Deep Learning - Group 2

André Dias, m20210668

André Pereira, m20210690

Guilherme Simões, m20211003

João Veloso, m20210696

Mariana Ramalho, m20210687

Counting Playing Cards Algorithm: An Image Recognition Approach

Identifying suits in playing cards with CNNs

Professor Mauro Castelli

Masters in Data Science and Advanced Analytics
NOVA IMS - Information Management School

1. Introduction

This paper presents an approach to identify suits of playing cards. After careful analysis, we noted many datasets already available for this project relied heavily on optimal camera positioning and on a never changing environment. We intend to introduce a more realistic and useful approach, as we try to identify the cards suits, regardless of the different camera angles, image backgrounds and if the cards are completely visible or might have been cropped. To do that, we will apply a **Convolutional Neural Network (CNN)** to a multi-class classification problem to recognize the images in a dataset built by us. Later, we will analyze the results and implement a counting cards prevention algorithm. The project was entirely made in *Google Colab* written in Python language.

Keywords: cards, suits, image recognition, cnn, accuracy, model, data algorithm.

2. Problem Definition and Algorithm

2.1 Task Definition

Identifying the suit of the card is the first step in the problem of recognizing playing cards. Although we have no difficulty identifying suits, we lack the capability to memorize the cards already played during several games. Our idea would eliminate the need to memorize the cards and could also provide useful probabilities of the cards yet to be played. Application examples of this project could be any type of card games played in casinos. So our final goal is to create a counting cards algorithm that provides a probability if a certain person is cheating or not.

Due to computational costs associated with the task in hand and the lack of data, we decided to only identify the suits on the cards, therefore every card will fall into one of 4 categories: **Clubs, Diamonds, Spades or Hearts.**

2.2 Approach

Our approach to this project started by **Preprocessing** the dataset, which was built specifically for it. When the dataset was in an acceptable format for the network, we moved onto the next step: **Model Assessment**. This was where we added extra complexity to a structured model similar to the ones built in class, by introducing more and different layers. **Parameter Tuning** was performed in an effort to find an optimal set of parameters with the goal of improving accuracy over the base model. Moving forward, we performed **Transfer Learning,** where a pre-trained model was used with our dataset. In the final step of our project, we compared all the models used, chose the one that best suited our dataset and finally fitted it with our **test set**.

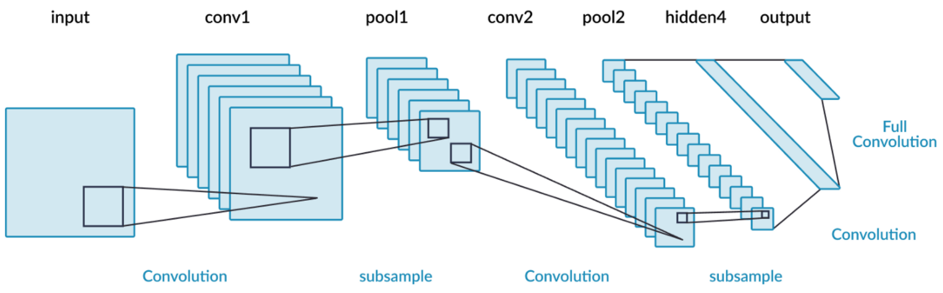


Figure 1. Example of a Convolutional Neural Network

3. Experimental Evaluation

3.1 Methodology

The choice of our final model was based mainly on **Accuracy** values. It's important to have a high value of this metric since we are creating a precise detection system for counting cards and the number in the card will later influence the decision making process of the algorithm. We also took into consideration the **F1-score**, along with the plotted confusion matrix when applying a prediction using the test set.

