Report Homework 2 Machine Learning

Edoardo Caciolo 1793918

09 Gennaio 2025

List of abbreviations

Acc Accuracy

AEM Average Evaluation Metrics

BS Batch Size

CNN Convolutional Neural Network

DQN Deep Q-Network

EP Epoches

F1 F1 Score

LR Learn Rate

MN Model Number

MPC Model Predictive Control

Pre Precision

Rec Recall

RL Reinforcement Learning

RNN Recurrent Neural Network

TR Total Reward

Syntax models:

m_	2	0005	32	30	.h5	→	m_	CNN	LR	BS	EP	.ext

Contents

Chapter 1

Introduction

This first chapter will describes the scope of the project and the models used.

1.1 Project Scope

The objective of this project is to solve a control problem for a racing car, in the Gymnasium virtual environment. Needing to operate in a discrete space of actions, two CNN models were developed, trained on a dataset of `6369` pre-classified images and labeled according to specific actions.
Once the two basic models were structured, it was a matter of choosing the right parameters so that these, when applied to the virtual model, would make the car perform at its best. The final choice was made based on the evaluation metrics, derived from tests on another dataset structured in the same way as the training dataset, the Total Reward recorded, and a visual check, made on the model implementation.

Discrete Action Space:

0	1	2	3	4
Nothing	Left Steering	Right Steering	Gas	Brake

1.2 CNNs structure

Neural networks are a subset of machine learning and play a key role in deep learning algorithms. They consist of layers of nodes that contain an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of a single node is above the specified threshold value, that node is activated, sending data to the next layer in the network.
There are various types of neural networks, used for different use cases and types of data. For example, recurrent neural networks (RNN) are commonly used for NLP and speech recognition, while convolutional neural networks (CNN) are more often used for classification and computer vision tasks.

In general, the structure of a CNN is as follows:

Convolutional layer (Conv2D)

The first layer of a convolutional neural network is always a convolutional layer. Convolutional layers apply a convolution operation to the input, then pass the result to the next one. It converts all the pixels in its receptive field to a single value. For example, if a convolution is applied to an image, it reduces the size of the image and combines all the information in the field into a single pixel. The final product of the convolution layer is a vector.

Activation Layer

Introduces nonlinearity into the model, allowing it to learn complex relationships. The `ReLU` (Rectified Linear Unit) is the most common. It can be integrated directly into convolutional layers.

Pooling Layer (MaxPooling2D)

Pooling layers, also called subsampling layers, perform dimensionality reduction, reducing the number of input parameters and consequently the number of parameters and computational load. It also helps make the representation more robust to small variations in the input.

Normalization Layer (BatchNormalization)

Normalizes the output of one layer per batch, improving the stability and speed of training.

Flattening Layer

Converts the 2D feature map data to a 1D vector. This step is necessary to connect the convolutional layers to the dense (fully connected) layers.

Dense Layer

These are fully connected layers that learn nonlinear relationships from the flattened data. Unlike partially connected layers, in the fully connected layer each node in the output layer connects directly to a node in the previous one. This layer performs the classification task based on the features extracted through the previous layers and their various filters. While convolutional and pooling layers tend to use `ReLu` functions, fully connected layers usually exploit a `Softmax` activation function to classify inputs appropriately, producing a probability of `0` to `1`.

Dropout Level

Helps prevent overfitting by randomly turning off some neurons during training, forcing the network not to rely too heavily on a specific input.

1.3 Models

The two CNN models used will be shown and explained below. The search for the best parameters applied to them will then be commented on.

