

**October University for Modern Sciences & Arts**

**CS363**

**Faculty:** Computer Science.

**Semester:** Spring 2023.

**Under Supervision of:** Professor Ammar Mohammed.

|  |  |  |
| --- | --- | --- |
| **Team Members** | **IDs** | **E-Mails** |
| Habiba Wael | 211921 | Habiba.wael1@msa.edu.eg |
| Mariam Ahmed | 213545 | Mariam.ahmed38@msa.edu.eg |
| Nathalie Amgad | 210357 | Nathalie.amgad@msa.edu.eg |
| Mahmoud Hossam | 222125 | Mahmoud.hossam2@msa.edu.eg |

**Introduction:**

This report discusses the results of 7 different classifiers with different cases to test a model’s potential. The model created is meant to classify the various images of brain scans into one of the 4 classes of the Alzheimer Disease.

**Data Description:**

The data is divided in 2 folders one for testing and the other for training, each folder is divided into 4 subfolders one for each class which are:

1. Moderate Demented class.
2. Mild Demented class.
3. Very Mild Demented.
4. Non-Demented.

The images are grayscale images with 208176 size. These images represent a

brain scan which differs from class to class indicating the Alzheimer stage of the patient.

**Preprocessing Steps:**

* The images are resized to the shape of 4848.
* Some other steps were done in different cases to compute variant results

the steps include horizontal flipping of the picture, changing the zoom range an image and changing the shear range to create different angles of the same image.

**Methodology:**

* **CNN\_SVM:**
  + In Case 1, a convolutional neural network (CNN) architecture was designed and implemented for the given data. The architecture consisted of two convolutional layers, each having 32 filters of size (3,3). The input data was padded with zeros to maintain the spatial dimensions after convolutional operations. In between the convolutional layers, two max pooling layers were inserted, each employing a max pooling operation of size (2,2) and strides=2 to perform down-sampling of the feature maps. Flattening operation was performed, then adding our Dense layer containing 128 neurons. Finally, a regulizer L2 was added to obtain a penalty on our thetas, when it reaches a high value to reduce overfitting.
  + In Case 2, Similar parameters were found and adding a Dropout Layer of rate 30%.
  + In Case 3, Similar parameters but, 2 more Dense layers were added each having 64 neurons and 32 neurons
  + In Case 4, learning rate was changed from 0.001 to 0.01 to explore the changes in performance.
  + In case 5, a dynamic method was applied , where it takes a varying number of filters varying between min\_number= 32 and max\_number=256. Then, iteratively adds more convolutional layers to the model with varying numbers of filters and ReLU activation functions.
* **CNN\_SVM:**
  + **Input Layer:** The input layer of a neural network can hold different sizes depending on the specific application and data being used. Therefore different sizing were applied, where the input images are resized to a specific shape, such as (48, 48, 1) or (150, 150), to match the requirements of the neural network architecture.
  + **Convolutions:** Different layers of convulsions with varying number of filters, with size (3, 3) and padding= ‘same’ to have the operation applied on our feature map to be equivalent to the dimensions of input.
  + **Max pooling:** Used to select the maximum activation value within a window of a specified size, in our case (2, 2). This operation effectively down samples the feature map, reducing its dimension while retaining the most important information. This technique is useful for reducing the spatial dimensions of the data, making it easier for subsequent layers to learn patterns in the data.
  + **Dropout Layer:** The dropout layer randomly drops out a specified portion of neurons during training, which reduces the contribution of individual neurons and helps prevent overfitting to the training data. This technique can be effective at preventing overfitting, especially in deep neural networks, by forcing the network to generalize to different combinations of neurons.
  + **Regularizers:** Improves the generalization of data, where a penalty is added to the objective function that is proportional to the square of the magnitude of the model parameters (thetas). This penalty term encourages the model to learn simpler relationships between the features and the target variable by penalizing large values of thetas. This helps to prevent overfitting and improve the model's ability to generalize to new data.
  + **SVM function:** The loss function that is used to train the SVM is typically the hinge loss function, which is used to find the hyperplane that best separates the classes in the input data. The squared hinge loss is another variant of the hinge loss that can be used in training an SVM.

