Project Report for Neural Computing

Cards Image Classification

Author Salvatore Romano
Neural Computing course
Prof. Sebastiano Battiato

MSc in Data Science for management

Contents

1 Introduction

Image classification is a task in computer vision and image processing where the goal is to assign a label or class to an input image. The process typically involves training a model on a large data-set of labeled images, and then using that model to predict the class of new, unseen images.

There are many different techniques and algorithms that can be used for image classification, but the most common approach is to use a CNN *(Convolutional Neural Network)*. CNNs are a type of deep learning model that are specifically designed to process data with a grid-like topology, such as an image. They are composed of multiple layers, each of which extracts increasingly complex features from the input image. The final layers of a CNN are typically fully connected layers, which are used to make the final class prediction.

The goal of this project is to use an images' data-set for classifying in a proper way the playing cards with the use of a CNN. The data-set can be easily reached by clicking here.

2 Background and Data

The goal of this image classification will be faced in Matlab, with a specific toolbox called ***Deep Learning Toolbox***, that allows you to build deep neural network from scratch or to use pre-trained ones.

In particular, a CNN will be trained both on train set and on validation set, then the prediction will be done on the test set. According to the requirements for this project, I decided to train, first a network by scratch and then a pre-trained one (***GoogLeNet***), also to see the differences in term of time and accuracy.

The data-set, is already divided in train set, validation set and test set. The images are $224x224x3$ in jpg format, they have been cropped so that, only the image of a single card is present and the card occupies well over 50% of the pixels in the image. There are 7624 training images, 265 test images and 265 validation images. The train, test and validation directories are partitioned into 53 sub directories, one for each of the 53 types of cards. By analysing the data-set, I noticed that there is already applied images' augmentation.

Before to start with the next section, it would be a good idea to give you some information about the classes of this data-set. As you already know the classes are 53, notice that inside the train set the "joker" card is in one folder that contains all the seeds, so for this reason there are 53 classes instead of 56.

Figure 1: Train data observations

As you can see from the Figure 1, the class with the most observations (181) is the *"Ace of Spades"*, instead the class with the fewer observations (108) is the *"Seven of Clubs"*.

3 Approaches

To manage and correctly perform this project, I will adopt two approaches that will guide me in this analysis.

This first approach regards the creation of a CNN from scratch. The complexity of the network will not be so high, due to computational power that is needed for the training phase.

The second approach regards the training of a pre-trained CNN, in particular I will use the ***GoogLeNet***.

4 My first CNN

At first I built a CNN from scratch; of course, the expectations are not high cause I already know that the building phase of a CNN requires a lot of instruments. Driven by my curiosity I decided to build a CNN regardless of the results, just to make a comparison with the results of the pre-trained one.

Starting from the configuration of my CNN, is made up of 34 layers and 35 connections. The following image is the representation of the network configuration.

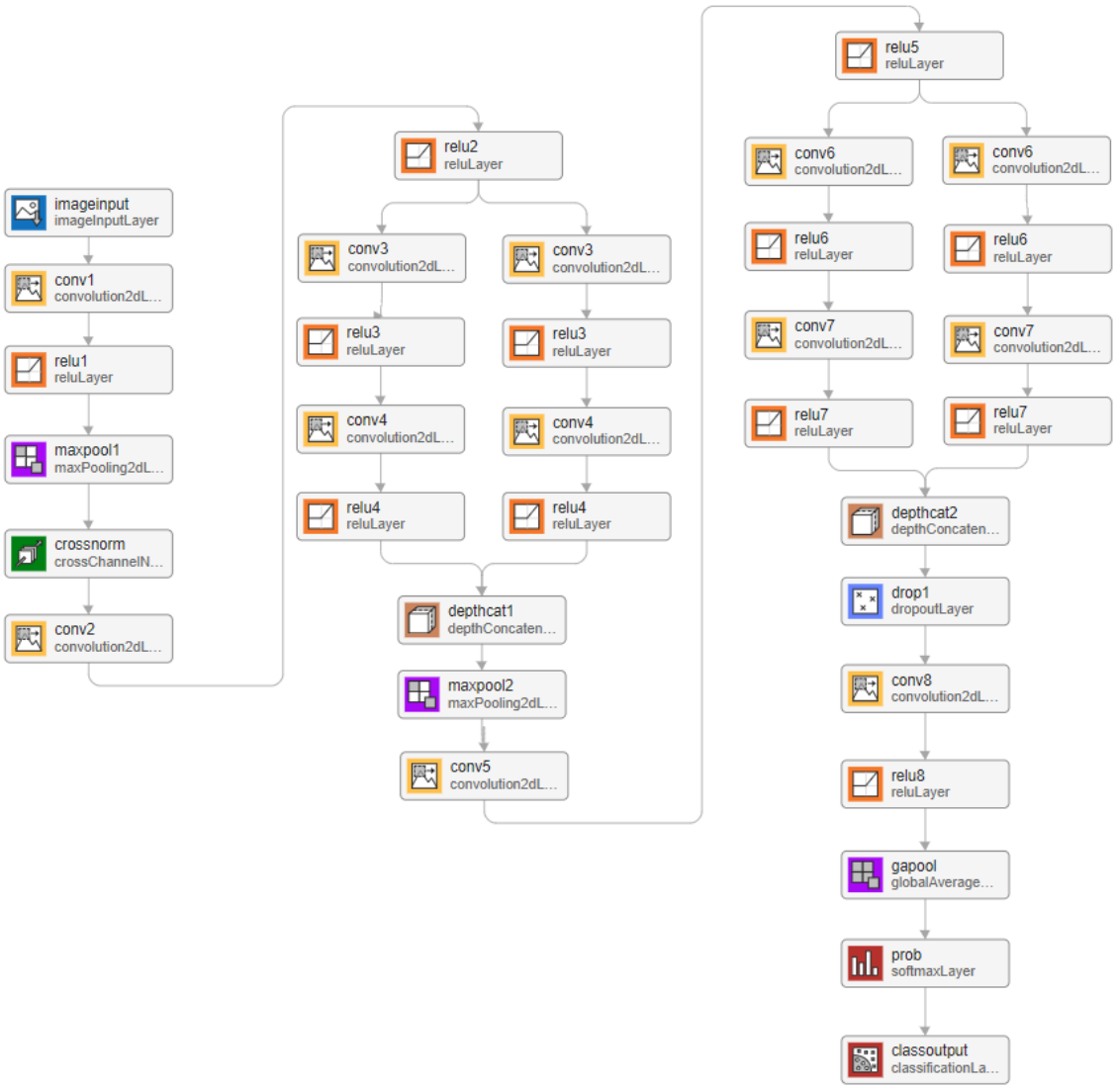

Figure 2: CNN configuration

The architecture is very simple and follows a logic pattern; it is being constructed using a combination of convolutional layers, ReLU activation layers, max pooling layers, and cross-channel normalization layers at the start. The model also uses depth concatenation layers and dropout layers to prevent over-fitting. The final layers include a global average pooling layer, a softmax layer, and a classification layer.