1.3.1 CNN1 Model

1. `Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)`
   (a) `Layer Type:` Convolutional 2D.
   (b) `Number of Filters:` 32.
   (c) `Filter Size:` 3x3.
   (d) `Activation Function:` ReLU.
   (e) `Input Shape:` Dimension of the image input.
   (f) `Description:` Applies 32 convolutional filters of size 3x3 to the input, each extracting specific features. The ReLU activation function adds non-linearity, enabling the learning of complex relationships.
2. `MaxPooling2D((2, 2))`
   (a) `Layer Type:` Max Pooling 2D.
   (b) `Pool Size:` 2x2.
   (c) `Description:` Reduces the spatial dimension of the output from the previous layer by taking the maximum value within 2x2 windows, reducing computation for subsequent layers and making the model less sensitive to feature positioning.
3. `Additional Conv2D and MaxPooling2D Layers`
   (a) `Structure:` Same structure but with different filters (64 and 128).
   (b) `Description:` Increases the number of filters to capture more complex features, making the image representation more abstract with each layer.
4. `Flatten()`
   (a) `Layer Type:` Flatten.
   (b) `Description:` Converts multi-dimensional spatial data into a one-dimensional vector, necessary for the transition from convolutional to dense layers.
5. `Dense(128, activation='relu')`
   (a) `Layer Type:` Dense.
   (b) `Number of Neurons:` 128.
   (c) `Activation Function:` ReLU.
   (d) `Description:` A fully connected layer that learns non-linear relationships from flattened data.
6. `Dropout(0.5)`
   (a) `Layer Type:` Dropout.
   (b) `Rate:` 0.5.
   (c) `Description:` Randomly excludes 50
7. `Dense(num_classes, activation='softmax')`
   (a) `Layer Type:` Dense.
   (b) `Number of Neurons:` Depends on the number of classes.
   (c) `Activation Function:` Softmax.
   (d) `Description:` Assigns a probability to each class, ensuring that the output is a probability distribution with values between 0 and 1, summing to 1.

1.3.2 CNN2 Model

1. `Conv2D(16, (5, 5), activation='relu', input_shape=input_shape)`
   (a) `Layer Type:` Convolutional 2D.
   (b) `Number of Filters:` 16.
   (c) `Filter Size:` 5x5.
   (d) `Activation Function:` ReLU.
   (e) `Input Shape:` Dimension of the im-

age input.

   (f) `Description:` Applies 16 convolutional filters of size 5x5 to the input, each extracting specific features. The ReLU activation function adds non-linearity to the model.

2. `BatchNormalization()`

   (a) `Layer Type:` Batch Normalization.

   (b) `Description:` Normalizes the activations of the previous layer at each batch, improving the stability and speed of the training process.

3. `MaxPooling2D((2, 2))`

   (a) `Layer Type:` Max Pooling 2D.

   (b) `Pool Size:` 2x2.

   (c) `Description:` Reduces the spatial dimensions of the output from the previous layer, thereby reducing the number of parameters and computational load.

4. `Conv2D(32, (5, 5), activation='relu')`

   (a) `Layer Type:` Convolutional 2D.

   (b) `Number of Filters:` 32.

   (c) `Filter Size:` 5x5.

   (d) `Activation Function:` ReLU.

   (e) `Description:` Applies 32 convolutional filters to further refine the feature extraction process.

5. `BatchNormalization()` (Repeated)

6. `MaxPooling2D((2, 2))` (Repeated)

7. `Conv2D(64, (5, 5), activation='relu')`

   (a) `Layer Type:` Convolutional 2D.

   (b) `Number of Filters:` 64.

   (c) `Filter Size:` 5x5.

   (d) `Activation Function:` ReLU.

   (e) `Description:` Applies 64 convolutional filters, enabling the extraction of even more complex features from the input.

8. `Flatten()`

   (a) `Layer Type:` Flatten.

   (b) `Description:` Converts the multi-dimensional feature maps into a one-dimensional vector, preparing the data for the dense layers.

9. `Dense(64, activation='relu')`

   (a) `Layer Type:` Dense.

   (b) `Number of Neurons:` 64.

   (c) `Activation Function:` ReLU.

   (d) `Description:` A fully connected layer that processes the flattened data, learning non-linear relationships.

10. `Dropout(0.3)`

   (a) `Layer Type:` Dropout.

   (b) `Rate:` 0.3.

   (c) `Description:` Randomly drops 30

11. `Dense(num_classes, activation='softmax')`

   (a) `Layer Type:` Dense.

   (b) `Number of Neurons:` Depends on the number of classes.

   (c) `Activation Function:` Softmax.

   (d) `Description:` This final dense layer assigns probabilities to each class using the softmax function, which converts the output into a probability distribution.

The main difference between CNN1 and CNN2 lies in the number and size of filters in the convolutional layers, as well as the presence of batch normalization in CNN2. CNN1 is designed to capture more detailed features, while CNN2 emphasizes larger features and potentially more stable and faster training due to batch normalization.

