Milestone 5 – Create a CNN and train and save, and with it create Confusion metrix using dataset#3.

(Parameter setting and reliability test of a sensor system for infant carrier car seat sensing in a car using a dashboard sensor)

Masters of Engineering
Information Technology

Aneeta Antony(1431461)
aneeta.antony@stud.fra-uas.de

Sonam Priya (1427129)
sonam.priya@stud.fra-uas.de

Nitu Shrestha (1428296)
nitu.shrestha@stud.fra-uas.de

I. IMPLEMENTATION -MILESTONE 5

A. *Create a CNN model in python programming.*

- Create a new CNN model for our experiment goal.
- Use the big dataset1 to train and optimize the CNN model created.
- Save this pre-trained CNN model.
- Use the Dataset#3 as input, load the pre-trained CNN Model and carryout prediction.
- Observed and submitted here with the Confusion matrix output and the accuracy range of the trained CNN model.

1) *CNN Model Creation*

a) Model Creation:

- A Sequential model is initialized using tf.keras.Sequential(), indicating that layers will be added sequentially.
- Input layer with shape (4,) is defined, suggesting the input shape of the data.
- Two dense (fully connected) layers with 64 and 32 neurons respectively, both using ReLU activation function and L2 regularization with a regularization parameter of 0.001.
- A dropout layer with a dropout rate of 0.5 is added to reduce overfitting.
- The output layer with a single neuron and sigmoid activation function, suitable for binary classification problems.

b) Model Compilation:

- The model is compiled using the Adam optimizer and binary cross-entropy loss function, which is common for binary classification tasks.
- Accuracy is chosen as the metric to monitor during training.

c) Model Training:

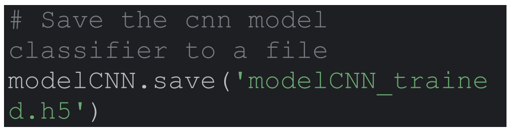
- The compiled model is trained on the training data (X_train and y_train) for 30 epochs with a batch size of 128.

The Code is as below:

```
# Separate features and target
X = combined_df.drop('Infant_Presence', axis=1)
y = combined_df['Infant_Presence']
# Splitting the dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Create a CNN model with increased complexity and
adjusted regularization
# Establish a more intricate CNN model
modelCNN = tf.keras.Sequential([
   tf.keras.layers.Input(shape=(4,)),
   tf.keras.layers.Dense(64, activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
   tf.keras.layers.Dropout(0.5),
   tf.keras.layers.Dense(32, activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
   tf.keras.layers.Dense(1, activation='sigmoid'),
])

modelCNN.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])

# Train the classifier with adjusted parameters
modelCNN.fit(X_train, y_train, epochs=30,
batch_size=128)
```

The accuracy and performance during traing observerd is like : Test Loss: 1.728192687034607, Test Accuracy: 0.9950894713401794, Accuracy: 1.00, Precision: 0.99 Recall: 1.00, F1-score: 1.00 .

d) The Trained CNN is saved for further use.



```
# Save the cnn model
classifier to a file
modelCNN.save('modelCNN_traine
d.h5')
```

2) The Confusion matrix of the CNN Mdel during the creation with data set1 is like show in figure below.

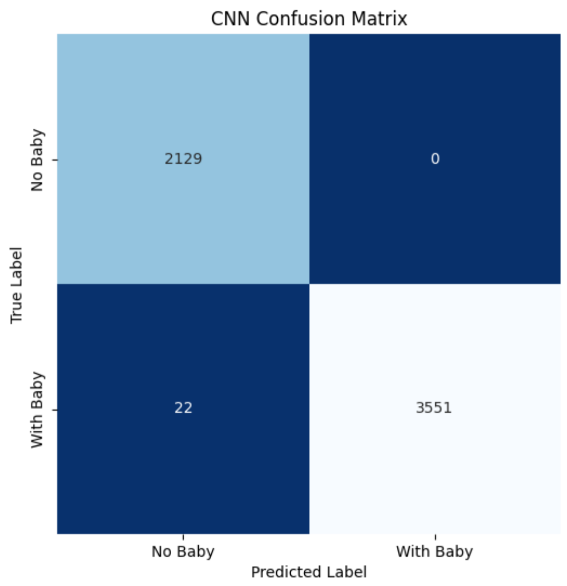


Figure 1 Confusion Matrix during training and testing of created CNN Model.

B. Confusion Matrix created from trained MLClassifier and RF Classifier

Toward this milestone we generated a confusion matrix from the saved pretrained CNN Model.

- Comparing the predicted labels with test labels.
- Predicted label and accuracy are found for the dataset#3 test data.
- Final optimized prediction Confusion Matrix looks like below Figure 2.

