CSC3432 report

Kristen Karsburg Arguello

December 2024

1 Task 1 - Network Visualization

1.1 Methods

1. Decompress the sequences file, split it into lines, and remove empty ones.
2. Set up the BioPython pairwise aligner with gap penalties of -10 and -0.5, use the BLOSUM-62 matrix, and set the mode to 'local', as exemplified in the module documentation.
3. For each sequence, compute alignment scores with others, normalize by the longest sequence length, as longer sequences have more opportunities for combinations, while shorter sequences may be more similar but have fewer opportunities for string matches, and store it in a zero-filled matrix.
4. Set a threshold based on score distribution (50% of max similarity) and create a graph with nodes as sequences and edges for scores above the threshold.
5. Use a community detection algorithm like Infomap, colour nodes by cluster, scale by degree, and plot the graph as in Figure 1.
6. Create a table of the network's topological attributes, as in Table 1, to identify clusters.

1.2 Results

The analysis of protein sequences through pairwise alignment and community detection reveals the presence of distinct clusters of homologous proteins. By employing a threshold, set at 50% of the maximum similarity score, we can connect proteins that exhibit an above-average similarity. This threshold allows for identifying homologous relationships, where proteins share a common evolutionary ancestor. Adjusting the threshold can yield different clusters; a higher threshold requires proteins to be extremely similar, while a lower threshold can capture more distant homologues.

In the constructed network, seen on Figure 1, clusters are visually represented by nodes of different colours and sizes. These clusters suggest evolutionary relationships, functional similarities, and structural conservation among the proteins. Proteins that do not form clusters are likely to lack strong similarity with others, possibly indicating a lack of common ancestry.

The network's topological attributes, seen on Table 1, such as transitivity, further support the presence of clusters. A high degree of transitivity (over 0.7) suggests that neighbouring nodes

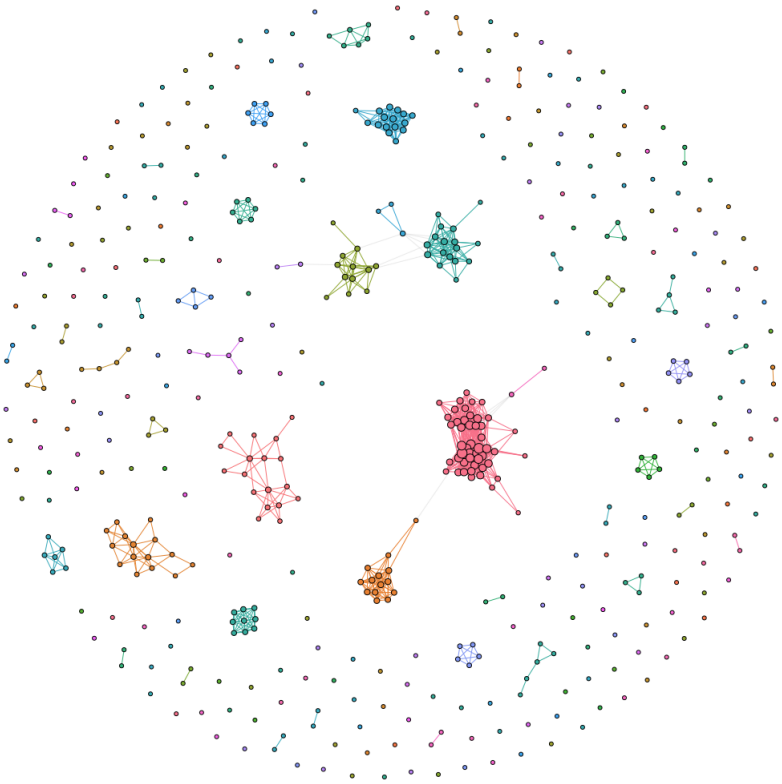

Figure 1: Visualization of the generated network with the nodes coloured and sized by their cluster memberships and connections.

are also neighbours, indicating a high level of clustering. This tendency can be attributed to common ancestry, where evolutionary processes lead to the formation of homologous protein clusters. Functional conservation also plays a role, as natural selection preserves sequences with beneficial functions across evolution. Additionally, gene duplication and divergence contribute to homologous clusters, where one gene copy retains the original function while the other undergoes mutations.

Overall, the presence of clusters in the network highlights the evolutionary and functional relationships among protein sequences, providing insights into their shared origins and conserved functionalities.

Metric	Value
Diameter	6
Girth	3
Average Path Length	2.326
Average Degree	3.953
Assortivity Degree	0.690
Transitivity Average	0.777
Transitivity Undirected	0.706

Table 1: Network topological properties.

2 Task 2 - simple ML

2.1 Methods

1. Load datasets, keep only CDR attributes from the first visit, and left join other visits using patient ID. Remove non-varying, non-patient-related, or identification columns (e.g., visit, MRI delay, patient ID, MRI ID). Fill missing data with the mode, correct CDR typos, and convert nominal (sex, hand) and ordinal categories (CDR levels, SES) to ordered integers.

2. Add a column indicating if a patient's condition worsened in any of the later visits compared to the first, to aim on preventing worsening at any given point, considering improvements as non-related.

3. Split the dataset: 20% for testing, the rest for training.

4. Create a pipeline with a scaler and logistic regression classifier, as shown in Figure 2. Perform grid search, with the listed values below, with shuffled cross-validation (15 folds) for each regularization type (None, L1, L2), measuring ROC-AUC and F1.

