Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Artificial Intelligence

FINAL PROYECT: Training a Machine Learning Model on Medical Images

Names: Ismael Gutierrez, Guillermo Farias, Jocelyn Benitez
ROLL NUMBER: 1995316, 2001715, 2109461

GROUP: 002 *DAY: THURSDAY*

HOUR: N4-N6

Docente:
M.C. Daniel Isaías López Páez

May 29, 2024

Contents

Training a Machine Learning Model on Medical Images

Ismael Gtz, Jocelyn Benitez, Guillermo Farias.

May 29, 2024

Gutiérrez Puente Ismael, Farias Domínguez Guillermo Sebastián, Benítez Ramírez Jocelyn

1 ABSTRACT

Este estudio presenta un enfoque dual para la clasificación y segmentación de imágenes de resonancia magnética (MRI) aplicadas al diagnóstico de la enfermedad de Alzheimer y la segmentación del hipocampo. En la primera parte del estudio, utilizamos un modelo de red neuronal convolucional (CNN) basado en la arquitectura Xception preentrenada para clasificar imágenes MRI en cuatro categorías: sano, leve, moderado y severo. El modelo se entrenó y validó utilizando técnicas de aumento de datos y se evaluó en un conjunto de prueba, mostrando una alta precisión y generalización.

En la segunda parte, abordamos la segmentación del hipocampo mediante un modelo U-Net, diseñado para extraer y segmentar características específicas del hipocampo en imágenes MRI. Las imágenes fueron preprocesadas y normalizadas antes de ser divididas en conjuntos de entrenamiento y validación. El modelo U-Net fue entrenado en 3000 imágenes y su desempeño fue monitoreado a través de la pérdida y la precisión en ambos conjuntos de datos.

Los resultados demostraron la eficacia del modelo de clasificación en la identificación de las diferentes etapas del Alzheimer, así como la capacidad del modelo U-Net para segmentar con precisión el hipocampo. Este enfoque combinado proporciona una herramienta robusta para el diagnóstico y la investigación de la enfermedad de Alzheimer, ofreciendo una base sólida para futuros desarrollos en la medicina personalizada y el análisis de imágenes médicas.

2 Introduction/Background

2.1 Machine Learning

Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior. Artificial intelligence systems are used to perform complex tasks in a way that is like how humans solve problems. The goal of AI is to create computer models that exhibit "intelligent behaviors" like humans, according to Boris Katz, a principal research scientist and head of the InfoLab Group at CSAIL. This means machines that can recognize a visual scene, understand a text written in natural language, or perform an action in the physical world. [14]

Machine learning is one way to use AI. It was defined in the 1950s by AI pioneer Arthur Samuel as:

"The field of study that gives computers the ability to learn without explicitly being programmed."

2.1.1 How does machine learning work?

UC Berkeley breaks out the learning system of a machine learning algorithm into three main parts.

1. **A Decision Process:** In general, machine learning algorithms are used to make a prediction or classification. Based on some input data, which can be labeled or unlabeled, your algorithm will produce an estimate about a pattern in the data.

2. **An Error Function:** An error function evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.

3. **A Model Optimization Process:** If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this iterative "evaluate and optimize" process, updating weights autonomously until a threshold of accuracy has been met.

2.2 Artificial Neural Networks

A neural network is a machine learning program, or model, that makes decisions in a manner like the human brain, by using processes that mimic the way biological neurons work together to identify phenomena, weigh options and arrive at conclusions. Artificial neural networks (ANNs) are computational models that attempt to emulate the architecture of the human brain. Data is processed in discrete "elements" or "neurons" and transferred amongst these elements through "connections" that simulate neural synapses. Artificial neurons are usually organized in the following layers: input, hidden and output. The neurons in the input layer compute the data provided by the investigators; the elements in the single or multiple hidden layers allow for the development of multiple variations of the data; while the output layer calculates the "answer" estimated by the model. Various mathematical functions are applied to the data, resulting in several ANN models, such as backpropagation, probabilistic and others.

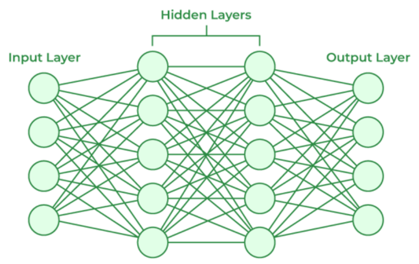


Figure 1. Neural Networks Architecture

Figure 1:

2.2.1 How do Artificial Neural Networks learn?

Artificial neural networks are trained using a training set. For example, suppose you want to teach an ANN to recognize a cat. Then it is shown thousands of different images of cats so that the network can learn to identify a cat. Once the neural network has been trained enough using images of cats, then you need to check if it can identify cat images correctly. This is done by making the ANN classify the images it is provided by deciding whether they are cat images or not. The output obtained by the ANN is corroborated by a human-provided description of whether the image is a cat image or not. If the ANN identifies incorrectly then back-propagation is used to adjust whatever it has learned during training. Backpropagation is done by fine-tuning the weights of the connections in ANN units based on the error rate obtained. This process continues until the artificial neural network can correctly recognize a cat in an image with minimal possible error rates. [1]

2.3 Deep Neural Networks

Deep learning is a subset of machine learning that uses multi-layered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain. By strict definition, a deep neural network, or DNN, is a neural network with three or more layers. In practice, most DNNs have many more layers. DNNs are trained on large amounts of data to identify and classify phenomena, recognize patterns and relationships, evaluate possibilities, and make predictions and decisions. While a single-layer neural network can make useful, approximate predictions and decisions, the additional layers in a deep neural network help refine and optimize those outcomes for greater accuracy.

2.3.1 How deep learning works

Deep learning works by mimicking the human brain using artificial neural networks. These networks process data through interconnected layers of nodes, where each layer refines predictions or categorizations.

- **Forward Propagation:** Data moves through the network from the input layer to the output layer, making initial predictions.
- **Backpropagation:** This process adjusts the network by calculating prediction errors and modifying weights and biases to improve accuracy. This cycle of forward and backward propagation trains the model.

Deep learning networks can vary in complexity and structure to address specific tasks:

- **Convolutional Neural Networks (CNNs):** Used in computer vision and image classification, CNNs detect patterns and features in images for tasks like object detection and recognition.
- **Recurrent Neural Networks (RNNs):** Used in natural language processing and speech recognition, RNNs handle sequential data to process and predict based on time-series information.