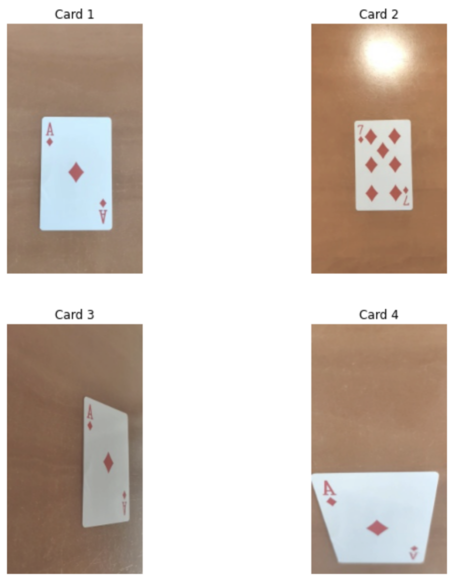
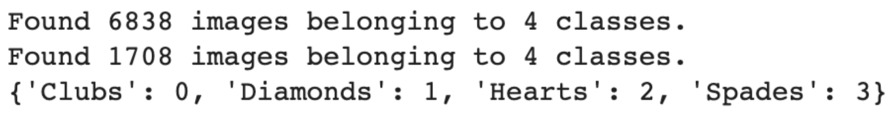


Figure 2. Four cards taken randomly from the Diamonds Suit class

Preprocessing

The full dataset is made up of **8994 original images** obtained by shooting the playing cards with different brightness and later by using a method called **Synthetic Image Generation**, where we convert a video of each card with various angles into frames in a python script and then save it in 4 different folders, according to the correspondent suit. Then, they were later resized to about 50% the original size (176, 320) with *openCV* library methods. In total we have around 2135 images for each class of suits: Hearts, Diamonds, Clubs and Spades**.** The Training consists of 80% of images, the Validation of 15% and the Test 5%.



```
Found 6838 images belonging to 4 classes.
Found 1708 images belonging to 4 classes.
{'Clubs': 0, 'Diamonds': 1, 'Hearts': 2, 'Spades': 3}
```

Figure 3. Split in Training and Validation

Data Augmentation and Callbacks

Implementing data augmentation brought variety into our dataset, since we were concerned with the amount of data that we had for the model to learn the figures patterns. That variety is introduced when we **flip, zoom, shear and so on**, making the number of total images in training not only higher, but also more complex. Two callbacks were also implemented, the **early stopping** to interrupt training if the validation loss is not improvement and the **model checkpoint** to save the weights at different points of training.

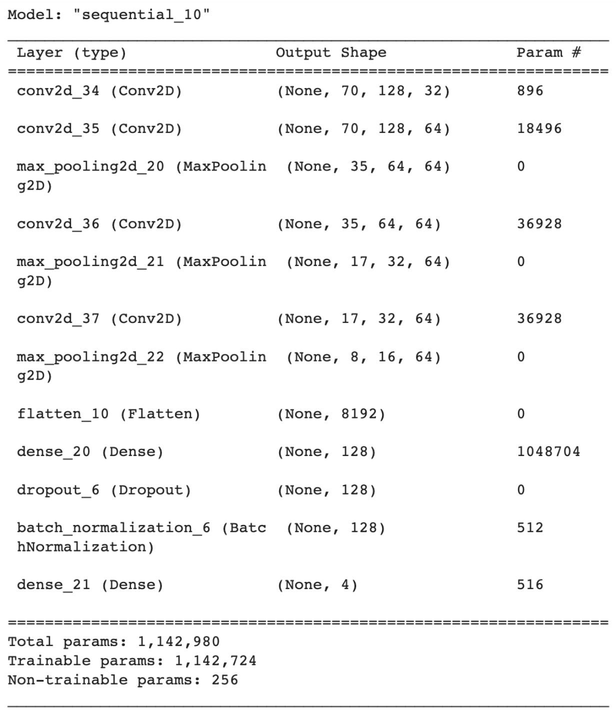
Model Assessment - Tuning Parameters

The strategy was based on increasing the CNN classifier complexity after training to improve the accuracy of our model. As we add more layers into the model and neurons in each one, the model tends to rapidly overfit. More layers will help extract more features, however we can only do that up to a certain extent. After that, instead of extracting features, we tend to **overfit the data**. We saw that when the validation error started increasing. In our specific case, instead of focusing on identifying the suit in the card, it would also memorize the number or even the details of a common background. The solution was to first decrease the number of epochs because, after a certain number, there is no reduction in training loss neither improvement in training accuracy; next, the overall complexity, meaning that we had too many layers and hidden units; after that, we created dropout layers after the Pooling layers and Batch Normalization after the Convolutional Layers.