    - No penalization grid: solvers (lbfgs, newton-cg), class-weight (balanced, none), fit-intercept (true, false).
    - L1 penalization grid: C (0.1 to 1, step 0.1), solver (liblinear), fit-intercept (true, false), intercept scaling (0.1 to 5, step 0.25), class-weight (balanced, none).
    - L2 penalization grid: C (0.1 to 1, step 0.1), solvers (lbfgs, liblinear, newton-cg, newton-cholesky), class-weight (balanced, none), fit-intercept (true, false).

5. Calculate mean learning performance with a 95% confidence interval for each regularization type.

6. Plot a heatmap of disease progression for each initial condition level and create a table showing dataset value count distribution.

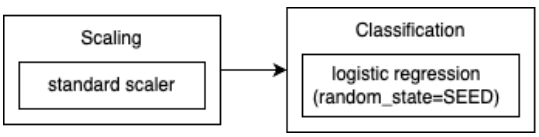

Figure 2: Set pipeline as a flow diagram.

2.2 Results

Predicting dementia worsening after a first visit is challenging, as the model's ROC-AUC scores are only slightly better than random guessing, indicating difficulty in distinguishing between worsening and non-worsening patients. Low F1 scores further highlight the model's struggle to accurately predict the positive class. These metrics suggest that while predictions are possible, they are not reliable or accurate enough for effective use.

A major issue is the dataset imbalance, with only 20% of patients experiencing worsening, as shown in Table 2. Figure 3 illustrates that most patients remain stable, with few worsening or improving. This imbalance biases the model towards predicting the majority class, leading to poor performance. Techniques like resampling, different evaluation metrics, or algorithms for

Worsening	Proportion
No	0.79
Yes	0.21

Table 2: Dataset distribution.

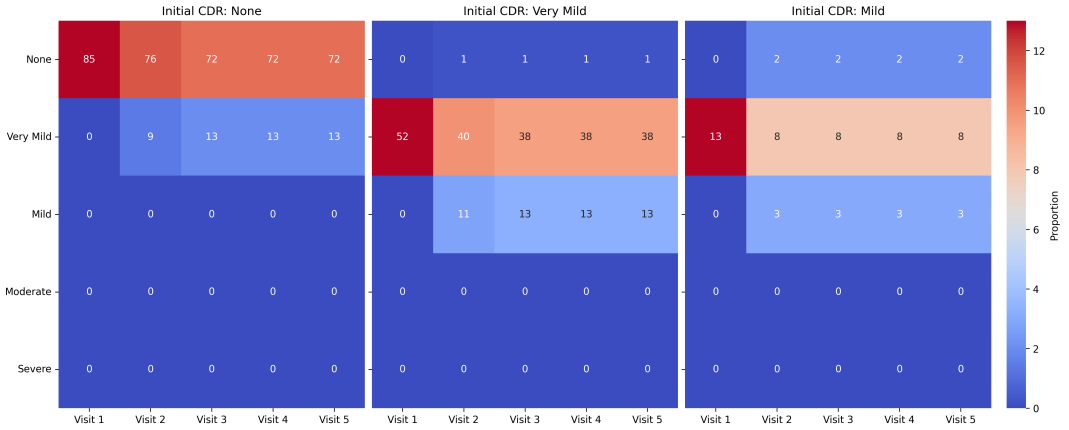

Figure 3: Disease progression over time grouped by started level condition.

imbalanced data could improve performance. The grid search identified balanced class weight as the best parameter to mitigate this imbalance.

Different regularization penalties were tested, with results in Table 2.2. The model without regularization achieved the highest mean ROC-AUC, slightly better than other penalties, followed by the L2 penalty. However, both scores are only marginally better than random guessing, indicating limited predictive power. The L1 penalty resulted in a mean ROC-AUC below random guessing, showing its ineffectiveness. These results highlight the need for careful regularization selection to optimize performance. Despite slightly better results with no regularization and the L2 penalty, overall performance is inadequate for clinical use risking misdiagnosis. Improving model performance and reliability is crucial before clinical deployment, requiring addressing data imbalance, enhancing training, and validating with comprehensive datasets.

Penalty	Mean ROC-AUC	ROC-AUC 95% CI	Mean F1	F1 95% CI
NO	0.554	[0.544 , 0.564]	0.225	[0.128 , 0.323]
L1	0.484	[0.481 , 0.488]	0.182	[0.173 , 0.192]
L2	0.552	[0.549 , 0.555]	0.210	[0.19 , 0.23]

Training scores for each regularization configuration.

Training scores for each regularization configuration.

Table 3:
Training scores for each regularization configuration.

3 Task 3 - advanced ML

3.1 Methods

1. Load the appointments and participants datasets. Perform a left join of participants into appointments, ensuring correct data types. Exclude patients with fewer than 5 appointments and remove the participant ID column, as IDs are irrelevant for model decisions.

2. Develop a pipeline for the column transformer (see Figure 4). Incorporate imputers, scalers, and encoders. Use a one-hot encoder for non-ordinal categories and handle unknown values to accommodate dataset variations.

