Alphabet Classification with ConvNets*

Nunzia Esposito, Mauro Vitrone, and Sara Volpe

University of Salerno, Course of FVAB {n.esp0624, m.vitrone95, sara1994.sv}@gmail.com

Abstract. Transfer Learning plays a crucial role when a given dataset has insufficient labeled examples to train an accurate model. In such scenarios, the knowledge accumulated within a model pre-trained on a source dataset can be transferred to a target dataset, resulting in the improvement of the target model. Due to their hierarchical architecture, Deep Neural Networks provide flexibility and customization in adjusting their parameters and depth of layers, thereby forming an apt area for exploiting the use of Transfer Learning. In this paper, we trained some convolutional neural networks to classify our dataset, composed of about 13 thousand high-resolution images of Alphabet letters, into 26 different classes. The first network chosen for transfer learning is Alexnet. At each training session we have changed some training options to achieve an accuracy rate of over 95%. Once we have found the optimal training options we have crystallized them for all the networks examined. This allowed us to compare the performances of the networks on equal terms of hardware and parameters. To make training faster, we used a very efficient GPU implementation of the convolution operation.

Keywords: Transfer Learning · ConvNets · Alphabet.

1 Introduction

Human learners appear to have inherent ways to transfer knowledge between tasks. That is, we recognize and apply relevant knowledge from previous learning experiences when we encounter new tasks. The more related a new task is to our previous experience, the more easily we can master it. Common Machine Learning algorithms, in contrast, traditionally address isolated tasks. Transfer Learning (TL) attempts to change this by developing methods to transfer knowledge learned in one or more source tasks and use it to improve learning in a related target task. Techniques that enable knowledge transfer represent progress towards making machine learning as efficient as human learning [1]. The possibility of approaching a machine to think like a human has been the main motivation of our work.

TL is a branch of Machine Learning which leverages the knowledge stored within a source domain and provides a means to transfer the same to a target domain where a domain could mean datasets, tasks, games, etc. It is based on

* Supported by BipLab

the fact that features that have been learned, for instance, to classify ripe and non ripe apples can also be used to classify pears or peaches - fruits of the same family. TL becomes indispensable in scenarios where a given task (referred to as the target task) does not have enough data required to train and accurate model. Under such situations, the knowledge stored within an off-the-shelf model which is trained on a source domain can be transferred to the target domain [2].

Convolutional Neural Networks (CNN) have long been a subject of interest especially in the context of TL, more specifically in the domain of image processing. Krizhevsky and Lee [3], [4] demonstrate how low-level neural layers in such networks can be successfully transferred for different tasks. In [5], [6] the transferability of each layer of a CNNs has been studied in where they have shown that while the lower layers learn features which are general across different tasks, the higher ones reflect the specific nature of the task at hand. For example, in the case of object classification, the hidden parameters of the output regression layer (generally the last layer) are highly specific to the number and type of labels to classify and hence are less likely to be transferable. Due to their stacked architecture, CNNs tends to provide flexibility while transferring knowledge stored within their layers, which in turn facilitates TL.

These concepts are the starting point for the fulfillment of this paper on TL. At first, the work presented herein endeavors to provide an appropriate replacement of high-level layers. As a result, we make an overview of the pretrained CNNs used to classify the alphabet and a selection of the best training options so that the TL is optimal. Also, we present a comparison of performance and accuracy value of the modify CNNs.

2 Transfer Method

The method we used for transfer learning regards the fine-tuning of pre-trained networks. Each of these models was previously trained on a subset of the ImageNet database, which is used in ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). The networks are trained on more than a million images and can classify images into 1000 object categories [7].

We employed five networks: AlexNet, GoogLeNet, Inception-v3, ResNet-50 e ResNet-101. Originally we also took in exam VGG-16 and VGG-19, but we tried without success the process of transfer learning of these models. The hardware at our disposal does not fulfill the networks requirements in terms of dedicated RAM for the GPU. Our experiment is based on the comparison of different networks, using the same work environment and parameters of training: for the consistency of our project we chose to remove VGG-16 and VGG-19 from the experiment.

For our work we chose the best laptop available, that is a DELL XPS 15, equipped with a graphic card nVidia GTX 1050 with 4 GB of RAM dedicated. The development environment used is MATLAB and the MathWorks documentation was our starting point. Our initial approach was the following: for each network, we extracted all layers except for the last three, which are configured

for 1000 classes; afterwards we replaced them with a fully connected layer, a softmax layer, and a classification output layer.