* **VGG-SVM:**
  + **VGG16:** The combination of the powerful feature extraction capabilities of the VGG16 model with the SVM's ability to perform effective classification, was a great approach. The architecture of the VGG16 model is relatively simple and is based on the use of small kernel sizes (3x3) in each convolutional layer, along with max pooling layers. The 16 layers are organized into 5 blocks. The first two blocks contain two convolutional layers each, followed by a max pooling layer. The remaining three blocks have three convolutional layers each, followed by another max pooling layer. Finally, the model has three fully connected layers, which end in a softmax layer.
  + **Input layer**: The original fully connected layer of the VGG16 model was replaced with custom layers that were specifically designed to perform the task at hand, which involved processing input with dimensions of (48, 48, 1).
  + **Weights**: The pre-trained weights of the ImageNet dataset that were used in the original VGG16 model were left unchanged. This is because the weights learned by the VGG16 model on ImageNet have been shown to be highly effective for a wide range of computer vision tasks. By leaving these weights unchanged, we can leverage the knowledge learned by the VGG16 model on a large dataset to improve the performance of our custom model. Additionally, freezing the pre-trained weights can prevent overfitting and help the model generalize better to new data.
  + **SVM** : SVM is particularly effective in separating data into different classes by constructing a hyperplane in a high-dimensional space that maximally separates the classes. The hyperplane is determined by selecting the parameters that maximize the margin between the classes. In our model, different kernels were applied with different C numbers with different ranges that outputed similar performance rates.
  + In Case 1, VGG16 model was used to extract features to input it in our SVM classifier with normal parameters used.
  + In Case 2, more layers were added to fine-tune our VGG16 model as Dense layer with 50 neurons and activation function =”RelU” , 32 neurons and also activation=’RelU’.
* **CNN:**
  + **Input Layer:** The input layer of a neural network can hold different sizes depending on the specific application and data being used. Therefore different sizing were applied, where the input images are resized to a specific shape, such as (48, 48, 1), to match the requirements of the neural network architecture.
* **Convolution Layer:** The fundamental component of a CNN is the convolutional layer. A set of kernels with different sizes in this case (3, 3) or (5, 5) are used to extract features from the input data. Each kernel applies itself to the input and produces a feature map that represents various regional patterns or features to capture a variety of features, several filters are employed.
* **Max-pooling Layer:** The max pooling layer as one of their components to downscale input feature maps while preserving crucial data. The feature maps are divided into not overlapping windows with size (2, 2) in our case, and the highest value is chosen for each window. This lowers the spatial dimensions, offers translation invariance, and aids in feature extraction. In CNNs, the max pooling layer contributes to hierarchical feature learning and is essential for pattern and object recognition.
* **Dropout Layer:** In order to improve generalization and lessen overfitting in neural networks, the dropout layer is a regularization technique. Randomly deactivating some nodes lessens their influence and promotes the acquisition of reliable characteristics. The dropout rate in this case (0.10 or 0.15) controls it while training a collection of subnetworks enhances performance. It is a helpful technique for enhancing the generality and functionality of CNNs.
* **Flatten Layer:** In order to preserve relationships between features while rearranging the elements into a linear sequence, the Flatten layer turns multidimensional input tensors into a flat vector. The flattened output frequently serves as input for fully linked layers, enabling the network to make intelligent evaluations and decisions.
* **Loss Function:** The “loss = 'categorical\_crossentropy'” argument sets the loss function to categorical cross-entropy. This loss function is commonly used for multi-class classification tasks when the target variable is one-hot encoded or categorical.
* **Optimizer:** The (optimizer="adam") argument specifies the Adam optimizer. Adam is an adaptive optimization algorithm that adjusts the learning rate based on the gradients of the model parameters. It is widely used in deep learning due to its effectiveness and efficiency.
* **Metrics:** The “metrics = ["accuracy","Precision","Recall"]” argument sets the evaluation metrics to accuracy, precision and recall while we calculate the F1\_score.
* **Dense Layers:** Represents the number of hidden layers in the model to which the output layer is connected.
* **Output Layer:** It has 4 outputs since our problem is multiclass and uses the softmax activation function which typically used for categorical multiclass data.
* **Cases:**
* **Case\_1:**
* 3 Convolution layers with filters (16, 32, 64) and size (3,3).
* 3 dropout layers with rates (0.15 and 0.10).
* A Flatten Layer.
* 512 Dense Layers.
* Preprocessing:
* Resizing to shape (48,48,1).
* Rescaling: 1/255.
* **Case\_2:**
* 4 Convolution layers with filters (16, 32, 64,128) and size (3,3).
* 3 dropout layers with rates (0.15 and 0.10).
* A Flatten Layer.
* 512 Dense Layers.
* Batch Size: 32.
* Preprocessing:
* Resizing to shape (48,48,1).
* Rescaling: 1/255.
* **Case\_3:**
* 4 Convolution layers with filters (16, 32, 64,128) and size (3,3).
* 4 dropout layers with rates (0.15 and 0.10).
* A Flatten Layer.
* 512 Dense Layers.
* Batch Size: 64.
* Preprocessing:
* Resizing to shape (48,48,1).
* Rescaling: 1/255.
* **Case\_5:**
* 3 Convolution layers with filters (16, 32, 64) and size (3,3).
* 3 dropout layers with rates (0.15 and 0.10).
* A Flatten Layer.
* 512 Dense Layers.
* Batch Size: 32.
* Preprocessing:
* Resizing to shape (48,48,1).
* Zoom\_Range =0.2.
* Shear\_Range=0.2.
* **VVG - 16 KNN:**
* **Explaining the VGG16 basis model:**
  + The "imagenet" dataset's VGG16 model with pre-trained weights is imported.
  + The 'include\_top=False' parameter excludes the top layers (completely linked layers).
  + The 'Input' keyword defines an input tensor with the form (48, 48, 3).
  + To produce the output tensor, the input tensor is routed through the base model.
  + 'Flatten' flattens the output tensor.
  + Finally, after using the input and output tensors, a new model is built.
* **Feature extraction from photos utilizing the VGG16 model:**
  + To hold the extracted features and labels, respectively, an empty list named "x\_train" and "y\_train" are initialised.
  + To display a progress bar, the code iterates through each filename in 'df.Filename' using 'tqdm'.
  + The picture is scaled to (48, 48) and loaded using the 'load\_img' function.
  + The picture is enlarged to match the input shape of the model before being transformed to a numpy array using the 'img\_to\_array' function.
  + To normalise pixel values, the picture array is preprocessed using 'preprocess\_input'.
  + To acquire features, the preprocessed picture array is sent through the VGG16 model using'model.predict'.
  + 'features.squeeze()' is used to eliminate single-dimensional dimensions from the retrieved features.
  + 'x\_train' has the squeezed characteristics added to it.
* **Using the 'MinMaxScaler' to scale the extracted features:**
  + A copy of the 'MinMaxScaler' class is generated and allocated to the  'scaler' object.
  + To determine the minimum and maximum values, the 'fit' method on  'scaler' is called with the input 'x\_train'.
  + 'x\_train' calls the 'transform' function on  'scaler' to scale the features.
  + The scaled features are moved to the 'x\_train' segment.
* **Splitting training and validation sets from the training data:**
  + The 'x\_train' and 'y' variables are separated into training and validation sets using the 'train\_test\_split' function.
  + 'x\_train', 'x\_valid', 'y\_train', and 'y\_valid' are the training and validation sets, respectively.
  + With a test size of 0.2,'stratify=y' to preserve class proportions, and an 8-state randomization for repeatability, the split is carried out.
* **Using the training data to develop a k-nearest neighbors classifier:**
  + When the k-nearest neighbors classifier ('KNeighborsClassifier') is created, the parameters 'n\_neighbors' and 'p' (which uses the Manhattan distance or Eclidean Distance metric).
  + The 'x\_train' and 'y\_train' inputs are passed to the classifier's 'fit' function in order to train the model.
* **Using the trained classifier to predict labels for the validation set:**
  + The classifier ('clf') is invoked using the input 'x\_valid' and the 'predict' method.
  + 'y\_pred' has been given the anticipated labels.
* **VGG16 - Naïve Bayes:**