3. Select algorithms to test: Support Vector Machine (SVM), Random Forest (RF), and Gradient Boosting (GB). Define parameter grids for each:

   - SVM grid: kernel (linear, rbf), C (0.1, 0.5, 0.7, 1), class-weight (balanced), gamma (scale, auto).
   - RF grid: n_estimators (300, 400), max_depth (None, 10), min_samples_leaf (1, 3), bootstrap (True), criterion (gini), class-weight (balanced).
   - GB grid: learning rate (0.1, 0.5), n_estimators (300, 400), max_depth (3, 5).

4. For each algorithm, construct a pipeline with the column transformer, feature selector, and grid search as the classifier (see Figure 4). Use nested cross-validation with 6 outer and 2 inner shuffled folds. Select the best algorithm based on the F1 score, which effectively measures positive class prediction (no-show).

5. Generate precision-recall curves and aggregate predictions from the best models to create a pooled confusion matrix for each outer fold. Use permutation importance to assess feature impact on positive class predictions with the best model.

6. Implement a Stacking Classifier with Logistic Regression as the final estimator. Stack the best models from all tested algorithms in the initial grid search, excluding the top-performing algorithm. Use the same pipeline setup as the best algorithm but replace the grid with the stacking classifier (see Figure 4).

7. Plot learning curves for both the top-performing algorithm and the stacking classifier.

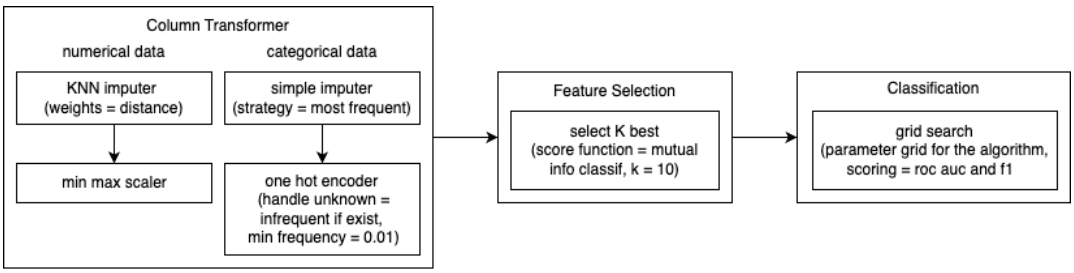

Figure 4: Set pipeline as a flow diagram.

3.2 Results

To find the best algorithm and hyperparameters for predicting missed appointments, various combinations were analyzed. Random Forest (RF) proved most effective, using these hyperparameters:

- Bootstrap: True;
- Class weight: balanced;
- Criterion: gini;
- Max Depth: 10;
- Min Samples per Leaf: 3;
- Estimators: 400;

RF outperformed Support Vector Machine (SVM) and Gradient Boosting (GB) in ROC-AUC and F1 scores (Table 4). The F1 score shows moderate prediction ability, while this score indicates that the model is the best among the tested options, it also indicates a margin for improvement.

Algorithm	ROC AUC	F1
SVC	0.662	0.394
RF	0.734	0.452
GB	0.7182	0.406

Table 4: Classifier algorithms performance (in the same pipeline).

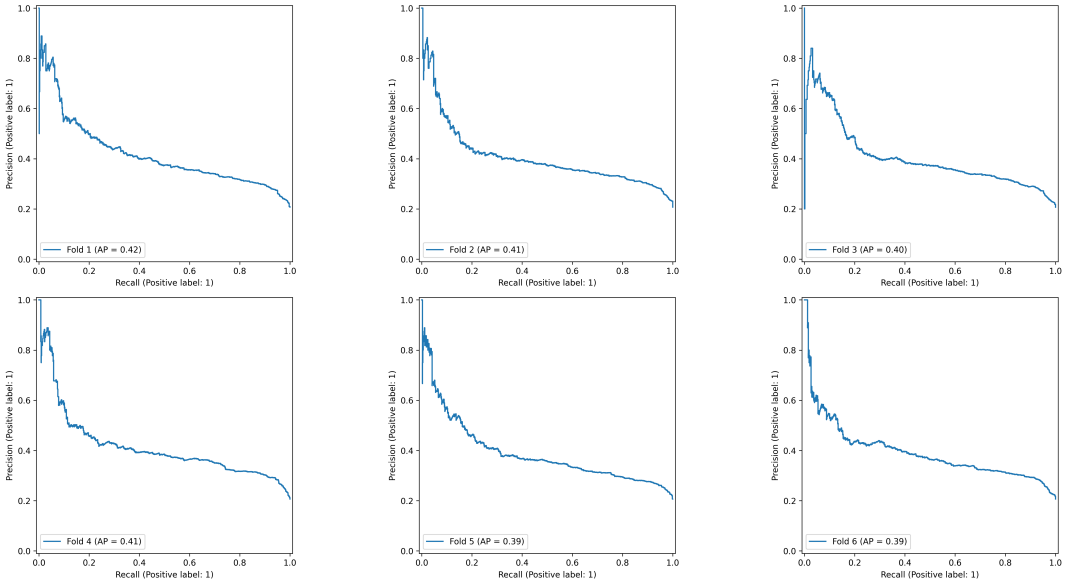

Figure 5: Single classifier precision-recall curves for each fold.