3.2 Results

The input shape that the network receives is 70px:128px. Larger images, such as the original size (352px:640px), would give us a better performance on our main metric measure, but we would have more memory consumption and a much higher runtime. The model has 4 Convolutional Layers, 3 Pooling Layers, 1 Flatten layer and 2 Dense layers (the second one being the output layer). In them, the activation function chosen was the *relu* for the intermediate layers and *Softmax* for the last dense layer since it is used for a multi-class classification problem. The reason behind having one more convolutional layer earlier is to pick-up lower-level features (lines and edges). At the end, we have a Dropout (p=0.4) and Batch Normalisation Layers.

The chosen optimiser was the *Adam* and the loss function was the *categorical_crossentropy.* Number of epochs was decided based on training and validation error rates, meaning that for us **10 epochs** was when it started to stabilize. The **default batch size** is made of 32 images. The shuffle was put equal to “True” on the training set and “False” on the validation and test set. When setting the *shuffle=True* in the validation set, the results were very poor (f1-score = 0.25). It happens that the prediction is correct but compared to wrong indices it led to misleading results.



```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_34 (Conv2D)          (None, 70, 128, 32)       896

 conv2d_35 (Conv2D)          (None, 70, 128, 64)       18496

 max_pooling2d_20 (MaxPoolin  (None, 35, 64, 64)       0
 g2D)

 conv2d_36 (Conv2D)          (None, 35, 64, 64)        36928

 max_pooling2d_21 (MaxPoolin  (None, 17, 32, 64)       0
 g2D)

 conv2d_37 (Conv2D)          (None, 17, 32, 64)        36928

 max_pooling2d_22 (MaxPoolin  (None, 8, 16, 64)        0
 g2D)

 flatten_10 (Flatten)        (None, 8192)              0

 dense_20 (Dense)            (None, 128)               1048704

 dropout_6 (Dropout)         (None, 128)               0

 batch_normalization_6 (Batc  (None, 128)              512
 hNormalization)

 dense_21 (Dense)            (None, 4)                 516

=================================================================
Total params: 1,142,980
Trainable params: 1,142,724
Non-trainable params: 256
_________________________________________________________________
```

Figure 4. Classifier Model Final Structure Figure

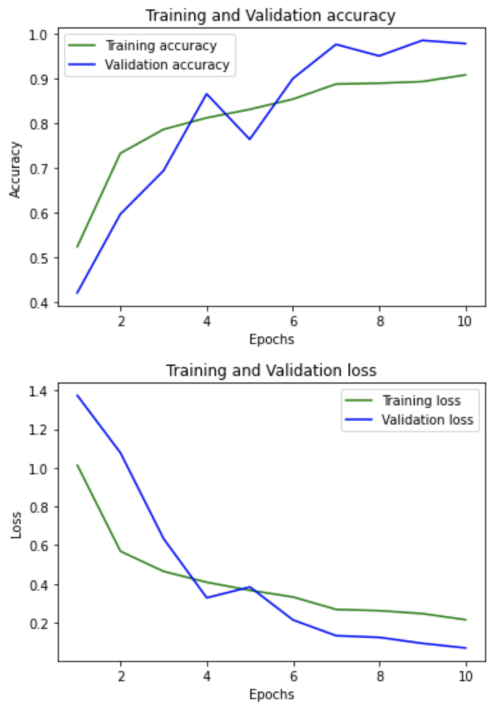


Figure 5. Plot Training and Validation Accuracy and Losso Structure

Cards Suits	Precision	Recall	F1-Score
Clubs	0.94	0.99	0.97
Diamonds	0.64	0.99	0.78
Hearts	0.98	0.38	0.55
Spades	0.99	1.00	1.00

Table 1. Confusion Matrix Final Results Test Set

Final Accuracy: 0.8408

3.3 Discussion

As stated before, our main measure of success is the model accuracy. At the beginning, the classifier was overfitting, meaning that we had a good accuracy on trained data but very less accuracy on validation due to the lack of data variety and quality. We also noticed that the validation error started to increase at a certain point.