4.1 Model Details

In this section there will be covered the details of model, in particular the hyper-parameters. The following analysis will be done layer by layer.

The first layer is an *image input layer* with the size of the input image, which is $224x224x3$.

Then, there is a *convolutional layer* with a kernel size of $3x3$ and 64 filters, with stride $2x2$. Next, a ReLu *(Rectified Linear Unit)* activation function is applied on the output of this layer.

A *max pooling layer* of size $3x3$ is applied with stride $2x2$, followed by a *cross channel normalization layer*.

Then, another *convolutional layer* with 32 filters, followed by a ReLu activation function.

Now there is the first convolutional block: 2 pairs of *convolutional layers* and ReLu activation function, with kernel size $3x3$, $5x5$ and 64,32 filters respectively.

A *depth concatenation layer* is applied to the output of the two last pairs of *convolutional layers*; a *max pooling layer* of size $3x3$ is applied with stride $2x2$, followed by a *convolutional layer* of 32 filters and ReLu activation function.

Then, there is the second convolutional block: 2 pair of *convolutional layers* of 128 and 64 filters respectively, with kernel size $3x3$ and $5x5$ respectively, are applied, each one followed by ReLu activation function.

As before, another *depth concatenation layer* is applied to the output of the two last pairs of *convolutional layers*, followed by a *dropout layer* with probability 0.5 and a *convolutional layer* of 53 filters and ReLu activation function.

Finally, a *global average pooling layer* is applied, followed by a *softmax layer* and a *classification layer*.

4.2 Training Options

In a CNN, the training options include the optimizer, the batch size, the learning rate etc. The optimizer is used to update the model's weights during training, such as using SGD *(stochastic gradient descent)* or Adam. The batch size determines the number of samples used in one forward/backward pass, which can impact the model's performance and computational efficiency. Other options such as learning rate, number of epochs and regularization also play important role in training a CNN.

Based on the upon configuration, the following image represents the training options used for the network.

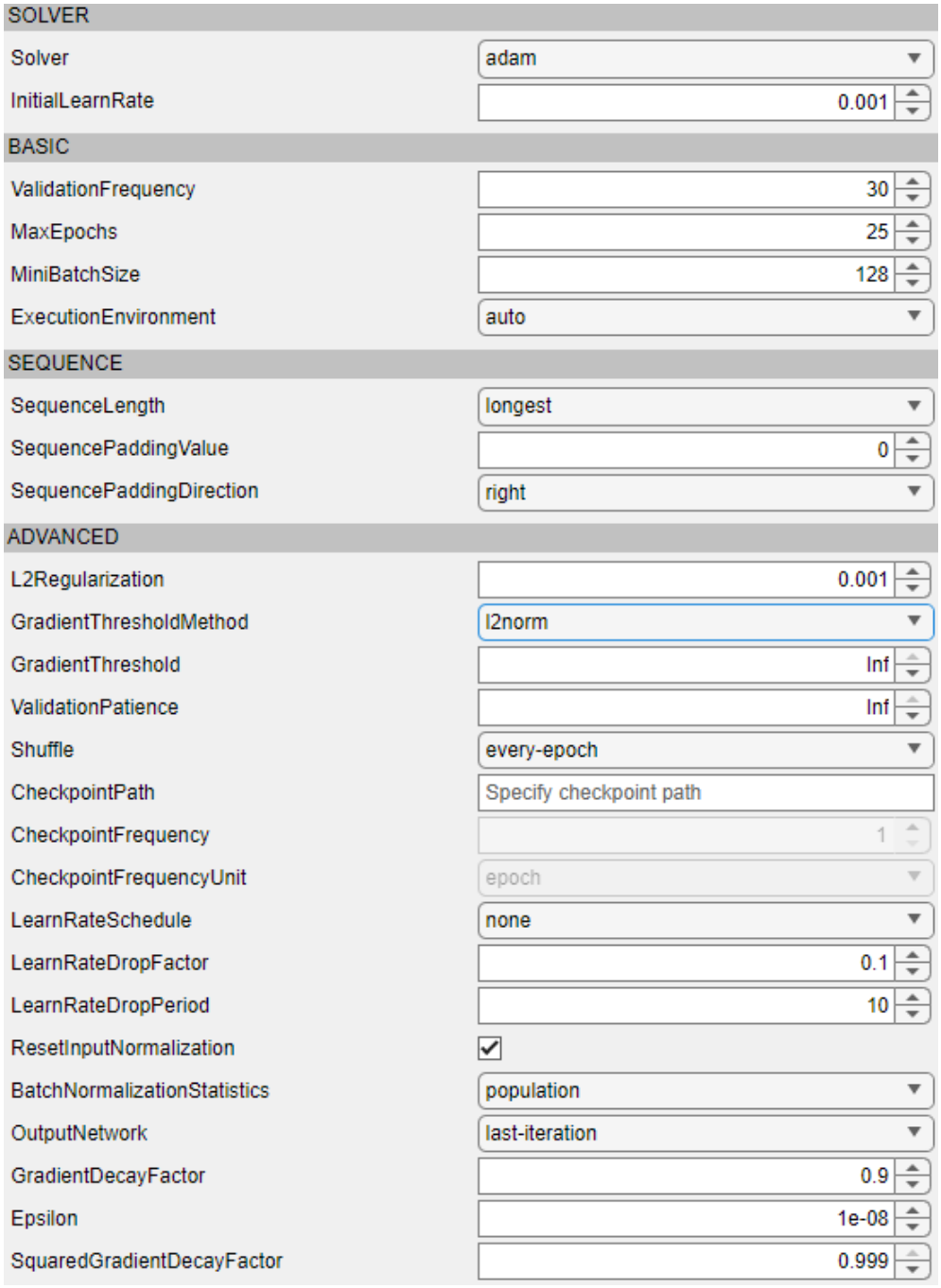

Figure 3: Training options

The Figure 3 represents the final training options that allow the CNN to reach the accuracy that I will show you in the next section. In particular, these options have been chosen by trying different combination of them.

For example, I tried the SGD with a specific learning rate, but as you know, it computes the gradient of the loss function with respect to the weights for a single training example at a time. This can make the optimization process more noisy and less stable compared to other optimization algorithms like Adam, that adapts the learning rate for each weight individually, which can lead to more stable and efficient optimization without

oscillations. Other change is, for example, the *L2Regularization* value, to better manage the over-fitting.

4.3 Final Results

As a reminder, the computation performed was both in train and validation set, with the training options specified before.

The results achieved are the following.

