

# A Deep CNN Approach with Transfer Learning for Image Recognition

Cristian Iorga

Department of Applied Electronics and Information Engineering  
“Politehnica” University of Bucharest  
Bucharest, Romania  
ubik3000@gmail.com

Victor-Emil Neagoe

Department of Applied Electronics and Information Engineering  
“Politehnica” University of Bucharest  
Bucharest, Romania  
victoremil@gmail.com

**Abstract**—This paper presents a model of Deep Convolutional Neural Networks (CNN) based on transfer learning for image recognition. This means to use a Deep CNN system pretrained on the large ImageNet dataset of 14 million images and 1000 classes in order to learn feature selection. The results of the pretraining phase are transferred to the problem of classification for the images belonging to the UC Merced Land Use dataset with 21 classes. As benchmark, we have considered a Deep CNN trained with a fraction of the same UC Merced Land Use dataset containing the test images for classification. The experimental results have pointed out the obvious advantage of the Deep CNN with transfer learning (accuracy of 0.87 using pretraining over 0.46 for fully training on the same dataset).

**Keywords**—Deep Learning (DL), Convolutional Neural Networks (CNN), image classification, Transfer Learning, Feature Selection

## I. INTRODUCTION

Much of the modern innovations in image recognition is reliant on Deep Learning technology. Deep learning is a class of Machine Learning algorithms that use Artificial Neural Networks with multiple layers, where progressively higher level features gets extracted from raw inputs. For example, in the case of image recognition, edges or curves are identified using the lower layers of Neural Networks, while human-identifiable items such as trees, cars, vegetation, etc. are identified in the higher layers. Convolutional Neural Networks (CNN) are a class of Artificial Neural Networks inspired by biology that are very effective for a variety of applications, especially for image recognition. Training CNNs, however, has a high cost: large quantities of data are required, and the process of learning takes large amounts of time, processing power and storage. High accuracy of classification necessitates hundreds of thousands or millions of samples (images) to train from.

There are specific cases where the image set used for training is orders of magnitude smaller than millions of samples, so how can effective performance be achieved from a deep neural network such as a CNN? One method that is used in the domain of Machine Learning is known as Transfer Learning, and it works by using a CNN trained on a large dataset to learn essential features that can be transferred to another related problem domain. The pretrained NN forms

the base upon which a more refined network is trained on a smaller dataset specific to the new problem domain.

The expression of Transfer Learning, in the case of image classification, is the use of **pretrained models**. A pretrained model is a model already trained on a large benchmark dataset, capable of solving problems similar to a newly encountered problem that needs solving. The common practice is to use models already proven and publicly available (e.g. [VGG](#), [Inception](#), [DenseNet](#)), because of the high computational cost of training such models or because a desired classification performance is required.

In this paper we propose to apply a Deep CNN model with Transfer Learning for image classification using feature selection.

The paper is organized as follows. Chapter II presents the proposed model. Experimental results are given in chapter III. Chapter IV is dedicated to concluding remarks.

## II. PROPOSED MODEL OF DEEP CNN WITH TRANSFER LEARNING FOR IMAGE RECOGNITION

The proposed model starts from a classical Deep Convolutional Neural Network. Deep neural models consist of feature detector units structured in layers. Lower layers detect simple features and using the model weights or parameters learned at that scale it feeds them as inputs into next higher layers in the hierarchy, which will detect more complex features.

Automatically learning **hierarchical feature representations** is an important characteristic of deep Convolutional Neural Networks, meaning that features extracted by the first layer are general and can be reused in other problem domains, while the next layers extract features that are more and more specific and are dependent of the dataset and task at hand. According to Yosinski et al. [1], ‘*if first-layer features are general and last-layer features are specific, then there must be a transition from general to specific somewhere in the network*’. In consequence, general features are extracted by CNN’s convolutional base (the lower layers following inputs) and, in contrast, some of the higher layers of the convolutional base and the classifier extract the specialized features (and of course, more complex).