Let's see their role:

- **fullyConnectedLayer:** A fully connected layer multiplies the input by a weight matrix and then adds a bias vector;
- **softmaxLayer:** A softmax layer applies a softmax function to the input. For each sample, the softmax function, evaluates the probability of belonging to the classes and returns the highest score;
- **classificationLayer:** A classification output layer holds the name of the loss function the software uses for training the network for multiclass classification, the size of the output, and the class labels [7].

We specified the size of the fully connected layer according to the number of classes in the new data and also increased the *WeightLearnRateFactor* and *BiasLearnRateFactor* in order to learn faster in the new layers than in the transferred layers: the software multiplies these factors by the global learning rate to determine the learning rate for the weights and bias in the layer.

We need to point out something about the architecture of the networks: AlexNet is the only one which has not a LayerGraph object; a layer graph describes the architecture of a directed acyclic graph (DAG) network. Because of this, the approach used for GoogLeNet and the other networks left is slightly different; we extracted the layers and stored them in a LayerGraph object, then we removed the last three layers. Next, we created new three layers (the same mentioned above) and added them to the layer graph. Finally we had to connect the last transferred layer remaining in the network (this last one is different according to the type of network) to the new layers.

For these networks we made a freezing operation of the weights of earlier layers by setting the learning rates in those layers to zero. During the training, the trainNetwork function does not update the parameters of the frozen layers. With this operation, the gradient of them do not need to be computed and it can significantly speed up the training. Moreover we want the network to focus on learning dataset-specific features in the remaining layers.

Our initial dataset had a size that did not correspond to that required by the networks. Every one of them has an image input size that differs from one another. To automatically resize the training images, we used an augmented image datastore. We performed additional augmentation operations like randomly flipping them along the vertical axis and translate them up to 30 pixels horizontally and vertically. The augmentation could prevent the network from overfitting and memorizing the exact details of the training images. Contrariwise, with the validation images, we did not perform any of the former augmentation operations, but we simply resized them.

The first training was performed with AlexNet. After many attempts, we found a good assignment of the *trainingOptions* function parameters, which brought to a very high accuracy of the network. Afterwards, our policy was to use the same values of *trainingOptions* and test them with all networks. We wanted to compare the performance of the models given the same environment, without

changing hardware and parameters. This is why we trained the networks with the same partitioning of the images dataset. The dataset was randomly divided in several parts once, and each of them used during the steps of the training. We wanted the models to use the same subsets of images for training, validation and prediction.

3 Transfer Experiments

At the beginning we have a dataset, supplied by BipLab, composed of almost 3 thousand images of handwritten capital letters divided in 26 folders, each one represents a letter of the alphabet. Afterwards, we found another dataset, called "Latin" [8], with the same structure but it also contained lower case letters. So we had to scroll through all the images and delete the useless ones. After that we could merged the two datasets obtaining a single set of about 13 thousand images. For our task we must divided it into two sets called "imdsDataSet" and "imdsPrediction", these are composed respectively by 85% and 15% of the previous dataset. The second set is used for assessment of the generalization error of the final chosen model, instead the first set has been divied into other two subsets: 70% of the images are included in "imdsTrain" while the other 30% are included in "imdsValidation". The training set is used to fit the models, the validation set is used to estimate prediction error for model selection.

The training options that we chose to make CNNs more efficient are:

- **MiniBatchSize:** Size of the mini-batch to use for each training iteration, specified as a positive integer. A mini-batch is a subset of the training set that is used to evaluate the gradient of the loss function and update the weights.
  After several tests we chose 64 as the best value.
- **MaxEpochs:** Maximum number of epochs to use for training, specified as a positive integer. An iteration is one step taken in the gradient descent algorithm towards minimizing the loss function using a mini-batch. An epoch is the full pass of the training algorithm over the entire training set.
  The value that has been chosen is 14.
- **InitialLearnRate:** Initial learning rate used for training, specified as a positive scalar. If the learning rate is too low, then training takes a long time. If the learning rate is too high, then training might reach a suboptimal result or diverge.
  After various tests we selected 0.001.
- **Shuffle:** Option for data shuffling, specified as one of the following:
  - once: Shuffle the training and validation data once before training.
  - never: Do not shuffle the data.
  - every-epoch: Shuffle the training data before each training epoch, and shuffle the validation data before each network validation.