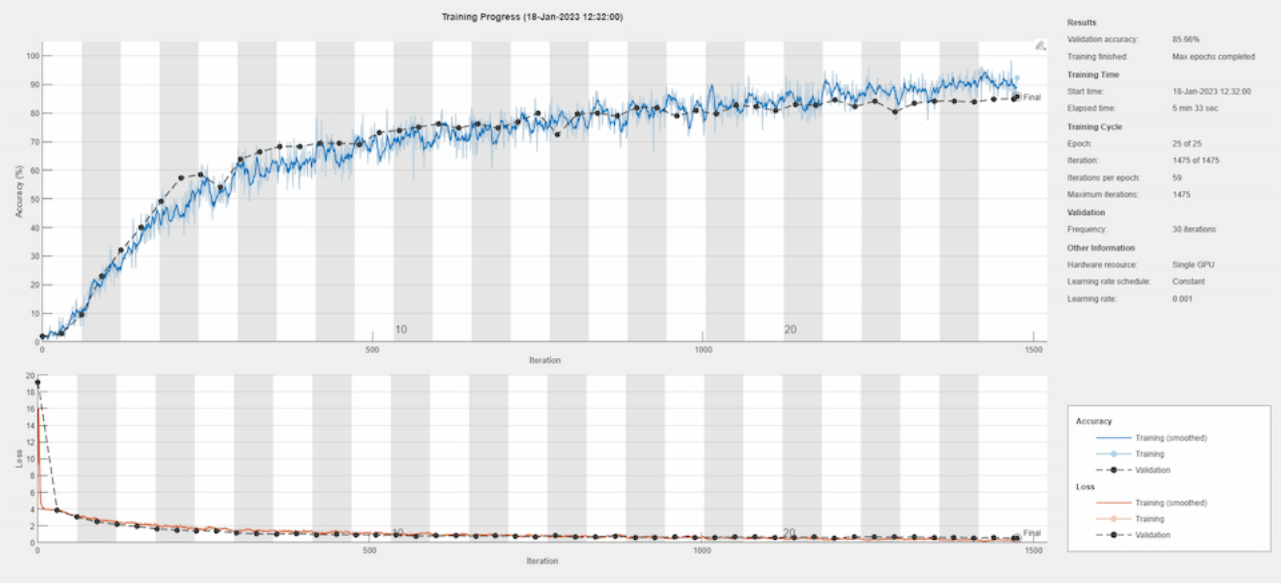

Figure 4: Final results

As it possible to notice from the Figure 4, the CNN reached an accuracy, on the validation set, of 85.66% in just 5 minutes. This can be a good value by considering that is a network created from scratch.

Now I will plot the confusion matrix's result based on the validation set, that is made up of 265 images.

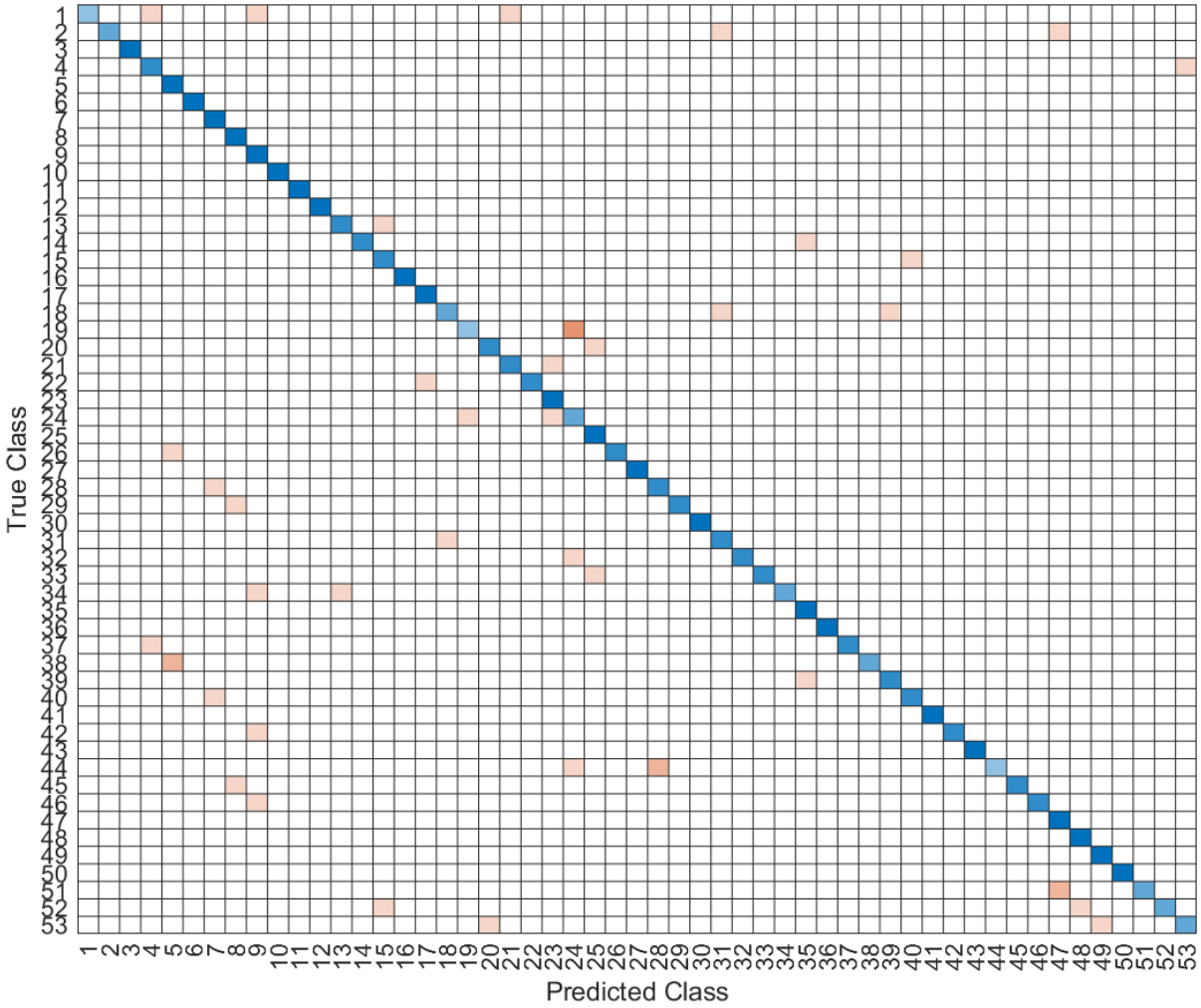

Figure 5: Confusion matrix of validation set

Due to the number of classes, is almost impossible to see in a proper way the confusion matrix in Figure 5; note that I can't reduce the number of classes because the data-set in based on playing cards, so each card is different from the other. The total number of misclassified images is 44 on 265.

Going on, it is necessary to compute the confusion matrix also for the test set, that is of the same size of the validation set, so 265 images.

