divided into the same four classes as our prior dataset. There were about 1.5k unique brains in this dataset, each corresponding with 61 successive axial scans at different heights. This structure allowed for the possibility of processing data into 3D brain volumes.

Modeling Approach

Dataset 1 Preprocessing

In our first dataset, we directed preprocessing efforts toward class balancing because prior preprocessing took care of the necessary transformations. After a 60/20/20 train/validation/test split, we randomly applied 5 augmentations to the training data until classes were evenly distributed. This maintained dataset integrity while improving model generalizability. Class distribution of the training set, pre and post-balancing, is visualized below:



Dataset 2 Preprocessing

For our second dataset, we split our preprocessing into two pipelines: 3D and 2D.

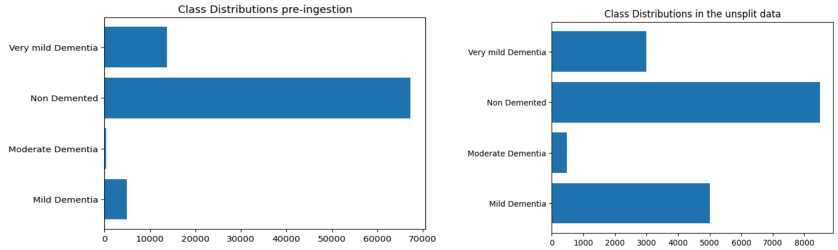
Dataset 2, 3D Preprocessing Pipeline

Our 3D pipeline involved stacking 61 axial cross-sections of brain scans into volumes representing full brains. However, inconsistent image naming conventions, time constraints, and limited computational resources posed consistent challenges in preprocessing. Nonetheless, transformations included downsizing, converting to grayscale, using memory mapping, and reducing pixels to the uint8 type. We experimented with different kinds of class merging and augmentation for model testing purposes. After

all of these steps, we wrote our transformed data to a local directory. To read data in for modeling, we set up a generator for more efficient memory (this limited modeling to CNN-based architectures).

Dataset 2, 2D Preprocessing Pipeline

In our 2D pipeline, class balancing occurred at the data ingestion phase due to massive class imbalances and computational limitations. We achieved this by randomly undersampling the two highly overrepresented non-demented and very mildly demented classes, which constituted around 90% of the dataset. After undersampling, this was the class distribution:



Next, we downsized, re-oriented, and normalized images. After these transformations, we merged the classes to yield binary outcomes. Then, we split our data 60/20/20. Finally, we augmented our training images by 1,000 per class, resulting in a balanced training set of exactly 12,000 images.

Model Development

For dataset 1, we developed 11 unique multiclass classification models, exploring various ML models and optimizing them through architectural adjustments and hyperparameter tuning. For dataset 2's 3D pipeline, we built 4 classification models, some multiclass and some binary. For the 2D pipeline, we built 6 binary classification models. This resulted in 21 models in total.

Model Validation

We applied validation for preliminary evaluations of performance and generalizability, allowing us to hyperparameter tune with an informed approach.

Model Evaluation

We defined model success as improvements in accuracy from our baselines. We also used confusion matrices to gain insights into performance across specific scenarios, paying attention to FNs. Accuracies are indicated in the appropriate modeling sections of the paper, and plots such as confusion matrices are detailed in the appendix.

Model Serving

Though we were not able to achieve model serving, we hope to deploy our models in a secure environment, develop an API to allow for easily inputting MRI scans and receiving insights, and build a monitor to track model performance and health so that retraining can be implemented when necessary.

Dataset 1 Modeling

Models 1 and 2 (Logistic Regression)

In our first set of models for our first dataset, we employed multiclass logistic regression due to its simplicity, computational efficiency, and effectiveness in handling multiclass classification tasks.

Model 1: Baseline Logistic Regression Model
Specifications: learning_rate = 0.01, epochs = 3, stochastic gradient descent
Performance: train accuracy: 45.66%, validation accuracy: 49.06%, test accuracy: 50.63

Model 2: Improved Logistic Regression Model
Specifications: learning_rate = 0.0001, epochs = 20, stochastic gradient descent with momentum 0.9, learning rate scheduling
Performance: train accuracy: 85.15%, validation accuracy: 85.55%, test accuracy: 86.09%

Models 1 and 2 Summary: Our logistic regression models exhibited improvements, with a 36% increase in test accuracy from 50% to 86%. This occurred through a reduction in the learning rate, an increase in epochs, incorporation of SGD momentum, and scheduling of the learning rate. Neither model suffered from overfitting, evidenced by consistent accuracies across the split data. These results underscore the pivotal role of hyperparameter tuning in maximizing performance, even within simple models like logistic regression.