2.4 Image Classification

Image classification is the task of categorizing and assigning class labels to groups of pixels or vectors within an image dependent on rules. The categorization law can be applied through one or multiple spectral or textural characterizations. [3]

2.4.1 Xception architecture

Xception is a deep convolutional neural network architecture that involves Depthwise Separable Convolutions. It was developed by Google researchers. Google presented an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution).

2.4.2 What does it look like?

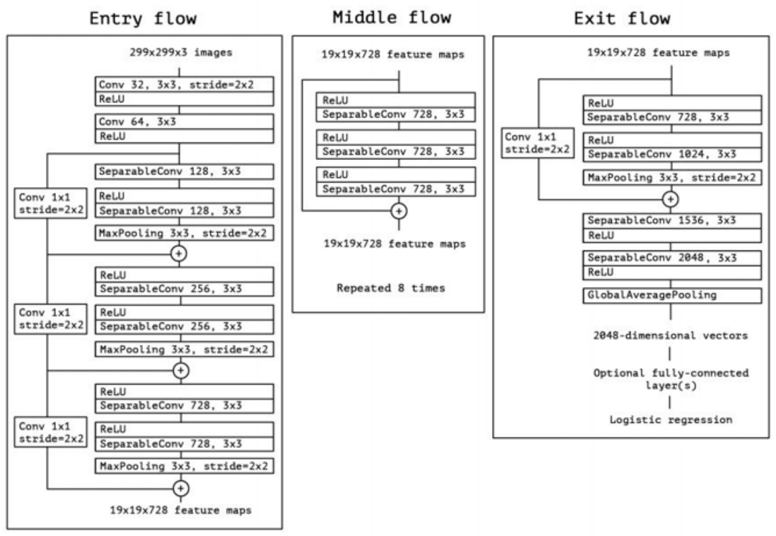


Figure 2. Overall Architecture of Xception (Entry Flow > Middle Flow > Exit Flow)

Figure 2:

The data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization. [10]

2.4.3 How does XCeption work?

XCeption is an efficient architecture that relies on two main points:

1. **Depthwise Separable Convolutions:** This is the core idea behind Xception. It splits the convolution operation into two separate layers:
    - **Depthwise Convolution:** Applies a single convolutional filter per input channel.
    - **Pointwise Convolution:** Uses a 1x1 convolution to combine the outputs of the depthwise convolution.
2. **Shortcuts between Convolution blocks as in ResNet**
    - **Residual Connections:** These shortcuts allow gradients to flow directly through the network, addressing vanishing gradient issues and improving training for deep networks.
    - **Function**

  - **Gradient Flow:** Ensures learning signals reach early layers effectively.
  - **Information Preservation:** Maintains feature integrity across layers.
- **Structure**
  - **Identity Mapping:** Directly adds input of a block to its output.
  - **Projection Mapping:** Adjusts dimensions when necessary, using convolutions.
- **Integration**
  - **Used alongside depthwise separable convolutions.**
  - **Enhances efficiency and performance by combining residual learning with computational benefits of separable convolutions.**

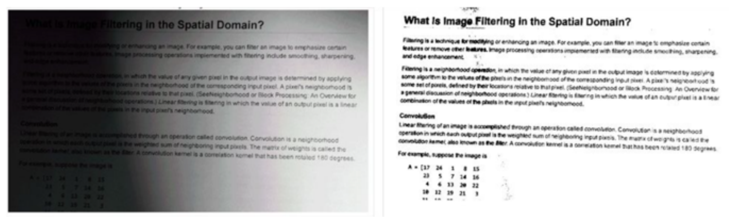
2.5 Image Segmentation

Image segmentation is a computer vision technique that partitions a digital image into discrete groups of pixels—image segments—to inform object detection and related tasks. By parsing an image's complex visual data into specifically shaped segments, image segmentation enables faster, more advanced image processing. [4]

2.5.1 How Image Segmentation Works

Image segmentation involves converting an image into a collection of regions of pixels that are represented by a mask or a labeled image. By dividing an image into segments, you can process only the important segments of the image instead of processing the entire image.

A common technique is to look for abrupt discontinuities in pixel values, which typically indicate edges that define a region. [5]



What Is Image Filtering in the Spatial Domain?

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel. (SeeNeighborhood or Block Processing: An Overview for a general discussion of neighborhood operations.) Linear filtering is filtering in which the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood.

Convolution

Linear filtering of an image is accomplished through an operation called convolution. Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. The matrix of weights is called the convolution kernel, also known as the filter. A convolution kernel is a correlation kernel that has been rotated 180 degrees.

For example, suppose the image is

```
A = [17  24   1   8  15
     23   5   7  14  16
      4   6  13  20  22
     10  12  19  21   3
```

Figure 3. Using thresholding to convert to a binary image to improve the legibility of the text in an image.

Figure 3:

Another common approach is to detect similarities in the regions of an image. Some techniques that follow this approach are region growing, clustering, and thresholding.

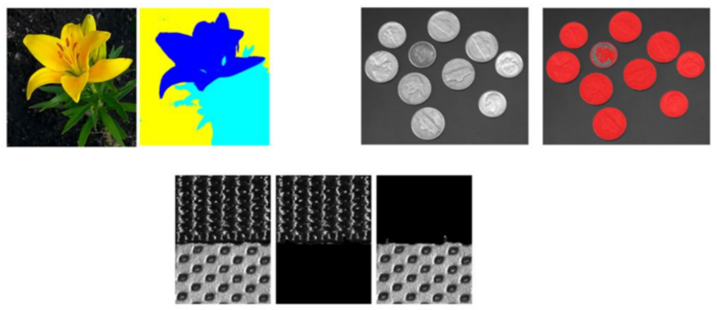


Figure 4. Segmenting regions based on color values, shapes, or texture.