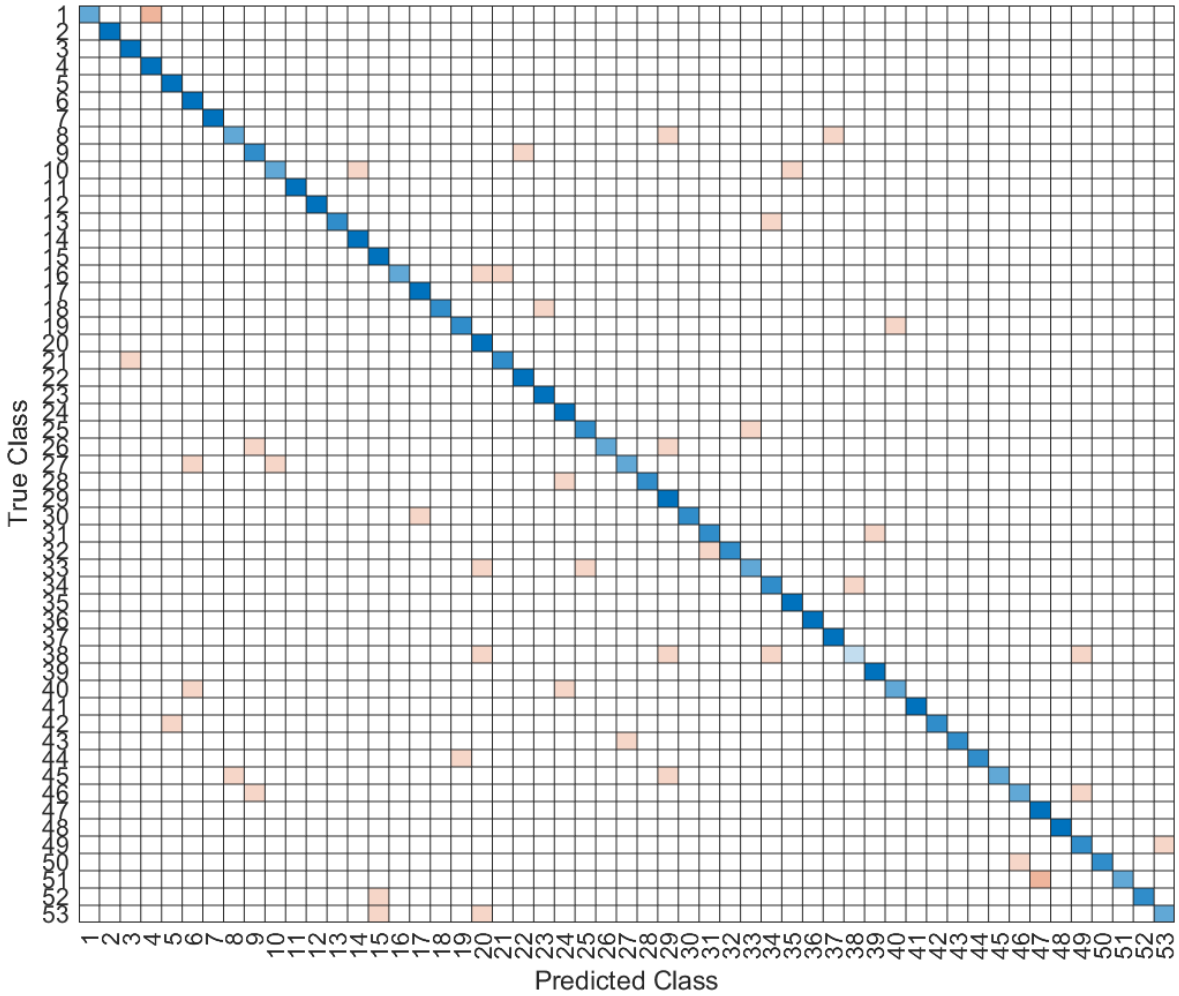

Figure 6: Confusion matrix of test set

As you can see in Figure 6, in this case the misclassified images are 45 on 265, so the result is almost similar to the one obtained in the validation set. The accuracy score computed in the test set has a value of 83.02%.

Before going on, it is interesting to display the misclassified images in MatLab, by comparing the true label in the test set, with the one predicted. Of course I will not provide all the 45 misclassified images, but only some of them.

Some misclassified images by this CNN are:

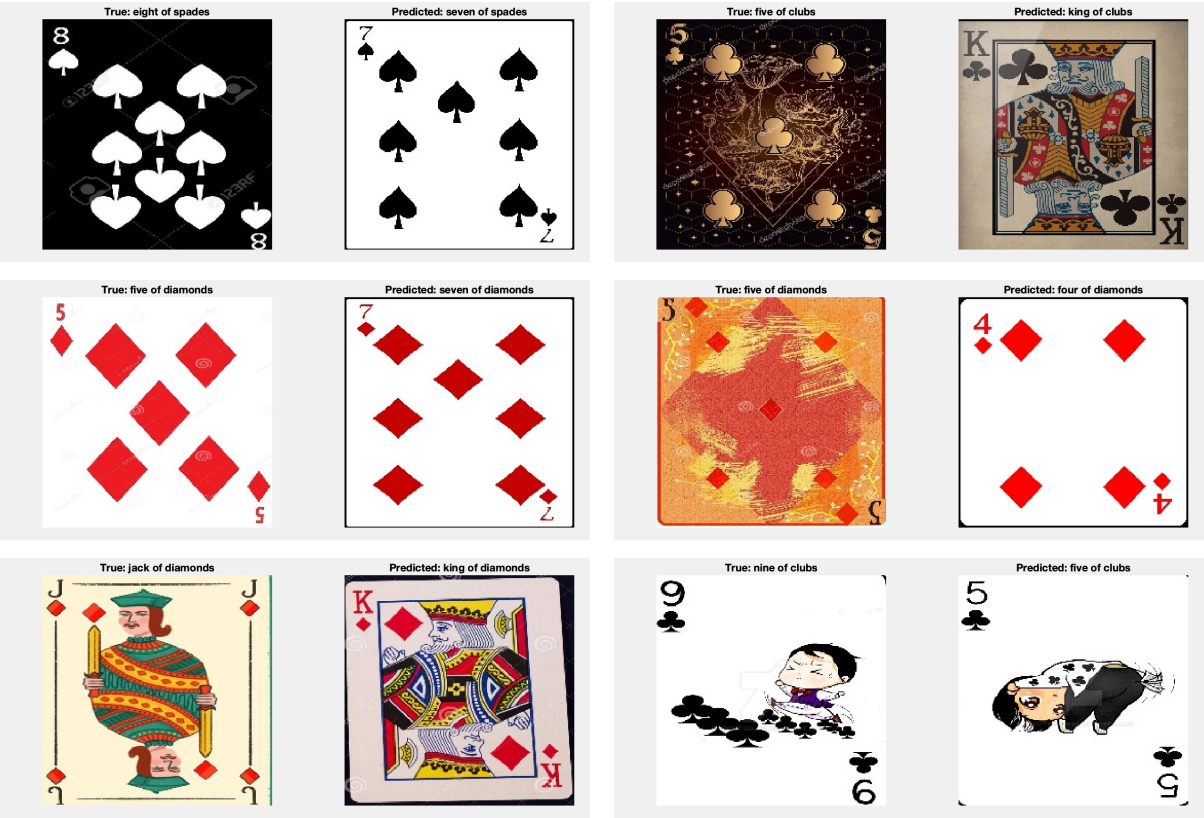

Figure 7: Examples of misclassified cards

5 GoogLeNet

GoogleNet, also known as Inception-v1, is a deep CNN architecture that was developed by Google and published in 2014. The network was trained on the ImageNet dataset, which contains more than 14 million images belonging to 1000 different classes. GoogleNet achieved a top-5 error rate of 6.67%, which was a significant improvement over the previous state-of-the-art at the time.

One of the key innovations of GoogleNet is the use of "Inception modules" which are designed to incorporate multiple convolutional and pooling layers in a single module. This allows the network to learn a more diverse set of features at different scales, which helps to improve its accuracy. The network also uses a "factorization" strategy to reduce the number of parameters and computational cost.

Another important feature of GoogleNet is the use of auxiliary classifiers in the Inception modules, which allow the network to learn more about the intermediate representations of the image and improve the overall accuracy.

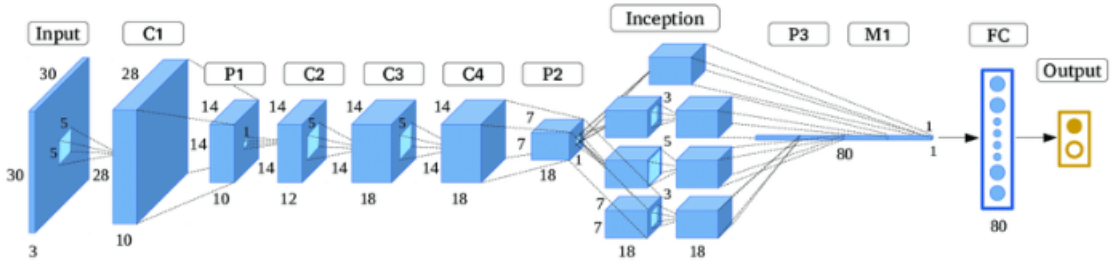

Figure 8: GoogleNet Configuration

I will not focus on the GoogleNet's hyper-parameters detail, cause I have already talked before about my CNN.

5.1 Final Results

In this section there will be explained the final results achieved by the GoogleNet on the cards image dataset. Noticed that the following results are based on these specification: optimizer Adam, learning rate of 0.001, mini_batch_size of 128. As always, the training of this pre-trained model has been performed both on train and validation set.

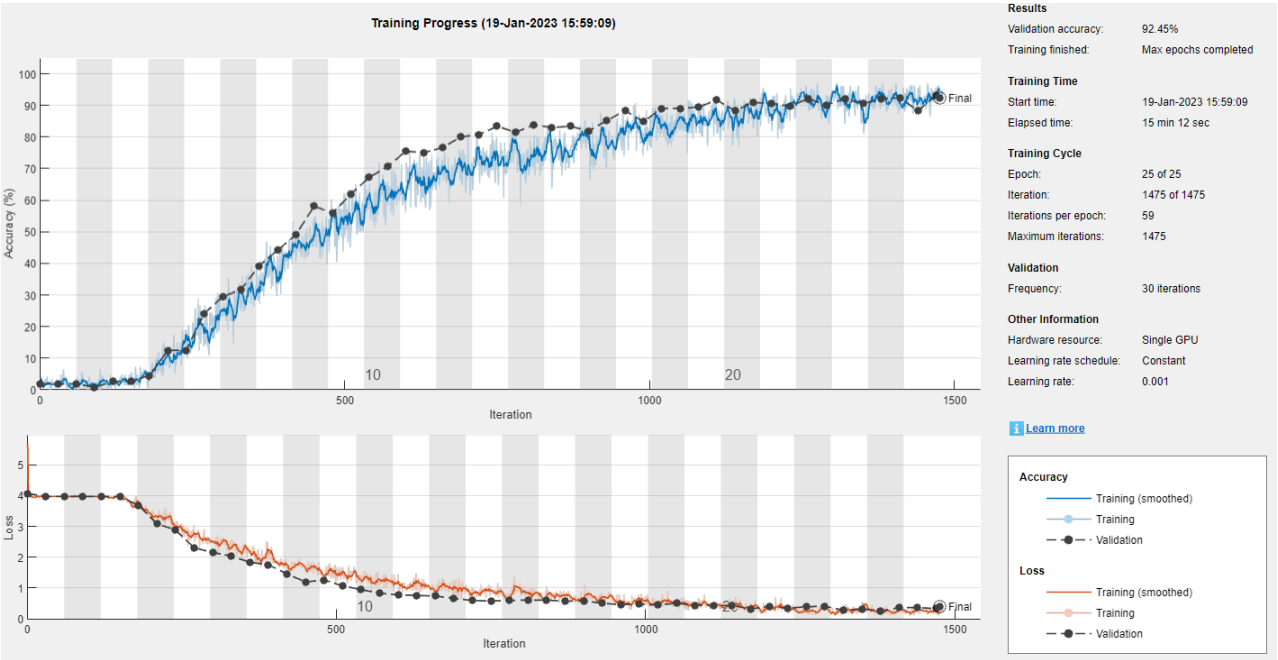

Figure 9: Final results

In Figure 9 is possible to observe that the final accuracy in the validation set is around 92.45%, achieved in 15 minutes. Also in this case I will provide you the confusion matrix for both validation and test set.

