Machine Learning
Homework 2

Simone Orelli
matricola 1749732

January 2024

Contents

1 Introduction

In this homework, we are invited to solve an image classification problem to learn the behavior of a racing car in a Gym environment. For image classification problem we mean to analyze the image and identify the class (label) to which it belongs. This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications. Gym, launched by OpenAI, is an open-source bank of artificial intelligence projects. This database has been put together for developers to use various artificial intelligence techniques such as reinforcement learning and computer vision to solve these environments. The environment explored in this study was CarRacing-v0, a 2D autonomous vehicle environment. Using machine learning an agent was trained to learn this track.

2 Image classification problem

The classification problem in its general form can be formulated as:

Definition 2.1 (Classification problem). Given a target function f, learning a function $\hat{f} : X \to Y$, with $Y = \{0, 1, ..., 5\}$ given a training set $D = \{x_i, t_i\}\{i = 1, ..., N\}$, such that the approximated function $\hat{f}$ returns values as close as possible to f, specially for samples not present in the dataset D.

In other words, the ML algorithm should analyze the training data and produce an inferred function, which can be used to label new examples outside the dataset. We want to make the function the more general possible, so that it's able to predict in the most correct way an unseen instance, while it's consistent with the dataset.

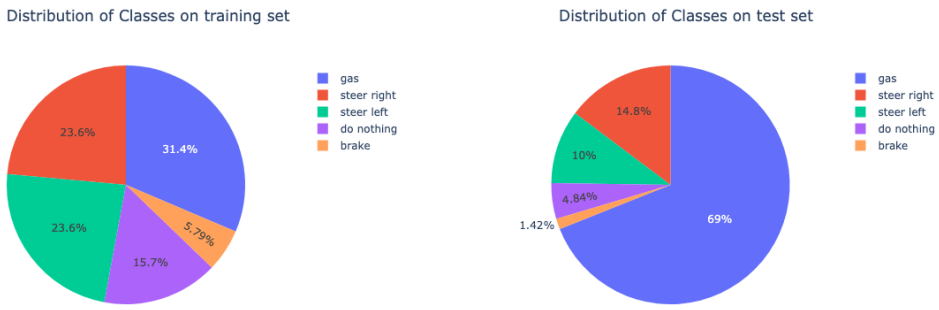

Figure 1: Distribution of images of the training set.

2.1 Dataset analysis

In our case the classes are divided into 5 possible actions:

1. Do nothing
2. Steer left
3. Steer right
4. Gas
5. Brake

The dataset is already divided into training and test sets, and it is composed by 9118 images of dimension $96 \times 96 \times 3$ (RGB). The dataset is divided into 6369 images for training set and 2749 images for test set. In the figure 1 is shown the distribution of the images of the training set and test set by classes. As we can see both the training set and the test set are unbalanced, but the test set is more than that. This could be a problem when solving the image classification problem. In the figure 2 is shown a random example of the image in the data set.

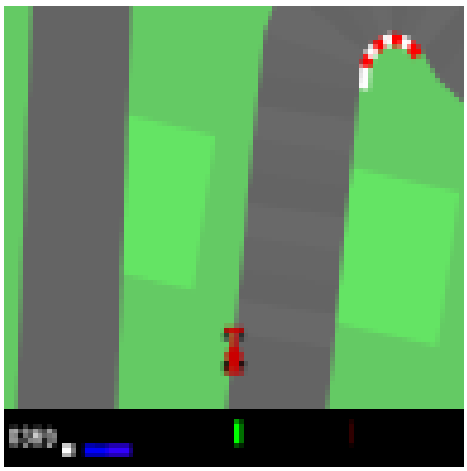

Figure 2: Random image of the dataset.

2.2 Preprocessing

As for all the data that fit any machine learning algorithm, either the images have to be preprocessed. Image preprocessing are the steps taken to format images before they are used by model training.

The aim of preprocessing is an improvement of the image data that suppresses unwilling distortions or enhances some image features important for further processing. The image preprocessing step is applied both to the training set and the test set and it consists to transform colored images into black and white images, removing the left down number. In the figure 3 is shown a random example of the preprocessed image.