  If the mini-batch size does not evenly divide the number of training samples, then trainNetwork discards the training data that does not fit into the final complete mini-batch of each epoch.

To avoid discarding the same data every epoch, we set the Shuffle value to every-epoch.

- **ValidationFrequency:** Frequency of network validation in number of iterations, specified as a positive integer. The 'ValidationFrequency' value is the number of iterations between evaluations of validation metrics.
  After several tests we chose 32 as the best value.
- **ValidationPatience:** Patience of validation stopping of network training, specified as a positive integer or Inf. The ValidationPatience value is the number of times that the loss on the validation set can be larger than or equal to the previously smallest loss before network training stops.
  To turn off automatic validation stopping, we specify Inf as the ValidationPatience value.

The experiments for each CNNs with the same training options, hardware and partition of the dataset are shown below.

AlexNet The Fig. 1 shows the AlexNet training plots: the plot above represents the validation accuracy through the iterations, instead the graphic below indicates how much the validation loss decreases. We reached, at the end of the training, a validation accuracy of 98.66% and a validation loss of 0.0747. Also, the execution time is of 18 min 46 sec.

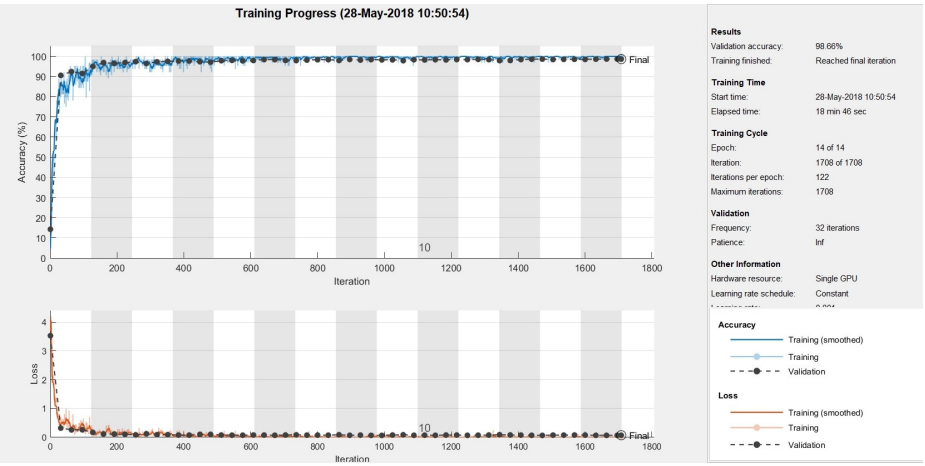

Fig. 1. AlexNet Training Plot.

We estimated the overall accuracy of the network based on a new images. We use the classify function with our network and the prediction subset as parameters; the output of the classify is a matrix containing the prediction labels and the scores. With the right subset labels aviable we could compare this values

with the output of classify function. The final accuracy of the network, that is the estimated mean of the correct predictions, is 0.9787.

To quickly show where our network failed in the prediction, we created the confusion matrix. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The elements on the diagonal are the right predictions: for example the first column represents the class of letter "A"; we can see that 79 images are right recognized and just one images is misclassified as "X". Furthermore, the worst misclassifications performed are "D", "V" and "X".

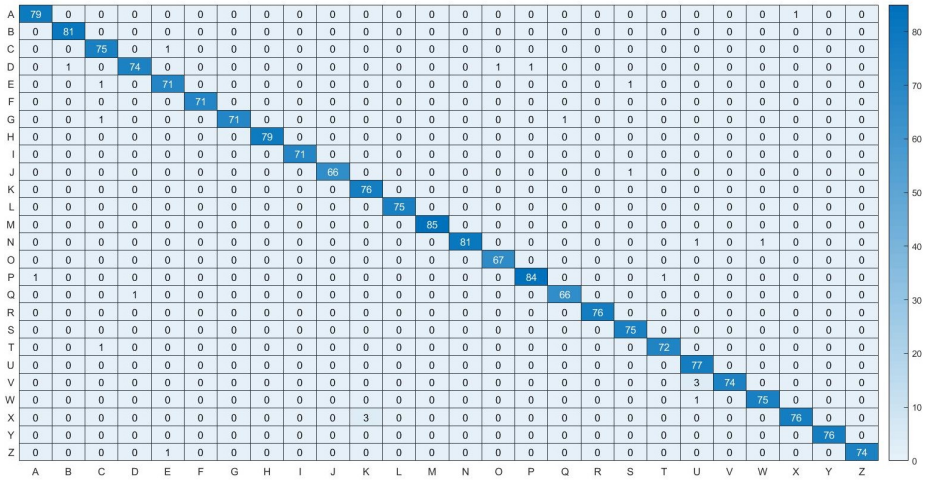

Fig. 2. AlexNet Confusion Matrix.