Key features include scheduling time, age, total visits, appointment date, and SMS reminders (Figure 6). These key features are indispensable for users to provide to the model, as they are the most impactful on predicting the no-show class, enabling insightful predictions and help in

targeted interventions like extra reminders for patients. The precision-recall curve (Figure 5) shows high precision but low recall for every cross validation fold, missing many positives, likely due to class imbalance or feature issues. The pooled confusion matrix (Figure 8) that mimics the behaviour for the whole model using each fold, shows 2,000 missed no-shows and 3,000 false positives, with 1,600 correct no-show predictions, further illustrating the challenge of correctly predicting that class.

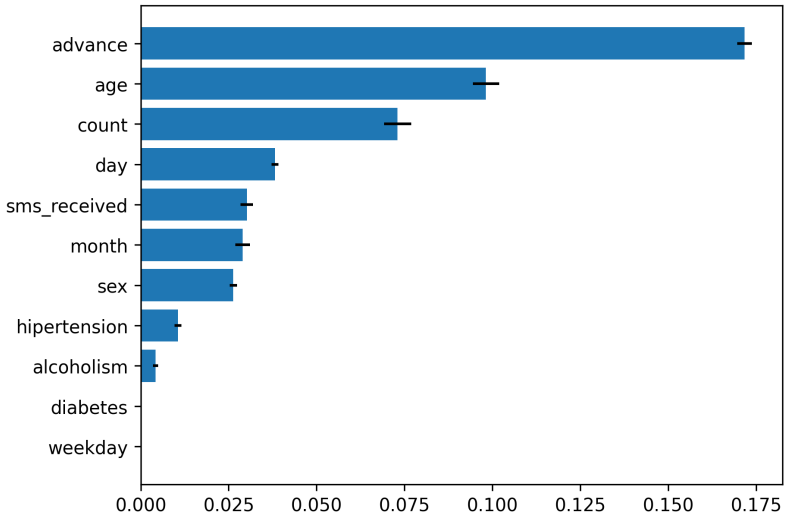

Figure 6: Features impact on the best model output (no-show class).

A stacking model was considered to improve the prediction ability of the model, but RF excelled in the validation scores, which is more important to determine the generalization capability of a model. The learning curve (Figure 7) shows RF's validation score improves with more data, reducing noise. The stacking model's learning curve showed faster validation improvement but was less robust than RF for this task when the data samples were all covered.

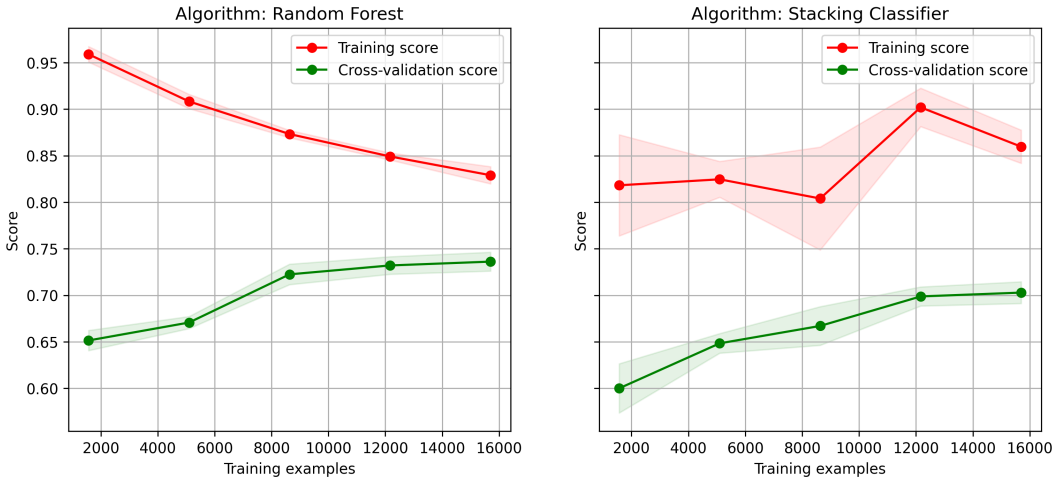

Figure 7: Single and stacked classifiers learning curves.

In summary, the RF algorithm, with tuned hyperparameters, is the best choice for predicting appointment no-shows, despite needing improved recall. Its robustness makes it valuable for healthcare providers, offering insights to reduce no-shows and conserve resources. By leveraging key features, it aids in deciding on patient reminders, though not for diagnosis. With no significant negative consequences, it is suitable for deployment, and its performance can improve with more no-show data. The current baseline provides a solid foundation for future enhancements.

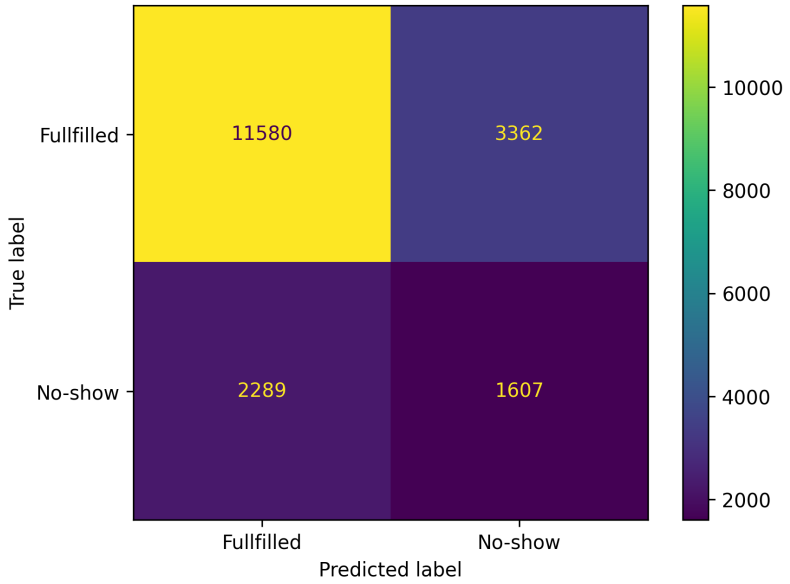

Figure 8: Single model pooled confusion matrix.