Finally, `Adam` was chosen as the optimizer because it has a good ability to adapt the learning rate for each parameter. As a result, it is ideal in situations where data can change very rapidly, as is typical in a racing simulation environment. This flexibility allows the model to optimize performance and improve learning.

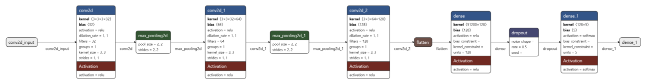

Figure 1.1: CNN1 model scheme - `m_10013210.h5`

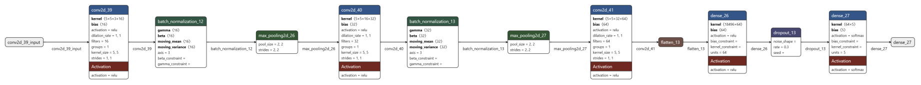

Figure 1.2: CNN2 model scheme - `m_200013210.h5`

The source code of the two CNN models used (Listing 2 and Listing 3) can be found in Appendix A.2.

The graphical structure of two tested CNN models is represented in Fig.1.1 and Fig.1.2 .

Chapter 2

Training

This chapter will explain the analysis techniques used and the partial results obtained. These will be preparatory to the choice of the optimal CNN model, used for the final application on the virtual Car-Racing environment.
For the sake of brevity, the source codes (*Section 4.2*) will be omitted; all results shown were produced by these attached scripts.

2.1 Methodology

To fit the produced models, hyperparameters were chosen through the use of the function `Grid Search`. This function trains the CNN with the dataset under consideration, by iterating a permutation of several chosen parameters. At the end of the process it generates a `.h5` model for each combination, in the case under consideration $3^3 * 2 = 54$ [1].
The parameters tested by Grid Search are:

1. `Batch Size [32, 64, 128]`: Determines the number of training samples processed before updating the internal model weights. Small batches can lead to faster but less stable training, while larger batches can stabilize training but require more memory.

2. `Learning rate [0.001, 0.0005, 0.0001]`: Parameter that sets the step size during weight updates. A value that is too high may prevent conver-

[1] Due to hardware resource problems, the generation of models without oversampling stopped at `39`.

gence, while a value that is too low may slow down training or cause blockages at local minima.

3. `Epoch [10, 20, 30]`: Represents a complete cycle of the entire training dataset passing through the CNN. A larger number of epochs gives the network more opportunity to learn and adjust the weights, but if it is excessive it can lead to "catastrophic forgetting."

It would have been possible to apply the same search function for other CNN parameters such as `Stride`, `Padding`, `Kernel`, `Dropout Rate, Activation Function`, and `Pooling Layer`. However, since these parameters require considerable computational resources, we chose to manually set the values that we felt would be useful for the success of the project.

For data preprocessing, normalization was applied by dividing all pixel values by `255.0`. This transforms the pixel values from a range of `0-255` to a range of `0.0-1.0`. This step reduces variance in the data and helps speed up training, making learning easier since it ensures that the images have the same size and scale of values before being processed.

Another preventive analysis was to verify that the dataset was balanced.

```
Image count by class:
{0:  1000, 1:  1500, 2:  1500, 3:  2000, 4:  369}
```

The results show an important imbalance among the five image classes. In theory, it would have been necessary to apply algorithms for oversampling (or undersampling), class weighting and synthetic data generation. This was to avoid problems of biased learning and overfitting. For this reason, a resampling function was introduced by exploiting the `sklearn` library. Thanks to it, each class was oversampled to have the same number of examples as the most represented class (in this specific case, the third one).

```
Image count by class after balancing:
{0:  2000, 1:  2000, 2:  2000, 3:  2000, 4:  2000}
```

However, it was decided not to apply any kind of balancing to the final model. The reason for this choice will be explained at the end of *Chapter 3 (Subsection 3.1.1)*, so that we have an overview of the project and do not generate confusion. Everything that follows therefore refers to CNN models without oversampling.

At this point, having obtained a series of models (`model.h5`), we proceeded with the extraction of the main evaluation metrics: `Accuracy`, `Recall`, `Precision`, `F1 Score`, `Confusion Matrix`. In classification tasks, metrics are commonly used to evaluate the performance of machine learning models.
Another extremely important metric in a scenario like this one, the `Total Reward`, was also added. However, while the classic metrics were extracted using the `sklearn.metrics` library, the Total Reward was implemented and extracted directly in the simulation environment, being part of the Car-Racing Gymnasium environment. Moreover, it was considered important, to choose the optimal model, to also extract data on the computing power used; this includes: computation time, data on RAM and CPU usage. Unlike other classification tasks, where CNN models were not used, these values have always been low, uniform, and therefore of little relevance (during the single training with defined parameters).