Model 3 (Decision Tree)

Our next model was a Decision Tree, though we acknowledged their limitations in learning pixel relationships in image data and their tendency to overfit. Nonetheless, we explored this model to gain insights into performance characteristics and potential applicability, given their interpretability and simplicity.

Model 3: Baseline Decision Tree Model
Specifications: input images flattened, unlimited number of leaf nodes, max_depth = None, criterion = “gini”, splitter = “best”, min_samples_splitint = 2, min_samples_leafint = 1, max_features = n_features
Performance: train accuracy: 100%, validation accuracy: 66.48%, test accuracy: 66.79%

Model 3 Summary: The decision tree model achieved perfect training accuracy but dropped on validation and test, with 66% accuracy each, indicating severe overfitting. This shows the limitations of trees for image data.

Models 4 and 5 (Random Forest)

Next, we tested Random Forests; similar to Decision Trees, we acknowledged the static feature space. Still, we hoped its ensemble approach would provide resilience against overfitting and allow room for interpretation.

Model 4: Baseline Random Forest Model
Specifications: input images flattened, n_estimators = 10, max_depth = None, criterion = “gini”, min_samples_split = 2, min_samples_leaf = 1, max_features = “sqrt”
Performance: train accuracy: 99.46%, validation accuracy: 78.52%, test accuracy: 76.33%

Hyperparameter Tuning: Our parameter grid for grid search was n_estimators: [20, 50], max_depth: [None, 10], min_samples_split: [2, 5], min_samples_leaf: [1, 2], max_features: [“auto”, “sqrt”]. Through grid search, we chose the specifications detailed below for our improved model.

Model 5: Improved Random Forest Model

Specifications: input images flattened, n_estimators = 50, max_depth = None, criterion = “gini”, min_samples_split = 2, min_samples_leaf = 1, max_features = “auto”
Performance: train accuracy: 100%, validation accuracy: 88.36%, test accuracy: 87.73%

Models 4 and 5 Summary: The Random Forest improved by 11% in test accuracy, from the baseline accuracy of 76% to the improved 88%. Adjustments included increasing the number of estimators by 40 and changing the method of feature splitting. Despite these changes, both models had overfitting. While the baseline model exhibited a substantial overfitting gap of 20%, the improved model managed to alleviate this, reducing it to 10%. These results emphasize the ongoing challenge of generalizability in decision-based models.

Models 6, 7, and 8 (CNN)

Next, we employed CNNs due to their proven effectiveness in handling complex spatial relationships within image data. We hypothesized that CNNs would be effective in detecting specific shapes and regions with atrophy, leveraging their ability to extract hierarchical features from raw pixel data.

Model 6: Baseline CNN Model
Specifications

- **Layers:** Conv2D (32, 3x3, ReLU), MaxPool (2x2), Conv2D (64, 3x3, ReLU), Flatten, Dense (128, ReLU), Dense (4, Softmax)
- **Training:** 3 epochs, Adam optimizer

Performance: train accuracy 83.60%, validation accuracy 71.41%, test accuracy 72.03%

Model 7: Improved CNN Model
Specifications

- **Layers**
    - Conv2D (32, 3x3, ReLU), MaxPool (2x2), Conv2D (64, 3x3, ReLU), MaxPool (2x2), Conv2D (64, 3x3, ReLU), Flatten, Dense (64, ReLU), Dense (4, Softmax)
- **Training:** 3 epochs, Adam optimizer

Performance: train accuracy 83.71%, validation accuracy 74.61%, test accuracy 74.92%

Model 8: Improved CNN Model 2
Specifications

- **Layers:** Conv2D (32, 3x3, ReLU), MaxPool (2x2), Conv2D (64, 3x3, ReLU), MaxPool (2x2), Conv2D (128, 3x3, ReLU), Flatten, Dense (256, ReLU), Dropout (0.3), Dense (4, Softmax)
- **Training:** 5 epochs, Adam optimizer

Performance: train accuracy 92.97%, validation accuracy 92.58%, test accuracy 92.97%