GoogLeNet The Fig. 3 shows the GoogLeNet training plots. We reached, at the end of the training, a validation accuracy of 96.83% and a validation loss of 0.1291. Also, the execution time is of 39 min 59 sec.

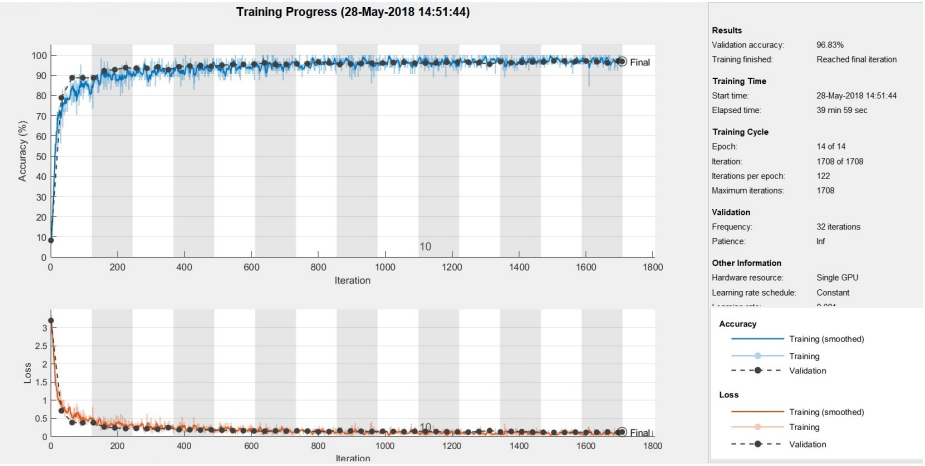

Fig. 3. GoogLeNet Training Plot.

The final accuracy of the network is 0.9635. In the Fig. 4 it is possible to see the confusion matrix of GoogLeNet. Making a comparison with AlexNet we can say that:

- Both misclassified the letter "X", but Alexnet is much better;
- GoogLeNet makes less mistakes only with "G";
- Both don't make errors in recognizing "F","L" and "Y".

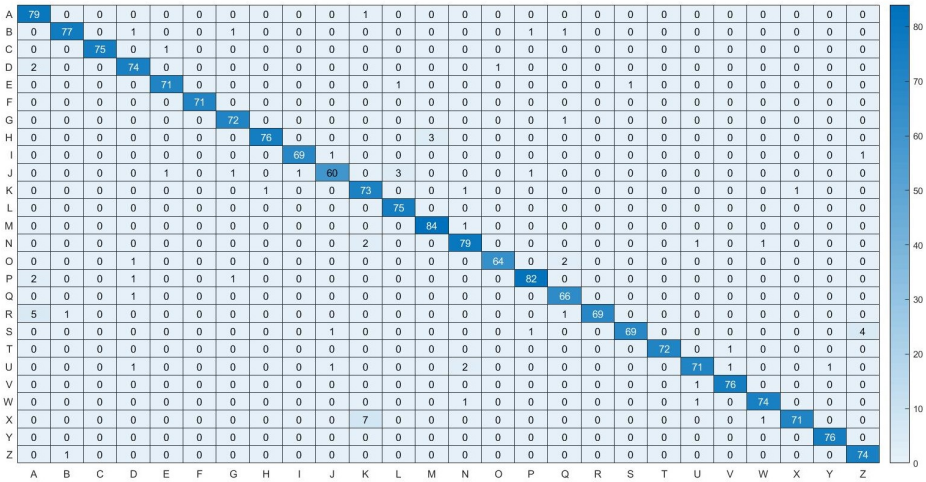

Fig. 4. GoogleNet Confusion Matrix.

Inception-v3 The Fig. 5 shows the Inception-v3 training plots. We reached, at the end of the training, a validation accuracy of 97.79% and a validation loss of 0.1076. Also, the execution time is of 148 min 9 sec.