2.2 Results

Below are the two tables with the results, about the training, obtained through the 39 models [2].
In Tab.**??** the evaluation metrics of each model and the Total Reward accumulated during the simulation phase are shown.

A first observation can be made about the `m_200013210.h5` model (CNN2 model). In fact, it turns out to be the best (among the other CNN2 models) in terms of evaluation metrics and Total Reward. Another observation that can be made is to note that the models that seem to have worked best, are those with lower `Epoches` (`10`); as well as with `Batch Sizes` no larger than `64`.

As for CNN1 models, on the other hand, `m_100056410.h5` turns out to be the best in terms of evaluation metrics, while `m_10013210.h5` in terms of Total

[2]The best results for each measurement are highlighted in light grey.

Reward.

Table 2.1: Results model simulation

Id	**Acc**	**Pre**	**Rec**	**F1**	**AEM**	**TR**	**Model**
1	0.58	0.40	0.52	0.42	0.48	878.06	m_200013210.h5
2	0.60	0.40	0.51	0.42	0.48	878.06	m_10013210.h5
3	0.59	0.40	0.51	0.42	0.48	857.86	m_100016420.h5
4	0.56	0.38	0.49	0.40	0.46	857.86	m_100013230.h5
5	0.57	0.39	0.51	0.41	0.47	834.29	m_100016430.h5
6	0.61	0.39	0.49	0.41	0.48	830.93	m_100013220.h5
7	0.66	0.41	0.47	0.43	0.49	810.72	m_100016410.h5
8	0.53	0.37	0.49	0.38	0.44	770.32	m_200013220.h5
9	0.52	0.34	0.43	0.35	0.41	719.81	m_200056430.h5
10	0.52	0.36	0.48	0.37	0.43	612.07	m_20013220.h5
11	0.53	0.38	0.49	0.39	0.45	612.07	m_200016430.h5
12	0.61	0.40	0.50	0.42	0.48	474.02	m_100112810.h5
13	0.65	0.42	0.50	0.45	0.51	379.75	m_100056410.h5
14	0.62	0.40	0.51	0.43	0.49	376.38	m_10016410.h5
15	0.60	0.40	0.51	0.42	0.48	376.38	m_100053210.h5
...	...	...	...	...	...	...	...
36	0.40	0.35	0.47	0.32	0.38	-95.00	m_200056420.h5
37	0.49	0.35	0.47	0.35	0.42	-95.00	m_200112810.h5
38	0.16	0.20	0.24	0.13	0.18	-95.00	m_20013210.h5
39	0.32	0.32	0.43	0.27	0.34	-95.00	m_20016430.h5

The complete table of results is shown in Fig.A.2

2.2.1 Graphs

Below are the Confusion Matrices 2.1 of the top three models.

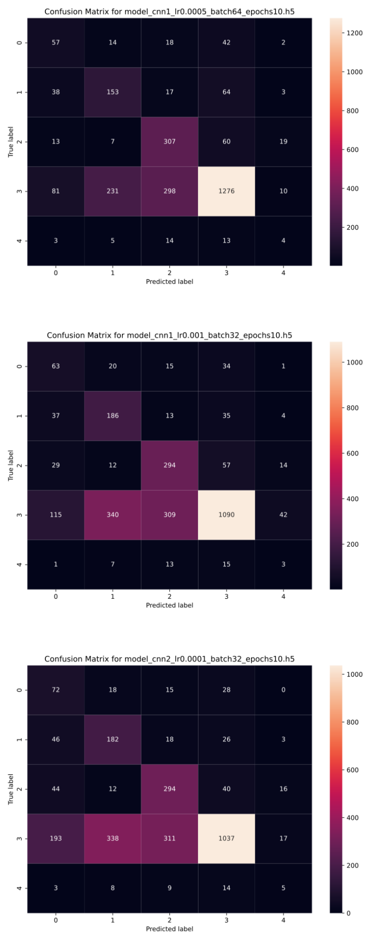

Figure 2.1: Confusion Matrices of the top three models

For all them, the biggest training problem occurred on the classification between labels `3` and `1-2`. In fact, there is a fair tendency to misclassify classes `1-2` as class `3`. This is evident from observation of the individual tables. The first idea was that this behavior was due to the preponderance of images belonging to `class 3` over the others. However, by producing the confusion matrices on the oversampled models, it was possible to verify that this is not true.

