
**University of Cagliari**

Master degree in Informatics

# Supermarket images classification using pretrained networks and linear SVM

Deep Learning Applications Project

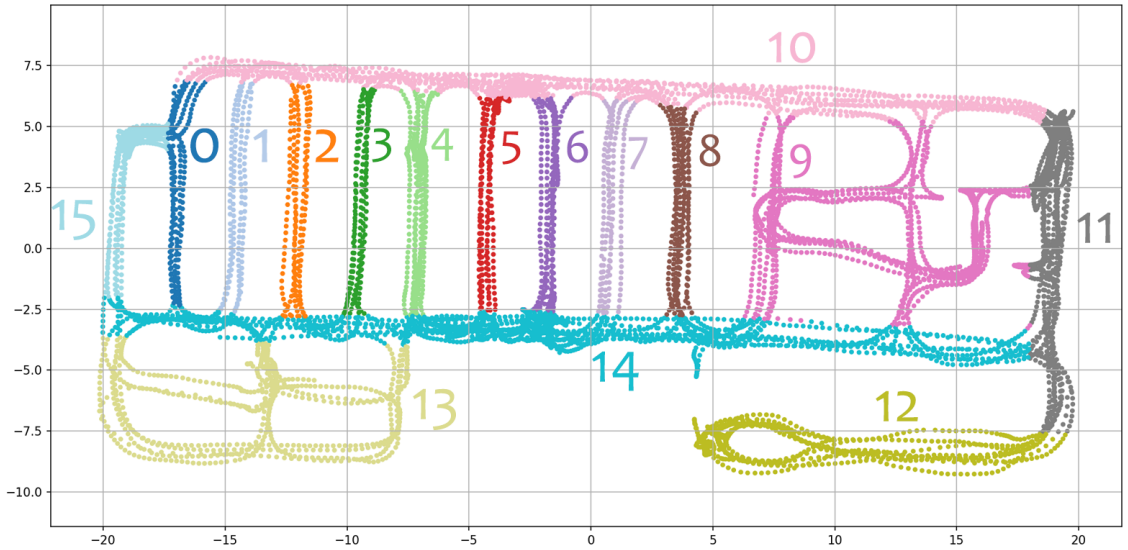
Andrea Piras 65211
Matteo Anedda 65212

# Index

# Introduction

The objective of this project is to try to understand the position of a person in a supermarket using image classification and deep learning.

The images of the supermarket are taken from a dataset. These images have been taken by a camera attached to a cart that followed some routes inside the supermarket. After the acquisition they have been divided into 16 classes/routes.




With this dataset we will perform three different tests:

- Feature extraction using three pretrained networks:
  - AlexNet
  - ResNet18
  - VGG16

  and perform the classification using linear SVMs.
- Fine tuning using an ImageNet pretrained network
- Classification creating a new network

# Useful Links

- Github project
- Original Dataset
- Original Full Size Dataset
- Organized Dataset

## Project Structure

```
.
| Folders
├── img   # Images for the readme
├── doc   # Report folder
│   └── Report.pdf   # Report
├── split  # Split folder
│   └── split.sh   # Split bash file
├── *TrainingSet*   # Folder with the images of the training set
│   ├── 00
│   ├── 01
│   ├── ...
│   ├── 14
│   └── 15
├── *TestSet*   # Folder with the images of the test set
│   ├── 00
│   ├── 01
│   ├── ...
│   ├── 14
│   └── 15
├── *ValidationSet*   # Folder with the images of the validation set
│   ├── 00
│   ├── 01
│   ├── ...
│   ├── 14
│   └── 15
|
| Markdown
├── README.md
├── PRETRAINED.md
├── FINETUNING.md
├── NEWNETWORK.md
|
| Liblinear
├── libsvmread.mexw64   # Read
├── libsvmwrite.mexw64   # Write
├── train.mexw64   # File with train function
├── predict.mexw64   # File with predict function
|
| Matlab Script
├── fine_tuning.mlx   # Script for the AlexNet fine tuning
├── new_network.mlx   # Script for new Network
└── pretrained_networks.mlx   # Script pretrained feature extraction
```

Folders with * are not included in the repository.

# Dataset Organization

The first step to do is to organize the images into folders. On the site it is possible to download the zip with the dataset. This file is a reduced version of another dataset and is made by a folder that contains all the images and three csv files:

- **training_set**: that contains all the images to use to train the pretrained network
- **validation_set**: that contains all the images to use to get an unbiased evaluation of the model
- **test_set**: that contains all the images to use to test the algorithm

The first two csv files have 6 columns:

- Image name
- x coordinate
- y coordinate
- u coordinate
- z coordinate
- Class

The last csv file has just the image file names, so there are no labels.

We used just the first two csv files and we removed all the coordinate columns from them, because the position in which the photo was taken it's not necessary. So just the name of the file and the class have been used.

In both the training and the validation set, using a bash script file, all the images have been divided into folders from 00 to 15 based on their class.

The **test_set** csv file hasn't been taken from the reduced dataset. So we downloaded the original dataset and we took the test set txt file. From this txt file we removed the coordinate columns and we used just the name of the file and the class.
At the end, using a bash script file, we divided the images into folders from 00 to 15 based on their class, like we did for the validation and training set.

So there are three folders:

1. **TestSet**: in which there are all the test set images
2. **ValidationSet**: in which there are all the validation set images
3. **TrainingSet**: in which there are all the training set images

The images folder won't be in the repository because the size is too high for github. The **organized dataset** that has been used can be downloaded here.

# Feature extraction using pre-trained networks

This section will show the pretrained networks that have been used in this project, what the code does and the result obtained.

## The pretrained networks

A pretrained network is a network that has already learned to extract powerful and informative features from natural images. It can be used as a starting point to learn a new task.

Many of the pretrained networks used in this project have been trained with the same ImageNet database. These pretrained networks can classify images into 100 object categories, such as keyboard, mouse, pencil and many animals.

### Alex Net

AlexNet is a convolutional neural network that is 8 layers deep.
The network has an image input size of 227x227.

To see the structure of the network in matlab you have to put these lines in the command window:

```
net = alexnet;
```

After

```
net.Layers;
OR
analyzeNetwork(net)
```

### Res Net 18

The ResNet-18 is a convolutional neural network that is 18 layers deep.
The network has an image input size of 244x244.

To see the structure of the network in matlab you have to put these lines in the command window:

```
net = resnet18;
```

After

```
net.Layers;
OR
analyzeNetwork(net)
```

### VGG 16

VGG16 is a convolutional neural network that is 16 layers deep. The network has an image input size of 224x224.

To see the structure of the network in matlab you have to put these lines in the command window:

```
net = vgg16;
```

After

```
net.Layers;
OR
analyzeNetwork(net)
```

## How the script works

This section will explain how the project works.

### Variables tuning

In the first part of the code it is possible to configure the code variables. This part is useful to enable or disable some parts of the code and to choose which pretrained network to use.

#### Print configuration

These variables enable or disable some part of the code.

1. Print random images of the training set (0 disabled / 1 enabled)

```
print_training_set = 0;
```

2. Print 12 random images of the test set (0 disabled / 1 enabled). For each of them it shows the number of the image, the prediction of the model and the correct class.

```
print_test_set = 0;
```

3. Print the confusion matrix (0 disabled / 1 enabled).

```
print_conf_matr = 0;
```

**Network selection**

It is possible to select one of the pretrained networks between **AlexNet**, **ResNet-18** and **VGG16**.

```
network = "alexnet";
% network = "resnet";
% network = "vgg"
```

## Import the dataset and split the training set

In the second part of the code there will be the import of all the images, using *imageDataStore*, a function that automatically labels all the images based on the folder names. The images will be stored into an *ImageDataStore* object.