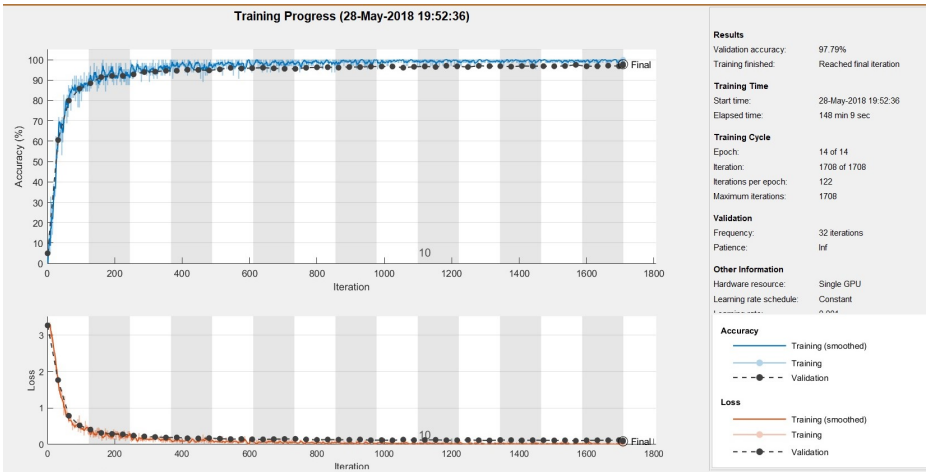

Fig. 5. Inception-v3 Training Plot.

The final accuracy of the network is 0.9817. In the Fig. 6 it is possible to see the confusion matrix of Inception-v3. Making a comparison with AlexNet we can say that:

- Inception-v3 improves the classification of some letters, in particular "V" and "X";
- AlexNet makes less mistakes on the letters "E" and "J", that are the worst misclassification of Inception-v3;

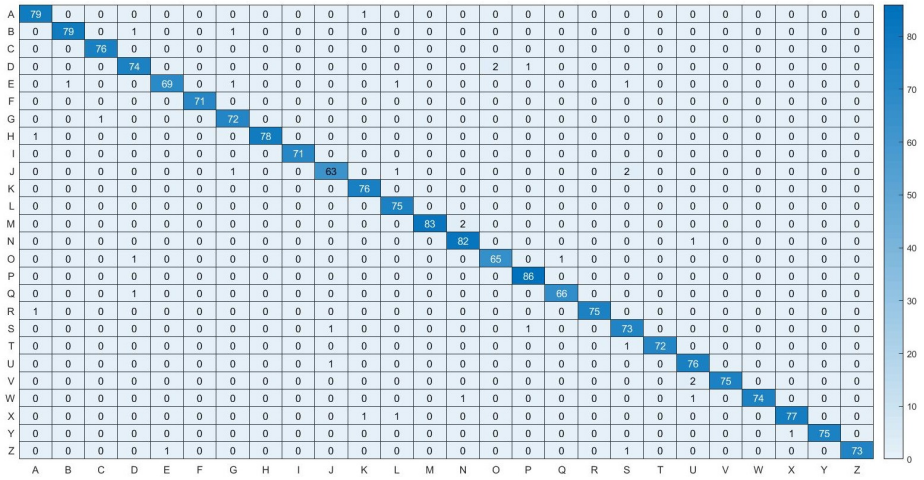

Fig. 6. Inception-v3 Confusion Matrix.

ResNet-50 The Fig. 7 shows the ResNet-50 training plots. We reached, at the end of the training, a validation accuracy of 97.76% and a validation loss of 0.1174. Also, the execution time is of 112 min 31 sec.