Models 6, 7, and 8 Summary: The three CNN models showed intriguing differences in performance. While the baseline and first improved model both had ~84% train accuracy, there were improvements in generalizability, with test and validation accuracies rising from 72% to 75%. Our first improvements to this CNN involved adding a third 2D convolutional layer with 64 filters and decreasing the units in the dense layer to 64. In the next model, we increased the third 2D convolutional layer’s filters to 128, we increased the dense layer units to 256, and added a dropout layer with a 30% rate. These adjustments led to improvements in generalizability and accuracy,

with accuracies across split data consistently at 93%. This highlights the well-suited nature of CNNs for deciphering complex pixel relationships and the potential for CNNs to excel in image-related tasks.

Model 9, 10, and 11 (Transfer Learning)

Next, we employed Transfer Learning because it allowed us to leverage pre-trained neural network architectures and weights, which have been trained on large-scale datasets. This allows us to benefit from the knowledge gained during training on previous similar tasks, speeding up our training and often yielding better performance.

Model 9: Baseline EfficientNetB0 Transfer Learning Model
Specifications

- **EfficientNetB0 parameters:** weights="imagenet", include_top=False, input_shape=(128, 128, 3), classes = 4, classifier_activation = “softmax”
- **Layers:** EfficientNetB0, Global Avg Pool, Dropout (0.1), Dense (4, Softmax)
- **Training:** Adam optimizer, fine_tune_at = 221, phase 1 learning_rate = 1e-3, phase 1 epochs = 1, phase 2 learning_rate = 1e-4, phase 2 epochs = 1

Performance: train accuracy 64.84%, validation accuracy 63.13%, test accuracy 64.69%

Model 10: Baseline VGG16 Transfer Learning Model
Specifications

- **VGG16 parameters:** weights="imagenet", include_top=False, input_shape=(128, 128, 3)
- **Layers:** VGG16, Flatten, Dense (1028, ReLU), Dense (4, Softmax)
- **Training:** Adam optimizer, fine_tune_at = 8, phase 1 learning_rate = 1e-4, phase 1 epochs = 10, phase 2 learning_rate = 1e-5, phase 2 epochs = 5

Performance: train accuracy 98.50%, validation accuracy 85.39%, test accuracy 85.62%

Model 11: Improved VGG16 Transfer Learning Model
Specifications

- **VGG16 parameters:** weights="imagenet", include_top=False, input_shape=(128, 128, 3)
- **Layers:** VGG16, Flatten, Dense (512, ReLU), Dense (4, Softmax)
- **Training:** Adam optimizer, fine_tune_at = 8, phase 1 learning_rate = 1e-4, phase 1 epochs = 10, phase 2 learning_rate = 1e-5, phase 2 epochs = 5

Performance: train accuracy 99.53%, validation accuracy 90.08%, test accuracy 90.31%

Models 9, 10, 11 Summary: For transfer modeling, we fine-tuned two baselines: the EfficientNetB0 pre-trained model and the VGG16 pre-trained model. VGG16 outperformed EfficientNetB0 by 20% on test data but showed minor overfitting, while EfficientNetB0 showed no signs of overfitting. Despite EfficientNetB0's complex architecture, we preferred VGG16 for its simplicity and efficacy. We continued with VGG16, improving test accuracy to 90%, a 5% increase. Overall, these models did not surpass the performance of our CNNs, hindered by the trade-off between complexity and computational capacity, which limited our ability to maximize the transfer models’ potential through fine-tuning. Still, they achieved good results.

Dataset 2 Modeling – 3D Pipeline

Models 1 and 2 (CNN)

Model 1: CNN for Multi-Classification

Specifications: This model shares the same base architecture as model #6, trained on our first, preprocessed dataset. This model intends to compare the difference in predictive accuracy in a model that works on reduced information in exchange for spatial relationships.
Performance: test accuracy 77.94%, test loss 0.7982

Model 2: CNN for Binary Classification

Specifications: Similar to model 1, this model shares the same base architecture. However, the data being passed in has been grouped for binary classification, to compare the performance of our 3D pipeline in different tasks.
Performance: test accuracy 76.47%, test loss 0.5505

Models 1 and 2 Summary: As mentioned earlier, in the process of building our image stacks, our data points were reduced to a tenth of the original size. Despite the compression and memory-reducing steps we took, both models gave promising results. The model had similar performance between binary and multi-classification in accuracy and loss. It is important to note the skewness of our confusion matrix (refer to Appendix C, Dataset #2: 3D Pipeline, Model 1**)** – specifically, our confusion matrix shows that our model lacks sensitivity in detecting 'Mild Dementia' and 'Very mild Dementia', likely due to the class imbalance within the dataset. Our subsequent models aim to address class imbalance using different augmentation methods.