So the program takes the test set images from the folder **TestSet** and it stores them into an *ImageDataStore* object. The same thing for the training set.

## Image resize

The networks require different input sizes, in this section the image will be resized to fit the first input layer. To automatically resize the training and test images before they are used as input by the network, the program creates augmented image datastores, it specifies the desired image size, and it uses these datastores as input arguments to activations.

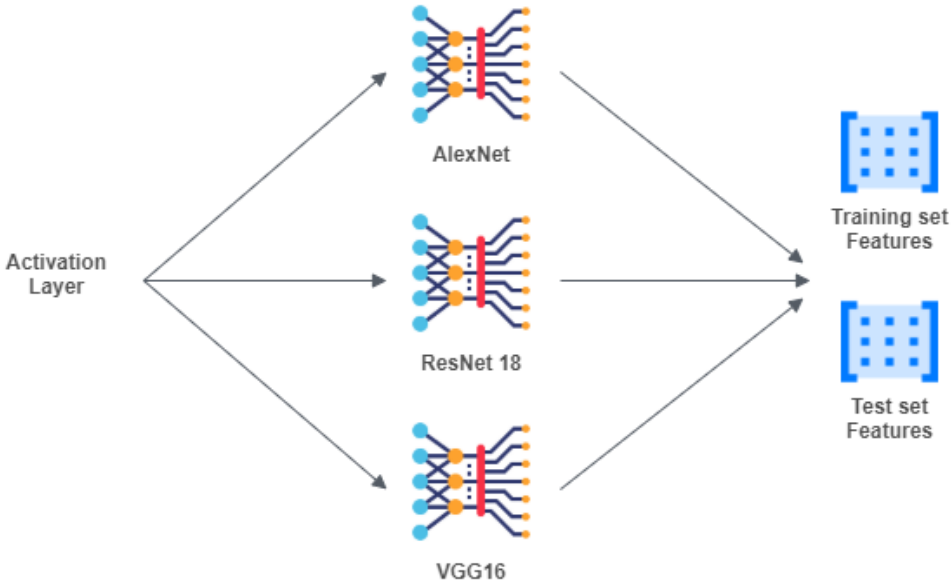

## Select the activation layer for the feature extraction and extract the features

The network constructs a hierarchical representation of input images. Deeper layers contain higher-level features, constructed using the lower-level features of earlier layers.

To get the feature representations of the training and test images, the program will use activations on different layers depending on the network used.

In the project case for **alexnet** is **fc7**, for **resnet18** is **pool5** and for **vgg16** is **fc7**.
This parameter can be changed. Basically the code is extracting the feature from the layer before the layer that actually classifies the things.

At the end of this step there will be the features of the training and test sets.



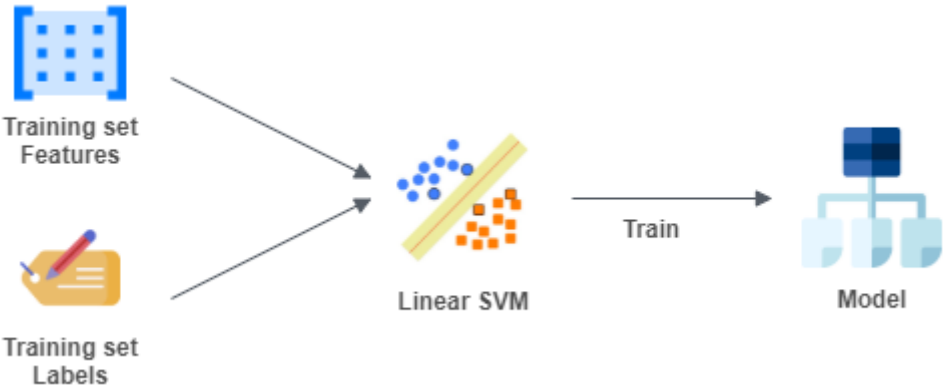

## Classification

The next step is to train the linear SVM model using the training set features and labels, and after to perform the classification using the model, the feature of the test set and the labels of the test set. At the end the program computes the accuracy.

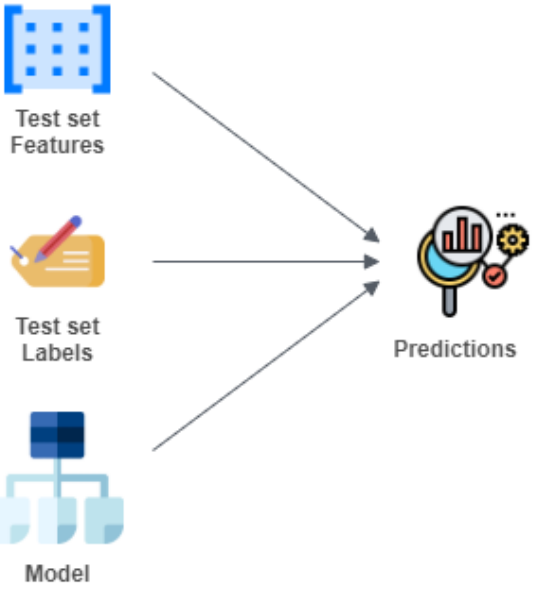
The library used to train linears svm and to perform the classification is liblinear. To use this library is necessary a conversion of the data to the one compatible with liblinear. After, the program trains the model passing the labels and the features. Next it performs the predictions using the labels and the features of the test set and the model generated before. At the end it computes the accuracy.

In the first rows of this section of the code there is the conversion of the data to the one compatible with liblinear.

In the next part of the code there is training of the model, passing to the function the labels and the features. To train the model there is an option (*-s 2*) to use the **L2-regularized L2-loss support vector classification (primal**). The default is **L2-regularized L2-loss support vector classification (dual)**, but it gives many warnings because it reaches the max number of iterations.




After there is the prediction using the labels and the features of the test set and the model generated before.




At the end the program computes the accuracy.

```
YTrain = double(YTrain(:,1));
YTest = double(YTest(:,1));
featuresTrain = sparse(double(featuresTrain));
featuresTest = sparse(double(featuresTest));
model = train(YTrain, featuresTrain, '-s 2');
YPred = predict(YTest, featuresTest, model);
accuracy = mean(YPred == YTest)
```

# Test and output analysis

The program ran with every net and for each of them the accuracy, the images correctly classified vs. the number of images and the time elapsed has been compared. It's important to say that AlexNet and ResNet18 ran using the GPU, VGG16 with CPU because it is too big for the GPU.