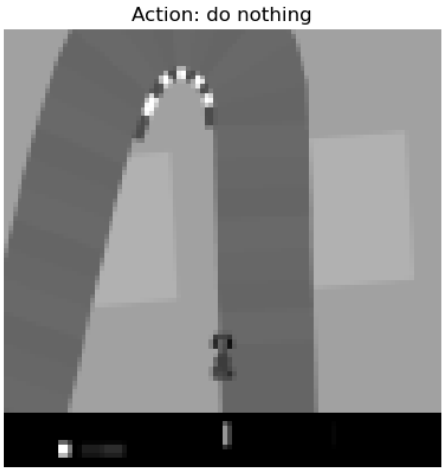

Figure 3: Random image of the preprocessed dataset

2.3 The models

The most commonly used image classification algorithm is the Convolutional Neural Network (CNN). A neural network is a function approximator represented by a parametric model. Given a target function f, the goal is to learn the parameters of the estimator $\hat{f}(x, \Theta)$ in such a way that $\hat{f}$ results to be a good approximation of the target function f. Convolutional neural networks are a special class concerning feedforward (FNN) neural networks, where a FNN is an artificial neural network wherein connections between the nodes do not form a cycle: the information moves in only one direction, from the input through the hidden layers to the output. In its general form a neural net consists of a list of layers that transform the input volume into an output volume. The layers are divided into input layer, hidden layers and an output layer. Each layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function. Typically, the input of a CNN is an image, the output is a vector defining the class to which the image belongs. The hidden layers are composed by the convolutional layers. Each convolutional layer is in turn composed of 3 stages:

- **Convolution stage:** the convolution operation is defined as a translation of a matrix, called kernel, to the input matrix and represents a filter. The resulting matrix is generally smaller than the input matrix;
- **Detector stage:** layer will apply an elementwise non-linear activation function, such as $ReLU$;
- **Pooling stage:** layer will perform a downsampling operation along the spatial dimensions;

As stated, in a convolutional neural network, the hidden layers include layers that perform convolutions. This is typically done by a layer that performs a dot product of the convolution kernel with the layer's input matrix. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. The input is originally an image (tensor) with a shape: *(input height)* $\times$ *(input width)* $\times$ *(input channels)*. After passing through a convolutional layer, the image becomes abstracted to a feature map with shape: *(feature map height)* $\times$ *(feature map width)* $\times$ *(feature map channels)*. In a neural network, the number of parameters to tune grows rapidly with the increase of the number of

layers (just think of a fully connected sequential net, if we have to connect n units of a layer with m units of the following one, we are adding $n \times m$ parameters to the model each time). Furthermore, in convolutional layers a parameter sharing scheme is used to control the number of free parameters. The result of convolution operations is an activation map, and the set of activation maps for each different filter are stacked together along the depth dimension to produce the output volume. To solve the image classification problem we will define two different models that share the same structure. The two models will be two CNN and they will differ from the structure.

2.3.1 Model A

In general, the structure of model A is composed of kernels that increase in layer after layer quantities. In particular they have medium sizes that decrease layer after layer. For the stride is worth a similar speech: it starts with a medium-sized stride that gradually decreases in size. As activation function is used Relu, as optimizer is used Adam and as pooling stage the Maximum Pooling is used. Finally, the network has two dense layers of connection. In the figure 4 is shown its structure. As

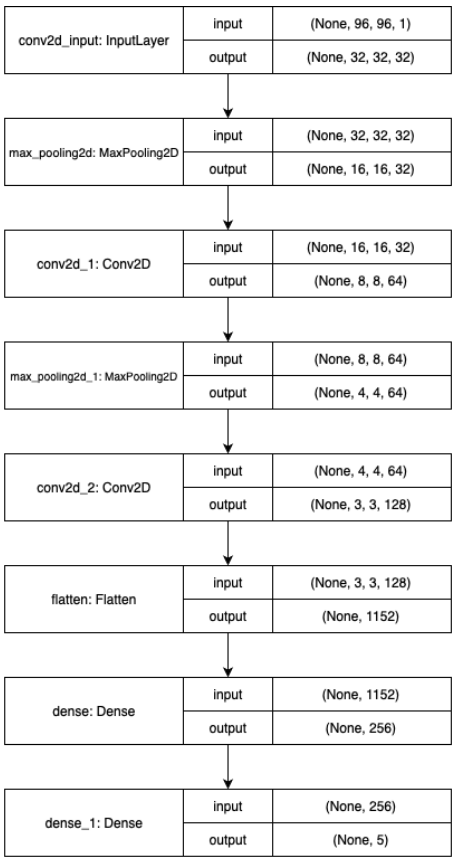

Figure 4: Architecture of the first model

shown in the figure 4 we have 3 convolutional layers (all of them have $ReLU$ as activation function):

- **conv2d_input:** this layer is composed by 32 kernels of dimension 5×5 with stride 3×3 and same padding[1];
- **conv2d_1:** this layer is composed by 64 kernels of dimension 3×3 with stride 2×2 and same padding;
- **conv2d_2:** this layer is composed by 128 kernels of dimension 2×2 with stride 2×2 without padding;

[1] **padding** represents the size of the outer edge applied to the input image; if 'same', padding is added so that the output has the same size as the original input.