4 Task 4 - deep learning

4.1 Methods

1. Gather domain names, superfamily names, and CATH IDs for each sequence. Join this data into one dataset, using CATH ID and domain name as keys. Exclude sequences in superfamilies with fewer than 1000 occurrences and those without a superfamily name. Retain only the sequences and superfamily names columns in the final dataset.

2. Convert labels to binary; transform sequences into vectors; pad sequences to a max length of 300 (Figure 9).

3. Use a random search tuner to build a transformer, optimizing validation loss over 10 trials. Decide on embedding dimensions (16 or 32), encoder layers (2 or 4), heads per layer (2 or 4), dense layers (1 or 2), units per layer (32 or 64), and learning rate (0.001 or 0.01) for the Adam optimizer. Use accuracy and AUC as metrics. Transformer architecture is in Figure 9.

4. Implement a 4-fold splitter for cross-validation, separating test and train datasets for evaluation and hyperparameter tuning.

5. For each fold, train the model for 50 epochs using the best hyperparameters. Record train, validation, and evaluation performance, and the top 5 tuner trials to assess hyperparameter impact.

6. Identify the best epoch by lowest validation loss. Repeat the previous step with optimized epochs.

7. Calculate fold means for cross-validation; plot performance graphs for chosen epochs and display tables with trial and fold results for analysis.

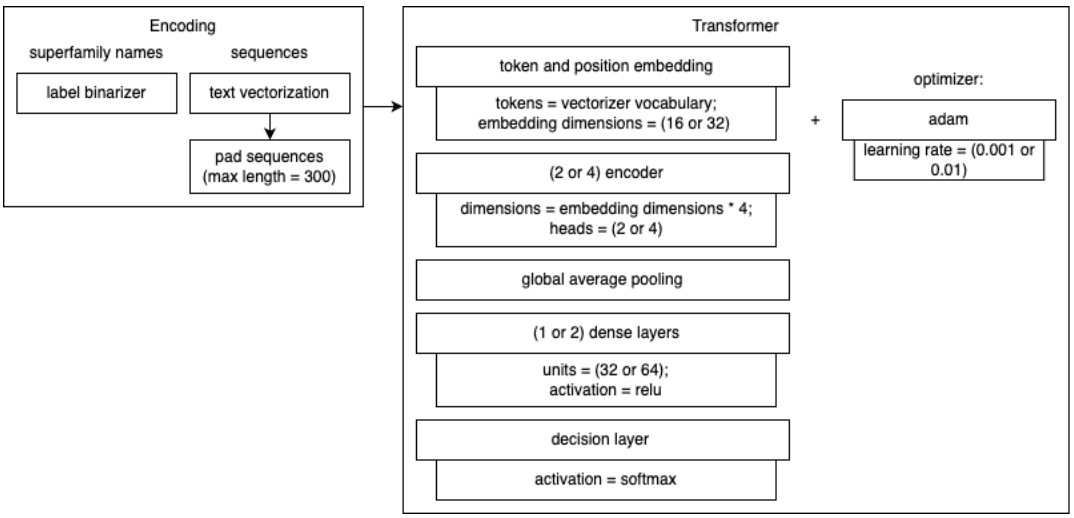

Figure 9: Preprocessing and transformer architecutre.

4.2 Results

Predicting protein function (CATH superfamily) from sequences using a neural network shows promise. The model uses a transformer architecture with encoder components to leverage attention mechanisms, focusing on each protein sequence letter to identify crucial patterns. This makes transformers ideal for sequence processing.

Performance was tested with various encoder layers, attention heads, dense layers, and learning rates (Table 5). Results show smaller learning rates improve loss, suggesting they allow more effective learning with more epochs. More attention heads and a single dense layer performed best with a higher learning rate.

Loss	Encoder Layers	Heads/encoder	Dense Layers	Units/dense layer	Learning Rate
2.378	2	2-4	1	32	0.001
2.847	2	4-4	2	32-32	0.01
2.996	2	2-4	1	64	0.01
3.094	2	2-2	2	64-64	0.01
3.299	4	2-2-4-2	1	64-32	0.01

Table 5: Different losses results from different hyperparameters combinations.

Fold	Loss	Accuracy	AUC
1	2.549	0.478	0.917
2	2.493	0.475	0.925
3	2.471	0.489	0.926
4	2.433	0.475	0.928
Mean	2.487	0.479	0.924

Table 6: Cross validation results.

Four-fold cross-validation with a separate test set was used to avoid data leakage during hyperparameter tuning. Overfitting was observed at 50 epochs, where training loss decreased but validation loss increased after a point (Figure 10). The optimal epoch count was found by

minimizing validation loss, and the model was retrained (Figure 11), enhancing generalization. The model's mean AUC was 0.920 (Table 6), showing strong protein function prediction, though accuracy was low, indicating room for improvement in prediction precision.