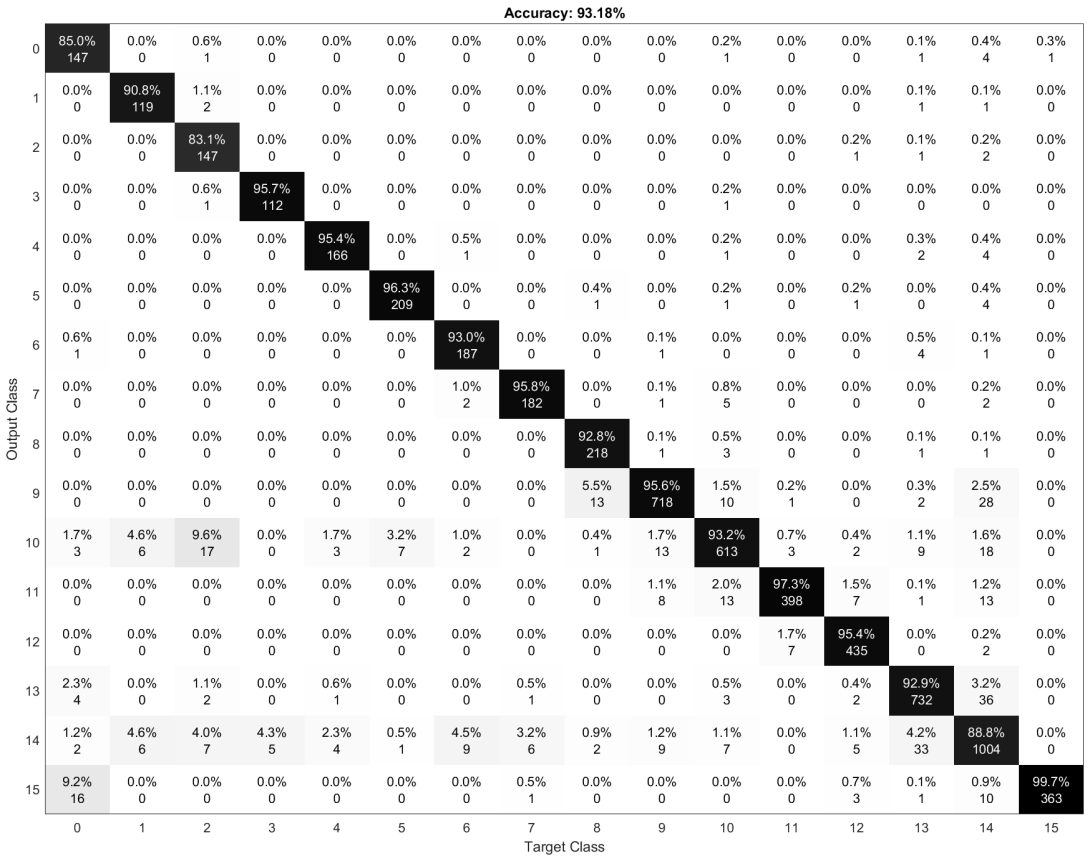
For each pretrained network the confusion matrix has been plotted to understand in which classes there are more errors.

| Pretrained Network | Accuracy | Correct classified vs. No. Images | Time elapsed (s) | Time elapsed |
|---|---|---|---|---|
| AlexNet | 93.18% | 5750 / 6171 | 63.71 | 1 min 3 s |
| ResNet18 | 91.41% | 5641 / 6171 | 25.06 | 25.06 s |
| VGG16 | 92.72% | 5722 / 6171 | 1976.52 | 32 min 56 s |

Here it's possible to see that the accuracy is higher on **AlexNet**, but **ResNet18** is the fastest in terms of time elapsed.

## Confusion matrix

### AlexNet

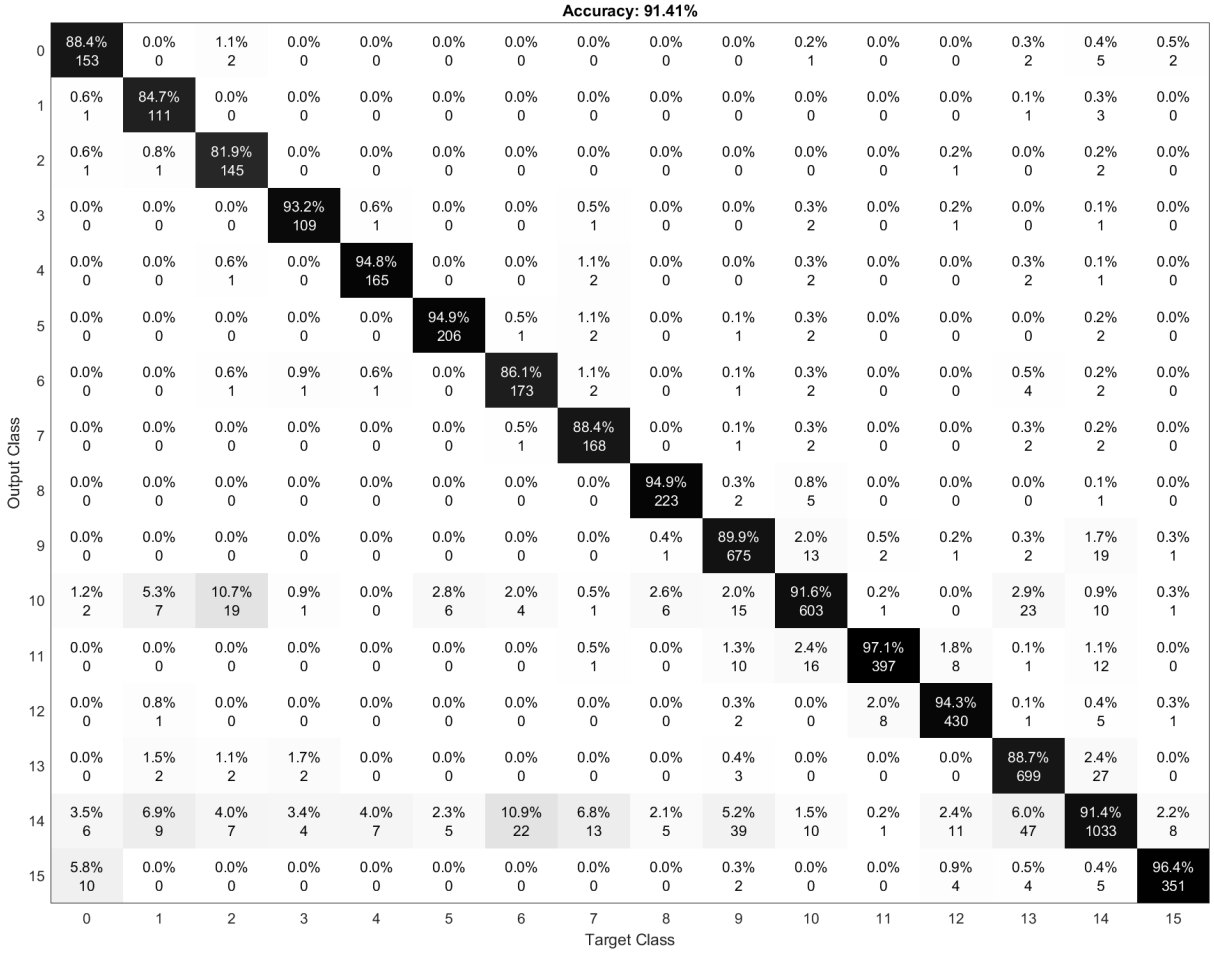


In this confusion matrix it's possible to see that there are many images of the class 0 classified as 15. The main reason is that 15 has a link with class 0. The errors come from photos that are between 0 and 15, for example a photo of the route 0 in which it is possible to see the route 15 and so on.

It is also possible to notice images of route 2 classified as 10. Analyzing the map it is possible to see that 2 and 10 are linking routes.

The routes 10 and 14 are the ones that produce the most errors, in fact analyzing row 10 and 14 of the confusion matrix we can see that for almost every class there are misclassified images.