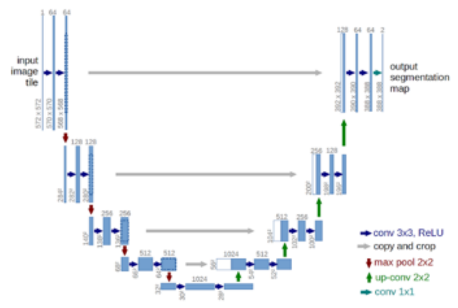
Figure 4:

2.5.2 UNET architecture

UNET is an architecture developed by Olaf Ronneberger et al. for Biomedical Image Segmentation in 2015 at the University of Freiburg, Germany. It is one of the most popularly used approaches in any semantic segmentation task today. It is a fully convolutional neural network that is designed to learn from fewer training samples. It is an improvement over the existing FCN — "Fully convolutional networks for semantic segmentation" developed by Jonathan Long et al. in (2014). [6]

2.5.3 What does it look like?

UNET is a U-shaped encoder-decoder network architecture, which consists of four encoder blocks and four decoder blocks that are connected via a bridge. The encoder network (contracting path) halves the spatial dimensions and doubles the number of filters (feature channels) at each encoder block. Likewise, the decoder network doubles the spatial dimensions and halves the number of feature channels. [7]



channels.

Figure 5. UNET architecture

Figure 5:

2.5.4 How does U-Net work?

1. **Encoder Network:** The encoder network acts as the feature extractor and learns an abstract representation of the input image through a sequence of the encoder blocks. Each encoder block consists of two 3x3 convolutions, where each convolution is followed by a ReLU (Rectified Linear Unit) activation function. The ReLU activation function introduces non-linearity into the network, which helps in the better generalization of the training data. The output of the ReLU acts as a skip connection for the corresponding decoder block. Next, follows a 2x2 max-pooling, where the spatial dimensions (height and width) of the feature maps are reduced by half. This reduces the computational cost by decreasing the number of trainable parameters.

2. **Skip Connections:** These skip connections provide additional information that helps the decoder to generate better semantic features. They also act as a shortcut connection that helps the indirect flow of gradients to the earlier layers without any degradation. In simple terms, we can say that skip connection helps in better flow of gradient while backpropagation, which in turn helps the network to learn better representation.

3. **Bridge:** The bridge connects the encoder and the decoder network and completes the flow of information. It consists of two 3x3 convolutions, where each convolution is followed by a ReLU activation function.

4. **Decoder Network:** The decoder network is used to take the abstract representation and generate a semantic segmentation mask. The decoder block starts with a 2x2 transpose convolution. Next, it is concatenated with the corresponding skip connection feature map from the encoder block. These skip connections provide features from earlier layers that are sometimes lost due to the depth of the network. After that, two 3x3 convolutions are used, where each convolution is followed by a ReLU activation function. The output of the last decoder passes through a 1x1 convolution with sigmoid activation. The sigmoid activation function gives the segmentation mask representing the pixel-wise classification.

[8]

3 Methodology

3.1 Datasets Methodology

3.1.1 Dataset "Augmented Alzheimer MRI Dataset"

The data consists of MRI images. The data has four classes of images both in training as well as a testing set:

1. Mild Demented

2. Moderate Demented
3. Non Demented
4. Very Mild Demented

The data contains two folders. One of them is augmented ones and the other one is originals.

1. Originals could be used for validation or test dataset.
2. Data is augmented from an existing dataset. Original images can be seen in Data Explorer.

[9]

3.1.2 Dataset "MRI Hippocampus Segmentation"

The hippocampus is a structure within the brain that plays important roles in the consolidation of information from short-term memory to long-term memory, and in spatial memory that enables navigation. Magnetic resonance imaging is often the optimal modality for brain medical imaging studies, being T1 ideal for representing structure. The hippocampus has become the focus of research in several neurodegenerative disorders. Automatic segmentation of this structure from magnetic resonance (MR) imaging scans of the brain facilitates this work, especially in resource poor environments. [12] **About This Dataset**

This dataset contains T1-weighted MR images of 50 subjects, 40 of whom are patients with temporal lobe epilepsy and 10 are nonepileptic subjects. Hippocampus labels are provided for 25 subjects for training. For more information about the dataset, refer to the original article.

3.2 CNNs Methodology

This project involves creating two Convolutional Neural Network (CNN) models for different tasks related to MRI images.

1. For dataset "Augmented MRI Dataset", we develop a CNN model using Xception architecture, with some improvements, to classify the MRI images into one of the four classes.
2. For dataset "MRI Hippocampus Segmentation Dataset", we develop a CNN model using the UNet architecture to segment hippocampus regions in the MRI scans, which is crucial for studying neurodegenerative disorders such as Alzheimer's disease.

4 Results

https://www.kaggle.com/code/guillefarias/pia-ia Para visualizarlo se necesita una cuenta en KAGGLE

4.1 First Model: Alzheimer's Classification

4.1.1 Preprocessing Applied to the Images

The dataset is split into training, validation, and test sets using the `train_test_split` function. The first split divides the data into training (70%) and test (30%) sets. The training set is further split into training (80%) and validation (20%) sets, maintaining the same random state for reproducibility.

An `ImageDataGenerator` is initialized with a preprocessing function specific to the MobileNetV2 model. This function scales the pixel values to the range required by MobileNetV2. Data generators for training, validation, and test sets are created using the `flow_from_dataframe` method, which generates batches of tensor image data with real-time data augmentation.

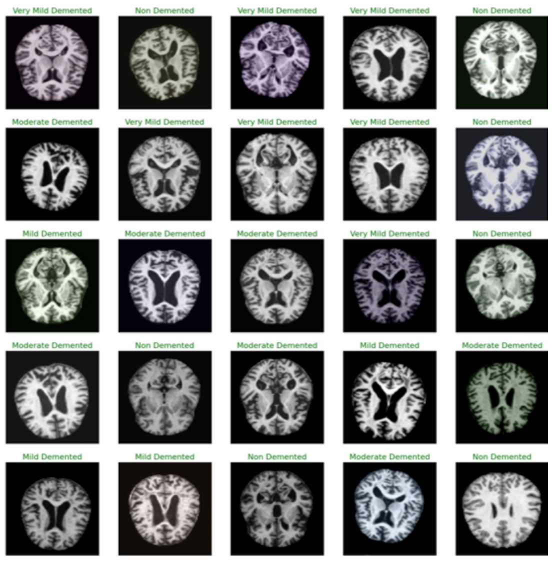


Figure 6. Examples of the dataset (Image and their respective label)

Figure 6:

4.1.2 CNN Architecture

We defined our input images to be 244x244 pixels with three color channels (RGB). We used the pre-trained Xception model from Keras, excluding its top layer, and applied global max pooling to its output. [11] Our custom model, built using Keras Sequential API, consists of:

- **Xception Base Model:** Pre-trained on ImageNet, used for feature extraction.
- **Flatten Layer:** Converts the multi-dimensional output of the base model into a one-dimensional vector.