Chapter 3

Model choice

This chapter will address the final choice of model among those analyzed, based on the considerations made in the previous chapter. This will then be used to run the car within its virtual track.

3.1 Choice and application

Based on the considerations so far, the choice of which model to use falls on one of the three algorithms mentioned above. Unlike other classification projects, however, the choice cannot be made without visually evaluating the behavior of the virtual model. Indeed, it is not difficult for a model that appears to have the upper hand on paper, to show abnormal behavior during simulation. Excessively high speeds, sudden braking, or excessive lateral deviations may be trivial in such an environment, but they have devastating effects when applied to a real physical model. However, it is clear that virtual behavior, no matter how well calibrated, will differ in one way or another from the real thing.
Below are the pros, cons and visual observations made on the simulator, borne by the respective models.

model m_10013210.h5

It starts with a low speed and seemingly good trajectory control. Shortly thereafter he increases his speed. At this point it slightly loses trajectory control and lateral deflection, but has good road recovery and cornering control, although it seems to brake a little sharply. The main problem is speed control.

model m_100056410.h5

Poor control, it goes off the track and cannot get back on. In fact, it has a relatively low total reward of about `379`.

model m_200013210.h5

Almost perfect. Partially goes off the track in a couple of places, but recovers quickly. Very good cornering grip, even in tight ones, without braking sharply. Very good speed control, almost constant.

As mentioned, this simulation demonstrates how the choice of reference metric depends heavily on the context and specific goals of the application. In the case under consideration, for example, the most indicative was Total Reward, proper in truth to the Reinforcement Learning algorithms upon which the Car-Racing Gymnasium environment was designed. In any case, given the analysis and proposed results, the algorithm chosen for this problem is `m_200013210.h5`

3.1.1 Model oversampling test

Section 2.1 introduced the presumed need to balance the dataset. Testing, however, showed that oversampling the training data resulted in significantly worse learning than with unbalanced classes. In Tab.A.1 it is evident how the best model (TR of `558.20` and an AEM of `0.43`) presents much lower results than its unbalanced counterpart. The reason for this evidence lies in the noncategorical need for balancing in any scenario. In some cases, such as simulated car racing, the natural distribution of the data is critical because it reflects the real-world conditions under which the model will be used; artificially balancing the dataset can therefore make the model less effective in handling real-world situations in which some classes are more frequent than others. In sequence-based learning problems or in dynamic environments such as games, where temporal relationships and sequentiality of events are critical, balancing classes can disrupt these important dynamics, leading to less effective learning.
In addition, different classes may have different costs or consequences; for example, in an automotive environment (virtual or even more physical), avoiding an obstacle may be more important than taking the perfect turn, making it critical

that the model learn to recognize and react appropriately to the most critical classes, even if they are less well represented. Forced balancing of a dataset can also introduce bias into the model, making it believe that all classes have the same probability, which is often not the case. Finally, oversampling can greatly increase the size of the dataset, increasing computational complexity and training time, a circumstance that has occurred.

The methodology applied for this test was the same as explained in the rest of the article. For completeness, a simulation test was also performed on the best algorithm found (`m_100016410.h5`). The implementation on the virtual environment showed that the car did not have good trajectory control, not always sufficient control of lateral deviation in cornering, and poor speed control, which converged to zero in some time intervals.

Chapter 4

Conclusions

This last chapter will list all the documents attached to the project for completeness. In addition, some future developments of possible interest will be presented.

4.1 Future developments

In previous chapters, reference has been made several times to Total Reward, the output of the Reward Function proper to Reinforcement Learning algorithms. The term has perhaps been misused in this context, but for a reason. The environment under consideration was created to be developed through RL or Deep Learning methodologies; the environment itself incorporates, as can be seen from the video, a Total Reward calculated on the behavior of the car. It is clear that this value is not entirely consistent. This could be visually verified in tests of other CNN models where the car remained stationary or even where the control went crazy causing the car to go against the road, spin, or completely off the road.