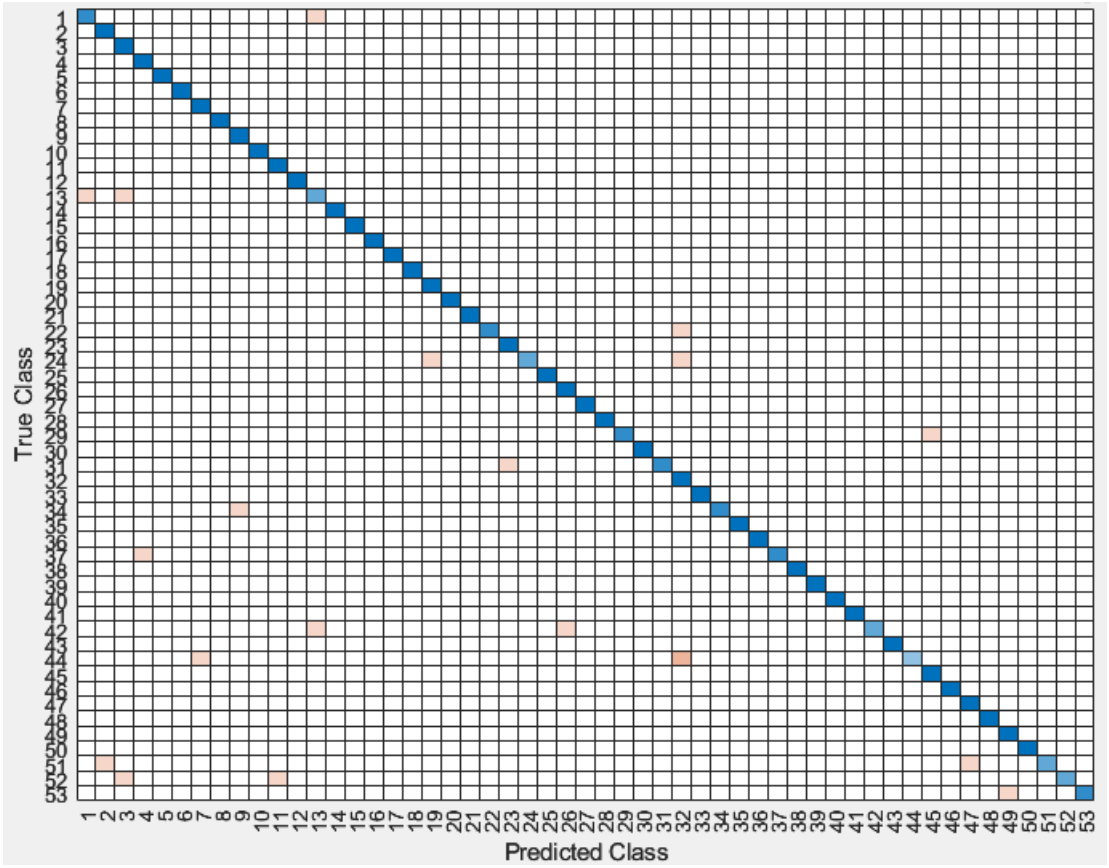

Figure 10: Confusion matrix of validation set

By analysing the Figure 10 is possible to see that only 20 images have been misclassified on a total of 265. Of course this is a better result compared to my CNN, that on the other hand, works better speaking about computation time.

Now I will display the confusion matrix of the test set. Note that the accuracy score on test set is around 90.20%.

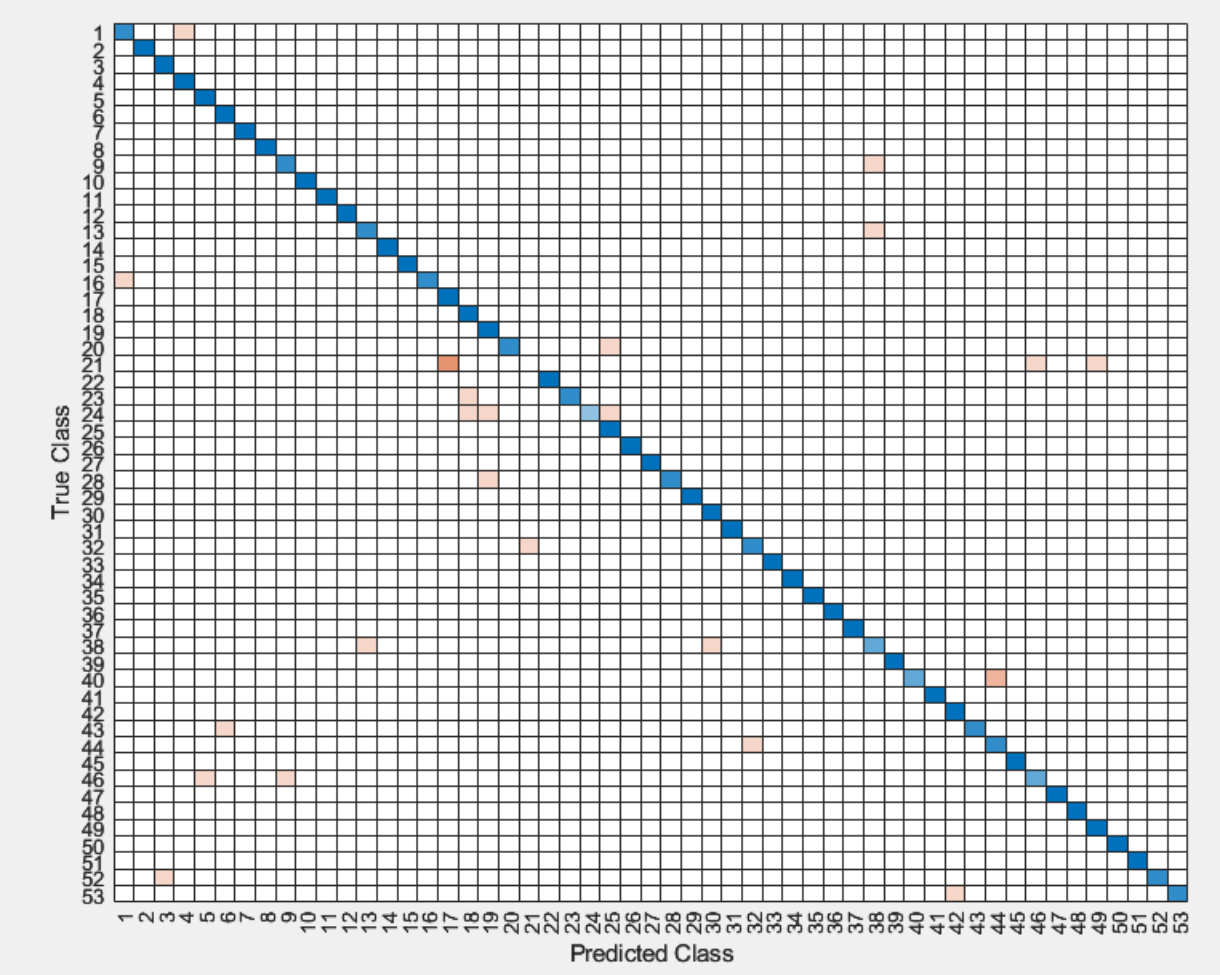

Figure 11: Confusion matrix of test set

As it possible to notice by looking at Figure 11, in this case the misclassified images are 26 on 265. Like for my CNN, I will show some of the misclassified images.

Some misclassified images by the GoogleNet are:

Figure 12: Examples of misclassified cards

6 Further Applications

Of course the project's goal has been reached! In this section I would like to investigates about the activation of different layers of a CNN; by analysing the layers' activation is possible to discover which features the network learns by comparing areas of activation with the original image. In brief, I will provide an image to the CNN in order to full fill the statement above.