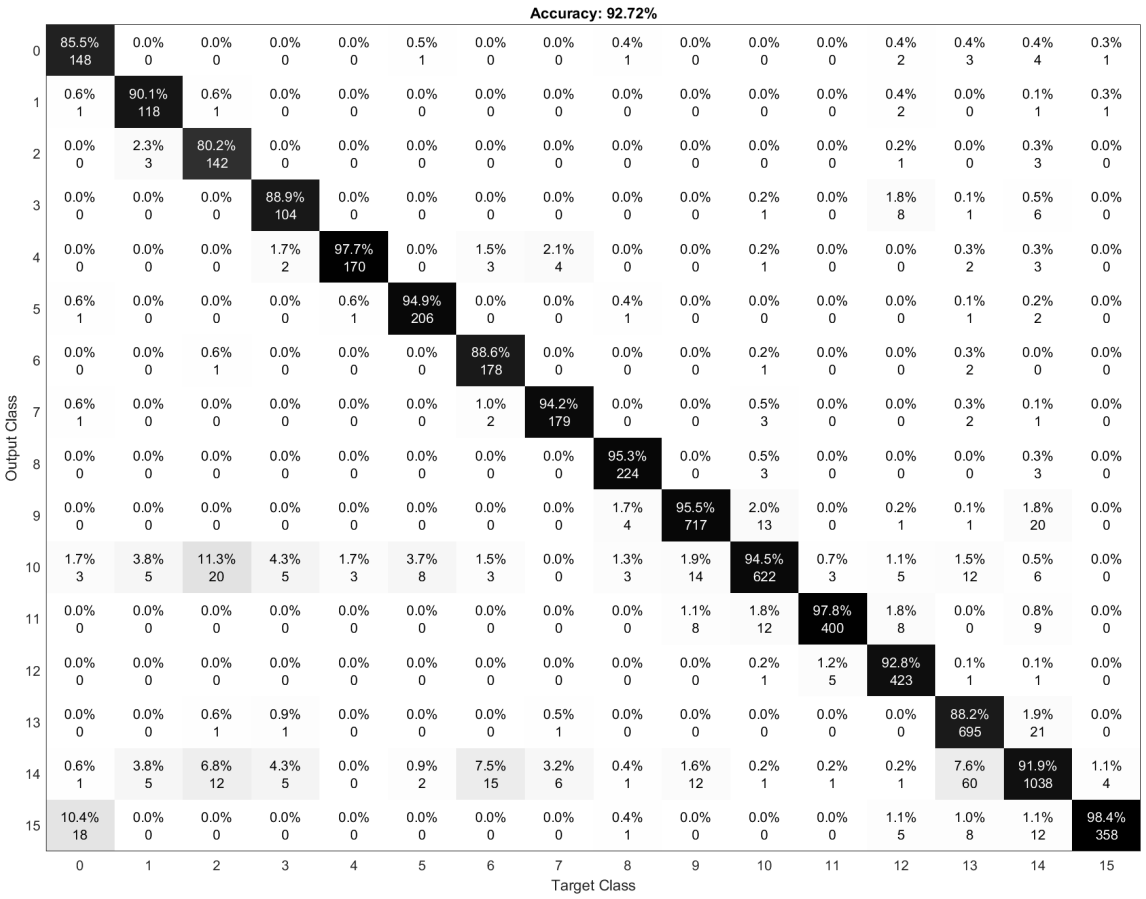
**ResNet 18**




Even in this confusion matrix we can see elements of class 0 classified as 15. The reason is the same as explained before.

We can also see that the routes 10 and 14 are the ones that produce the most errors, in fact analyzing row 10 and 14 of the confusion matrix we can see that for almost every class there are misclassified images.

### VGG




Here we can see the class 0 classified as 15 and the errors on rows 10 and 14. The errors of this network come from the same reason explained before.

## Error Analysis

At the end an analysis of the images misclassified has been done, almost the totality of them comes from images taken between two routes.
For example in some images of route 0 we could see route 15.

Image from the training set of class 00

Image from the test set of class 15

It's possible to see that the images are almost the same.
These errors could be fixed by removing these images, because they generate many errors.

# Fine tuning using an ImageNet pretrained network

This section will explain the fine tuning applied to the pretrained network AlexNet.

## Transfer Learning and Fine tuning

Transfer learning consists of taking features learned on one problem, and leveraging them on a new, similar problem.

The most common incarnation of transfer learning in the context of deep learning is the following workflow:

4. Take layers from a previously trained model.
5. Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
6. Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
7. Train the new layers on your dataset.

A last, optional step, is **fine-tuning**, which consists of unfreezing the entire model obtained above (or part of it), and re-training it on the new data with a very low learning rate. This can potentially achieve meaningful improvements, by incrementally adapting the pretrained features to the new data.

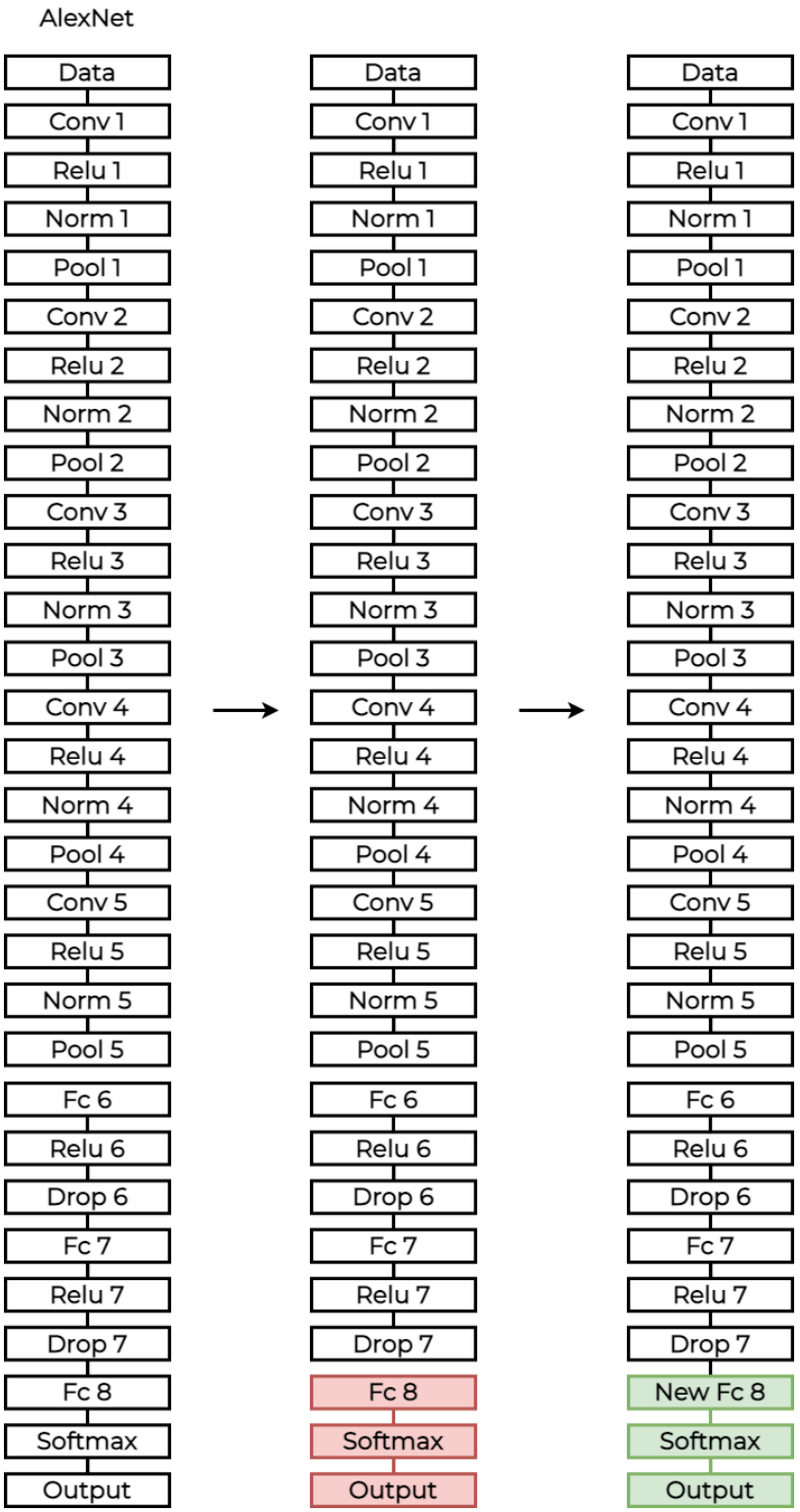
## Fine tuning applied to AlexNet

In this section we will explain the change applied to the AlexNet.

The thing we did was to remove the last three layers of the network, because they are configured for 1000 classes. These three layers must be fine-tuned for the new classification problem.