Then there are 2 MaxPooling layers of 2×2 filters of stride 2 and in the end there are 2 FC (fully connected layers) followed by a *softmax* for output. Finally, we compile the model using Adam as optimizer[2] with 0.001 as learning rate. In any case, to avoid overfitting, some regularization techniques have been implemented, such as:

- **Batch Normalization:** Normalize the output of each layer so that it has a zero average and a unit standard deviation during training;
- **Dropout**: During training, it randomly "deactivates" some neurons, forcing the network to develop more robust representations. In our case we have chosen 0.8 as rate dropout;
- **Weight decay:** Called also weight regularization, it is a technique that introduces a penalty to the weights of the model during training, helping to limit excessive growth. In practice, a penalty is added to the cost function during optimization. The term regularization of weights can be implemented in different forms, such as regularization $L1$ or $L2$. Regularization L1 adds the absolute sum of weights, while regularization L2 adds the sum of squares of weights. The addition of this penalty encourages the network to use smaller weights, thus reducing the complexity of the model and improving its generalization ability on unseen data. In our case we have implemented the regularization $L2$ (ridge) with rate equal to 0.001;

This model produces:

```
Total params: 349,573
Trainable params: 349,125
Non-trainable params: 448
```

2.3.2 Model B

In general, the structure of model B is composed of kernels that increase in layer after layer quantities. In particular they start from large sizes that decrease layer after layer. For the stride is a similar matter: it starts with a large stride that gradually decreases in size. The Relu is used as an activation function, SGD is used as an optimizer and Average Pooling is used as a pooling stage. Finally, the network has two dense layers of connection. In the figure 5 is shown its structure. As shown in the figure 5 we have 3 convolutional layers (all of them have $ReLU$ as activation function):

- **conv2d_input:** this layer is composed by 32 kernels of dimension 11×11 with stride 5×5 and without padding;
- **conv2d_1:** this layer is composed by 64 kernels of dimension 5×5 with stride 1×1 and without padding;
- **conv2d_2:** this layer is composed by 128 kernels of dimension 3×3 with stride 1×1 and same padding;

Then there are 2 AvgPooling layers of 2×2 filters of stride 2 and in the end there are 2 FC (fully connected layers) followed by a *softmax* for output. Finally, we compile the model using SGD as optimizer with 0.001 as learning rate. In any case, to avoid overfitting, we have implemented the same regularization techniques implemented for model A:

- **Batch Normalization:** Normalize the output of each layer so that it has a zero average and a unit standard deviation during training;
- **Dropout**: rate dropout equal to 0.5;
- **Weight decay:** for model B was implemented the regularization $L2$ with rate equal to 0.001;

This model produces:

[2]When building a model in a deep learning library like Keras, the optimizer is one of the key components that defines how the model will be trained. The optimizer is responsible for updating the model weights so as to reduce the cost function during training. In simpler terms, the optimizer determines how the model learns from the data during the training process. The choice of the optimizer can affect the convergence rate of the model, the ability of the model to exit from local minima, and its ability to generalize to new data.

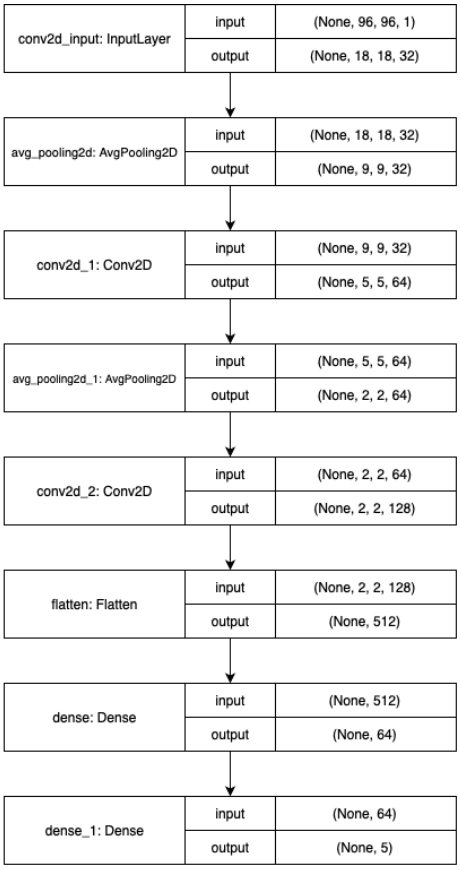

Figure 5: Architecture of the second model

```
Total params: 163,077
Trainable params: 162,629
Non-trainable params: 448
```