Model 3 and 4 (CNNs with Image & Volume Augmentation)

Dataset 2 had a large class imbalance. Now working with 3D spatial data, we faced the decision of whether to augment each slice independently before forming our brain volumes or augment the entire volume uniformly after stacking. We hypothesized that the first method could lead to a more resilient model that could handle noisy and missing data, but we also thought it could lose benefits gained from stacking our brain slices, particularly the spatial relations between each slice. On the other hand, the second method could maintain these spatial relationships but introduce unrealistic artifacts within the entire volume, misleading the model about the true form of brain structures. To analyze trade-offs, we built a different model for each approach and compared their predictive accuracies.

Model 3 Performance: test accuracy 44.5%, validation accuracy 47.0%, test loss 2.378, validation loss 1.322
Model 4 Performance: test accuracy 55.3%, validation accuracy 69.3%, test loss 1.707, validation loss 1.058

Models 3 and 4 Summary: Unlike the first two models, Models 3 and 4 experience a singular training epoch. Augmentation to balance classes drastically increases the amount of data, creating further strain on computational resources. Hence, while these models underperform in comparison to our previous two, which do not attempt class balancing, they still demonstrate the value of image augmentation in 3D modeling. Furthermore, these models allow us to compare different augmentation strategies in the 3D pipeline. Although Model 4 crashed partway through training the second epoch, it reached test accuracies of ~70% (performing far better than Model 3). These initial results indicate to us that our strategy of augmenting entire volumes is better suited for training models in the pipeline because this augmentation strategy maintains spatial relations between each slice, which is valuable to our model.

Dataset 2 Modeling – 2D Pipeline

Models 1 and 2 (Logistic Regression)

Model 1: Baseline Logistic Regression Model
Specifications: learning_rate = 0.01, epochs = 1, batch_size = 100, Adam optimizer
Performance: Train Accuracy 68.23%, Validation Accuracy 81.57%, Test Accuracy 81.93%

Model 2: Improved Logistic Regression Model
Specifications: learning_rate = 0.01, epochs = 20, stochastic gradient descent with momentum 0.9, learning rate scheduling
Performance: train accuracy 93.12%, validation accuracy 90.49%, test accuracy 91.12%

Models 1 and 2 Summary: These logistic regression models exhibited notable differences in performance, with test accuracy improving from 81% to 91%. This was achieved through a learning rate reduction, increase in epochs, incorporation of SGD momentum, and learning rate scheduling. As in our initial dataset, the models did not suffer from major overfitting issues. These results confirm the power of simple hyperparameter tuning in enhancing performance, particularly in simple models like logistic regression. Furthermore, this performance signposts the immense potential for CNNs, which go beyond logistic regression in their layered architecture.

Models 3 and 4 (CNN)

Model 3: Baseline CNN Model
Specifications

- **Layers:** Conv2D (32, 3x3, ReLU), MaxPool (2x2), Conv2D (64, 3x3, ReLU), MaxPool (2x2), Conv2D (64, 3x3, ReLU), Flatten, Dense (128, ReLU), Dropout (0.2), Dense (1, Sigmoid)
- **Training:** 3 epochs, Adam optimizer

Performance: train accuracy 92.86%, validation accuracy 94.23%, test accuracy 94.64%

Model 4: Improved CNN Model
Specifications

- **Layers:** Conv2D (32, 3x3, ReLU), MaxPool (2x2), Conv2D (64, 3x3, ReLU), MaxPool (2x2), Conv2D (128, 3x3, ReLU), Flatten, Dense (256, ReLU), Dropout (0.3), Dense (1, Sigmoid)
- **Training:** 5 epochs, Adam optimizer

Performance: train accuracy 99.03%, validation accuracy 98.38%, test accuracy 98.62%

Models 3 and 4 Summary: From our baseline to improved CNN models, we see an increase in test accuracy from 95% to 99%, highlighting the effectiveness of tuning. Specifically, doubling the number of filters in the final 2D convolutional layer, increasing dense layer units, and adjusting the dropout rate from 20% to 30% substantially improved model performance. Moreover, the minimal differences across split data accuracies suggest robust generalizability. Overall, these CNN models demonstrated unparalleled performance in terms of test accuracies and generalizability, underscoring the robustness of CNNs in image classification.