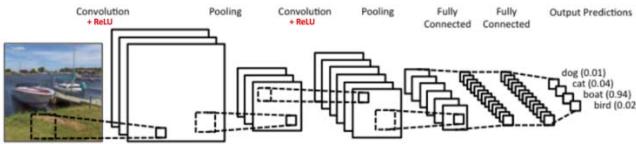


Fig. 1. Typical CNN architecture

In order to use Transfer Learning methodology, a VGG16-based architecture CNN, pretrained on ImageNet, loaded from `torchvision.models` of PyTorch ML framework, has been used, where the final classifier of the model has been replaced by a **Multilayer Perceptron** (MLP) with 3 fully connected layers, with ReLU activation function and dropout added.

VGG16 is a CNN model that has been proposed by K. Simonyan and A. Zisserman [2]. It achieves for top-5 test accuracy a percentage of 92.7% in ImageNet. VGG16 has been trained for weeks using NVIDIA Titan Black GPUs.

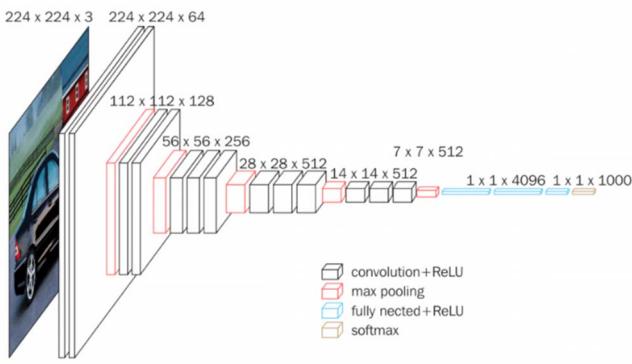


Fig. 2. Architecture of VGG16

A fixed size  $224 \times 224$  RGB image is the input to conv1 convolutional layer. Input image passes through a stack of convolutional layers, where the filters have a very small receptive field of  $3 \times 3$  size (the smallest size capable of capturing notions of center, up/down, left/right). For one of the configurations,  $1 \times 1$  convolution filters are used, representing a linear transformation of the input channels (with non-linearity following). The stride of convolution is fixed to 1 pixel; spatial padding of the convolutional layer input is constructed such that after convolution the spatial resolution is maintained (for  $3 \times 3$  convolutional layers padding is 1-pixel). Five max-pooling layers carry out spatial pooling. Some of the convolutional layers (not all) are followed by max-pooling layers, which perform over a  $2 \times 2$  pixel window, with stride dimension 2.

The stack of convolutional layers (with different depth depending on the architecture specifics) are followed by 3 fully-connected (FC) layers: the first two layers each having 4096 channels, while the third one performs classification into 1000 categories and in consequence has 1000 channels. A soft-max layer is the final one. The non-linearity of all hidden layers is computed using the rectification (ReLU)

function. VGG16 contains a total of 16 layers, 13 of those are convolutional and the rest of 3 are fully connected ones.

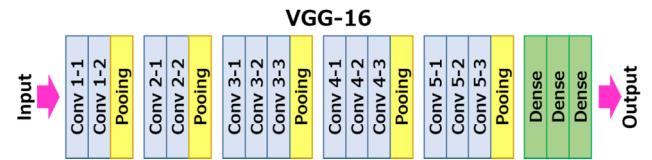


Fig. 3. VGG16 arch details

*PyTorch* is a Python-based open-source machine learning framework that supports research and development in Computer Vision, NLP and other domains of AI/Machine Learning. It mainly provides two high-level features:

- GPU-accelerated Tensor computation;
- Deep neural networks models built on a tape-based autograd system.

The package `torchvision` has several Deep CNN already defined (*Inception*, *VGG*, etc) and ready to use, and every model can be used either pretrained or not.

All models of `torchvision models` have been pretrained on the 1000-class Imagenet dataset. **ImageNet** is an image database of over 14 million samples organized according to the [WordNet](#) hierarchy (currently, the classes are defined only as nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently there is an average of over five hundred images per node. The images have been collected from the web and labeled by humans using Amazon's Mechanical Turk crowd-sourcing tool.