2.4 Results

In this section we are going to explore the results reached by the two models. The results are evaluated in terms of driving performance of the car and in terms of accuracy, f1 score, precision and recall. In order to evaluate the two models we have to fit they. The fitting is made by the function `model.fit()` where `model` is a `sequential` object of the `keras.model` library. The fit function is executed by setting 50 epochs, 128 batch size and by putting two techniques. One of them is useful for avoiding overfitting and the other is useful since we are working with an unbalanced training set. They are:

- **Early stopping:** it's a technique used when training machine learning models to avoid overfitting. Its basic idea is to stop training when model performance on a validation set stops improving. In practice, the model is continuously evaluated on a validation data set during training, and if its performance does not improve for a number of consecutive epochs, training is interrupted. This technique is useful to prevent the trained model from learning the training data too well, adapting to noises or peculiarities specific to that data set but losing the ability to generalize to new unseen data. Early stopping helps to find the optimal point where the model has the best performance on validation data. In our case we have implemented the early stopping with 10 as patience and it is started after the first 15 epochs;
- **Class weight:** it's a technique used when working with unbalanced datasets, in which some classes have significantly more or fewer examples than other classes. In this context, assigning

different weights to classes during training can help the model to give greater importance to the less represented classes. When class weight is used, a different weight can be assigned to each class when filling in the model. For example, if a class has fewer examples, its weight can be increased, while for a class with more examples the weight can be reduced. This allows the model to "pay more attention" to the less represented classes during training. The use of class weight is useful when you want to ensure that the model is not dominated by the majority classes and that it is able to generalize well even on less represented classes. In our case we have chosen the weights shown in the figure 6;

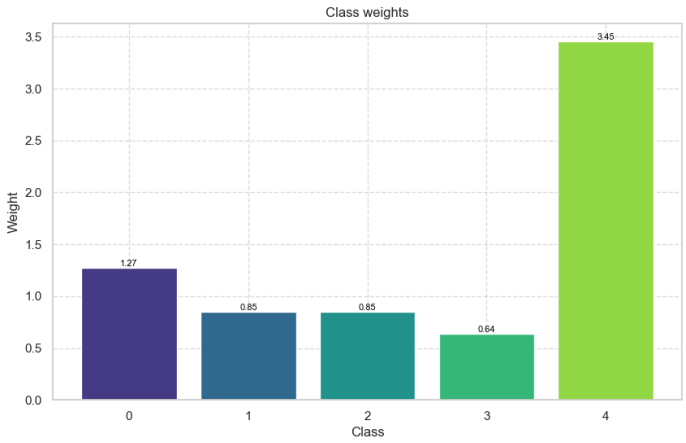

Figure 6: Class weights

2.4.1 Model A

For fitting the first model we need of 48.3 seconds. The results are evaluated using the cross entropy loss function and accuracy. The results are shown in figure 7. In particular, for the first model we