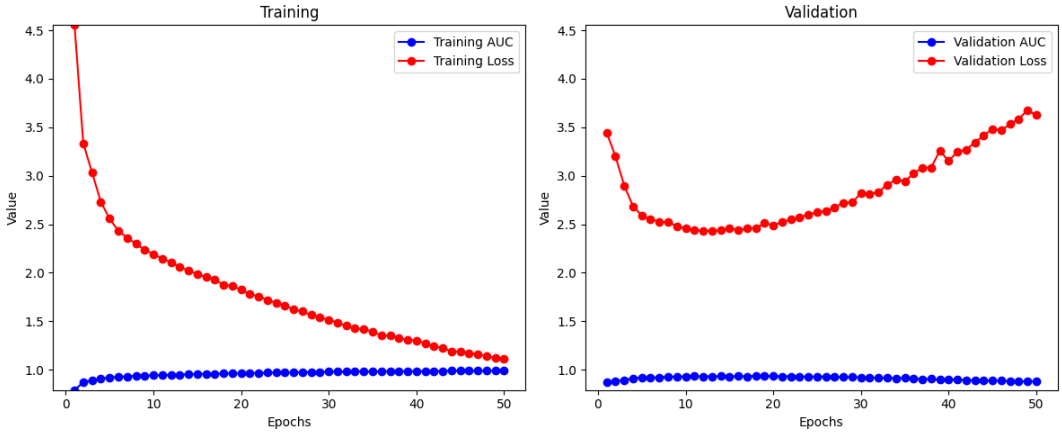

Figure 10: Model performance with 50 epochs training.

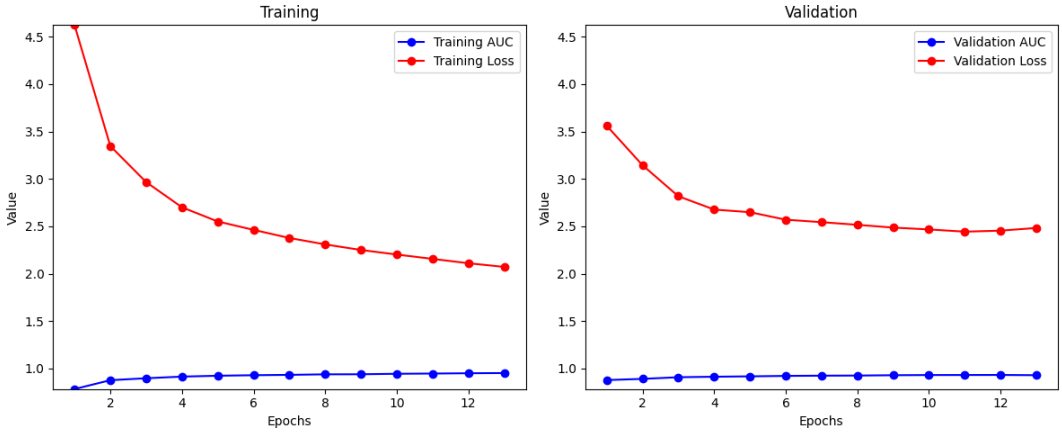

Figure 11: Optimized model performance with the best amount of epochs.

The dataset's characteristics influenced model performance. The S60 dataset, with more occurrences per protein family, helped the model capture sequence patterns better than the non-redundant S40 dataset. This richer data improved the model's ability to predict protein functions, achieving an AUC near 95%, showing promise for medical applications like disease understanding and therapy development. However, despite its strong predictive capabilities, the model isn't infallible. Experimental validation and further analyses are necessary to ensure accuracy and reliability in medical contexts.

5 Task 5 - image classification

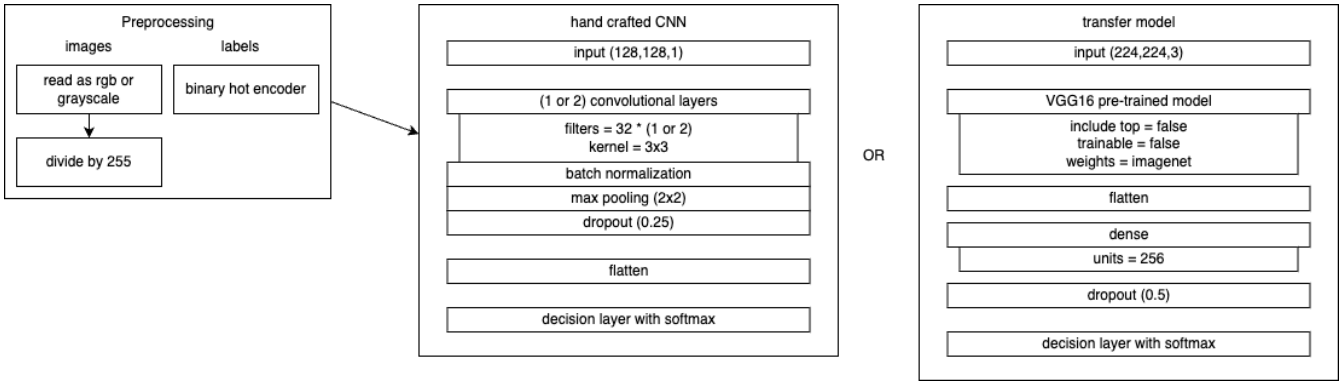

Figure 12: Preprocessing and neural networks set.