So we add again a fully connected layer with the right number of classes and some parameters tuned: weight and bias learn rate factor. These two parameters are useful if we want the network to learn faster in the new layers than in the transfered.
We also add the softmax layer and the output layer at the end.

With fine tuning it is also possible to change the weights of the first layers, but we decided to leave them unchanged.




When the new network is ready we set all the parameters like the optimizer, learning rate, the validation set, the mini batch size, the epochs and so on.

After that, we train the new network using the layers, training set and the options (learning rate, optimizer and so on).

# How the script works

This section will explain how the project works.

## Variables tuning

In the first part of the code it is possible to configure the code variables.

This part is useful to enable or disable some parts of the code and to choose which pretrained network to use.

### Print configuration

These variables enable or disable some part of the code.

8. Print random images of the training set (0 disabled / 1 enabled)

```
print_training_set = 0;
```

9. Print 12 random images of the test set (0 disabled / 1 enabled). For each of them it shows the number of the image, the prediction of the model and the correct class.

```
print_test_set = 0;
```

10. Print the confusion matrix (0 disabled / 1 enabled).

```
print_conf_matr = 0;
```

### Other variables

The last variable is the number of classes

```
numClasses = 16;
```

## Import the datasets

In the second part of the code there will be the import of all the images, using *imageDataStore*, a function that automatically labels all the images based on the folder names. The images will be stored into an *ImageDataStore* object.

So the program takes the test set images from the folder **TestSet** and it stores them into an *ImageDataStore* object. The same thing for the training and validation set.

## Image resize

The networks require different input sizes, in this section the image will be resized to fit the first input layer. To automatically resize the training, validation and test images before they are used as input by the network, the program creates augmented image datastores, it specifies the desired image size, and it uses these datastores as input arguments to activations.


## Fine-Tuning

We take all the layers except the last 3. The last three layers of the pretrained network are configured for 1000 classes. These three layers must be fine-tuned for the new classification problem.

After composing the network with the old and the modified layers. The last three layers have been replaced with a fully connected layer, a softmax layer, and a classification output layer.

Set the fully connected layer to have the same size as the number of classes in the new data.

To learn faster in the new layers than in the transferred layers, increase the `WeightLearnRateFactor` and `BiasLearnRateFactor` values of the fully connected layer.

```
freezedLayers = net.Layers(1:end-3);
layers = [
    freezedLayers
    fullyConnectedLayer(numClasses, ...
        'WeightLearnRateFactor',20, ...
        'BiasLearnRateFactor',20)
    softmaxLayer
    classificationLayer];
```

## Tune the training options

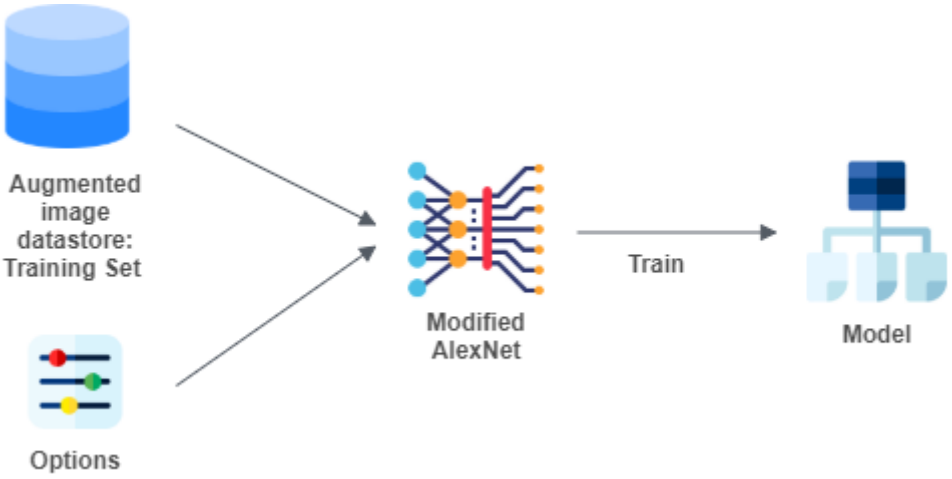
In this section we configure the training options of the network. Our optimizer is **sgdm**. The number of epochs is 6, the mini batch size is 100, the learning rate is 0.0001 and the validation frequency is 3.

In this section we have also to specify where to take the validation data. We used our validation set.

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize',100, ...
    'MaxEpochs',6, ...
    'InitialLearnRate',1e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',3, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

## Train the network

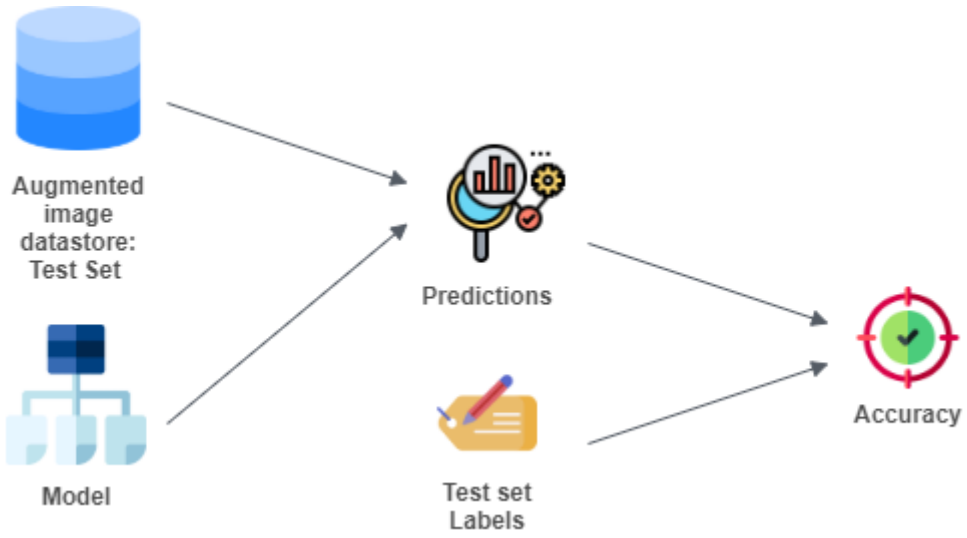
Train the network using the training set, the layers and the options configured before.




```
netTransfer = trainNetwork(augimdsTrain,layers,options);
```

## Classification

Perform the classification using the test set features and the model. After with the predictions check how many of them are right using the test set label and compute the accuracy.




```
[YPred,scores] = classify(netTransfer,augimdsTest);
YPred = double(YPred(:,1)) -1;
```

## Test and output analysis

In this section we want to analyze one of the best results obtained by the network.

We analyzed the accuracy, the images correctly classified vs. the number of images and the time elapsed.