A promising advance could be the integration of a CNN, a DQN and an MPC to develop an extremely robust model that can be used in both virtual and physical environments.

DQNs, crucial in Deep Reinforcement Learning, learn optimal decision-making strategies through continuous interactions with the environment, adapting to maximize long-term gains. This results in the ability to plan actions with future implications, despite the fact that they require a significant amount of training data and are sensitive to parameter choice.

CNNs, which are excellent in image processing, are well suited for analyzing and interpreting visual data from the environment. These networks can handle complex visual data, allowing the model to make decisions based on the current visual conditions of the track. However, they have no inherent consideration of time or sequence, limiting their ability to plan long-term strategies.

MPC, with its forecasting and optimization-based approach and supported by an accurate physical and mathematical model, calculates the optimal trajectory and control actions taking into account specific constraints and objectives. Its effectiveness, however, is closely related to the accuracy of the physical model of the system and the ability to quickly solve complex optimization problems.

The integration of DQN, CNN, and MPC could offer a synergistic approach, leveraging the strengths of each methodology. CNNs would analyze track images, identifying obstacles and relevant features, while DQNs would learn optimal decision-making policies by interpreting the state of the environment provided by CNNs. The MPC, used in tandem with the DQN, would refine actions based on vehicle dynamics and system constraints. This integration would allow realistic and safe trajectories to be planned while maximizing performance and meeting physical constraints.

In summary, the integration of each of these techniques would compensate for the limitations of the others, improving image processing, strategic decision making, and optimized control. This approach could not only improve effectiveness in a virtual environment such as CarRacing, but also pave the way for implementation on physical models, shifting the boundary between virtual simulation and practical application in autonomous driving.

4.2 Attached files

The following files are attached:

1. `ML_Homework2_SourceCode_1793918.py`:
   CNN models generation code.
2. `ML_Homework2_evaluation_metrics_1793918.py`:
   Evaluation code for the created models.
3. `ML_Homework2_play_policy_template_1793918.py`:
   Simulation code of the created models in the virtual environment Car-Racing Gymnasium [box2d].
4. `Readme.md`:
   Readme file with useful instructions.
5. `models_Oversampling_OFF`:
   Folder containing 39 CNN models without oversampling.
6. `models_Oversampling_ON`:
   Folder containing 54 CNN models with oversampling.
7. `MER_ML_Homework_2_1793918.xlsx`:
   Contains 4 spreadsheets showing all evaluation metrics and confusion matrices with and without oversampling.
8. `ML_Homework_2_1793918_m_200013210_simulation_video_.mp4`:
   Video of the simulation with the best model (`m_200013210.h5`) without oversampling.

All resources are available in the public GitHub repository, within their respective `.ZIP` archives:

Project 2 Machine Learning GitHub Repository

Appendix A

Source Code

A.1 Libraries used

```
import os
import sys
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
import gymnasium as gym
```

Listing 1: Libraries used

A.2 CNN Model Source Code

```
def model_cnn1(input_shape, num_classes):
    model = Sequential([
            Conv2D(32, (3, 3), activation='relu',
                               input_shape=input_shape),
            MaxPooling2D((2, 2)),
            Conv2D(64, (3, 3), activation='relu'),
            MaxPooling2D((2, 2)),
            Conv2D(128, (3, 3), activation='relu'),
            Flatten(),
            Dense(128, activation='relu'),
            Dropout(0.5),
            Dense(num_classes, activation='softmax')
            ])
    return model
```

Listing 2: CNN Model 1

```
def model_cnn1(input_shape, num_classes):
    model = Sequential([
            Conv2D(32, (3, 3), activation='relu',
                               input_shape=input_shape),
            MaxPooling2D((2, 2)),
            Conv2D(64, (3, 3), activation='relu'),
            MaxPooling2D((2, 2)),
            Conv2D(128, (3, 3), activation='relu'),
            Flatten(),
            Dense(128, activation='relu'),
            Dropout(0.5),
            Dense(num_classes, activation='softmax')
            ])
    return model
```