The quantity of images available plus the quality when automatically resized, were not enough for the CNN to learn the exact figures shapes, so we decided to increase our dataset using the SIG method with not only different camera angles, but also distinct lighting conditions. Eventually, some images ended up blurred due to the camera trying to focus on the card. This happens to be something that lowered the model's final accuracy. After applying Data Augmentation in our bigger dataset, we had to **improve the complexity of our model**. That was when we started to implement layers such as the Dropout and Batch Normalization Layers: The first one randomly ignores some layers output known as nodes, meaning that these ignored nodes are not taking part in training the model; the second, normalizes the outputs of the previous layer. Both made the validation accuracy move randomly when in intermediate layers, so we decided to only keep the last two, after the Dense layer. That way we got the best optimal result.

4. Counting Cards Prevention Algorithm

The main objective here is to train a Convolutional Neural Network to recognize and classify each card of a playing card deck. Our program should look for the card’s corners, because in real conditions, cards may show partially covered and the corner of a card is the minimum information you need to identify it. At this point we could have different python scripts to handle the card identification as we desire. For this example we will focus only on a python script to count cards to get an advantage while playing Blackjack.

The process is the following: We initialize a variable count as 0, then we start counting cards. If a 2, 3, 4, 5, 6 is drawn we increase count by 1; if a 7, 8, 9 is drawn the count stays the same; if a 10, J, Q, K, A is drawn we decrease count by one. The last step is to divide the count for the number of decks remaining to get a more accurate prediction (absolute count). This way we know when the absolute count has a high value, it is more likely for high cards to be drawn.

Now that the algorithm has enough knowledge on how to count cards, the goal is to analyze the consecutive moves of a targeted player. It will receive as an input the playing cards dealt. While the rounds accumulate, it gives a probability if the player is indeed counting cards by checking if it’s doing the same plays as our machine.

5. Future Work

Our model is far from perfect and we can improve it a lot. Some of the improvements we thought of are listed below:

- Adding each card to the final dataset to do a multi-class classification problem with the output being the 52 cards available in a playing cards deck;
- Increase variety in terms of backgrounds to each card. Allied with Data Augmentation we would be able to build a richer dataset;
- In the Casino certainly it wouldn't happen, but while testing the algorithm if a different card would appear (a joker, for example) we should teach our model how to react to this kind of situation;
- Trying a different approach to the problem by using YOLO (You Only Look Once) for real-time image detection. This is a very fast and accurate object recognition algorithm with excellent learning capabilities. It uses a single neural network to perform both classification and prediction of bounding boxes for detected objects;
- There are tables known as the "Illustrious 18" and "Fab 4" that index some combination of cards and complement the card counting algorithm. We could train the model better so it can recognize these patterns and make better decisions by itself.

6. Conclusion

The process of creating an image recognition model started early when we decided to create our own playing cards dataset. Collecting data means that we had to take into consideration the images size, quality and at the end, labeling. It's a very time consuming process, but essential for a good final result. It enabled us to see the changes in results depending on the background that we used. After applying all the preprocessing associated with resizing and splitting into train, validation and test, we started to try different layers to see how the classifier would improve the accuracy. Afterwards, we tested with pre-trained models to see if we could improve the model performance. Finally, we tested the model and plotted the results.

In the minetime, we hope to improve the model by adding high quality images, variety in terms of backgrounds and finally testing it with the finalized Counting Cards Prevention Algorithm.

The dataset is available through the following link:
https://drive.google.com/file/d/1hUj9WQ8Fn6vN2k-KkNLb07IQ5a8sC4xH/view?usp=sharing

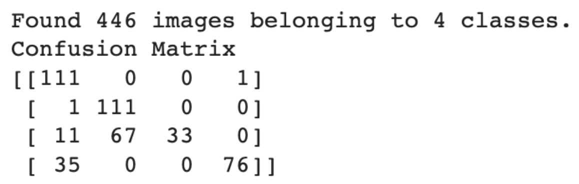
7. Bibliography

Goodfellow, I., Bengio, Y., & Courville, A. (2017). Deep learning (Adaptive Computation and Machine Learning series). November 18, 2016.

Chollet François. (2021). Deep learning with python. December 22, 2017.

