Soccer Event Detection using Deep Learning

Marwan Agourram

Department of Computer Science, LM Data Science, Via Celoria 18, Milan, 20133, Italy.

Contributing authors: marwan.agourram@studenti.unimi.it;

Abstract

Event detection is an important machine learning task that is gaining academic attention due to its relevant applications. In this report, I proceed to perform image classification using deep learning techniques, in particular, Convolutional Neural Networks (CNN), which have been proved to perform best for pursued task. The main source of images used in this report consists of image frames of several football matches.
This report draws its inspiration from the paper "Soccer event detection using deep learning", Karimi, A., Toosi, R., & Akhaee, M. A. (2021)*.

Keywords: Soccer event detection, Deep learning, Convolutional Neural Networks

*arXiv preprint arXiv:2102.04331

Contents

1 Introduction **3**

2 Dataset **4**

3 Convolutional Neural Networks **4**
3.1 Models 5

4 Methodology **7**

5 Results **8**
5.1 Model 1 8
5.2 Model 2 9
5.3 Model 3 9
5.4 Model MobileNetV2 9

6 Conclusion **9**

7 Figures **11**

List of Figures

1 Images from the dataset after resize operation. 11
2 Plots of the performance related to `model_1`. 12
3 Plots of the performance related to `model_2`. 13
4 Plots of the performance related to `model_3`. 14
5 Plots of the performance related to `model_mbnv2`. 15

1 Introduction

Football is known to be one of the most important sports in human history. It is played across the whole globe mainly due to the fact that it requires little effort in order to set up the proper game. However, this simplicity has been replaced in certain areas of this sport, in particular in the professional field. Big football clubs compete on the pitch and outside the pitch to achieve the best possible performance, both in terms of sport and in terms of revenue. In fact, it is estimated that approximately 47 billions of Euro are generated across the whole world, mainly coming from direct derivations of the sport (i.e., merchandising, tickets), but also from other sources that are linked to it (i.e., betting).
In particular, almost every professional team has introduced in the technical staff a new figure, the match analyst, whose main task is to help the team manager develop new strategies that cannot be easily understood directly from the game itself. In fact, this technical role works mainly with data generated from the match in order to perform his analysis. This kind of aspects highlights how important is nowadays to develop new techniques that allows clubs to perform better.

In this report I proceeded to implement different convolutional neural network in order to classify images extracted from several videos of football matches. These image frames represents different aspects of the game, such as a free-kick, penalties or referees showing a yellow or red card. The goal of this report is to correctly classify this images. This task, which now can be handled with machine learning tools, was once performed by human annotators. It is understood how important it became to correctly develop models that can classify images in (almost) real-time. Moreover, this task can also be seen as an important step towards innovation: in the last years football federations introduced the VAR[1] system, which consist of two additional referees who controls monitors in order to help the game referee in depicting important events. Image classification performed with the appropriate infrastructure and the appropriate models can be implemented in real-time game, enhancing the transition of football towards an innovative approach[2].

The report is structured as follows: in Section 2 I provide a short explanation on the structure of the dataset used; in Section 3 I provide an overview about Convolutional Neural Networks, the main layers that are used and other characterizing features about the CNNs, focusing on describing the different CNN models used; in Section 4 I provide an explanation on the implemented procedure; in Section 5 I report the obtained results from the implemented models; Conclusions and a brief discussion is reported in 6.

[1] Video Assistant Referee: a match official in association football who assists the referee by reviewing decisions using video footage and providing advice to the referee based on those reviews.
[2] This is a personal consideration of the author.

2 Dataset

This report is based on data from the "Soccer Event Dataset (Image)"[3] dataset, which contains images taken from UEFA Champions League (UCL) and UEFA European League (EL) football matches. These images were collected through:

- Web crawling and collection of images related to events.
- Watching the videos of UCL and EL football matches to extract the relevant frames.

The collected images regards mainly eight events (corner kick, penalty, free-kick, red-card, yellow-card, tackle, substitution, and cards), with the addition of other three events that consists of frames depicting different camera shots (left penalty area, right penalty area, center circle).