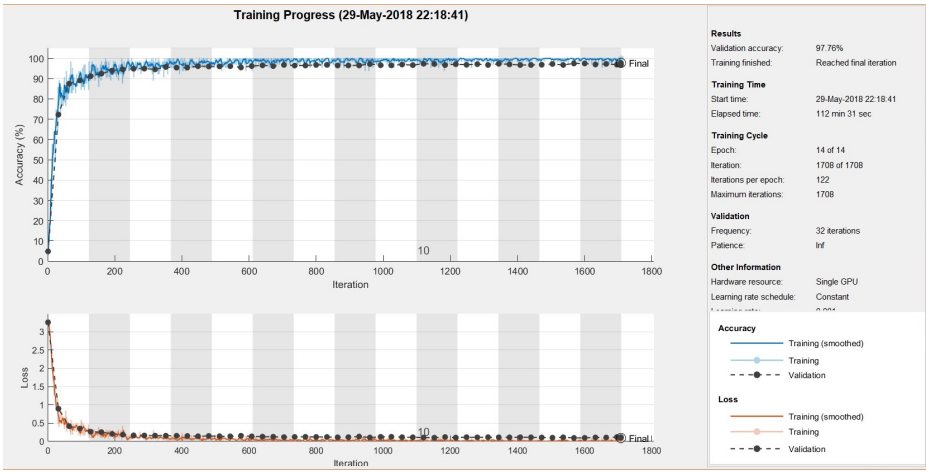

Fig. 7. ResNet-50 Training Plot.

The final accuracy of the network is 0.9802. In the Fig. 8 it is possible to see the confusion matrix of ResNet-50. Making a comparison with AlexNet we can say that:

- Inception-v3 improves the classification of the letters "G", "P" and "V";
- AlexNet makes less mistakes on the letters "J", "N" and "X", that are the worst misclassification of ResNet-50;

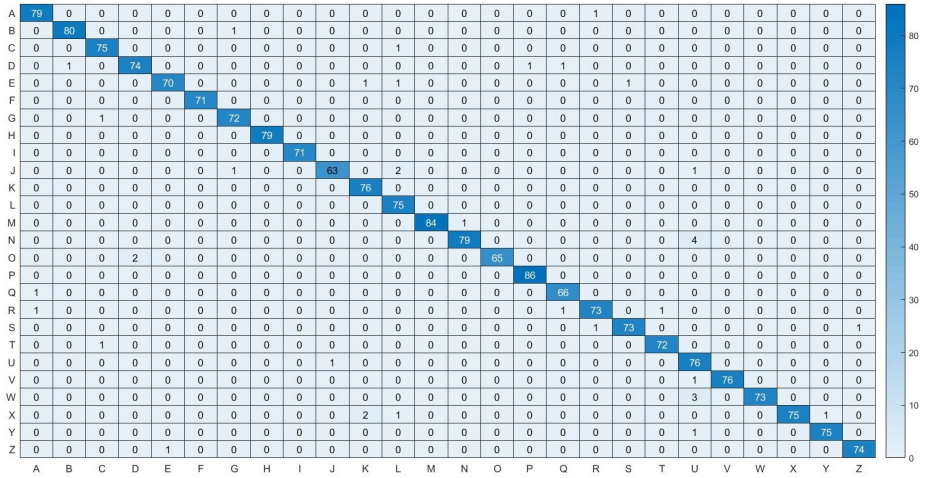

Fig. 8. ResNet-50 Confusion Matrix.

ResNet-101 The last network we trained is ResNet-101. In the Fig. 9 is displayed its training progess. This network is the slowest among the five: it took more than two hours of training, surpassing Inception-v3. The validation accuracy obtained is 97.22% and the validation loss 0.1686; the final accuracy of the network is 0.9701.

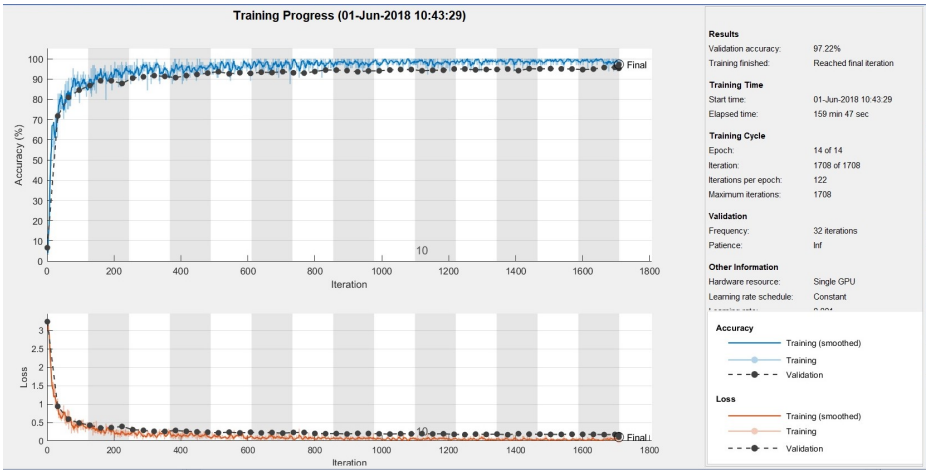

Fig. 9. ResNet-101 Training Plot.