Similar steps for KNN except for the following steps:

* **Using the training data to develop Naïve Bayes classifier:**
* NB = GaussianNB() creates an instance of the Gaussian Naive Bayes classifier.
* The fit() method is used to train the classifier by providing it with the training data (x\_train and y\_train).
* The code y\_pred = NB.predict(x\_valid) predicts the target labels for the validation data (x\_valid) using the trained classifier.
* The predicted labels are stored in the variable y\_pred.
* The predict() method is called on the trained classifier (NB) to make predictions.
* The predicted labels can be further used for evaluation or analysis.
* **VGG19 – Random Forest:**

Similar steps for KNN/Naïve Bayes except for the following steps:

* **Using the training data to develop the Random Forest classifier:**
* rfc = RandomForestClassifier() creates an instance of the Random Forest classifier.
* The hyperparameters used are:
  + n\_estimators = 400
    - The number of trees in the forest.
  + min\_samples\_split = 2
    - The minimum number of samples required to split an internal node
  + min\_samples\_leaf = 1
    - The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
  + max\_features = 'sqrt'
    - The number of features to consider when looking for the best split
  + max\_depth = None
    - The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.
  + bootstrap = False
    - Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.
* The fit() method is used to train the classifier by providing it with the training data (x\_train and y\_train).
* The code y\_pred = rfc.predict(x\_valid) predicts the target labels for the validation data (x\_valid) using the trained classifier.
* The predicted labels are stored in the variable y\_pred.
* The predict() method is called on the trained classifier (rfc) to make predictions.
* The predicted labels can be further used for evaluation or analysis.
* **VGG19 – Decision Tree:**