Models 5 and 6 (Transfer)

Model 5: Baseline EfficientNetB0 Transfer Learning Model
Specifications

- **EfficientNetB0 parameters:** weights="imagenet", include_top=False, input_shape=(244, 122, 3), classes = 2, classifier_activation = “sigmoid”
- **Layers:** EfficientNetB0, Global Avg Pool, Dropout (0.1), Dense (2, Sigmoid)
- **Training:** Adam optimizer, fine_tune_at = 221, phase 1 learning_rate = 1e-3, phase 1 epochs = 1, phase 2 learning_rate = 1e-4, phase 2 epochs = 1

Performance: train accuracy 82.05%, validation accuracy 87.16%, Test Accuracy 87.90%

Model 6: Improved VGG16 Transfer Learning Model
Specifications

- **VGG16 parameters:** weights="imagenet", include_top=False, input_shape=(244, 122, 3)
- **Layers:** VGG16, Flatten, Dense (512, ReLU), Dense (2, Sigmoid)
- **Training:** Adam optimizer, fine_tune_at = 8, phase 1 learning_rate = 1e-4, phase 1 epochs = 10, phase 2 learning_rate = 1e-5, phase 2 epochs = 5

Performance: train accuracy 98.82%, validation accuracy 96.91%, test accuracy 97.14%

Models 5 and 6 Summary
For these transfer models, we again used the two pre-trained models EfficientNetB0 and VGG16. The VGG16 model exhibited a 10% superior performance on the test data than EfficientNetB0. Neither showed significant differences in training, validation, and test accuracy, unlike our prior dataset, where VGG16 suffered from overfitting. Overall, our VGG16 transfer models fell slightly short of matching the performance of our advanced CNN model for this dataset’s 2D pipeline, but only by 1%. Although we were limited in our ability to fully utilize the transfer models’ potential through fine-tuning, these models still achieved impressive results.

<u>Conclusion</u>
In both datasets, our advanced CNNs proved to deliver the best performance, as seen by their generalizability and high test accuracy. Among the models in our first dataset, Model 8 had the highest test accuracy at 92.97%, while among the 2D pipeline models in our second dataset, Model 4 had the highest test accuracy at 98.62%. Despite these two models sharing similar architectures, the superior performance of Model 4 in our second dataset's 2D pipeline is due to preprocessing in downsampling the underrepresented class during data ingestion, resulting in a highly balanced test set, as well as converting model outcomes from categorical (4) to binary. Following closely were Transfer Models, with our top-performing model being Model 6 from our second dataset's 2D pipeline, with a test accuracy of 97.14%. Overall, our 22 models exhibit a blend of breadth and depth in our approaches: breadth in our diverse selection of models and depth in our focus on deep learning methodologies. Although our 3D pipeline approach did not yield the expected results, we intend to persist in testing these methods while ensuring integrity in our approach to truly assess performance.

Appendix

A. Data Sources

1. **Alzheimer MRI Preprocessed Dataset | Kaggle**

This dataset, collected from several sources such as websites and hospitals, contains 6,400 Preprocessed MRI Images, split into 4 classes: “Mild Demented” (896 images), “Moderate Demented” (64 images), “Non Demented” (3,200 images) and “Very Mild Demented” (2,240 images).

2. **OASIS Alzheimer's Detection**

This dataset contains unprocessed MRI images from 1,500 patients, featuring a selection of slices that depict various sections of the brain along the z-axis. The images have been classified into 4 classes: “Mild Dementia” (5,002 Images), “Moderate Dementia” (488 images), “Non Demented” (67,000 images), and “Very mild Dementia”(13,700 images).

B. Notebooks

alzheimers_severity_ML_models

This directory contains the following 4 notebooks, which include preprocessing and modeling efforts performed on our first dataset, sourced from here, and our second dataset, sourced from here.

1. **dataset1_models.ipynb**

This Python notebook houses our 11 models for our first dataset, ranging from traditional ML models to deeper exploration of deep learning models such as CNNs and Transfer models.