All of these images comes in two different folders that serves two different tasks: Train folder, which contains 60.500 images[4], and Test folder, which contains 5.500 images[5]. Moreover, both the aforementioned folders comes with different zip-folders, each one named after the depicted event.

Finally, the dataset is balanced since each category is well represented by the same number of images both in the training and in the testing data. As we can see from Figure 1, there is a (recurrent) element throughout most of the images, that is the green pitch, as well as various subjects (players, referee, assistants) that are performing some actions that are related (directly, or indirectly) to the labelled event. Some of the pictures turns out to be difficult to classify even for a human, since it is not possible to depict the event by observing it directly. For example, the event 'To Substitute' can be referred to images representing the fourth official handling the substitution panel, or again, the event 'Yellow-Card' can be represented by images of the referee showing the card to the player, which however is masqueraded by a yellow t-shirt worn by some fan in the stands. This inherent problem increases the difficulty of the task, since there is the need to correctly depict the most important features that defines a certain event.

3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are considered to be one of the most efficient deep learning techniques when it comes to event detection and, in particular, for image detection. This peculiarity arises from the structure of these models, which derives from regular neural networks, and achieves best results with particular types of data, such as images.
CNN are feed-forward neural networks, that is, the flow is uni-directional (forward) from the input layers towards the output layer. The architecture consists of the following layers:

- Input layer: the input layer is the starting point of the CNN. It takes the input data, which in our case, is an image.

[3] The GitHub repository is found at: http://tinyurl.com/soevd.
[4] Each of the eleven category previously described as 5.500 images each.
[5] Each of the eleven category previously described as 500 images each.

- Hidden layer(s): hidden layers contains one or more convolutional layers. These layers perform convolutional operations in order to extract the main features from the input. Filters (i.e., kernels) are also applied to detect other relevant patterns, such as objects or textures. Finally, after each convolutional operations, an activation function (such as ReLU[6] in our case) is included in order to learn complex relationships by introducing non-linearity.
- Pooling layer(s): pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. In our case *Max Pooling*[7] is used, with a pooling size of (2,2).
- Flattening layer: after the last convolutional layer, a flattening operation is performed on the output in order to reduce the dimension of the data to a one-dimensional vector of a specified size.
- Fully connected layer: this layer connects every neuron in one layer to every neuron in another layer.
- Output layer: finally, the CNN provides the output which has the same size as the number of categories (in our case, the different events) in the input data.

Each CNN model goes through a `compile` stage, which consists in configuring the model for the training phase by specifying additional settings that are necessary during this stage. Some of the most important aspects to consider in this stage are:

- Optimizer Selection: it adjust the weights of the network based on the computed gradients during back-propagation[8]. In this report, the optimizer used are `Adam`[9] and `RMSprop`[10].
- Loss Function: it is a measure of how well the network is performing. During training, the goal is to minimize this function. Since the task is to classify images, the chosed loss function for this report is the `sparse_categorical_crossentropy`.
- Metrics: metrics are used to evaluate the performance of the model during training and testing. Common metrics for classification tasks include accuracy, precision, recall, and F1-score. In this report, accuracy is the metrics that is used across all models.

3.1 Models

In order to perform the task, four different CNN models have been implemented. The details of each of the proposed models are reported in this subsection.

Model 1

The first model (`model_1`) has five convolutional layers. The first convolutional layer has filter size of 32 and a kernel size of (3,3), with the activation function being the ReLU. The second layer has 64 filters, the third and the fourth layers have 128 filters,

[6] Rectified Linear Unit.
[7] Max Pooling uses the maximum value of each local cluster (in our case, of size $(2, 2)$) of neurons in the feature map.
[8] Back-propagation is a gradient estimation method used to train neural network models. The gradient estimate is used by the optimization algorithm to compute the network parameter updates.
[9] Adaptive Moment Estimation.
[10] Root Mean Square Propagation.

and the fifth convolutional layer has 256 filters, with the same kernel size and activation function used across all of the convolutional layers. Each convolutional layer has a Max Pooling layer that operates with a pool size of (2,2).
After the fifth convolutional layer, a flattening layer and the first dense layer (which has 256 neurons and ReLU activation function). Finally, a dropout layer (with a dropout size of 0.5) is followed by the second dense layer, which have the same number of neurons as the number of categories in the input data, and uses the SoftMax[11] activation function, which returns the probability associated with the input data being in a specific class.
The model is then compiled using the RMSprop optimization function, accuracy as the metric and the 'sparse categorical cross-entropy' as the loss function.