Listing 3: CNN Model 2

A.3 Tables

Table A.1: Model evaluation results with oversampling

Id	Acc	Pre	Rec	F1	AEM	TR	Model
1	0.46	0.38	0.51	0.37	0.43	558.20	m_100016410.h5
2	0.55	0.36	0.44	0.37	0.43	447.09	m_200053210.h5
3	0.49	0.37	0.47	0.36	0.42	362.91	m_200112830.h5
4	0.53	0.38	0.48	0.38	0.44	83.45	m_200016410.h5
5	0.56	0.40	0.46	0.38	0.45	69.98	m_200056410.h5
6	0.41	0.36	0.49	0.34	0.40	36.31	m_1000512830.h5
7	0.46	0.36	0.48	0.35	0.41	19.48	m_10013230.h5
8	0.44	0.37	0.49	0.35	0.41	16.11	m_100053210.h5
9	0.46	0.36	0.48	0.36	0.42	-27.66	m_200013230.h5
10	0.61	0.40	0.49	0.42	0.48	-64.70	m_200112810.h5
11	0.43	0.38	0.51	0.36	0.42	-68.06	m_100016420.h5
12	0.47	0.36	0.46	0.36	0.41	-81.53	m_20013210.h5
13	0.44	0.36	0.49	0.35	0.41	-91.63	m_1000512820.h5
14	0.44	0.35	0.47	0.34	0.40	-91.63	m_100056420.h5
15	0.43	0.35	0.47	0.34	0.40	-91.63	m_100112830.h5
16	0.41	0.36	0.48	0.33	0.40	-91.63	m_10013210.h5
17	0.45	0.36	0.48	0.35	0.41	-91.63	m_10016420.h5
18	0.50	0.37	0.48	0.37	0.43	-91.63	m_200016430.h5
19	0.62	0.42	0.45	0.42	0.48	-91.63	m_2000512810.h5
20	0.47	0.38	0.51	0.38	0.43	-95.00	m_1000112810.h5
21	0.40	0.37	0.51	0.34	0.41	-95.00	m_1000112820.h5
22	0.45	0.37	0.49	0.36	0.42	-95.00	m_1000112830.h5
23	0.49	0.38	0.50	0.38	0.44	-95.00	m_100013210.h5
24	0.44	0.37	0.49	0.35	0.41	-95.00	m_100013220.h5

Id	Acc	Pre	Rec	F1	AEM	TR	Model
25	0.40	0.37	0.51	0.34	0.40	-95.00	m_100013230.h5
26	0.42	0.37	0.50	0.35	0.41	-95.00	m_100016430.h5
27	0.40	0.37	0.48	0.34	0.40	-95.00	m_1000512810.h5
28	0.40	0.35	0.48	0.33	0.39	-95.00	m_100053220.h5
29	0.46	0.36	0.48	0.35	0.41	-95.00	m_100053230.h5
30	0.40	0.36	0.46	0.33	0.39	-95.00	m_100056410.h5
31	0.39	0.35	0.47	0.32	0.38	-95.00	m_100056430.h5
32	0.44	0.37	0.50	0.36	0.42	-95.00	m_100112810.h5
33	0.41	0.35	0.47	0.33	0.39	-95.00	m_100112820.h5
34	0.42	0.36	0.50	0.34	0.40	-95.00	m_10013220.h5
35	0.42	0.36	0.49	0.34	0.40	-95.00	m_10016410.h5
36	0.45	0.35	0.47	0.34	0.40	-95.00	m_10016430.h5
37	0.40	0.37	0.48	0.34	0.40	-95.00	m_2000112810.h5
38	0.45	0.37	0.47	0.35	0.41	-95.00	m_2000112820.h5
39	0.43	0.36	0.50	0.35	0.41	-95.00	m_2000112830.h5
40	0.43	0.37	0.51	0.35	0.42	-95.00	m_200013210.h5
41	0.44	0.36	0.50	0.35	0.41	-95.00	m_200013220.h5
42	0.42	0.36	0.49	0.34	0.40	-95.00	m_200016420.h5
43	0.44	0.37	0.51	0.35	0.42	-95.00	m_2000512820.h5
44	0.42	0.36	0.45	0.34	0.39	-95.00	m_2000512830.h5
45	0.50	0.37	0.47	0.37	0.43	-95.00	m_200053220.h5
46	0.40	0.35	0.46	0.32	0.38	-95.00	m_200053230.h5
47	0.53	0.37	0.48	0.38	0.44	-95.00	m_200056420.h5
48	0.38	0.35	0.47	0.31	0.38	-95.00	m_200056430.h5
49	0.25	0.32	0.41	0.24	0.31	-95.00	m_200112820.h5
50	0.46	0.34	0.44	0.34	0.39	-95.00	m_20013220.h5