5.1 Methods

1. Load the train, validation, and test sets, storing labels and images separately. Load images in two formats: as grayscale with a resolution of 128x128 for the single model (grayscale is chosen because retinal OCT images are typically grayscale, and converting them reduces noise, aiding pattern recognition), and as RGB with a resolution of 224x224 for the transfer model (required for the transfer model's input layer). Compute class weights to ensure balanced tuning and training. Normalize images to a [0,1] range by dividing each pixel by 255 and convert classes to binary class matrices.

2. Develop the CNN model as illustrated in Figure 12. Use an Adam optimizer with a tuned learning rate (options: 0.001 and 0.01). Set accuracy and AUC as metrics. Implement a random search tuner aimed at maximizing validation AUC, with a maximum of 10 trials.

3. After hyperparameter tuning with the specified tuner and a batch size of 128, train the CNN for 36 epochs using the same batch size and the computed class weights. Save the top 3 trials and evaluate the network with the test subsets.

4. Construct the transfer model as depicted in Figure 12, utilizing the VGG16 pre-trained model without its top layer, and set its weight to not trainable, to leverage proper feature extraction. Use an Adam optimizer with a learning rate of 0.000001. Employ the same metrics as the handcrafted network and train the transfer model using the RGB 224x224 subsets for the same number of epochs as before.

5. For both models, plot the performance metrics for each epoch, generate the confusion matrix for the test subset, and calculate occlusion sensitivity for each class prediction.

Metric	Transfer Model (VGG16)	Simple CNN
Loss	0.780	0.667
Accuracy	0.864	0.860
AUC	0.969	0.959

Table 7: Comparison of evaluation metrics between Transfer Model (VGG16) and Simple CNN.

5.2 Results

Classifying retinal images to identify eye disorders using deep neural networks yields promising results, particularly when comparing handcrafted Convolutional Neural Networks (CNNs) and transfer learning models. The handcrafted CNN, despite starting with a high initial loss, achieves commendable performance metrics, with an accuracy of 86% and an AUC of 95.9%, indicating its capability to classify retinal images effectively (Table 7). However, the initial high loss suggests that the model requires more epochs to stabilize and reach optimal performance, as seen in the progression of the training process (Figure 13).

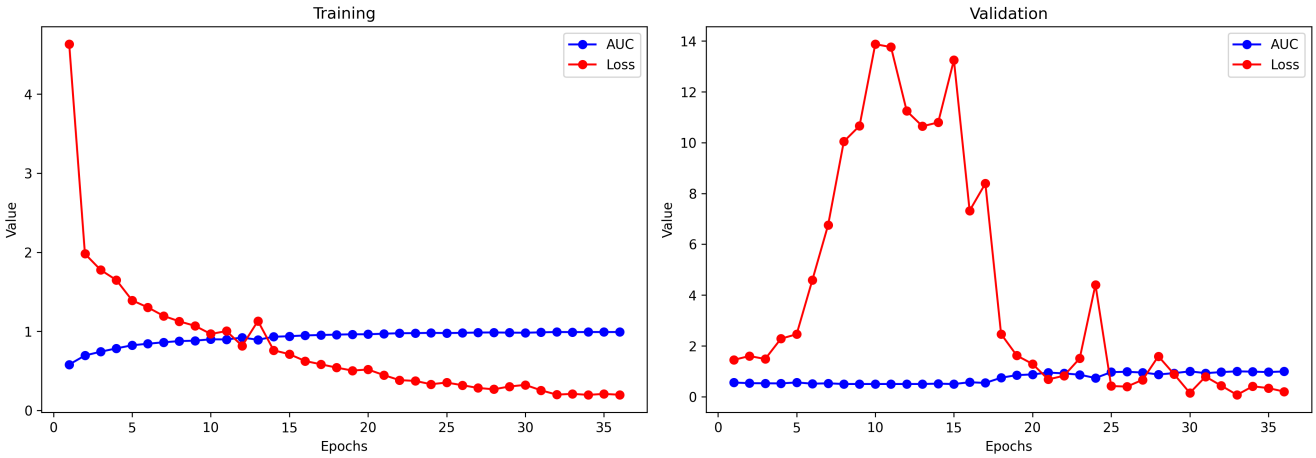

Figure 13: Performance of simple CNN for each epoch.

In contrast, the transfer learning model, built using a pre-trained VGG16 architecture, starts with a significantly lower initial loss (Figure 14), suggesting a more efficient learning process from the outset. The model's ability to maintain a pattern of decreasing loss and increasing AUC over epochs, as seen in the epochs plot (Figure 14), highlights its potential for further improvement and refinement.