Model 2

The second model (`model_2`) maintains the same layering structure as the previous model. However, there are little changes in the filters size: from the first to the fifth convolutional layer the filters are in order 64, 128, 128, 256, 512. The activation functions and the kernel size remains unchanged with respect to the previous model. Consequently to the reported changes, the first dense layer has now 512 neurons.
The model is compiled exactly as for `model_1`.

Model 3

The third model (`model_3`) has various changes with respect to the previous two models. Three convolutional layers are used, with filter size being (in order): 64, 128, 128. Each of the three convolutional layers has a Max Pooling with pool size (2,2). After the last convolutional layer, a flattening layers is followed by two dense layers, the first of them having 512 neurons, while the second having the same amount of neurons as the categories in the input data.
The major implementations introduced in this model are:

- Introduction of strides[12]: in the first convolutional layer, a stride of size (2,2) is introduced. This parameter allows the convolutional operations to 'skip' a number of pixels in both vertical and horizontal direction as specified by its dimension, in this case (2,2).
- The importance of the Flatten layer followed by a larger Dense layer: in this model, the flatten layers plays a crucial role, since it serves to reduce the spatial structure after the last convolutional layer before the output. The difference in the output provided by the last convolutional layer (128) and the input fixed for the first dense layer (512) allows to create a more densely connected layer, which is capable to capture more complex patterns in the data.

[11] The function representing the SoftMax activation function is:

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{N} \exp(y_i)}.$$

[12] Stride is a parameter that dictates the movement of the kernel, or filter, across the input data. When performing a convolution operation, the stride determines how many units the filter shifts at each step. This shift can be horizontal, vertical, or both, depending on the stride's configuration. This can lead to a faster reduction in spatial resolution compared to using the default stride of (1, 1).

- Introduction of the Adam optimiser: the last difference resides in the different optimization function introduced in the compiling stage. Compared to the previous models, `model_3` compiles optimizing with respect to the Adam algorithm which shares different similarities with the RMSprop algorithm. However, differences can be found with respect to key aspects such as update rules, bias correction and hyper-parameters.

MobileNetV2 Model

MobileNetV2 (`model_mbvn2`) is a convolutional neural network architecture that seeks to perform well on mobile devices. It is designed for efficient and lightweight deep learning models, especially targeted at devices with limited computational resources. Initially, a MobileNetV2 pre-train model is introduced, loading pre-trained weights on the ImageNet dataset. From the parameters of this pre-trained model, it is excluded the final fully connected layer, which it will be later added. An important aspect is that of freezing layers, which prevents weights from being updated throughout the training phase.
Then, a `GlobalAveragePooling` is added on top of the pre-trained model in order to reduce the spatial dimension of the base model output to a one-dimensional vector, obtained by averaging all the values. Finally, a dense layers with 256 neurons and ReLU activation function is followed by a dropout layer with rate 0.5.
The compilation of the model is the same as the one used in the first two models.

4 Methodology

As introduced in Section 2, the dataset comes in two different folders which contains eleven zipped file each representing the class of images contained in the files. The first operation was to extract the zipped files into other writable folders.
Then, I proceeded by inspecting the two folders '`train_extracted`' and '`test_extracted`' and, after noticing a subtle difference in the name of the `to substitute` between the two folders, I changed the name of the one in the test set.
Then, a brief exploration of the images was performed, plotting an image from each of the different categories from both the extracted folders. Finally, a counting of the images contained in each sub-folder was carried out.
The second phase consisted in creating tensors from the image data that are later used as input for the CNN models. This task was performed both for the train and the test data. Moreover, the train data is also used to obtain a sample for validating the CNN models. The tensors are obtained by running the function `resize_images`, which goes through the specified folder path and transforms each PIL image in an array, after having it resized into a (120, 120) pixel dimension. Each image data is then appended to a related variable `#_img` and also the corresponding class of the image data is appended to the related variable `#_lbl`. This is done for both the train and the test data.
After having extracted the arrays from the images, I proceeded to perform a `train_test_split` operation on the train images, in order to obtain data for the training and for the validation phase. The splitting ratio is 0.09, which allows to have a validation set of approximately 5.500 images, that is, 500 images in each class.