- **Dropout Layers:** Included to prevent overfitting, with rates of 0.3 and 0.25.
- **Dense Layer:** Contains 128 neurons with ReLU activation to learn complex patterns.
- **Output Layer:** Consists of 4 neurons with softmax activation for multi-class classification.

[13]

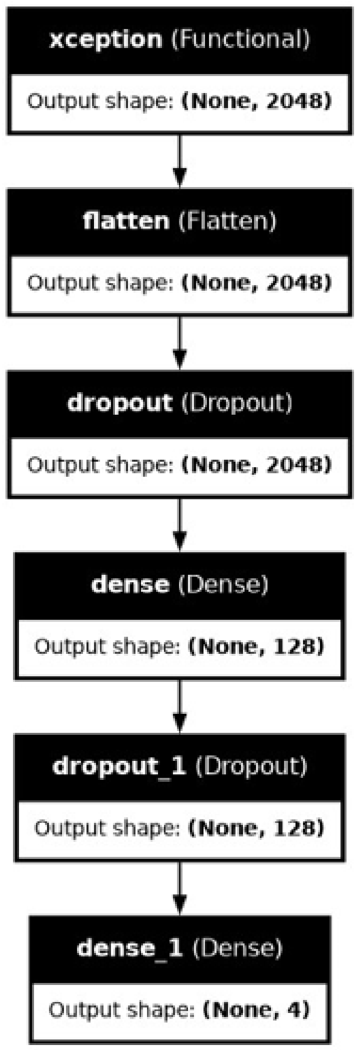

xception (Functional)
Output shape: (None, 2048)
flatten (Flatten)
Output shape: (None, 2048)
dropout (Dropout)
Output shape: (None, 2048)
dense (Dense)
Output shape: (None, 128)
dropout_1 (Dropout)
Output shape: (None, 128)
dense_1 (Dense)
Output shape: (None, 4)

Figure 7. CNN Architecture

Figure 7:

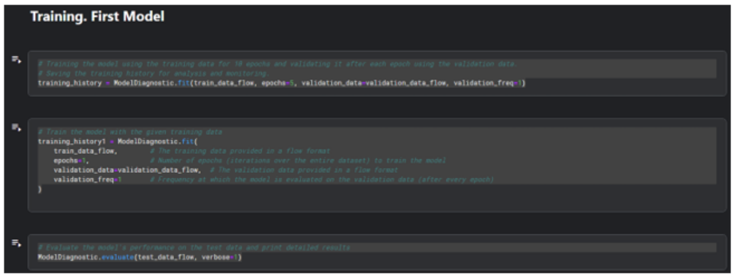
The model was compiled with the Adamax optimizer (learning rate of 0.001), categorical crossentropy loss function, and accuracy as the evaluation metric. This setup leverages the powerful feature extraction capabilities of the Xception model and adapts it to our specific classification task.

4.1.3 CNN Training

The model, referred to as *ModelDiagnostic*, was trained using the training data (`train_data_flow`) for a total of 5 epochs. After each epoch, the model's performance was validated using the validation data (`validation_data_flow`). This process allows for monitoring the model's learning progression and ensuring that it generalizes well to unseen data. The training history was saved for subsequent analysis and monitoring, providing insights into the model's behavior over the training period.

Subsequently, an additional training iteration was performed using the same training and validation data flows but for only 1 epoch. This single epoch training aimed to further fine-tune the model parameters based on the training data while evaluating the model on the validation data at the end of the epoch. The training history from this iteration was also recorded.

After training, the model's performance was assessed using the test data (`test_data_flow`). The evaluation was performed to measure how well the trained model generalizes to new, unseen data. Detailed results of this evaluation were printed, providing comprehensive metrics that summarize the model's performance.


Training. First Model

```
# Training the model using the training data for 10 epochs and validating it after each epoch using the validation data.
# Saving the training history for analysis and monitoring.
training_history = ModelDiagnostic.fit(train_data_flow, epochs=5, validation_data=validation_data_flow, validation_freq=1)
```

```
# Train the model with the given training data
training_history1 = ModelDiagnostic.fit(
    train_data_flow,         # The training data provided in a flow format
    epochs=1,                # Number of epochs (iterations over the entire dataset) to train the model
    validation_data=validation_data_flow,  # The validation data provided in a flow format
    validation_freq=1        # Frequency at which the model is evaluated on the validation data (after every epoch)
)
```

```
# Evaluate the model's performance on the test data and print detailed results
ModelDiagnostic.evaluate(test_data_flow, verbose=1)
```

Figure 8. CodeSnap

Figure 8:

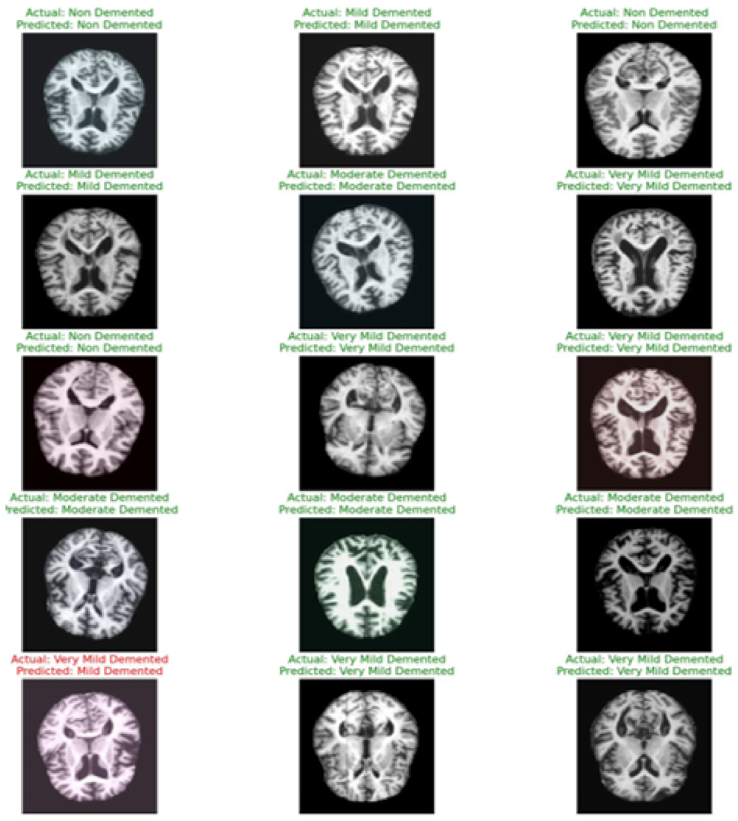

Actual: Non Demented
Predicted: Non Demented
Actual: Mild Demented
Predicted: Mild Demented
Actual: Non Demented
Predicted: Non Demented
Actual: Mild Demented
Predicted: Mild Demented
Actual: Moderate Demented
Predicted: Moderate Demented
Actual: Very Mild Demented
Predicted: Very Mild Demented
Actual: Non Demented
Predicted: Non Demented
Actual: Very Mild Demented
Predicted: Very Mild Demented
Actual: Very Mild Demented
Predicted: Very Mild Demented
Actual: Moderate Demented
Predicted: Moderate Demented
Actual: Moderate Demented
Predicted: Moderate Demented
Actual: Moderate Demented
Predicted: Moderate Demented
Actual: Very Mild Demented
Predicted: Mild Demented
Actual: Very Mild Demented
Predicted: Very Mild Demented
Actual: Very Mild Demented
Predicted: Very Mild Demented

Figure 9. Predictions of the CNN

Figure 9:

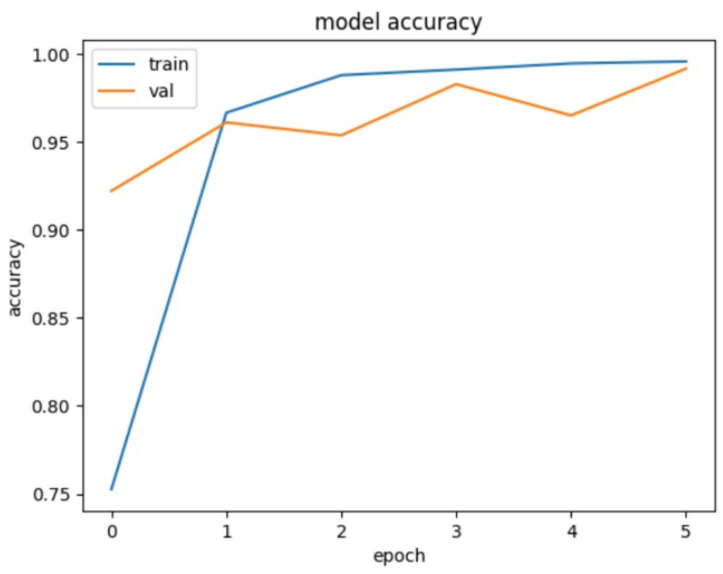


Figure 10. Training and Validation accuracy of the model.

Figure 10:

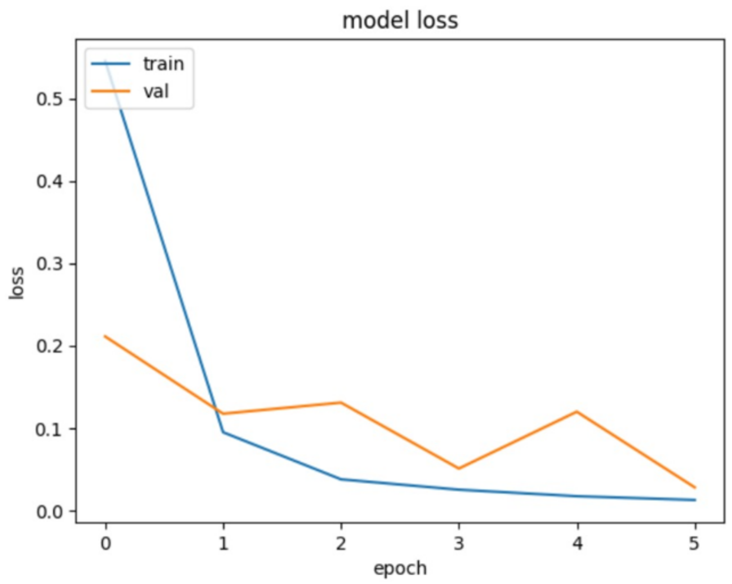


Figure 11. Training and Validation loss of the model.

Figure 11:

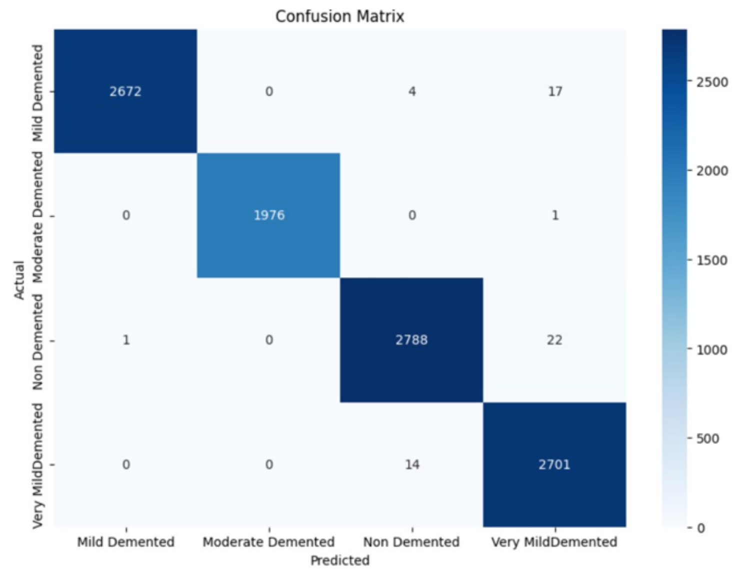


Figure 12. Confusion Matrix of the model.

Figure 12:

4.2 Second Model: Hippocampus Segmentation

4.2.1 Preprocessing Applied to the Images

First, we define the paths to the directories containing the label images and the original MRI images. These paths, labeled as `Label_Path` and `Image_Path`, point to the respective directories holding the data. Using these paths, we retrieve lists of all file paths for the label and image files using the `glob` method, which searches for all .jpg files recursively within the specified directories. These lists are then converted into pandas Series objects (`Label_Series` and `Image_Series`), with paths converted to strings for easier manipulation.