2. **dataset2_3d_models.ipynb**

This Python notebook houses our first two models for the 3D pipeline of our second dataset, focusing CNNs for stacked image volumes, or 3D images, with different classes collapsing – one version with three classes and one version with binary classes.

3. **dataset2_3d_augmented_models.ipynb**

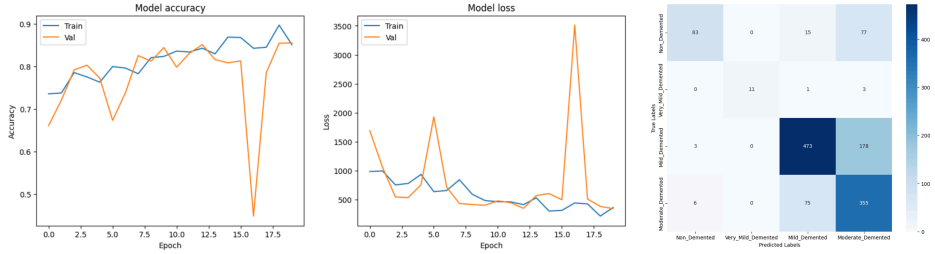
This Python notebook houses our second two models for the 3D pipeline of our second dataset, focusing CNNs for stacked image volumes, or 3D images, with different types of augmentation.

4. **dataset3_2d_models.ipynb**

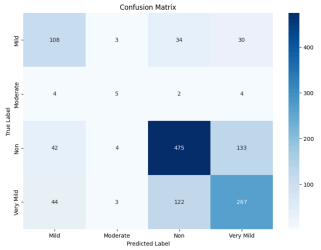
This Python notebook houses our 6 models for our second dataset’s 2D pipeline, featuring two logistic regression models, two CNNs, and two transfer models.

C. Model Results & Figures

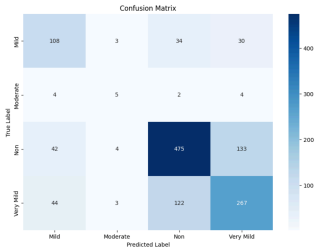
- **Dataset #1:**
  - **Model 2: Improved Logistic Regression Model**



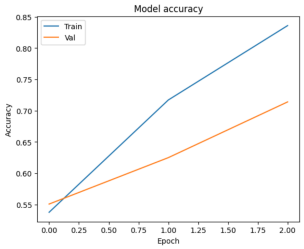
  - **Model 3: Baseline Decision Tree Model**

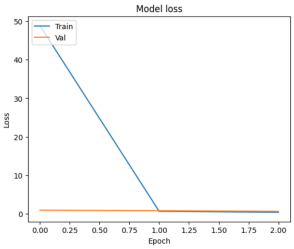


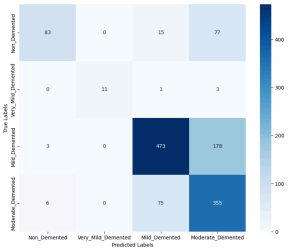
- **Model 5: Improved Random Forest Model**