On the training data, an operation of data augmentation is performed with the parameters reported in Table 1:

Parameter	Value
`rotation_range`	20
`width_shift_range`	0.2
`height_shift_range`	0.2
`shear_range`	0.2
`zoom_range`	0.2
`horizontal_flip`	`True`
`rescale`	1./255

Table 1: Parameters for Data Augmentation on Training data.

Finally, the training image data is loaded in memory, generating batches (`batch_size = 32`) of augmented data.
The validation and the test data are only re-scaled, using the same scaling factor as for the training data, and then both loaded in memory.

5 Results

The four CNN models described in Section 3 have been trained for 10 epochs. Then, for each model, a plot for the training and validation loss and one for the training and validation accuracy is displayed. Finally, on the test dataset, a confusion matrix is computed as well as three different metrics: Accuracy, F1 score, and Recall score.

5.1 Model 1

The first CNN model results are reported in Figure 2. Starting from the loss (2a), the model shows an initial high value for the loss which is followed by a constant decrease for both training and validation data, therefore highlighting the improvement of the performance. However, from the 8th epoch, there is an increase in the loss that peaks for the validation at the 9th epoch. This phenomenon is considered to be related with overfitting phenomena, which is confirmed by the important increase in the validation loss in the 9th epoch that is not properly matched by the one for the training data.
For the accuracy (2b), we can confirm the overfitting phenomena, since the validation accuracy is higher than the one for training. However, at the 9th epoch, the accuracy for validation drops.
Finally, as shown in the confusion matrix in Figure 2c, we can see that the model has some difficulties when it comes to distinguish images labelled as 'Center', wrongly assigning them the class 'Right'. On the other hand, comparing the various results obtained, we can see that the only value in the main diagonal[13] that stands out in terms of 'good' performance is the one associated with the class 'Tackle'.

[13]The values in the main diagonal of the confusion matrix represents the portion of images that are correctly classified.

5.2 Model 2

The performance of the second CNN model are shown in Figure 3. The loss (3a) for the training shows an initial decrease, which is then followed by an inversion of the trend. The validation loss initially follows the same pattern as the training, but around the 4th epoch shows an irregular pattern that ends with an increase. The accuracy (3b) shows that the model performs well in the first four epochs, and then for the training it decreases to the initial value. Once again, the model presents overfitting, since the validation accuracy is always bigger than the one for the training.
Finally, from the confusion matrix in Figure 3c, the model confirms a worst performance compared to `model_1`. In particular, different images have been classified either to be 'Right', 'Corner', or 'Red Cards'. In particular, we can see how different 'Penalty' images have been classified as 'Right' images. Moreover, the model is not capable to detect patterns that classifies images 'Cards', 'Corner', 'Left', or 'Penalty.

5.3 Model 3

The third CNN model shows an increase in the performance compared to the two previously mentioned models (see Figure 4). Starting from the loss (4a), the model seems to improve its performance. In particular, the training loss shows a smooth declining curve, while the validation loss follows (approximately) the same pattern. The accuracy of the model matches the decrease in the loss, since both training and validation increases the accuracy of `model_3`. However, once again, there is still room for overfitting.
Finally, the confusion matrix in Figure 4c shows an overall good balance in classifying images, except from the fact that 'Cards' and 'Tackle' classes are not properly detected by the model, which instead tends to classify the majority of the images as 'Yellow-cards'.

5.4 Model MobileNetV2

The last model implements a pre-trained model of `MobileNetV2`. The model results are shown in Figure 5. Starting from the loss (5a), the model exhibits an initial decrease in the training loss, which is not matched by the validation one, which instead shows an increase. The accuracy of the model (5b) highlights the presence of overfitting. Both the loss and the accuracy shows a flattening of the performance after the second epoch, highlighting the inefficiency of the proposed model. This could be due the architecture that was implemented (see Section 3.1), since the freezing layer do not allows weights to be updated after each epoch.
The confusion matrix (5c) highlights the poor performance of this model, which is capable of classifying images mainly as 'Left' or as 'Yellow-cards'. This could also be justified by the inability in detecting the proper patterns in the input data.