Id	Acc	Pre	Rec	F1	AEM	TR	Model
51	0.50	0.36	0.47	0.37	0.43	-95.00	m_20013230.h5
52	0.34	0.34	0.45	0.29	0.35	-95.00	m_20016410.h5
53	0.54	0.36	0.44	0.37	0.42	-95.00	m_20016420.h5
54	0.41	0.33	0.43	0.31	0.37	-95.00	m_20016430.h5

Table A.2: Model evaluation results without oversampling

Id	Acc	Pre	Rec	F1	AEM	TR	Model
1	0.58	0.40	0.52	0.42	0.48	878.06	m_200013210.h5
2	0.60	0.40	0.51	0.42	0.48	878.06	m_10013210.h5
3	0.59	0.40	0.51	0.42	0.48	857.86	m_100016420.h5
4	0.56	0.38	0.49	0.40	0.46	857.86	m_100013230.h5
5	0.57	0.39	0.51	0.41	0.47	834.29	m_100016430.h5
6	0.61	0.39	0.49	0.41	0.48	830.93	m_100013220.h5
7	0.66	0.41	0.47	0.43	0.49	810.72	m_100016410.h5
8	0.53	0.37	0.49	0.38	0.44	770.32	m_200013220.h5
9	0.52	0.34	0.43	0.35	0.41	719.81	m_200056430.h5
10	0.52	0.36	0.48	0.37	0.43	612.07	m_20013220.h5
11	0.53	0.38	0.49	0.39	0.45	612.07	m_200016430.h5
12	0.61	0.40	0.50	0.42	0.48	474.02	m_100112810.h5
13	0.65	0.42	0.50	0.45	0.51	379.75	m_100056410.h5
14	0.62	0.40	0.51	0.43	0.49	376.38	m_10016410.h5
15	0.60	0.40	0.51	0.42	0.48	376.38	m_100053210.h5
16	0.64	0.40	0.48	0.43	0.49	369.65	m_100013210.h5
17	0.59	0.36	0.46	0.38	0.45	316.72	m_200053210.h5
18	0.49	0.35	0.45	0.35	0.41	83.45	m_20016420.h5

Id	**Acc**	**Pre**	**Rec**	**F1**	**AEM**	**TR**	**Model**
19	0.47	0.35	0.45	0.35	0.40	73.35	m_20013230.h5
20	0.55	0.38	0.50	0.40	0.46	59.88	m_200016410.h5
21	0.49	0.36	0.48	0.36	0.42	59.88	m_10013230.h5
22	0.50	0.37	0.51	0.38	0.44	53.15	m_10016420.h5
23	0.52	0.38	0.51	0.39	0.45	22.85	m_100112820.h5
24	0.50	0.37	0.50	0.37	0.43	19.48	m_100053220.h5
25	0.49	0.37	0.51	0.37	0.43	19.48	m_10013220.h5
26	0.50	0.37	0.50	0.37	0.43	19.48	m_10016430.h5
27	0.66	0.38	0.34	0.34	0.43	-74.13	m_20016410.h5
28	0.49	0.37	0.51	0.37	0.44	-74.80	m_200016420.h5
29	0.49	0.36	0.49	0.36	0.42	-91.63	m_100053230.h5
30	0.45	0.36	0.51	0.36	0.42	-91.63	m_100056430.h5
31	0.51	0.37	0.50	0.38	0.44	-95.00	m_100056420.h5
32	0.44	0.35	0.48	0.34	0.40	-95.00	m_200013230.h5
33	0.42	0.35	0.47	0.33	0.39	-95.00	m_200053220.h5
34	0.40	0.34	0.46	0.32	0.38	-95.00	m_200053230.h5
35	0.21	0.35	0.38	0.22	0.29	-95.00	m_200056410.h5
36	0.40	0.35	0.47	0.32	0.38	-95.00	m_200056420.h5
37	0.49	0.35	0.47	0.35	0.42	-95.00	m_200112810.h5
38	0.16	0.20	0.24	0.13	0.18	-95.00	m_20013210.h5
39	0.32	0.32	0.43	0.27	0.34	-95.00	m_20016430.h5