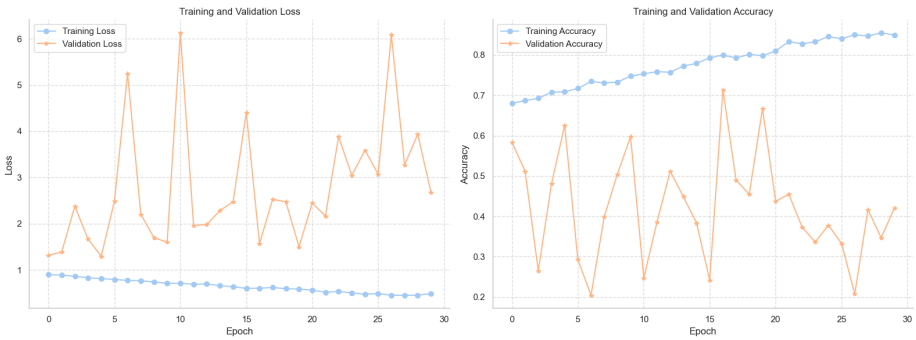

Figure 7: Test and Loss accuracy for model A

have:

```
Test loss: 1.4987
Test accuracy: 0.6668
```

It can be noted that this model is affected by overfitting despite the various regularization techniques applied. This can be seen by observing the trend of the accuracy curves on the training set and test set. In particular, you can see that the accuracy curve on the training set increases without the test curve actually converging. This means that the model has learned and adapted too well to the training data, but struggles to generalize well to new unseen data. To heal our model from overfitting one might consider to apply data augmentation on training set, in order to expand the dataset and improve the model generalization. Another idea might be to reduce the learning rate to

make optimization more stable and allow the model to converge better however without exaggerating otherwise the model might suffer from underfitting[3]. Otherwise you might think of adding more dropout layers and/or batch normalization layers in the model. The classification report is:

	precision	recall	f1-score	support
0.0	0.20	0.23	0.21	133
1.0	0.31	0.48	0.38	275
2.0	0.58	0.37	0.45	406
3.0	0.79	0.80	0.80	1896
4.0	0.31	0.10	0.15	39
accuracy			0.67	2749
macro avg	0.44	0.40	0.40	2749
weighted avg	0.68	0.67	0.67	2749

Finally, in figure 8 is shown the confusion matrix of this model.

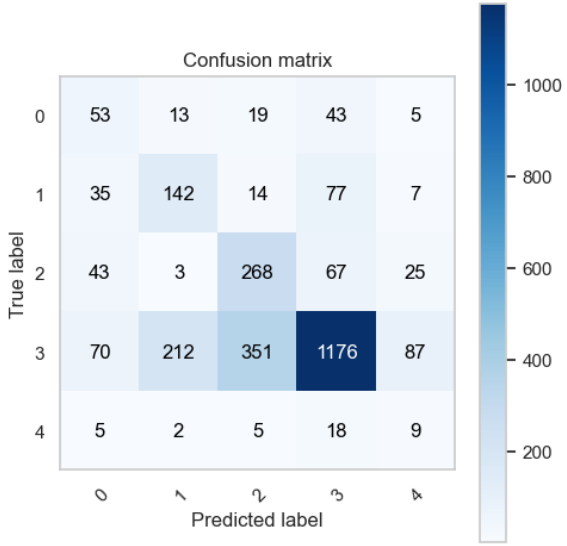

Figure 8: Confusion matrix for model A

2.4.2 Model B

For fitting the first model we need of 58.1 seconds and the results are shown in figure 9. In particular, for the first model we have:

```
Test loss: 1.1791
Test accuracy: 0.6151
```

We can see that this model is affected by overfitting less than the first. The classification report is:

	precision	recall	f1-score	support
0.0	0.26	0.35	0.30	133
1.0	0.34	0.53	0.41	275
2.0	0.43	0.68	0.53	406
3.0	0.86	0.64	0.73	1896

[3]Underfitting is a condition in which a machine learning model fails to adequately capture the complexity of training data, leading to unsatisfactory performance on both training and test data. In other words, the model is too simple to correctly represent the relationship between input and output variables in your problem.

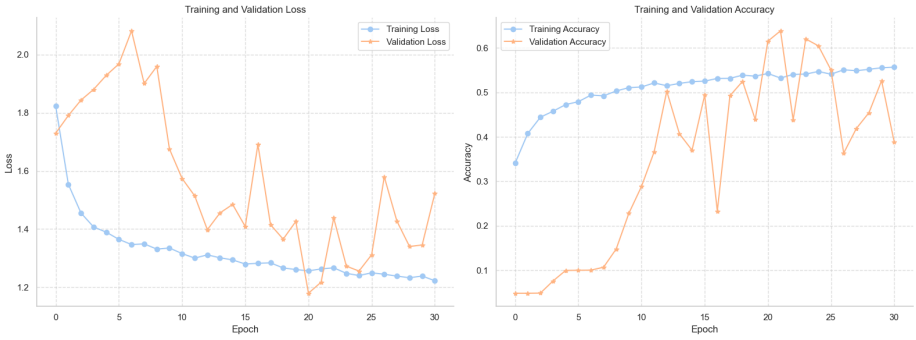

Figure 9: Test and Loss accuracy for model B

4.0	0.11	0.23	0.15	39
accuracy			0.62	2749
macro avg	0.40	0.48	0.42	2749
weighted avg	0.70	0.62	0.64	2749

Finally, in figure 10 is shown the confusion matrix of this model.