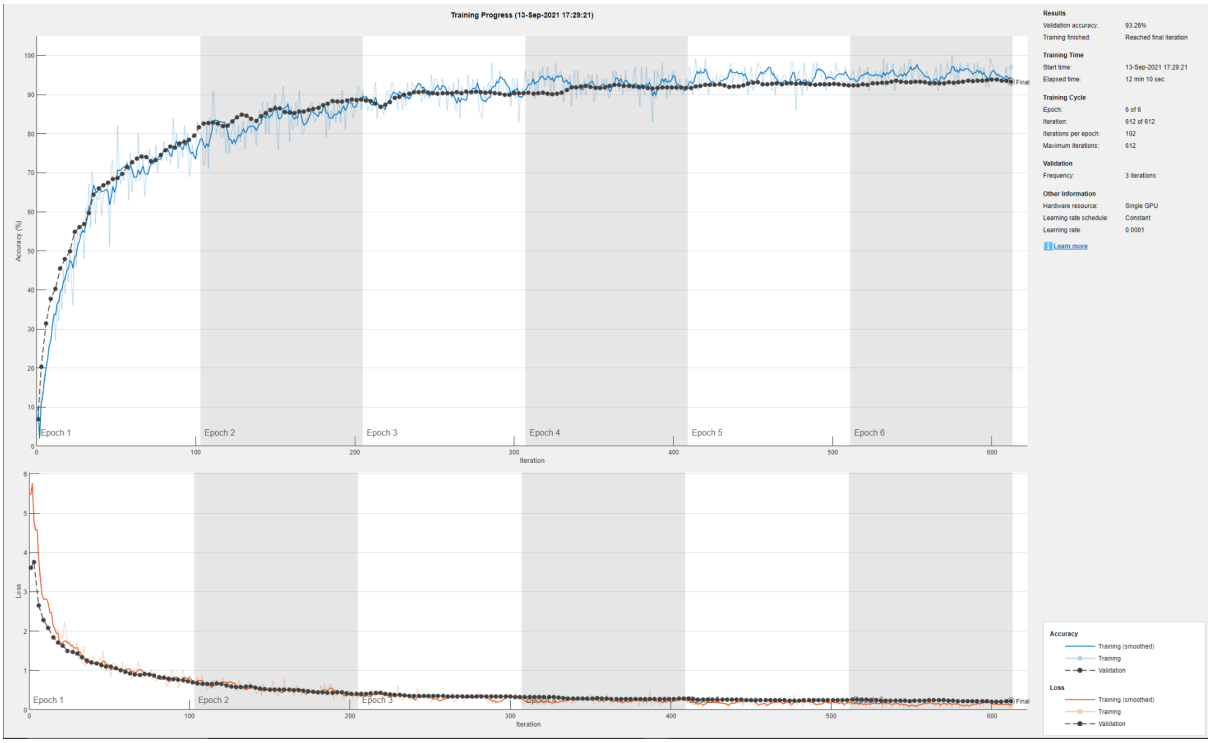
We also analyzed the confusion matrix to understand in which class there are more errors and why.

| Pretrained Network | Accuracy | Correct classified vs. No. Images | Time elapsed (s) | Time elapsed |
|---|---|---|---|---|
| AlexNet Fine Tuning | 93.02% | 5740/6171 | 752.53 | 12 min 32 s |

We also analyzed the accuracy and the loss graphs of the network.

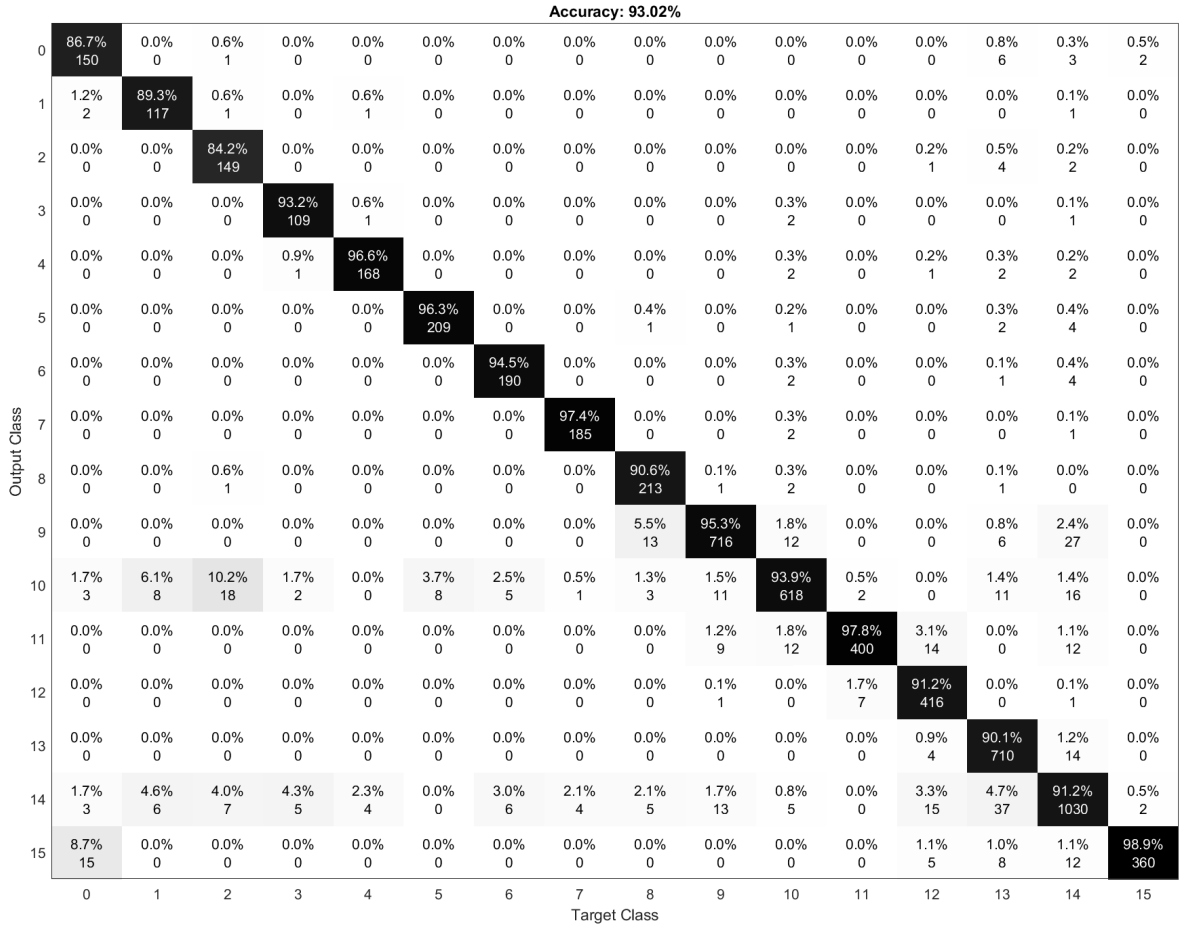
As we can see from the accuracy graph in the next page, the validation and the training lines follow almost the same path.

The same thing can be said about the loss graph. The validation set line and the training set line are almost the same. This means that there's no overfitting or underfitting.




## Confusion matrix

In this section we analyze the confusion matrix of the network.



In this confusion matrix it's possible to see that there are many images of the class 0 classified as 15. The main reason is that 15 has a link with class 0. The errors come from photos that are between 0 and 15, for example a photo of the route 0 in which it is possible to see the route 15 and so on.

It is also possible to notice images of route 2 classified as 10. Analyzing the map it is possible to see that 2 and 10 are linking routes.

The routes 10 and 14 are the ones that produce the most errors, in fact analyzing row 10 and 14 of the confusion matrix we can see that for almost every class there are misclassified images.

## Error Analysis

As we did with the other test, at the end we analyzed the misclassified images and we saw that the major part of them comes from images taken between two routes.

For example in some images of route 0 we could see route 15.

Image from the training set of class 00

Image from the test set of class 15

It's possible to see that the images are almost the same.