6 Conclusion

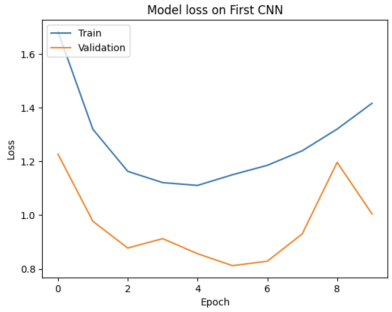
This report focused on understanding some of the possible techniques that could be implemented in order to perform image classification. As specified by the initial

purpose, the main technique used across the report consisted in Convolutional Neural Networks, which are one of the most important tools in deep learning field to perform this specific task.
Four different models were implemented and evaluated in the previous sections: `model_1`, which is to be considered as the starting point, allows to understand the challenges that should be faced when performing image classification on a large dataset. `model_2` tries to increase the performance obtained by the first model by proposing a different structure, which however did not return better results. `model_3` is to be considered as the best model in this report, due to the capacity to learn across epochs by following a smooth pattern. However, it is still far from being considered as a 'well-performing' model, since it did not solve the issue related to overfitting which is recurrent in all of the proposed models. Finally, a pre-trained model is implemented, which did not provide acceptable results.

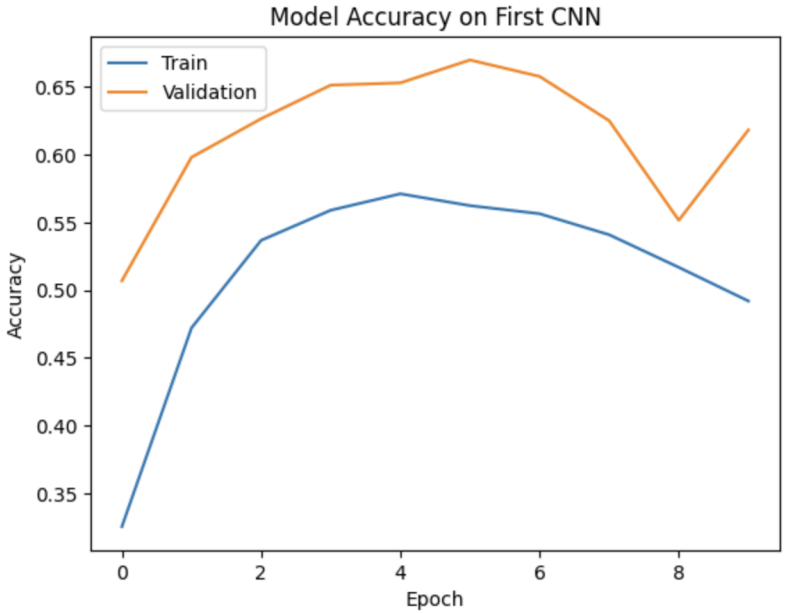
The proposed models could surely provide more accurate results if some subtle changes were to be introduced: first of all, an appropriate reduction on the dimension of the original dataset could be considered in order to reduce the computational power when it comes to train the models, therefore allowing to increment the complexity of the CNNs. Secondly, a different approach in the pre-processing of the images or different parameters for the data augmentation stage could be considered in order to provide better results.

7 Figures

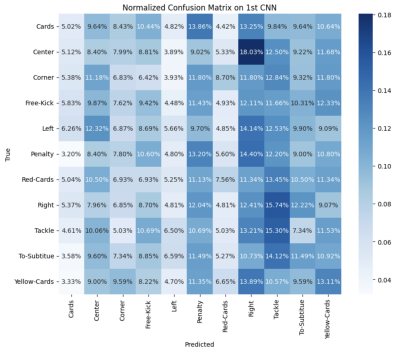
Fig. 1: Images from the dataset after resize operation.