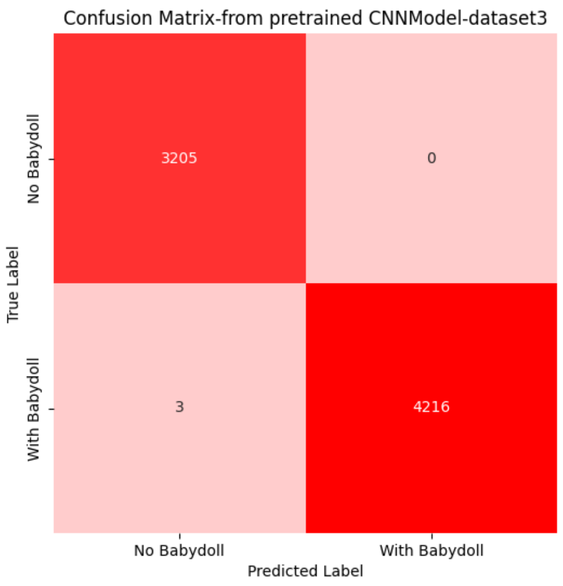


Figure 2 Confusion Matrix from CNN Model

1) Accuracy and performance on CNN with Dataset#3

- Accuracy: 1.00
- Precision: 1.00
- Recall: 1.00
- F1-score: 1.00

C. The Hyperparameter modified for the CNN Model created in our experiment using tenserflow.

1) Number of Layers and Neurons: Experiment with different architectures by varying the number of layers and the number of neurons in each layer (Dense layers).

- We tried adding multiple dense layers adding more layers, increasing or decreasing the number of neurons in each layer, or changing the activation functions.

2) Regularization: Adjust the regularization strength (e.g., L2 regularization parameter) to prevent overfitting. In *this model, regularization is applied using the kernel_regularizer argument in the Dense layers.*

3) Dropout Rate: Dropout is a regularization technique used to prevent overfitting by randomly setting a fraction of input units to 0 during training. The dropout rate (probability of dropping out a neuron) can be adjusted using the Dropout layer. Experiment with different dropout rates to see how they affect the model's performance.

4) Learning Rate: The learning rate determines the step size during optimization. The default learning rate for the Adam optimizer (adam) is usually a good starting point, but you can experiment with different learning rates to find the optimal value for your model. We tried to check with others like learning rate schedules or decay to adjust the learning rate over time.

5) Optimizer: In this case, the Adam optimizer (adam) is used, which is a popular choice for training neural networks. However, you can experiment with other optimizers such as SGD (Stochastic Gradient Descent), RMSprop, or Adam with different learning rates and momentumvalues.

6) Batch Size: The batch size specifies the number of samples used in each mini-batch during training. Larger batch sizes can speed up training but may require more memory. Experiment with different batch sizes to see how they affect training stability and convergence.

7) Number of Epochs: The number of epochs defines how many times the entire training dataset is passed forward and backward through the neural network. Training for too few epochs may result in underfitting, while training for too many epochs may result in overfitting. Experiment with different numbers of epochs to find the optimal balance between underfitting and overfitting.

D. Dataset#3 , FFT Dataset#3.

- Already submitted in the previous milestone.

The dataset#3 both ADC Data and Processed transformed fft can be found in the OneDrive Link - Dataset3 (FFT Data_data folder and ADC Data Folders) .The fft data is processed and saved as numpy dataframe arrays are saved for the with-baby-doll and with-out-baby- doll namely withbaby_npy_array_DS§ & withoutbaby_npy_array_DS3 .

CreateCNN-ML5.py

CNN creation , Python code.

CNN-load-DS3-confusionmatrix.py

CNN prediction Python code.

References

[1] K. S. Gill, V. Anand, R. Gupta and P. -A. Hsiung, "Detection of Malware Using Machine Learning techniques on Random Forest, Decision Tree, KNeighbour, AdaBoost, SGD, ExtraTrees and GaussianNB Classifier," 2023 4th IEEE Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2023, pp. 1-7, doi: 10.1109/GCAT59970.2023.10353495. keywords: {Codes;Digital systems;Computational modeling;Malware;Stability analysis;Data models;Pattern recognition;Data Science;Behaviour Analysis;Malware Attacks;Anomaly Detection;Classifcation;Machine Learning;Deep Learning;Artifcial Intelligence;Classifers}

[2] K.N. Fang, J.B. Wu, J.P. Zhu and B.C. Xie, "A review of technologies on random forests", *Statistics and Information Forum*, vol. A26, pp. 32-38, 2011.

[3] D. Yuan, J. Huang, X. Yang and J. Cui, "Improved random forest classification approach based on hybrid clustering selection," 2020 Chinese Automation Congress (CAC), Shanghai, China, 2020, pp. 1559-1563, doi: 10.1109/CAC51589.2020.9326711. keywords: {Random forests;Decision trees;Clustering algorithms;Classification algorithms;Indexes;Vegetation;Partitioning algorithms;random forest algorithm;clustering ensemble selection;personal indoor thermal preference}

[4] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html