Next, we categorize the images into left and right images. We initialize empty lists to store the paths of left (`L_IMG`), right (`R_IMG`), and total (`Total_IMG`) images. By iterating over the paths, we split each path using predefined delimiters to extract relevant parts, such as the main path and target path for both labels and images. Based on these extracted parts, we categorize the images into left or right images and add them to their respective lists. If an image path does not match the expected pattern, an error message is triggered.

After categorizing the images, we sort the lists alphabetically to ensure a consistent order. This step is crucial for maintaining the correct pairing between the original images and their corresponding labels. We then initialize empty lists (`X_Image` and `X_Hippocampus`) to store the preprocessed image data.

For each set of corresponding original, left, and right images, we read the images using OpenCV and convert them from BGR to RGB format. The left and right images are combined using `cv2.addWeighted` to create a composite hippocampus image. This combined image represents the hippocampus region more effectively by integrating information from both the left and right views. We then normalize the pixel values for both the original and hippocampus images by dividing by 255, ensuring that the data is scaled appropriately for input into the CNN model. The normalized images are appended to their respective lists (`X_Image` for original images and `X_Hippocampus` for hippocampus images).

Additionally, for the first 1500 images, we repeat a similar procedure to generate a separate subset of processed images, stored in lists (`Image_New` and `Hippocampus_Mask`). This step provides a focused dataset that can be used for further validation or testing purposes. [2] Finally, the lists of images (`X_Image` and `X_Hippocampus`) are converted into NumPy arrays of type `float32`. Assuming that the hippocampus images are in RGB format, they are converted to grayscale by averaging the RGB channels. This transformation simplifies the data, focusing on the intensity values which are critical for segmentation tasks. The additional subsets (`Image_New` and `Hippocampus_Mask`) are also converted to NumPy arrays, ensuring they are ready for use in training the UNet model.

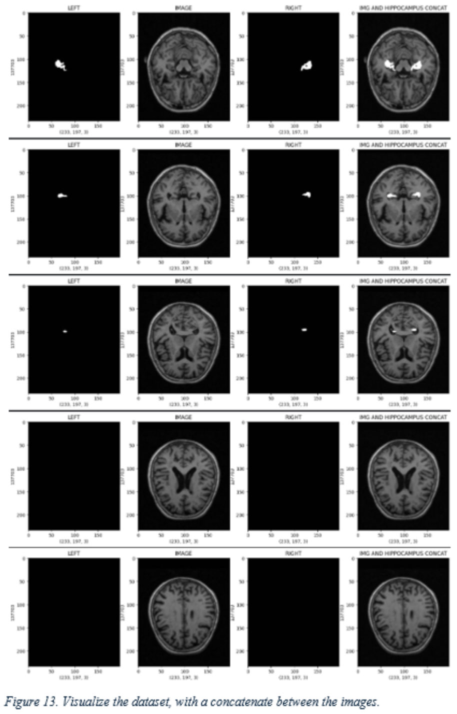


Figure 13. Visualize the dataset, with a concatenate between the images.

Figure 13:

4.2.2 CNN Architecture

Our model is implemented using the Sequential API from TensorFlow's Keras library. The architecture comprises two main parts: the encoder and the decoder.

Encoder The encoder part of the model extracts features from the input images and progressively reduces their spatial dimensions while increasing the depth of the feature maps. This is achieved through a series of convolutional layers, batch normalization, and activation functions:

- **First Convolutional Block:**
  - Conv2D with 32 filters and a kernel size of (2,2), using 'he_normal' initialization.
  - BatchNormalization to normalize the output and stabilize the training process.
  - ReLU activation to introduce non-linearity.
- **Second Convolutional Block:**
  - Conv2D with 64 filters and a kernel size of (2,2), using 'he_normal' initialization.
  - BatchNormalization and ReLU activation.
- **Third Convolutional Block:**
  - Conv2D with 128 filters and a kernel size of (2,2), using 'he_normal' initialization.
  - BatchNormalization and ReLU activation.

Decoder The decoder part of the model upsamples the feature maps back to the original image dimensions, using transposed convolutions to recover the spatial resolution:

- **First Transposed Convolutional Block:**
  - Conv2DTranspose with 64 filters and a kernel size of (2,2).
  - ReLU activation.
- **Second Transposed Convolutional Block:**
  - Conv2DTranspose with 32 filters and a kernel size of (2,2).
  - ReLU activation.
- **Final Transposed Convolutional Block:**
  - Conv2DTranspose with 1 filter and a kernel size of (2,2) to output the segmented image.
  - ReLU activation.

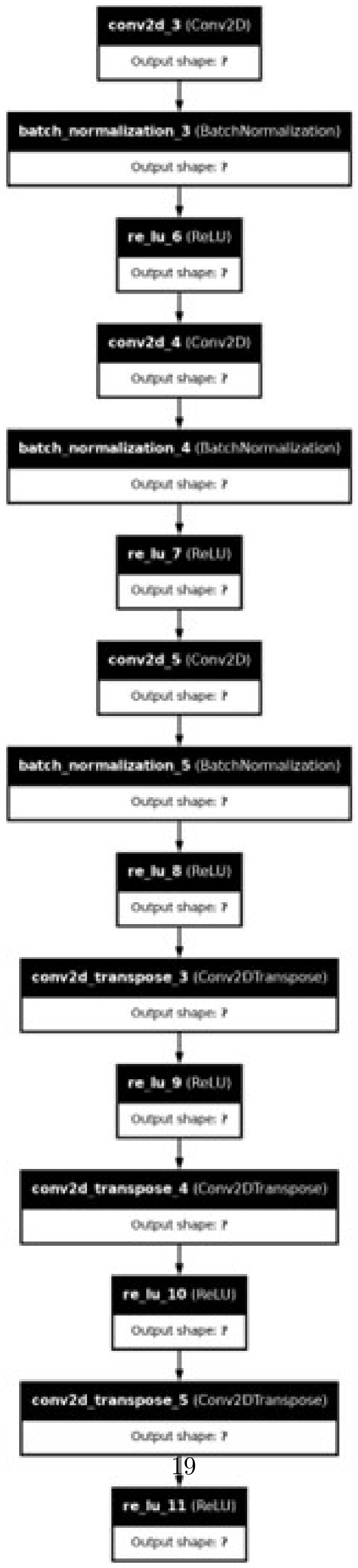

conv2d_3 (Conv2D)
Output shape: ?
batch_normalization_3 (BatchNormalization)
Output shape: ?
re_lu_6 (ReLU)
Output shape: ?
conv2d_4 (Conv2D)
Output shape: ?
batch_normalization_4 (BatchNormalization)
Output shape: ?
re_lu_7 (ReLU)
Output shape: ?
conv2d_5 (Conv2D)
Output shape: ?
batch_normalization_5 (BatchNormalization)
Output shape: ?
re_lu_8 (ReLU)
Output shape: ?
conv2d_transpose_3 (Conv2DTranspose)
Output shape: ?
re_lu_9 (ReLU)
Output shape: ?
conv2d_transpose_4 (Conv2DTranspose)
Output shape: ?
re_lu_10 (ReLU)
Output shape: ?
conv2d_transpose_5 (Conv2DTranspose)
Output shape: ?
re_lu_11 (ReLU)
Output shape: ?