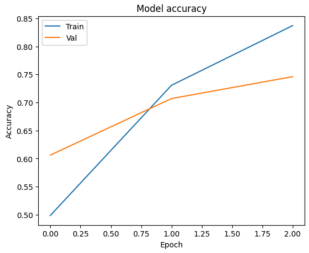
- **Model 6: Baseline CNN Model**

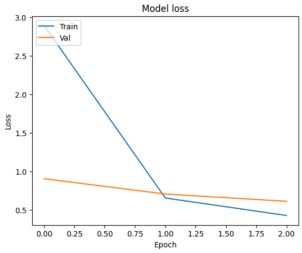


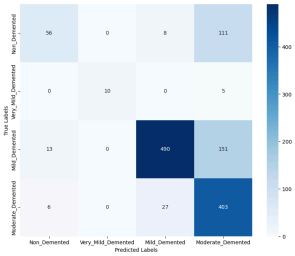




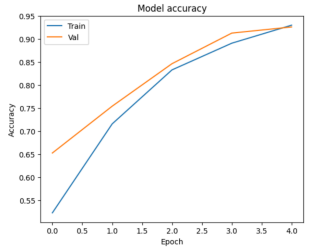
- **Model 7: Improved CNN Model**

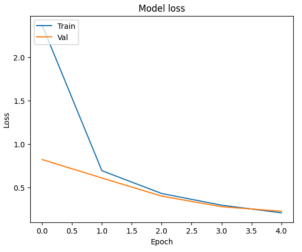


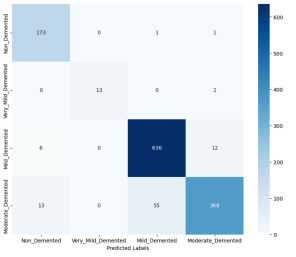




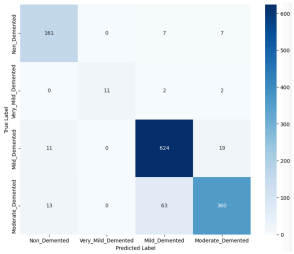
- **Model 8: Improved CNN Model 2**



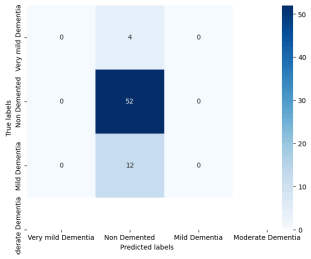




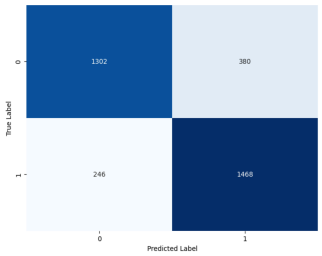
- **Model 11: Improved VGG16 Transfer Model**



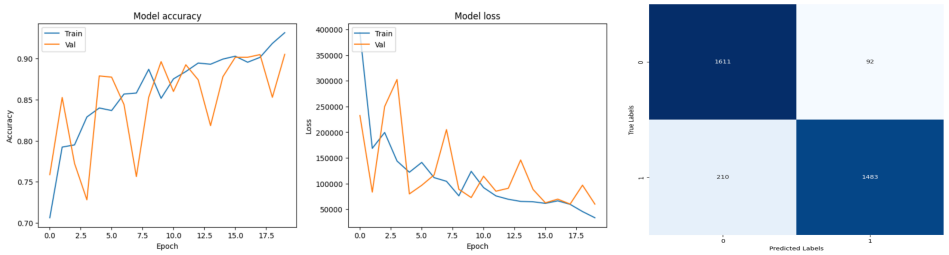
- **Dataset #2: 3D Pipeline**
  - **Model 1: CNN for Multi Classification**



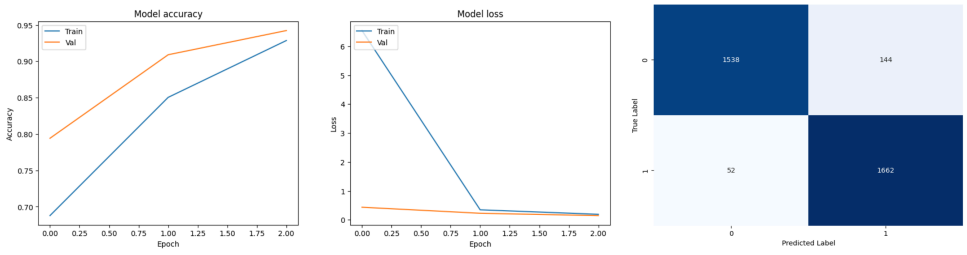
- **Dataset #2: 2D Pipeline**
  - **Model 1: Baseline Logistic Regression Model**



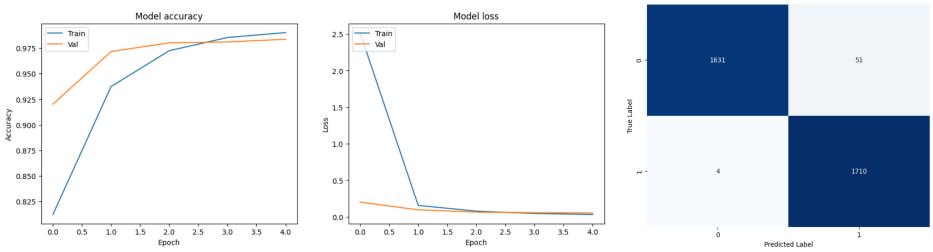
  - **Model 2: Improved Logistic Regression Model**



  - **Model 3: Baseline CNN Model**



  - **Model 4: Improved CNN Model**



- **Model 6: Improved VGG16 Transfer Model**