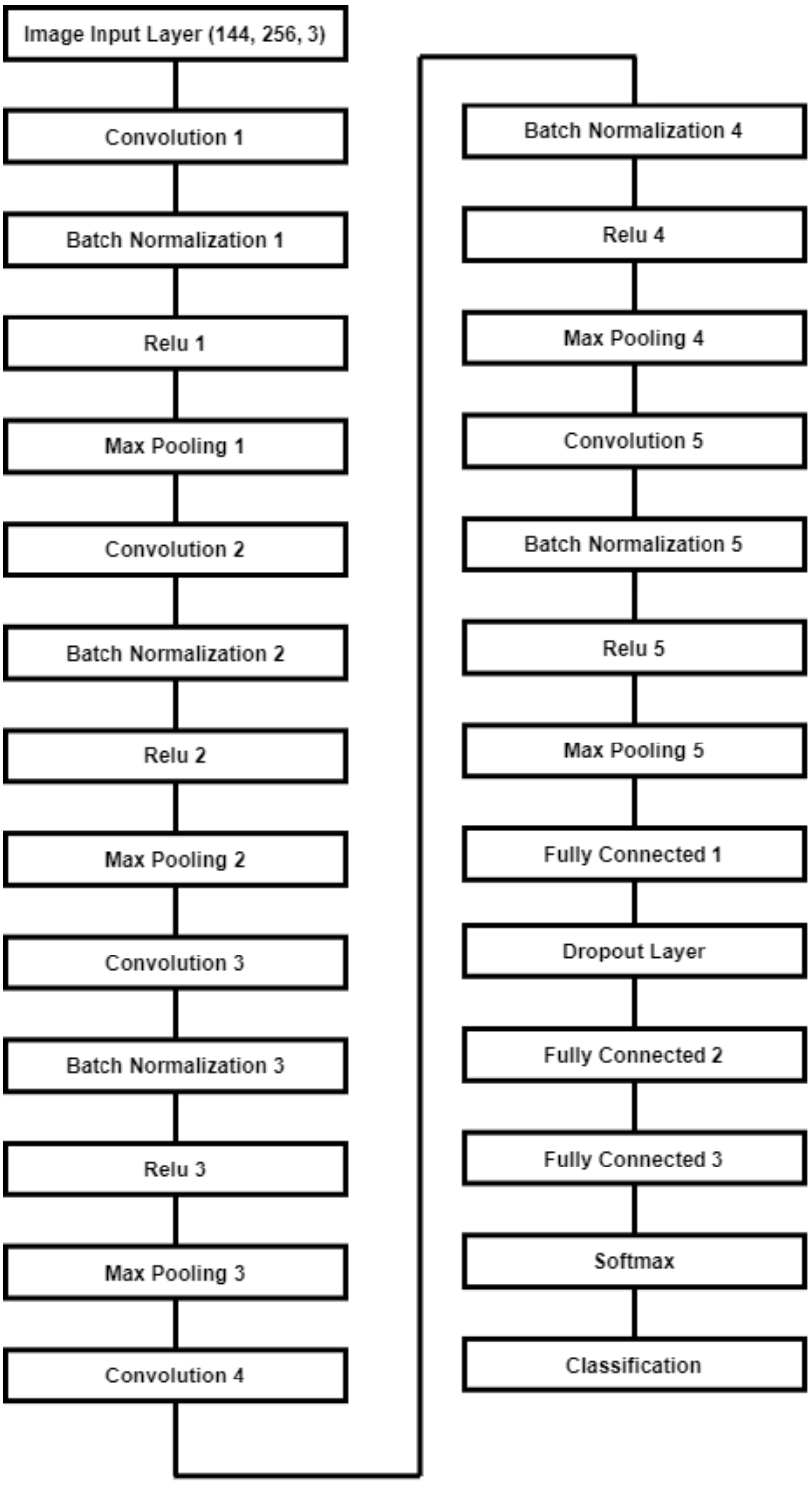
These errors could be fixed by removing these images, because they generate many errors.

# Creating a new network

The last test consists of creating a new convolutional neural network.

## Network Structure

The network we created has 27 layers.



In our first try we started with a simple network composed by: a convolution 2d layer + batch normalization layer + relu layer + max pooling layer, then it follows a fully connected layer + softmax and at the end the layer that classifies the feature produced from the convolutional part of the net.

After we added convolutional layers and changed some parameters, like the dimension of the filter, until we reached an optimal result as accuracy.

The final network created has 5 combinations of:


At the end there are 3 fully connected layers, a softmax layer and the classification one. Between the first two fully connected layers there's a dropout layer.

# How the script works

This section will explain how the project works.

## Variables tuning

In the first part of the code it is possible to configure the code variables. This part is useful to enable or disable some parts of the code and to choose which pretrained network to use.

### Print configuration

These variables enable or disable some part of the code.

1. Print random images of the training set (0 disabled / 1 enabled)

```
print_training_set = 0;
```

2. Print 12 random images of the test set (0 disabled / 1 enabled). For each of them it shows the number of the image, the prediction of the model and the correct class.

```
print_test_set = 0;
```

3. Print the confusion matrix (0 disabled / 1 enabled).

```
print_conf_matr = 0;
```

## Import the datasets

In the second part of the code there will be the import of all the images, using *imageDataStore*, a function that automatically labels all the images based on the folder names. The images will be stored into an *ImageDataStore* object.

So the program takes the test set images from the folder **TestSet** and it stores them into an *ImageDataStore* object. The same thing for the training and validation set.

## Image Resize

There is no need to resize the images because the first layer takes in input rgb images with size 256x144, exactly the ones that we have in the dataset.

## Network Creation

In this section we will create the network. So we will show which layers and what parameters are in the network.

```
layers = [
    imageInputLayer([144 256 3])

    convolution2dLayer(12,64,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2)

    convolution2dLayer(8,96,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2)
    convolution2dLayer(6,96,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2)

    convolution2dLayer(5,128,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2)

    convolution2dLayer(3,64,'Padding','same')
```

```
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)

    fullyConnectedLayer(128)
    dropoutLayer(0.25)
    fullyConnectedLayer(128)
    fullyConnectedLayer(16)
    softmaxLayer
    classificationLayer];
```

## Training options

We chose **sgdm** as an optimizer. We setted the mini batch size at 100, the number of epochs at 4, the learning rate at 0.001 and the validation frequency at 5.

Next we specify all the training options, the validation set is taken before from the training set .

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize',100, ...
    'MaxEpochs',4, ...
    'InitialLearnRate',0.001, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',5, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

## Train the network

Using the training set, the network and the training options we train the network.

```
netTransfer = trainNetwork(imdsTrain,layers,options);
```

## Classification

At the end we perform the classification.

```
[YPred,scores] = classify(netTransfer,imdsTest);
YPred = double(YPred(:,1)) -1;
```