Figure 14:

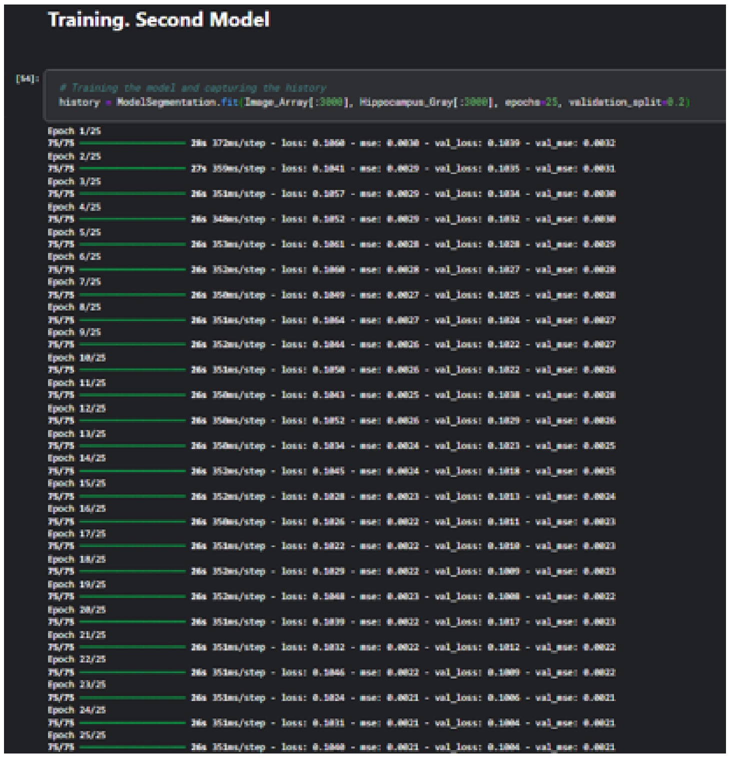
4.2.3 CNN Training

We utilize the `fit` method to train the model, with the following parameters:

- **Input Data:** The first 3,000 images and their corresponding segmentation masks from the dataset.
- **Epochs:** The training is conducted over 25 epochs. An epoch refers to one complete pass through the entire training dataset.
- **Validation Split:** 20% of the training data is set aside for validation. This means that out of 3,000 images, 2,400 images are used for training, and 600 images are used for validating the model's performance.

The `fit` method returns a history object that records the training metrics, including:

- **Loss:** Measures the error of the model's predictions during training and validation.
- **Accuracy:** Evaluates the proportion of correct predictions out of total predictions.
- **Validation Loss and Accuracy:** These metrics help monitor the model's performance on the validation set to ensure it generalizes well to unseen data.