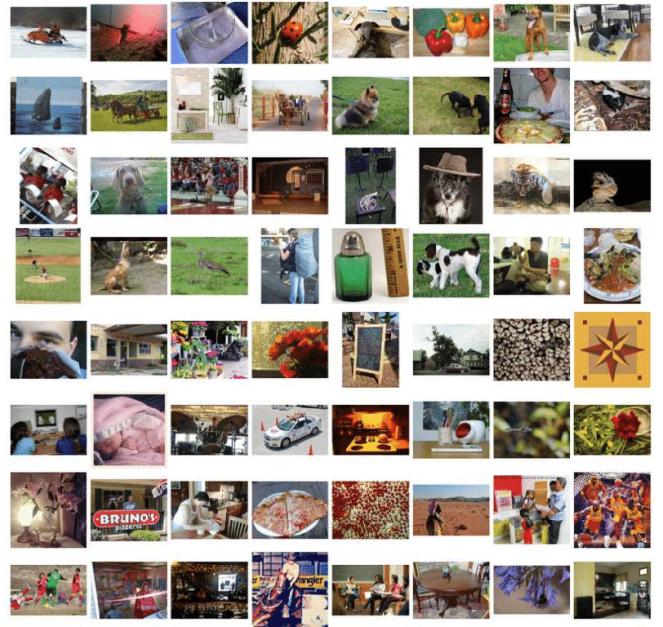


Fig. 4. Examples of images from Imagenet dataset

The Transfer Learning method applied in the case of our model is repurposing a pretrained model. This method involves replacing the original classifier with a new classifier fit to the current purpose. Fine-tuning of the model was performed using the strategy of freezing the convolutional base. The chosen strategy corresponds to an extreme of the trade-off between train and freeze. The base concept here is to keep the original form of the convolutional base and then feed the classifier with the outputs of the frozen convolutional base. The pretrained model is used as a fixed feature selection mechanism, which is useful or even necessary if there is a shortage of computational power or in case of small datasets, and/or the pretrained model solves a problem similar to the new one.

Image classification models resulted using a transfer learning method based on pretrained CNNs usually contain two parts:

1. **Convolutional base**, performing feature selection.
2. **Classifier**, recognizing the input image based on the features previously extracted.

Since we are now focusing on the classifier part, the approach followed here was to use a neural network with fully connected layers. In the domain of image classification, one approach is to use a stack of fully-connected layers followed by a final layer which is softmax activated [2-4]. The probability distribution over each class is provided by the outputs of this final layer.

The process to use a pretrained model is as follows:

1. Load in pretrained weights from a network trained on a large dataset, in our case VGG16 pretrained on ImageNet;
2. Freeze all the weights in the lower (convolutional) layers: the layers to freeze are adjusted depending on similarity of new task to original dataset, in our case all convolutional layers have been frozen;
3. Replace the upper layers of the network with a custom classifier; the number of outputs must be set equal to the number of classes;
4. Train only the custom classifier for the current task, in consequence optimizing the model for the current dataset.

Loading in a pretrained model in PyTorch is simple:

```
from torchvision import models
model = models.vgg16(pretrained=True)
```

VGG16 model has over 130 million parameters, but will train only the very last few fully-connected layers. Initially, we freeze all of the model's weights:

```
# Freeze model weights
for param in model.parameters():
    param.requires_grad = False
```

Then, we add on our own custom classifier with the following layers:

```
(classifier): Sequential(
    (dropout): Dropout(p=0.5)
    (inputs): Linear(in_features=25088,
        out_features=84, bias=True)
    (relu1): ReLU()
    (hidden_layer1): Linear(in_features=84,
        out_features=63, bias=True)
    (relu2): ReLU()
    (hidden_layer2): Linear(in_features=63,
        out_features=42, bias=True)
    (relu3): ReLU()
    (hidden_layer3): Linear(in_features=42,
        out_features=21, bias=True)
    (output): LogSoftmax()
)
```