In the Fig. 10 it is possible to see the confusion matrix of ResNet-101. If we compare the results of the present network with the AlexNet ones we see that:

- The classification capacity of D is decreased even more, the mistakes made are 8, in fact Inception-v3 saw 6 "D" as "O" and "P" and 2 as "Q";
- AlexNet made errors, apart from "D", also with "V" and "X" and the situation with Inception is unchanged;
- In addition, the current network misclassified 5 times "N" and "O";
- Fortunately, there are some improvements with Inception, it does not make errors with one letter, the "C", it is also better than AlexNet by one with the letters "P" and "V";

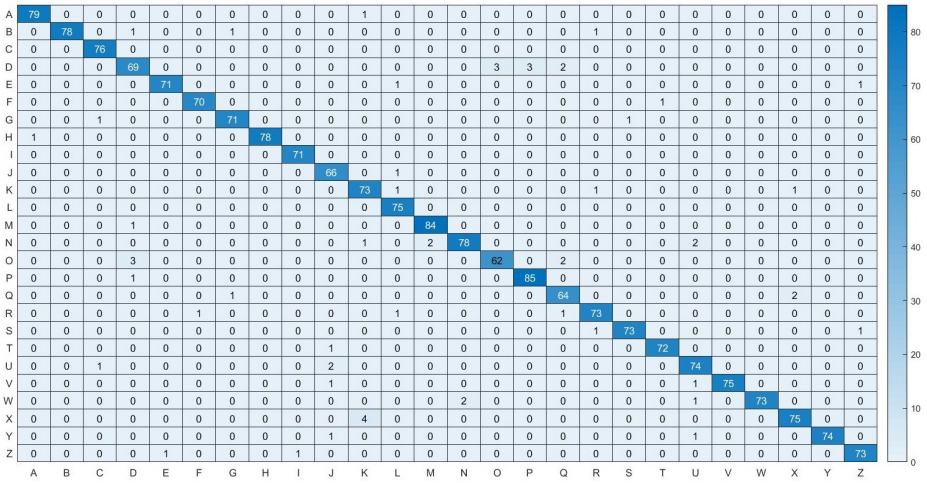

Fig. 10. ResNet-101 Confusion Matrix.

4 Conclusion

In this project, we have performed empirical investigations on Transfer Learning for the classification of handwritten letters. Our experiments were conducted on one big database, result of the merge of two collections of images. The purpose of this project was to study the performance of networks, given the same hardware and parameters. We chose the best parameters for AlexNet and then trained the other networks with them; even in these conditions, they reached an accuracy above the 90Further developments could see the usage of a different machine, more powerful in terms of hardware; this could allow the training of VGG-16 and VGG-19, which we discharged because of their heavy computational burden. For the future, a bigger dataset could be used or a different approach of comparison: for example, instead of using the same parameters for all the models, for each network, it could be found the right combination, that could lead to the highest accuracy. Finally, we can say that the best network is AlexNet, because it was the fastest network to learn and the more accurate of all in the recognition of the letters. Even if the other models accuracy was fair, it cannot justify the computational cost too heavy and the long execution time.

References

1. Lisa Torrey and Jude Shavlik, "Transfer Learning", IGI Global, 2009.
2. T.Semwal, G.Mathur, P.Yenigalla and S.B.Nair, "A Practitioners' Guide to Transfer Learning for Text Classification using Convolutional Neural Networks", 2018.
3. A.Krizhevsky, I.Sutskever and G.E.Hinton, "Imagenet classification with deep convolutional neural networks", NIPS, 2012.

4. H.Lee, R.Grosse, R.Ranganath and A.Y.Ng,"Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations", ICML, 2009.
5. P.Sermanet, D.Eigen, X.Zhang, M.Mathieu, R.Fergus and Y.LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks", CoRR, 2013.
6. J.Yosinski, J.Clune, Y.Bengio and H.Lipson, "How transferable are features in deep neural networks?", NIPS, 2014.
7. MathWorks, https://it.mathworks.com/.
8. GitHub, https://github.com/GregVial/CoMNIST/tree/master/images.