Similar steps for KNN/Naïve Bayes/Random Forest except for the following steps:

* **Using the training data to develop the Decision Tree classifier:**
* dtc = DecisionTreeClassifier () creates an instance of the Random Forest classifier.
* The fit() method is used to train the classifier by providing it with the training data (x\_train and y\_train).
* The code y\_pred = dtc.predict(x\_valid) predicts the target labels for the validation data (x\_valid) using the trained classifier.
* The predicted labels are stored in the variable y\_pred.
* The predict() method is called on the trained classifier (dtc) to make predictions.
* The predicted labels can be further used for evaluation or analysis.
* **VGG16:**
* The preprocessing function to be applied to the incoming data is specified by the "preprocessing\_function" parameter.

* tf.keras.applications.mobilenet.preprocess\_input is employed in this situation to normalise and resize the input image.
* Using a directory that contains subdirectories of images, the "flow\_from\_directory()"  function creates batches of data. A separate class is represented by each subfolder.
* The "batch\_size" parameter indicates the number of images in each batch, while the "target\_size" argument defines the intended size of the output images in the batch.
* The train\_batches generator's next batch of data and labels is returned by the function next(train\_batches). Instead of putting the complete dataset at once, this can be useful for iterating through the entire dataset in batches.
* “vgg16\_model=tf.keras.applications.vgg16.VGG16()”:  this function initializes the model's weights using pre-trained weights from the ImageNet dataset, and then we can print the entire summary of the architecture of the model using “vgg16\_model.summary()”.
* Builds a new instance of a Keras Sequential model using “tf.keras.models.Sequential()”. “for layer in vgg16\_model.layers[:-1]” loops through all of the pre-trained model's layers but the final one, this is used  to reuse the pre-trained model's convolutional layers, which can accelerate training and enhance model performance.
* “for layer in model.layers: layer.trainable=False”: changes the trainable property of each layer in the model object to False after it iterates through all the levels. This means that the pre-trained values of these layers' weights will be maintained during the training procedure.
* “model.compile(tf.keras.optimizers.SGD(learning\_rate=0.0001),loss='categorical\_crossentropy', metrics=['accuracy'])”: configures the optimizer, loss function, and evaluation metric to get the model ready for training. During training, the SGD optimizer modifies the network's weights by determining the gradients of the loss function relative to the weights. The step size for these weight updates is determined by the learning rate parameter.
* Model is evaluated using the accuracy,f1score, recall and precision.

**Results and performance evaluation:**

This table represents the best case of each algorithm.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Train Accuracy** | **Test**  **Accuracy** | **Train F1Score** | **Testing**  **F1Score** | **Train**  **Precision** | **Test**  **Precision** | **Train**  **Recall** | **Test**  **Recall** |
| KNN | 80% | 49.4% | 30% | 28% | 32% | 40% | 45% | 50% |
| Naïve Bayes | 13% | 47% | 12% | 29% | 35% | 30% | 33% | 29% |
| CNN | 97% | 68% | 68% | 67% | 97% | 68% | 97% | 67% |
| CNN\_SVM | 89% | 68% | 67% | 61% | 92% | 69% | 92% | 68% |
| VGG16 |  |  | 38% | 27% | 70% | 27% |  | 25% |
| VGG16\_SVM | 58% | 50% | 36% | 16% | 60% | 67% | 70% | 64 |
| Random Forest | 69% | 50% | 52% | 27% | 48% | 28% | 80% | 31% |
| Decision Tree | 58% | 37% | 50% | 20% | 50% | 21% | 46% | 21% |

**Interpretation:**

These results indicate that the CNN model was the best classifier of the provided dataset, this was CNN case\_1 with 3 convolution layers,3 maxpooling layers,3 dropout layers, Flatten layer and 512 dense layers with no pre-processing other than resizing the images to the shape (48,48,1) and normalizing the features. The Following is the CNN-SVM case\_ which has .

**Discussion and interpretation of results:**

**References used in the projects**

1. **VGG16 – KNN:** <https://www.kaggle.com/code/duynm619/vgg-knn>.
2. **CNN:** <https://www.kaggle.com/code/ahmadjaved097/multiclass-image-classification-using-cnn>.
3. **CNN-SVM:** <https://github.com/krishnaik06/Complete-Deep-Learning/blob/master/Image%20Classification%20Using%20SVM.ipynb>.