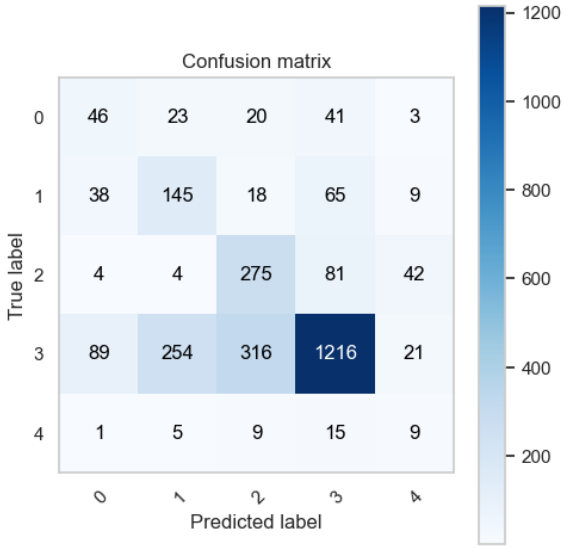

Figure 10: Confusion matrix for model B

3 Results and conclusions

In order to solve the problem assigned to us, we built two CNN with slightly different structures, trying to highlight how the size of the kernels, their stride and the presence or not of padding can affect the results obtainable. In particular, we have achieved a result that is acceptable to both networks. However, the first model I think can drive the car slightly better (as shown in the video). That is why we have chosen to examine the first model in greater depth. In particular, an investigation was conducted of how the profile of the test accuracy and test loss varies with the weight decay parameter. This survey is shown in the figure 11. In figure 12 we monitored the average values of precision, recall and f1-score of the model. In particular we can notice that choosing the weight decay rate of 0.001 is not a bad choice; it is with this value that we can get the best values of accuracy and loss on the test set. In any case, both models can offer excellent performance in

terms of driving and this is probably due to their similar structure. In fact, both models have large kernels in the first layers that decrease in size layer by layer. Probably this allows models to capture large-scale image characteristics (such as track or grass recognition).

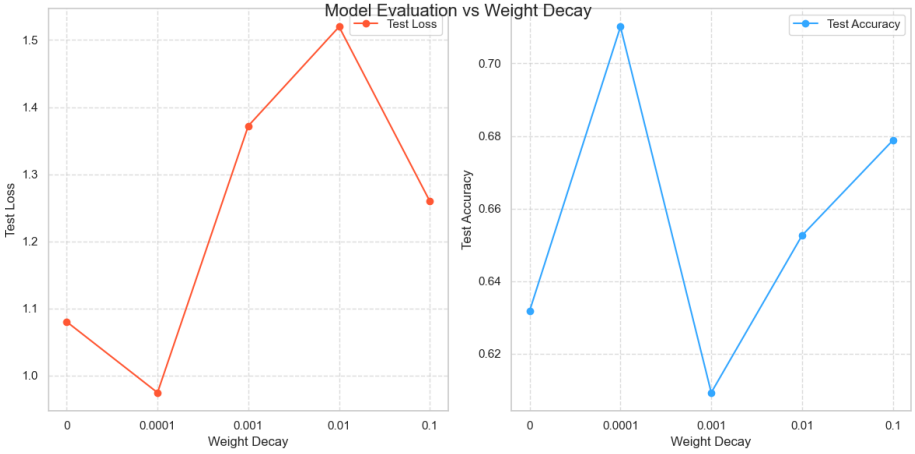

Figure 11: Test loss and test accuracy of model A

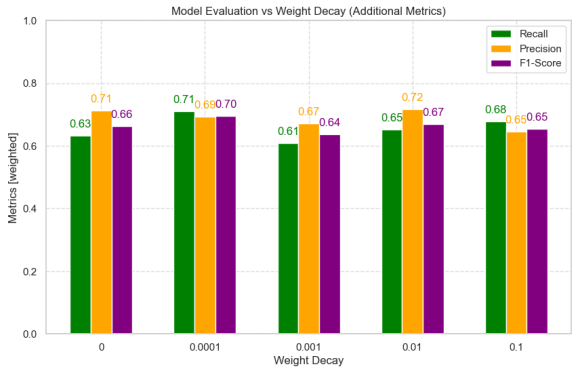

Figure 12: Precision, F1 score and recall of model A