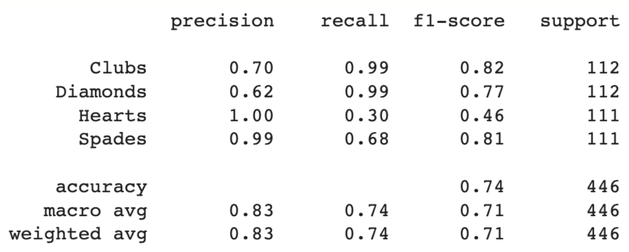
8. Annexes

Figure 3. Classifier Model Final Structure



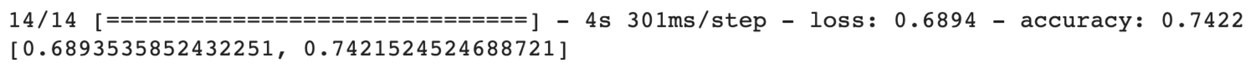
```
Found 446 images belonging to 4 classes.
Confusion Matrix
[[111   0   0   1]
 [  1 111   0   0]
 [ 11  67  33   0]
 [ 35   0   0  76]]
```

Figure 6. Advanced Model 03 - Confusion Matrix



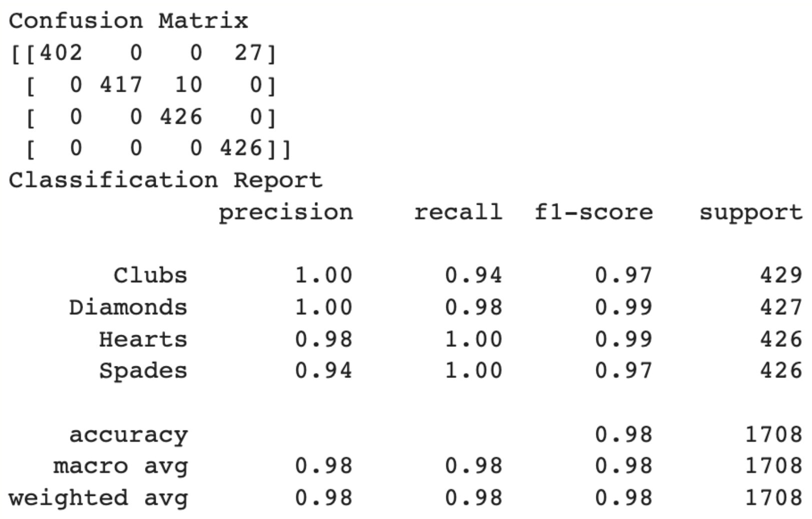
```
              precision    recall  f1-score   support

       Clubs       0.70      0.99      0.82       112
    Diamonds       0.62      0.99      0.77       112
      Hearts       1.00      0.30      0.46       111
      Spades       0.99      0.68      0.81       111

    accuracy                           0.74       446
   macro avg       0.83      0.74      0.71       446
weighted avg       0.83      0.74      0.71       446
```

Figure 7. Advanced Model 03 - Classification Report



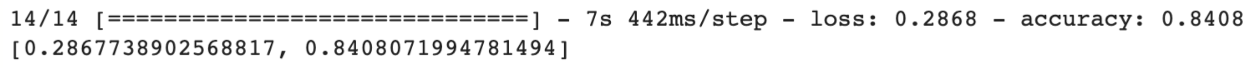
```
14/14 [==============================] - 4s 301ms/step - loss: 0.6894 - accuracy: 0.7422
[0.6893535852432251, 0.7421524524688721]
```

Figure 8. Advanced Model 03 - Accuracy Values



```
Confusion Matrix
[[402   0   0  27]
 [  0 417  10   0]
 [  0   0 426   0]
 [  0   0   0 426]]
Classification Report
              precision    recall  f1-score   support

       Clubs       1.00      0.94      0.97       429
    Diamonds       1.00      0.98      0.99       427
      Hearts       0.98      1.00      0.99       426
      Spades       0.94      1.00      0.97       426

    accuracy                           0.98      1708
   macro avg       0.98      0.98      0.98      1708
weighted avg       0.98      0.98      0.98      1708
```

Figure 9. Advanced Model 03 - Accuracy Values



```
14/14 [==============================] - 7s 442ms/step - loss: 0.2868 - accuracy: 0.8408
[0.2867738902568817, 0.8408071994781494]
```

Figure 10. Classifier Model - Accuracy Values