The card that will be provided to the CNN is the *"Jack of Hearts"*. In particular:

Now the image will pass through the network to examine the output activations of the first convolutional layer of my CNN.

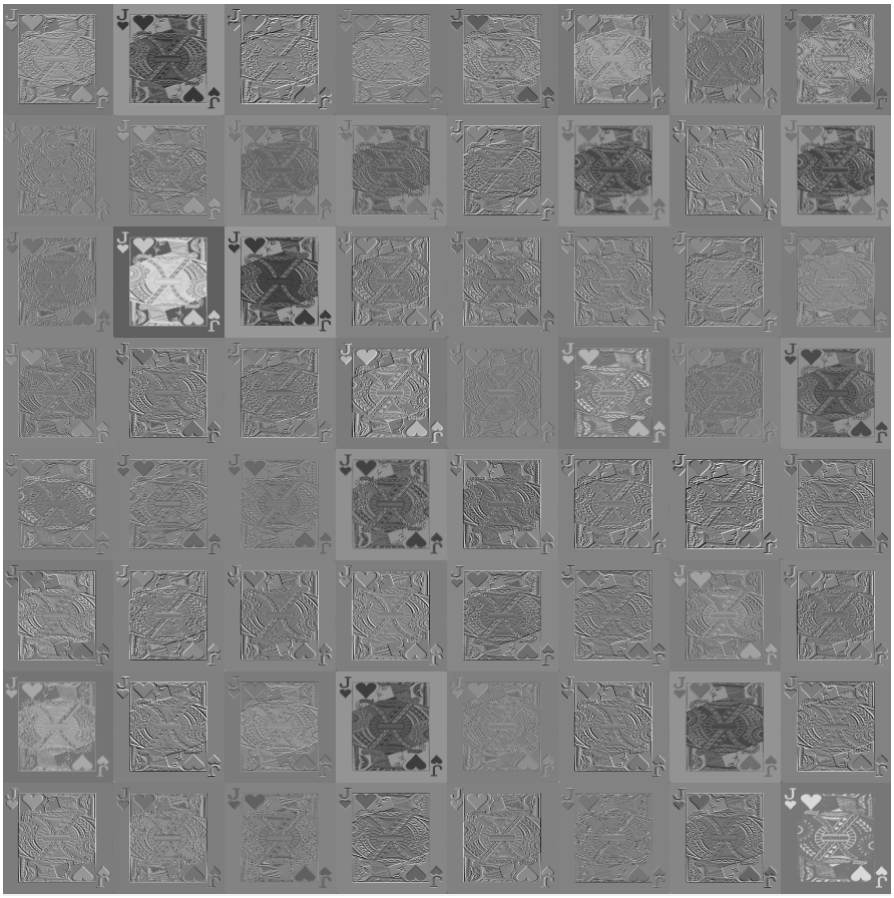

Figure 13: First convolutional layer's activation

By observing Figure 13, it can be notice that each tile in the grid of activations is the output of a channel in the first convolutional layer. White pixels represent strong positive activations and black pixels represent strong negative activations.

Now let's see if the channel detects the edge of the image.

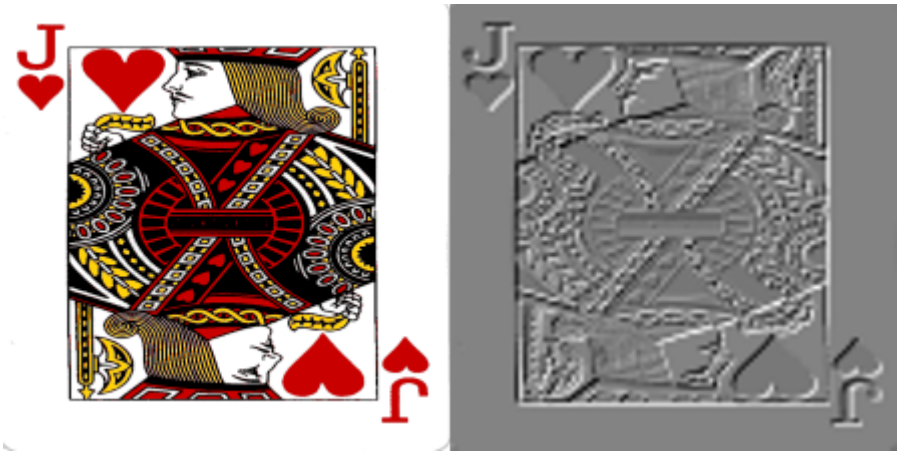

Figure 14: Edges activation

In Figure 14 you can see that this channel focuses more on the edges.

Of course, you can analyse whatever layer you want; in deeper layer the network learns to detect more complicated features. For example, we can analyse a deeper layer to see what it takes in count.

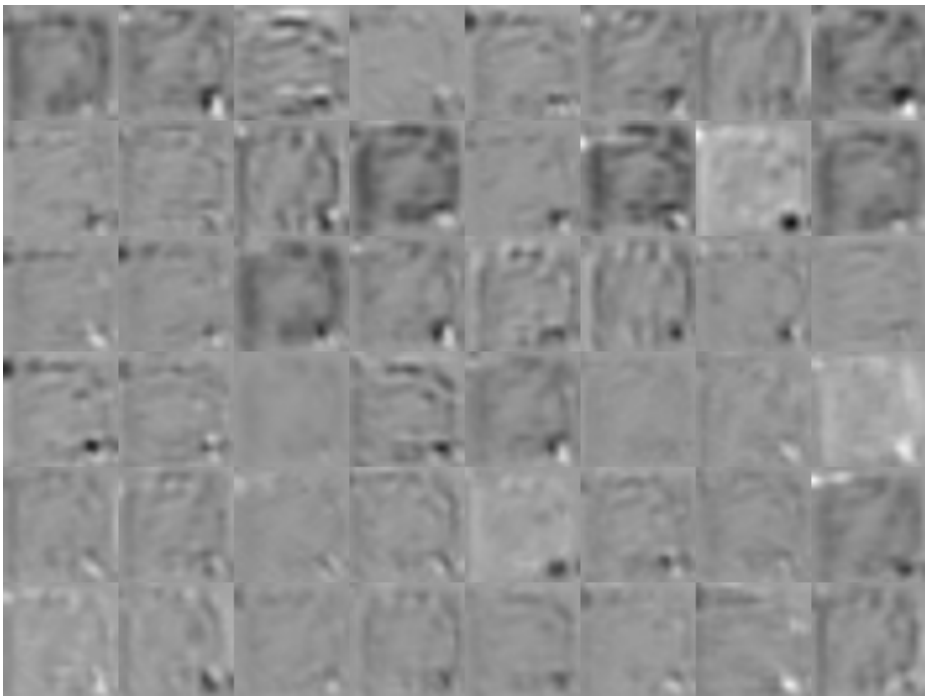

Figure 15: Seventh convolutional layer's activation

After all these procedure, it is interesting to discover if the CNN is able to recognizes the image!