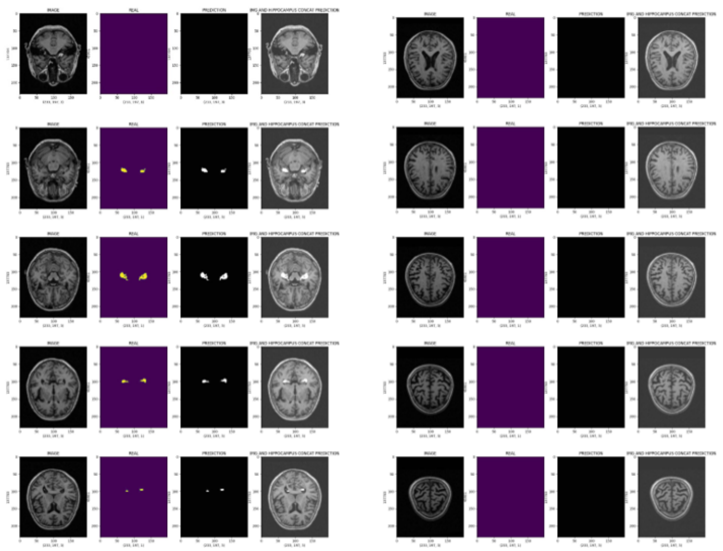


Training. Second Model

```
[54]: # Training the model and capturing the history
      history = ModelSegmentation.fit(Image_Array[:3000], Hippocampus_Gray[:3000], epochs=25, validation_split=0.2)
Epoch 1/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 28s 372ms/step - loss: 0.1060 - mse: 0.0030 - val_loss: 0.1039 - val_mse: 0.0032
Epoch 2/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 27s 359ms/step - loss: 0.1041 - mse: 0.0029 - val_loss: 0.1035 - val_mse: 0.0031
Epoch 3/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1057 - mse: 0.0029 - val_loss: 0.1034 - val_mse: 0.0030
Epoch 4/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 348ms/step - loss: 0.1052 - mse: 0.0029 - val_loss: 0.1032 - val_mse: 0.0030
Epoch 5/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 353ms/step - loss: 0.1061 - mse: 0.0028 - val_loss: 0.1028 - val_mse: 0.0029
Epoch 6/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 352ms/step - loss: 0.1060 - mse: 0.0028 - val_loss: 0.1027 - val_mse: 0.0028
Epoch 7/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 350ms/step - loss: 0.1049 - mse: 0.0027 - val_loss: 0.1025 - val_mse: 0.0028
Epoch 8/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1064 - mse: 0.0027 - val_loss: 0.1024 - val_mse: 0.0027
Epoch 9/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 352ms/step - loss: 0.1044 - mse: 0.0026 - val_loss: 0.1022 - val_mse: 0.0027
Epoch 10/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1050 - mse: 0.0026 - val_loss: 0.1022 - val_mse: 0.0026
Epoch 11/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 350ms/step - loss: 0.1043 - mse: 0.0025 - val_loss: 0.1038 - val_mse: 0.0028
Epoch 12/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 350ms/step - loss: 0.1052 - mse: 0.0026 - val_loss: 0.1029 - val_mse: 0.0026
Epoch 13/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 350ms/step - loss: 0.1034 - mse: 0.0024 - val_loss: 0.1023 - val_mse: 0.0025
Epoch 14/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 352ms/step - loss: 0.1045 - mse: 0.0024 - val_loss: 0.1018 - val_mse: 0.0025
Epoch 15/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 352ms/step - loss: 0.1028 - mse: 0.0023 - val_loss: 0.1013 - val_mse: 0.0024
Epoch 16/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 350ms/step - loss: 0.1026 - mse: 0.0022 - val_loss: 0.1011 - val_mse: 0.0023
Epoch 17/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1022 - mse: 0.0022 - val_loss: 0.1010 - val_mse: 0.0023
Epoch 18/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 352ms/step - loss: 0.1029 - mse: 0.0022 - val_loss: 0.1009 - val_mse: 0.0023
Epoch 19/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 352ms/step - loss: 0.1048 - mse: 0.0023 - val_loss: 0.1008 - val_mse: 0.0022
Epoch 20/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1039 - mse: 0.0022 - val_loss: 0.1017 - val_mse: 0.0023
Epoch 21/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1032 - mse: 0.0022 - val_loss: 0.1012 - val_mse: 0.0022
Epoch 22/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1046 - mse: 0.0022 - val_loss: 0.1009 - val_mse: 0.0022
Epoch 23/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1024 - mse: 0.0021 - val_loss: 0.1006 - val_mse: 0.0021
Epoch 24/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1031 - mse: 0.0021 - val_loss: 0.1004 - val_mse: 0.0021
Epoch 25/25
75/75 ━━━━━━━━━━━━━━━━━━━━ 26s 351ms/step - loss: 0.1040 - mse: 0.0021 - val_loss: 0.1004 - val_mse: 0.0021
```

Figure 15. CodeSnap

Figure 15:



gure 16. Segmentation Prediction of the model.

Figure 16:

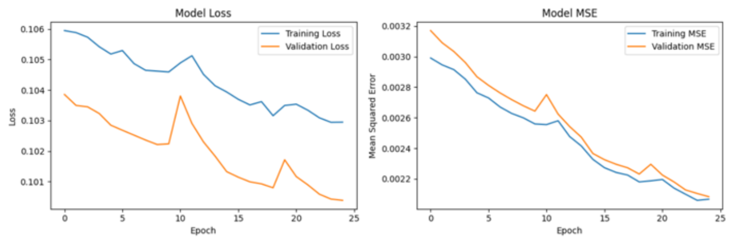


Figure 17. Loss and MSE of the model in training and validation.

Figure 17:

5 Discussion and Conclusions

In this study, we developed and evaluated two CNN models for medical image analysis: one for Alzheimer's classification and another for hippocampus segmentation.

The first model, leveraging the Xception architecture, demonstrated robust performance in classifying Alzheimer's disease stages with good accuracy and generalizability. The use of data augmentation and dropout layers effectively mitigated overfitting, ensuring the model's robustness. The training and validation accuracies and losses, along with the confusion matrix, provided comprehensive insights into the model's performance, highlighting areas for potential improvement.

The second model employed a UNet architecture for the segmentation of the hippocampus in MRI images. This model effectively segmented hippocampus regions, facilitating detailed study of neurodegenerative disorders. The preprocessing steps, including image normalization and combining left and right hippocampus images, were crucial in preparing the data for effective model training. The training and validation metrics confirmed the model's ability to generalize well to unseen data.

Future work could focus on enhancing these models by incorporating more diverse datasets, exploring different architectures, and applying advanced techniques such as transfer learning and fine-tuning to improve performance further. Overall, the results indicate promising potential for using CNNs in medical image analysis, contributing to more accurate and efficient diagnosis and study of neurodegenerative diseases.

6 REFERENCIAS BIBLIOGRÁFICAS

References

[1] G. Boesch. *A complete guide to image classification in 2024*. https://viso.ai/computer-vision/image-classification/. Accessed: 2024-05-28. May 2024.

[2] Maël Fabien. *XCeption model and depthwise separable convolutions*. https://maelfabien.github.io/deeplearning/xception/#what-does-it-look-like. Accessed: 2024-05-28. Mar. 2019.

[3] GeeksforGeeks. *Artificial Neural Networks and its Applications*. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/. Accessed: 2024-05-28. June 2023.

[4] IBM. *What is Deep Learning?* https://www.ibm.com/topics/deep-learning. Accessed: 2024-05-28.

[5] IBM. *What is image segmentation?* https://www.ibm.com/topics/image-segmentation. Accessed: 2024-05-28.

[6] IBM. *What is machine learning (ML)?* https://www.ibm.com/topics/machine-learning. Accessed: 2024-05-28.

[7] Alberto M. Marchevsky. "The use of artificial neural networks for the diagnosis and estimation of prognosis in cancer patients". In: *Elsevier eBooks*. Elsevier, 2007, pp. 243–259. DOI: 10.1016/b978-044452855-1/50011-8.

[8] MATLAB Simulink. *Image segmentation*. https://la.mathworks.com/discovery/image-segmentation.html. Accessed: 2024-05-28.

[9] MIT News. *Explained: Neural networks*. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. Accessed: 2024-05-28. Apr. 2017.

[10] Andrew MVD. *Hippocampus segmentation in MRI images*. https://www.kaggle.com/datasets/andrewmvd/hippocampus-segmentation-in-mri-images. Accessed: 2024-05-28. Apr. 2020.

[11] S. Pawan and J. Rajan. "Capsule networks for image classification: A review". In: *Neurocomputing* 509 (2022), pp. 102–120. DOI: 10.1016/j.neucom.2022.08.073.

[12] MIT Sloan. *Machine learning, explained*. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. Accessed: 2024-05-28. Apr. 2021.

[13] N. Tomar. *What is UNET? - Analytics Vidhya - Medium*. https://medium.com/analytics-vidhya/what-is-unet-157314c87634. Accessed: 2024-05-28. Feb. 2024.

[14] Uraninjo. *Augmented Alzheimer MRI Dataset.* `https://www.kaggle.com/datasets/uraninjo/augmented-alzheimer-mri-dataset`. Accessed: 2024-05-28. Sept. 2022.