by using:

```
def model_setup(dropout=0.5, hidden_dim =
len(class_indices), lr = 0.0001):

    model = models.vgg16(pretrained=True)

    for param in model.parameters():
        param.requires_grad = False

    from collections import OrderedDict
    classifier = nn.Sequential(OrderedDict([
        ('dropout', nn.Dropout(dropout)),
        #dense121:('inputs', nn.Linear(1024,
        hidden_dim * 4)),
        ('inputs', nn.Linear(25088, hidden_dim * 4)),
        ('relu1', nn.ReLU()),
        ('hidden_layer1', nn.Linear(hidden_dim * 4,
        hidden_dim * 3)),
        ('relu2', nn.ReLU()),
        ('hidden_layer2', nn.Linear(hidden_dim * 3,
        len(class_indices) * 2)),
        ('relu3', nn.ReLU()),
        ('hidden_layer3', nn.Linear(hidden_dim * 2,
        hidden_dim)),
        ('output', nn.LogSoftmax(dim=1))
    ]))

    model.classifier = classifier
    criterion = nn.NLLLoss()
    optimizer = optim.Adam(model.classifier.parameters(), lr )
    model.to(device)
    print(model)
    return model, optimizer, criterion
model, optimizer, criterion = model_setup()
```

When the extra layers are added to the model, they are set to trainable by default (`require_grad=True`). For the VGG16, we're only changing the very last original fully-connected layer. All of the weights in the convolutional layers and the first 5 fully-connected layers are not trainable.

The final architecture of our model as presented by PyTorch is:

```

VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2,
    padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=2, stride=2,
    padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=2, stride=2,
    padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=2, stride=2,
    padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): MaxPool2d(kernel_size=2, stride=2,
    padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (dropout): Dropout(p=0.5)
    (inputs): Linear(in_features=25088,
    out_features=84, bias=True)
    (relu1): ReLU()
    (hidden_layer1): Linear(in_features=84,
    out_features=63, bias=True)
    (relu2): ReLU()
    (hidden_layer2): Linear(in_features=63,
    out_features=42, bias=True)
    (relu3): ReLU()
    (hidden_layer3): Linear(in_features=42,
    out_features=21, bias=True)
    (output): LogSoftmax()
  )
)

```

Then, the final classifier layers (the 3 fully connected ones) of the pretrained CNN have been trained using backpropagation. Loss and accuracy on the validation set were tracked to determine the best hyperparameters.

### III. EXPERIMENTAL RESULTS

Accuracy of image classification has been used to evaluate the performance for our model. This accuracy is expressed by both correct classification rate and also by confusion matrix. A number of 15 epochs has been used for the training phase. We have chosen ADAM optimizer for CNN learning. The training and evaluation phases of the model were ran using GPU acceleration.

#### A. Data set

For training, [UC Merced Land Use Dataset](#) was used and it has 21 categories, with each category containing 100 images. Images of the dataset were extracted from large images present in the collection of “USGS National Map Urban Area Imagery”, reflecting various urban areas around the USA. The pixel resolution corresponds to 1 foot. The dataset has been split further along into 3 partitions:

- Training set (80%)
- Validation set (10%)
- Tests set (10%)

The classes of the considered dataset are given in Table I.

TABLE I. CLASSES OF UC MERCED LAND USE DATASET

Categories	Categories	Categories
- agricultural	- forest	- overpass
- airplane	- freeway	- parkinglot
- baseballdiamond	- golfcourse	- river
- beach	- harbor	- runway
- buildings	- intersection	- sparseresidential
- chaparral	- mediumresidential	- storagetanks
- denseresidential	- mobilehomepark	- tenniscourt

Each image measures 256x256 pixels, with RGB channels. In Table II. there are shown 3 examples of classes.

TABLE II. CATEGORIES EXAMPLES FOR UC MERCED LAND USE DATASET

Category	Image
airplane	
intersection	
river	

We have used a VGG16-based CNN architecture as benchmark model; this has been completely trained on our UC Merced Land Use dataset. On this benchmark model the final classifier has been replaced by an MLP with 3 fully connected layers, using ReLU activation function and dropout. Thus, the architecture of the proposed and benchmarked model remains the same.

For comparative analysis purpose, a classic Multilayer Perceptron (MLP) having three hidden fully-connected layers and also trained on UC Merced Land Use dataset has been used as a secondary benchmark model. The results of performance evaluation are shown in Table III.