Figure 16: Predicted result

The last experiment that I would like to do is to provide always this image to the CNN, but this time I will remove 2/3 of the image; this because the CNN has been trained on a data-set that contains images also with data augmentation (modified image like different color, background , cropped etc..). Let's try this:

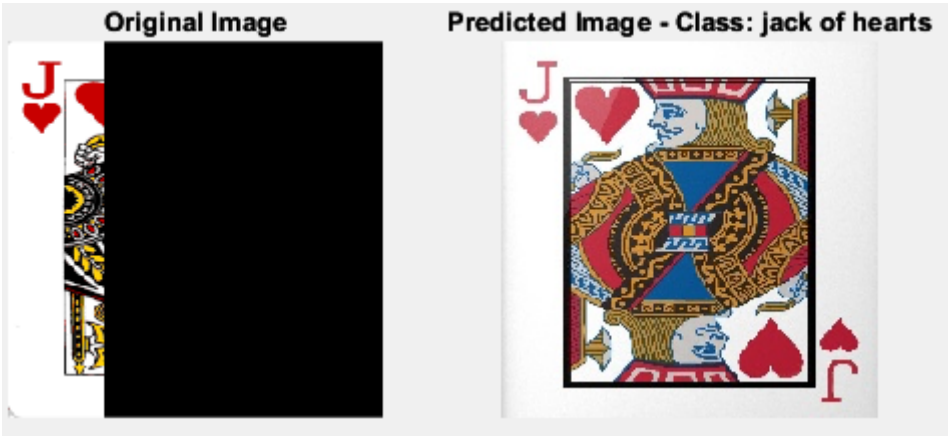

Figure 17: Cropped image prediction

7 Appendix

```
test=imageDatastore('/Users/turex/Documents/Neural Computing/
   test', "IncludeSubfolders", true,"LabelSource", "
   foldernames");
true_test_labels = test.Labels;
[pred_test_labels,score_test] = classify(finalnet, test);
accuracy_test = mean(true_test_labels == pred_test_labels);
C_test = confusionmat(true_test_labels, pred_test_labels);
confusionchart(C_test)
heatmap(C_test)
 valid=imageDatastore("/Users/turex/Documents/Neural
    Computing/valid", "IncludeSubfolders", true,"LabelSource
    ", "foldernames");
 true_validation_labels = valid.Labels;
 [pred_validation_labels,score_validation] = classify(
    finalnet, valid);
 accuracy_validation = mean(true_validation_labels ==
    pred_validation_labels);
 C_valid = confusionmat(true_validation_labels,
    pred_validation_labels);
 confusionchart(C_valid)

misclass_indices = find(true_test_labels ~= pred_test_labels)
   ;
for i = 1:numel(misclass_indices)
    subplot(1,2,1)
    imshow(readimage(test, misclass_indices(i)));
    true_label = true_test_labels(misclass_indices(i));
    title(sprintf('True: %s', true_label));

    pred_label = pred_test_labels(misclass_indices(i));
    pred_idx = find(pred_test_labels == pred_label);
    subplot(1,2,2)
    imshow(readimage(test, pred_idx(2)));
    title(sprintf('Predicted: %s', pred_label));

    input("Press Enter to continue...");
end
 true_validation_labels = valid.Labels;
 [pred_validation_labels1,score_validation1] = classify(
    googlenet, valid);
 accuracy_validation1 = mean(true_validation_labels ==
    pred_validation_labels1);
```

```
 C_valid1 = confusionmat(true_validation_labels,
    pred_validation_labels1);
 confusionchart(C_valid1)

true_test_labels = test.Labels;
[pred_test_labels1,score_test1] = classify(googlenet, test);
accuracy_test1 = mean(true_test_labels == pred_test_labels1);
C_test1 = confusionmat(true_test_labels, pred_test_labels1);
confusionchart(C_test1)

misclass_indices1 = find(true_test_labels ~=
   pred_test_labels1);
for i = 1:numel(misclass_indices1)
    subplot(1,2,1)
    imshow(readimage(test, misclass_indices1(i)));
    true_label = true_test_labels(misclass_indices1(i));
    title(sprintf('True: %s', true_label));

    pred_label = pred_test_labels1(misclass_indices1(i));
    pred_idx = find(pred_test_labels1 == pred_label);
    subplot(1,2,2)
    imshow(readimage(test, pred_idx(1)));
    title(sprintf('Predicted: %s', pred_label));
    input('Press Enter to continue..')
end

im = imread('Jack of Hearts.png');
imshow(im)
imgSize = size(im);
imgSize = imgSize(1:2);
act1 = activations(finalnet,im,'conv');
sz = size(act1);
act1 = reshape(act1,[sz(1) sz(2) 1 sz(3)]);
I = imtile(mat2gray(act1),'GridSize',[8 8]);
imshow(I)

[maxValue,maxValueIndex] = max(max(max(act1)));
act1chMax = act1(:,:,:,maxValueIndex);
act1chMax = mat2gray(act1chMax);
act1chMax = imresize(act1chMax,imgSize);

I = imtile({im,act1chMax});
imshow(I)
act6 = activations(finalnet,im,'conv_7');
sz = size(act6);
```

```
act6 = reshape(act6,[sz(1) sz(2) 1 sz(3)]);

I = imtile(imresize(mat2gray(act6),[64 64]),'GridSize',[6 8])
    ;
imshow(I)
[predictedLabel,score] = classify(finalnet,im);
subplot(1,2,1);
imshow(im);
title("Original Image");

predictedImage = readimage(test, find(pred_test_labels ==
    predictedLabel, 1));
subplot(1,2,2);
imshow(predictedImage);
title(sprintf('Predicted Image - Class: %s', predictedLabel))
    ;


i_aug = imread('Jack of Hearts.png');

mask = zeros(size(i_aug));
mask(:,1:floor(size(i_aug,2)/3.9),:) = 1;
img_half_covered = i_aug .* uint8(mask);

[predictedLabel,score] = classify(finalet,img_half_covered);
subplot(1,2,1);
imshow(img_half_covered);
title("Original Image");

predictedImage = readimage(test, find(pred_test_labels ==
    predictedLabel, 1));
subplot(1,2,2);
imshow(predictedImage);
title(sprintf('Predicted Image - Class: %s', predictedLabel))
    ;
```

References

[1] https://www.kaggle.com/datasets/gpiosenka/cards-image-datasetclassification

[2] https://mathworks.com/products/matlab.html

[3] https://www.researchgate.net/figure/GoogleNet-like-architecture_fig7_320723863

[4] https://it.mathworks.com/help/deeplearning/ug/visualize-activations-of-a-convolutional-neural-network.html