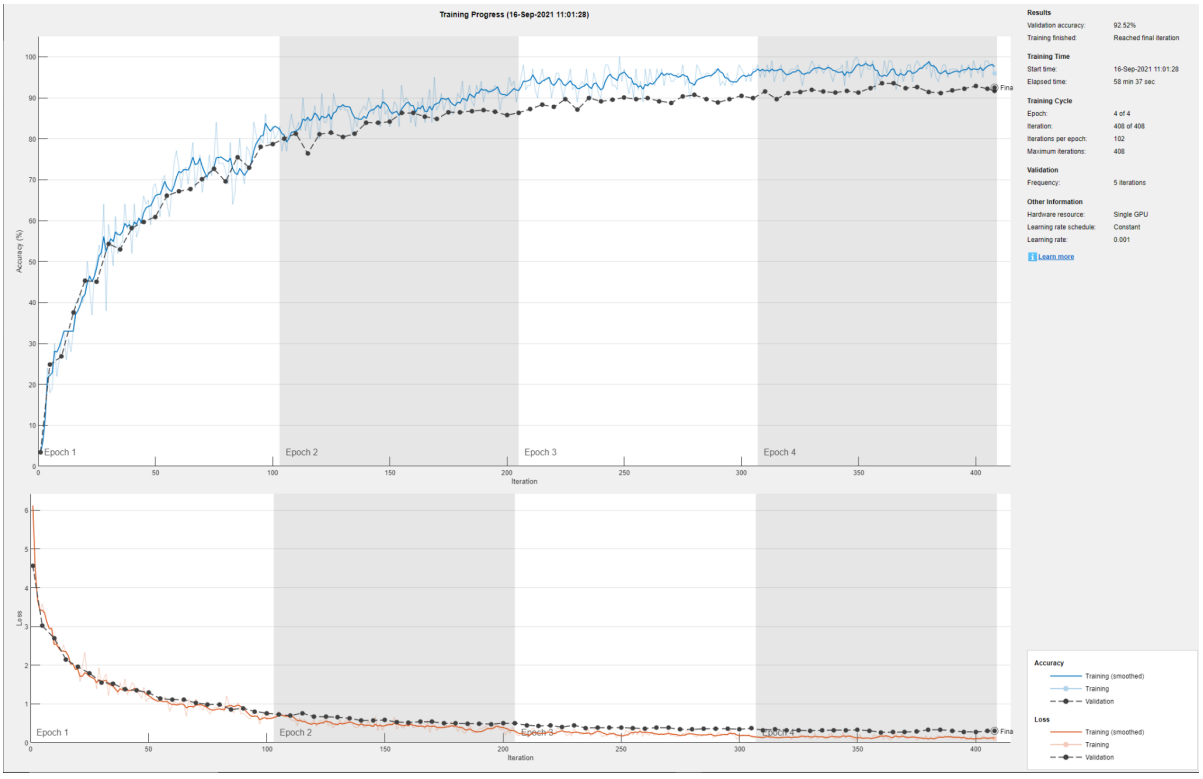
# Test and output analysis

In this section we want to analyze one of the best results obtained by the network. We analyzed the accuracy, the images correctly classified vs. the number of images and the time elapsed. We also analyzed the confusion matrix to understand in which class there are more errors and why.

The training of the network has been performed using an Nvidia 2060 GPU.

| Pretrained Network | Accuracy | Correct classified vs. No. Images | Time elapsed (s) | Time elapsed |
|---|---|---|---|---|
| New Network | 90.52% | 5585 / 6171 | 3685.58 | 1 h 1 m 25 s |

We also analyzed the accuracy and the loss graphs of the network.

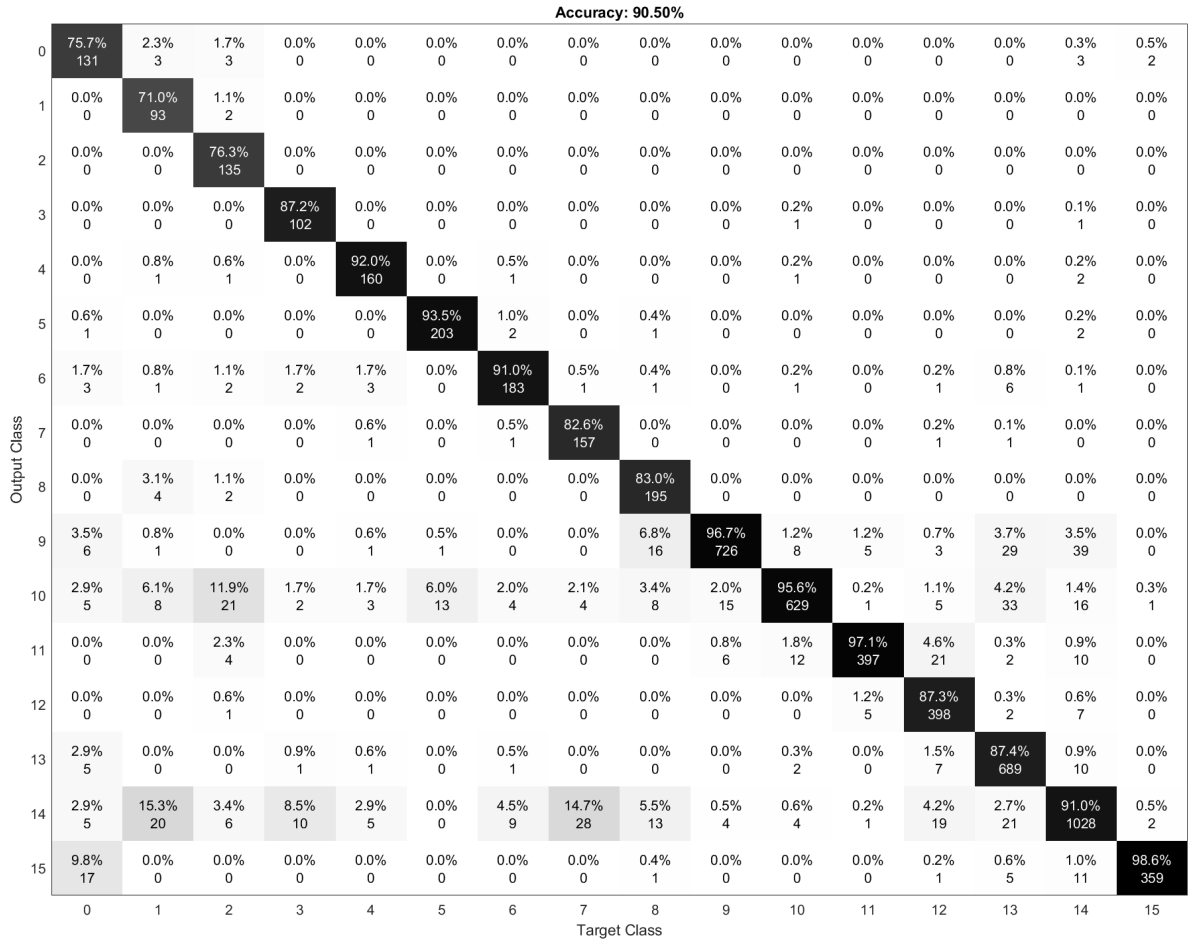



As we can see the accuracy graph shows that the validation and the training lines follow almost the same path.

The same thing can be said about the loss graph. The validation set line and the training set line are almost the same. This means that there's no overfitting or underfitting.

## Confusion matrix

In this section we analyze the confusion matrix of the network.




Even in this network we can see the classes 0 classified as 15 and many errors regarding rows 10 and 14. The reason for these errors is the same explained in previous chapters.

We can also see that in the first 3 classes this network is the worst in terms of accuracy (almost 75 vs almost 80).

# Error Analysis

As we did with the other test, at the end we analyzed the misclassified images and we saw that the major part of them comes from images taken between two routes.
For example in some images of route 0 we could see route 15.

Image from the training set of class 00

Image from the validation set of class 15

It's possible to see that the images are almost the same.

These errors could be fixed by removing these images, because they generate many errors.

# How to run the project

This section will explain how to run the project

## Preliminary steps

1. Clone the repository

```
git clone https://github.com/thisispivi/Deep-Learning-Supermarket
```

2. Open one of the matlab script file:

    a. `pretrained_networks.mlx`: the file with the feature extraction using pretrained networks and with the classification using linear SVM

    b. `fine_tuning.mlx`: the file with the fine tuning

    c. `new_network.mlx`: the file with the new network

3. Install the matlab Statistic and Machine Learning Toolbox Add On: Home > Add-On > Statistic and Machine Learning Toolbox
4. Install the matlab Deep Learning Toolbox Model for AlexNet Network Add On: Home > Add-On > Deep Learning Toolbox Model for AlexNet Network
5. Install the matlab Deep Learning Toolbox Model for ResNet-18 Network Add On: Home > Add-On > Deep Learning Toolbox Model for ResNet-18 Network
6. Install the matlab Deep Learning Toolbox Model for VGG-16 Network Add On: Home > Add-On > Deep Learning Toolbox Model for VGG-16 Network
7. Install the Plot Confusion Matrix Add On: Home > Add-On > Plot Confusion Matrix by Vane Tshitoyan

## Dataset organization

Here there are two options:

### Download our organized dataset

1. Download the organized dataset we used from this link.

### Download and manually organize the original dataset

1. Download the original dataset
2. Insert the images in the split folder
3. Run the `split.sh` files
4. Move the Training, Validation and Test set folders in the root of the project

## Variables configuration

The next step is to configure the variables of the first section.

## Run the script

The only thing left is to run the script