(a) Loss

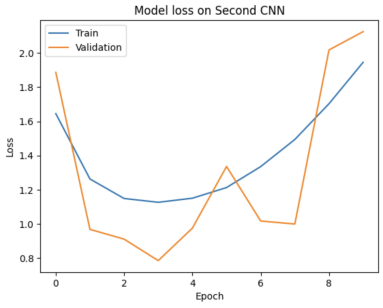


(b) Accuracy

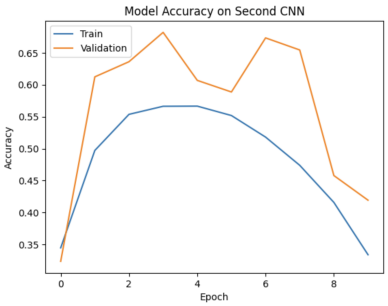


(c) Correlation Matrix (Normalized)

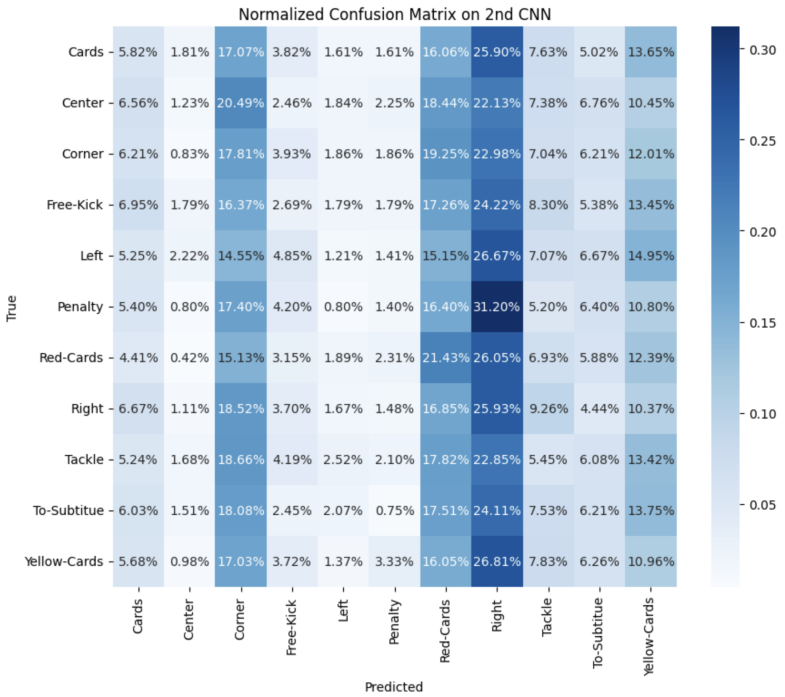
Fig. 2: Plots of the performance related to `model_1`.



(a) Loss

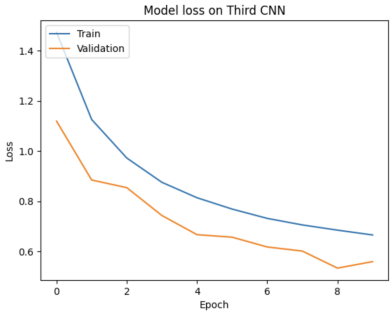


(b) Accuracy

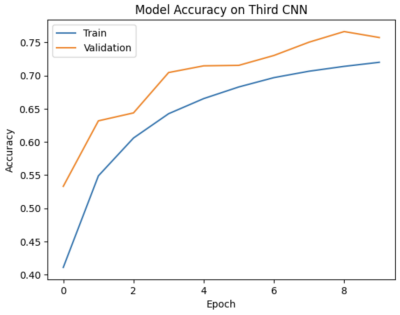


(c) Correlation Matrix (Normalized)