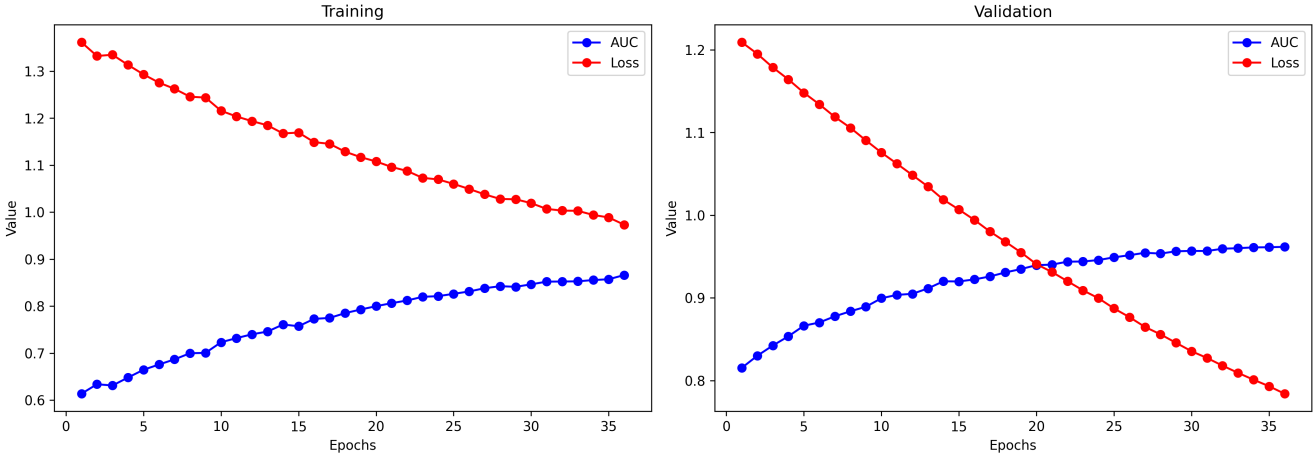

Figure 14: Performance of transfer model for each epoch.

The transfer model architecture uses VGG16 as a feature extractor (Figure 12), flattens the output, and feeds it into dense layers with dropout to prevent overfitting, ending with a softmax decision layer. The hand-crafted CNN starts with an input layer, followed by 1-3 convolutional layers (determined by a hyperparameter) with increasing filters, batch normalization, max

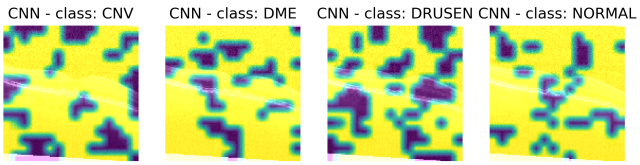

Figure 15: Simple CNN occlusion sensitivity per class.

pooling, and dropout, to avoid overfitting, then flattens the output and passes it through a decision dense layer.

Examining occlusion sensitivity shows that the normal class has fewer focus areas, indicating simpler decision-making, while other classes show more focus areas in both CNN models (Figure 15). The transfer model has smoother attention areas, suggesting refined feature extraction (Figure 16), whereas the handcrafted CNN has pixelated attention areas, indicating less sophisticated analysis. The simpler occlusion sensitivity for the normal class is due to its distinct, easily recognizable features.

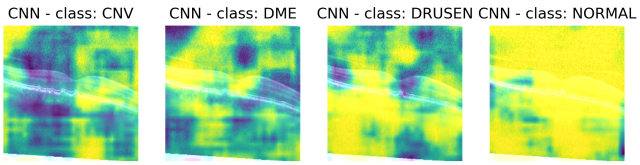

Figure 16: Transfer Model occlusion sensitivity per class.

The confusion matrices for both models indicate high accuracy with minimal errors, demonstrating their effectiveness in classifying retinal images across four classes (Figures 17). However, the transfer model's ability to start with a better loss value and its potential for improvement make it a more suitable choice for this task.

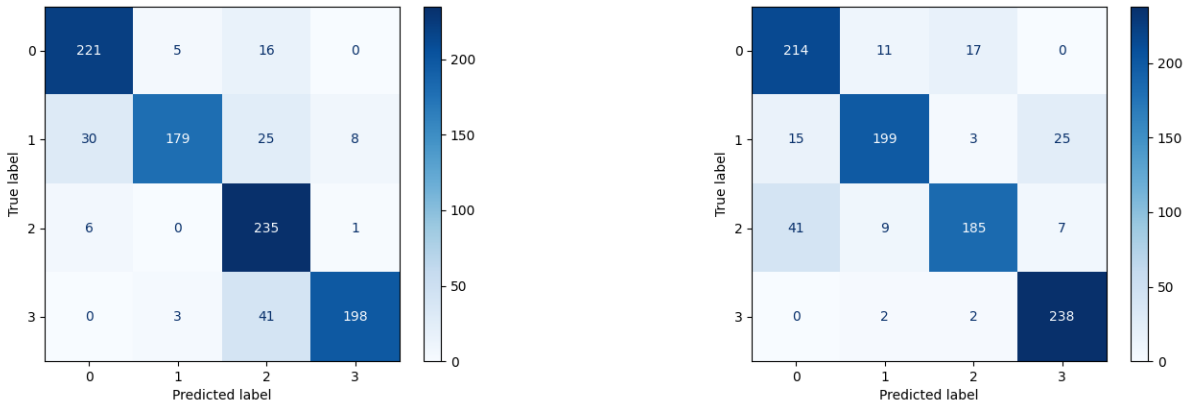

Figure 17: Comparison of confusion matrices for simple CNN (left) and transfer model (right).

In conclusion, while both the handcrafted CNN and the transfer learning model perform well in classifying retinal images to identify eye disorders, the transfer model offers a more robust

starting point and greater potential for enhancement. Its ability to capture patterns and make accurate predictions with less initial loss underscores its suitability as the best model for this task. The deep neural network's capacity to classify retinal images effectively is evident, with the transfer model showing more promise for future developments in this field, making it a great tool for medical purposes of diagnosing diseases using retinal images.