TABLE III. CORRECT CLASSIFICATION RATE FOR THE PROPOSED AND BENCHMARKED MODELS

Architecture \ Accuracy	VGG16 pretrained	VGG16 fully-trained	MLP 3 FC hidden layers
Validation	0.9464	0.4509	0.3125
Test	0.8661	0.4598	0.3170

### B. Confusion Matrix

Confusion matrix has been computed for the models on the test partition of the dataset (see Table IV and Table V).

TABLE IV. CONFUSION MATRIX OF THE PROPOSED DEEP CNN MODEL

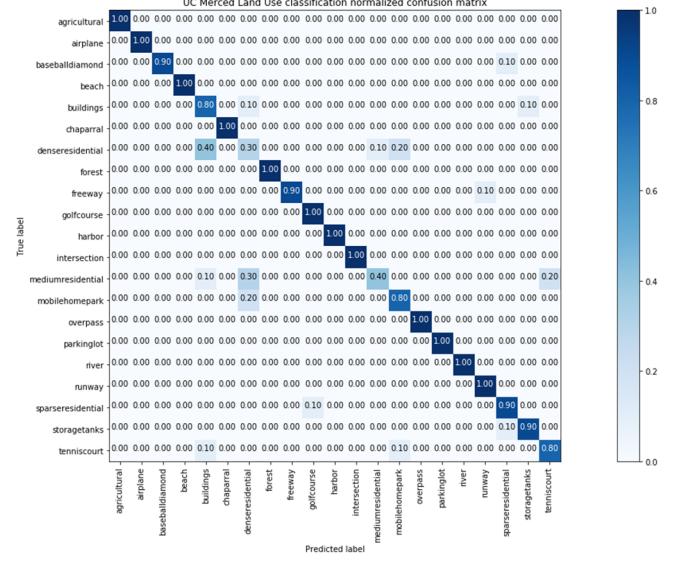
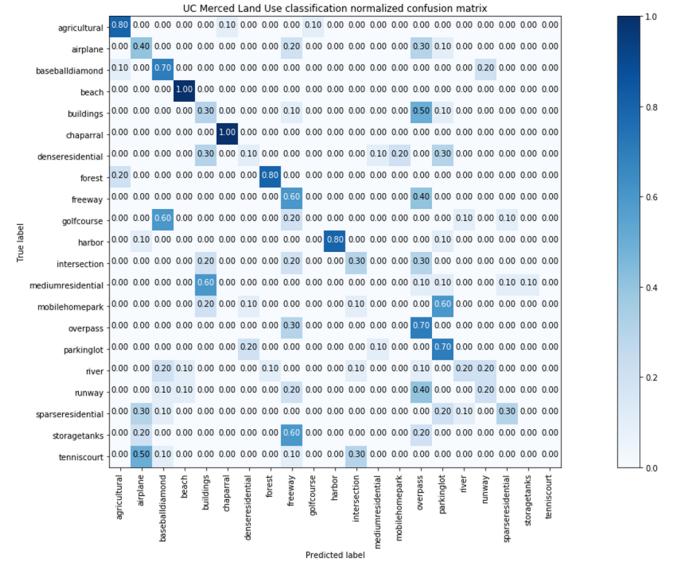


TABLE V. CONFUSION MATRIX OF THE BENCHMARK MODEL



## REFERENCES

### C. Detection example for correct classification

In Fig. 5. we can evaluate the advantage of the proposed pretrained model by comparison with the fully-trained benchmark one, consisting in its ability to provide a class prediction with stronger accuracy.