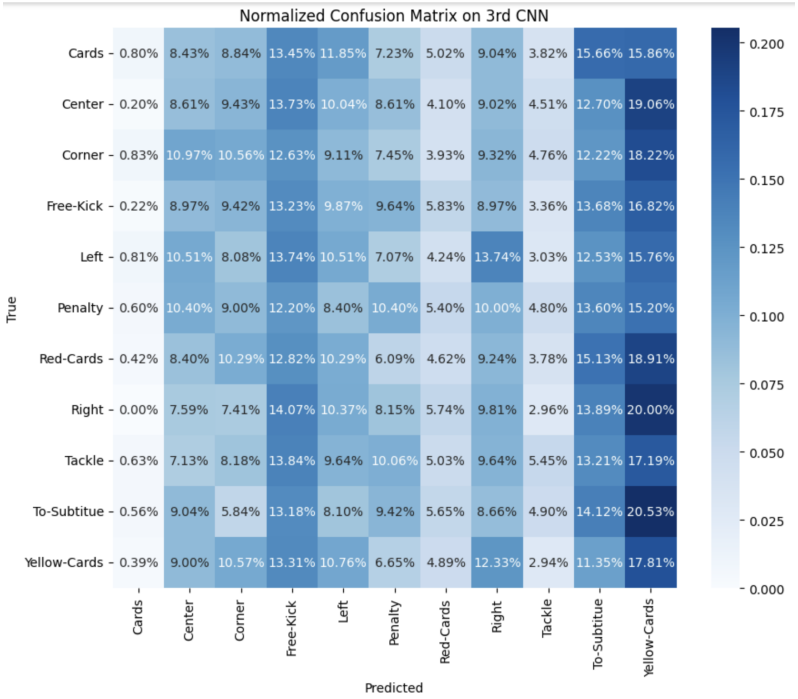
Fig. 3: Plots of the performance related to `model_2`.



(a) Loss

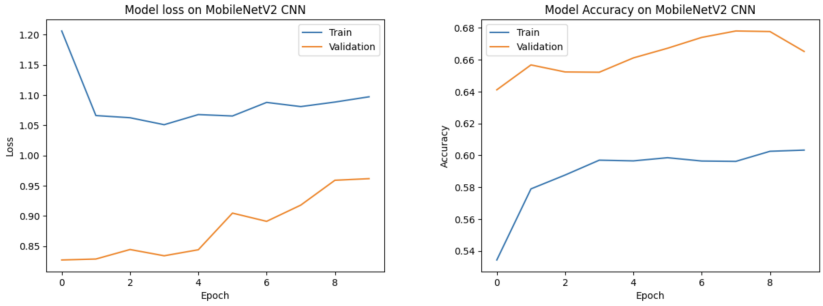


(b) Accuracy

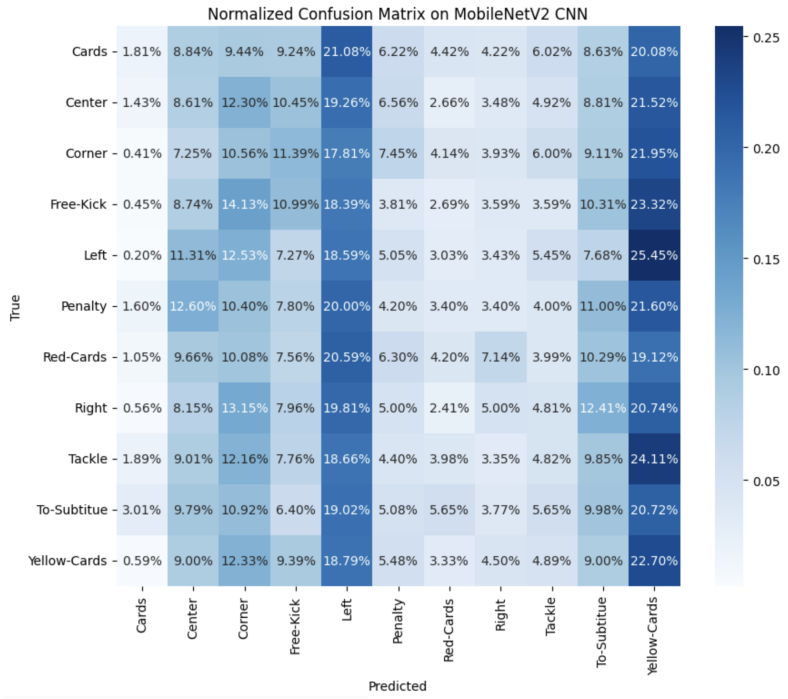


(c) Correlation Matrix (Normalized)

Fig. 4: Plots of the performance related to `model_3`.



(a) Loss (b) Accuracy



(c) Correlation Matrix (Normalized)

Fig. 5: Plots of the performance related to `model_mbnv2`.