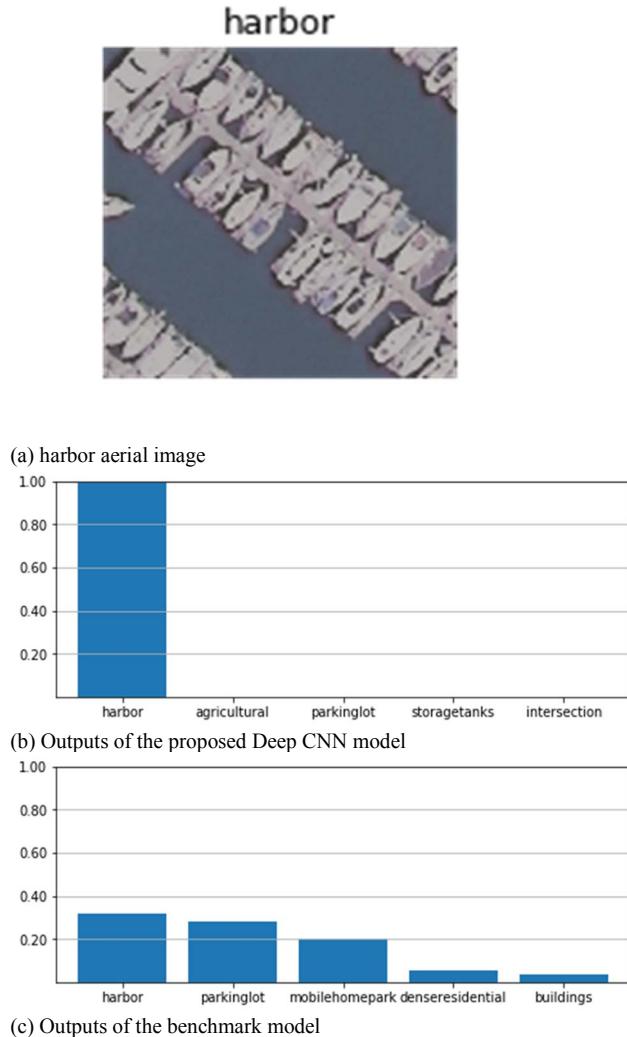


Fig. 5. Detection examples for correct classification

## IV. CONCLUDING REMARKS

In this paper we have applied a model of Deep CNN with Transfer Learning for image recognition. The CNN system with an architecture of 16 layers is previously trained on the big **ImageNet** dataset of 14 million images for learning feature selection. Then, the pretrained CNN system has been trained and tested using the UC Merced Land Use dataset with 21 classes. The CNN with transfer learning has shown its performance on the test set, leading to an accuracy of 0.87, by comparison to a CNN system with the same architecture fully trained on the same UC Merced Land Use dataset, that has led to an accuracy of only 0.46.

- [1] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?”, in Advances in neural information processing systems, pp. 3320–3328, 2014.
- [2] K. Simonyan, and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, CoRR, 2014.
- [3] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
- [4] M.D. Zeiler, and R. Fergus, 2014, “Visualizing and understanding convolutional networks”, in European conference on computer vision, pp. 818–833, Springer, Cham, 2014.
- [5] A. Krizhevsky, I. Sutskever, G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, in Advances in Neural Information Processing Systems 25 (NIPS’11), pp. 1106-1114, 2012.
- [6] J. Patterson, A. Gibson, Deep Learning-A Practitioner’s Approach, O’Reilly Media, Inc., USA, 2017.
- [7] N. Buduma, N. Lacascio, Fundamentals of Deep Learning, O’Reilly Media, Inc., USA, 2017.
- [8] C. Bishop, Pattern Recognition and Machine Learning, Springer, New York, 2006.
- [9] S. Shalev-Shwartz, S. Ben-David, Understanding Machine Learning-From Theory to Algorithms, Cambridge Press, 2014.
- [10] V. Neagoe, Retele neurale pentru explorarea datelor (Neural Networks for Data Mining), MatrixRom, Bucharest, 2018.
- [11] W.L. Chao, Machine Learning Tutorial, National Taiwan University, 2011.
- [12] A. P. Barar, V. E. Neagoe, N. Sebe, “Image Recognition with Deep Learning Techniques”, in Recent Advances in Image, Audio, and Signal Processing, Proc. 1st International Conference on Image Processing and Pattern Recognition (IPPR13), pp. 126-132, Budapest, Dec. 10-12 2013.
- [13] V. E. Neagoe, A. P. Barar, N. Sebe, P. Robit, “A Deep Learning Approach for Subject Independent Emotion Recognition from Facial Expressions”, in Recent Advances in Image, Audio, and Signal Processing, Proc. 1st International Conference on Image Processing and Pattern Recognition (IPPR13), pp. 93-98, Budapest, Dec. 10-12, 2013.